ANDROID UI

“Fizemos os botões na tela parecerem tão bons que você vai querer lambê-los.” (Steve Jobs)

CONTEÚDO

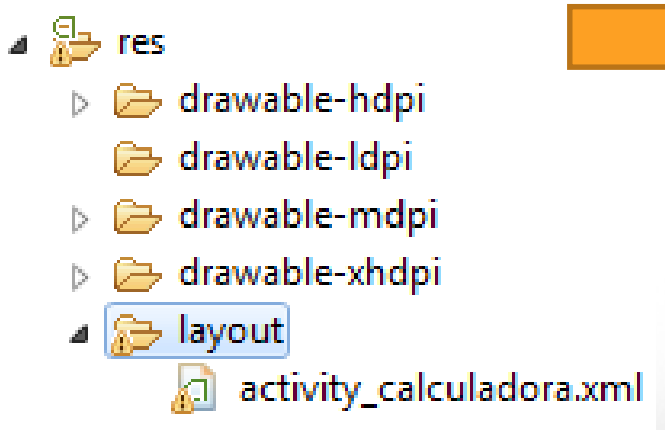
- LAYOUTS(VIEWGROUPS)
- VIEWS:
  - LISTA DE SELEÇÃO
  - PICKERS
  - CAIXAS DE DIÁLOGO
  - LISTs
  - MENUS

LAYOUTS – O QUE É?

- O layout é uma arquitetura para organizar a interface de usuário no Android
- O layout pode ser construído de duas formas: a) forma visual utilizando os recursos de *drag and drop* de views(widgets); b) de forma declarativa utilizando o arquivo XML que representa o layout;
- Elementos como views e viewgroups podem ser construídos no Android em tempo de execução;
- Sua declaração em arquivos XML permite uma melhor separação entre a parte de apresentação e o código que implementa os comportamentos.

LAYOUTS – LOCALIZAÇÃO

- Os layouts no Android ficam localizados no diretório ***res/layout*** do projeto;
- Todos os arquivos possuem a extensão XML;
- Quando um novo arquivo de layout é criado, ele é registrado na classe R.




```
public static final class layout {
    public static final int activity_calculadora=0x7f030000;
}
```

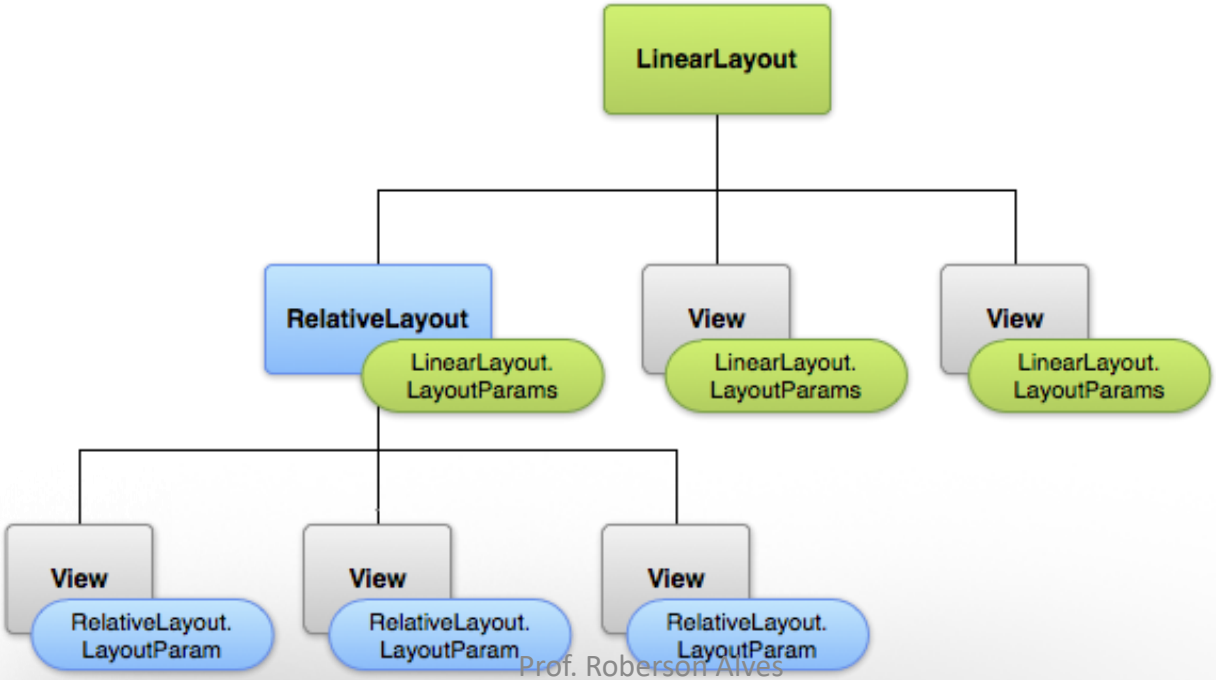
LAYOUTS – CARREGANDO

- A carga de um layout é feita através do método **setContentView()**;
- A carga deve ser feita dentro do callback **onCreate()** da activity;
- Não existe relação direta entre o nome do arquivo de layout e o nome da activity.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_calculadora);
}
```

LAYOUTS – PARÂMETROS

- Os layouts permitem a definição de uma série de atributos, normalmente no formato: ***layout_algumacoisa;***
- Estes atributos definem parâmetros de apresentação das views contidas no layout;



LAYOUTS – PARÂMETROS

- Dentre estes atributos estão os de definição de largura e altura;
- **Unidades de medida disponíveis:** px (pixels), dp (density-independent pixels), sp (scaled pixels based on preferred font size), in (inches), mm (millimeters);
- Geralmente utilizam-se as constantes:
  - **wrap_content:** ajustar o tamanho de acordo com o conteúdo;
  - **match_parent(fill_parent):** preencher, ou seja, usar o tamanho máximo permitido pelo layout.

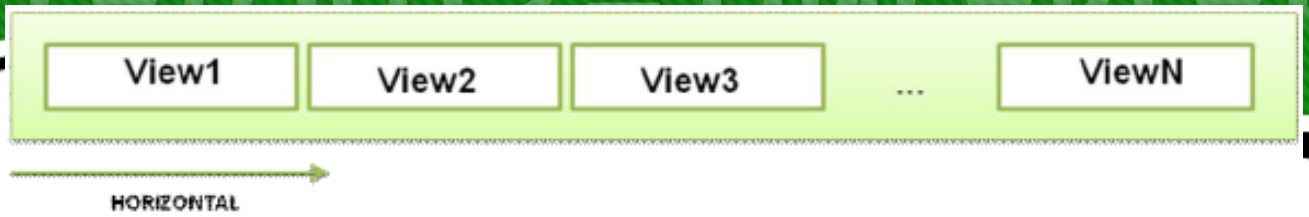
LAYOUTS - TIPOS

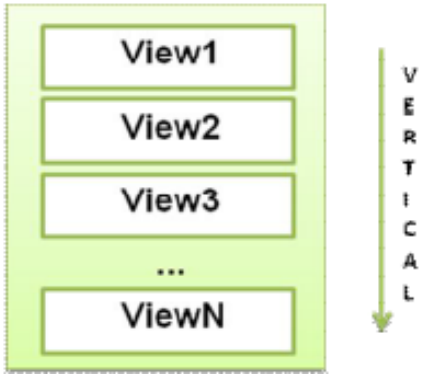
- A seguir, serão descritos os principais layouts e suas características:
  - **LinearLayout**
  - **FrameLayout**
  - **AbsoluteLayout(*deprecated*)**
  - **RelativeLayout**
  - **TableLayout**
  - **Fragments**

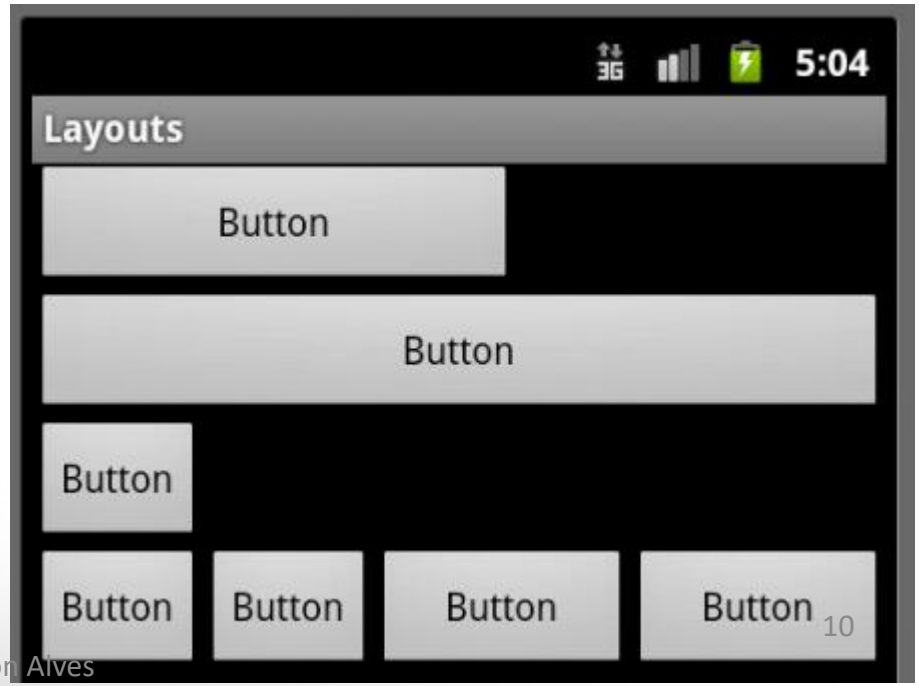
LAYOUTS – LINEARLAYOUT

- Este layout é utilizado para dispor seus componentes em uma única direção (por meio do atributo ***android:layout_orientation***): vertical ou horizontal;
- Este layout respeita as margens entre os seus filhos e alinhamento (ao centro, à esquerda ou à direita. Em Android, chamamos este atributo de ***android:layout_gravity***);
- É possível também atribuir, individualmente, pesos(***android:layout_weight***) para os componentes(views) para que estes possam ocupar o restante do espaço do layout.

LAYOUTS – LINEARLAYOUT



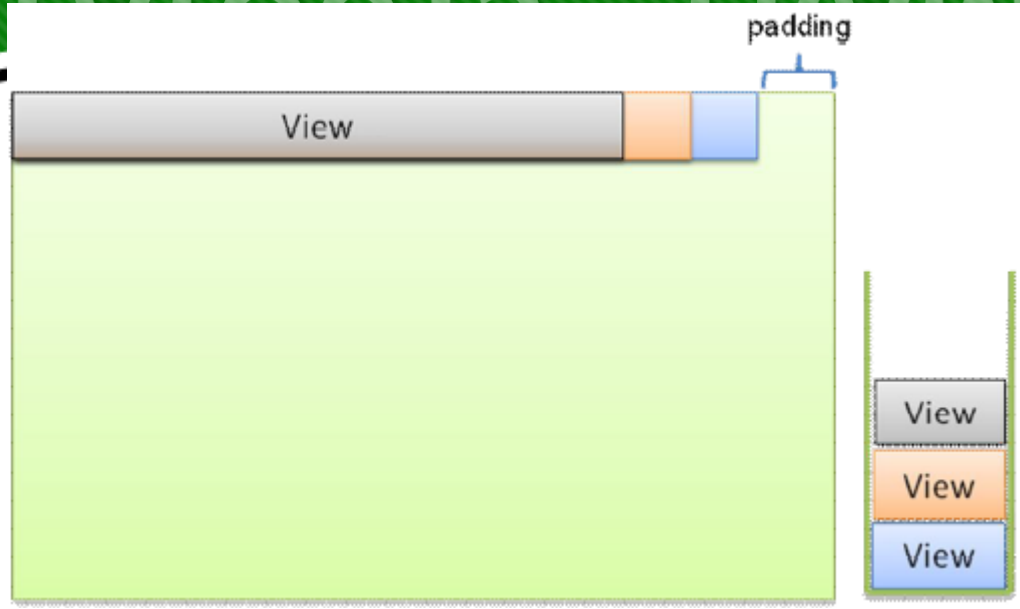


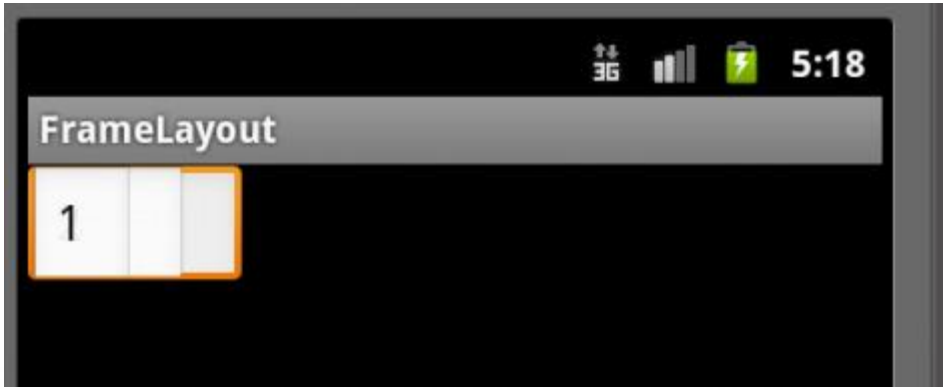


LAYOUTS – FRAMELAYOUT

- Arranja seus filhos de acordo com uma pilha de componentes que são adicionados, sendo que o topo da pilha contém o objeto que foi adicionado por último;
- Podemos utilizá-lo quando, por exemplo, queremos usar várias imagens, onde uma é trocada (sobreposta) pela outra (como um slide de imagens) conforme vão sendo adicionadas;
- O tamanho total de um FrameLayout é definido pelo seu maior filho mais o espaçamento (***padding***);
- Todos os componentes são agrupados no canto superior esquerdo do layout.

LAYOUTS – FRAMELAYOUT

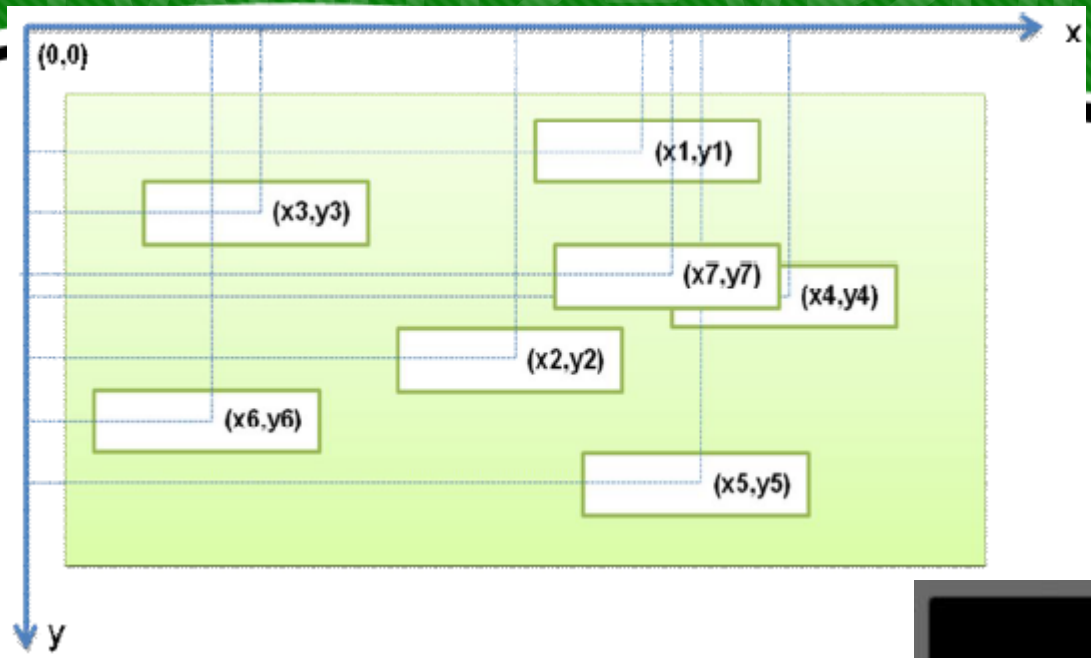




LAYOUTS – ABSOLUTELAYOUT

- Este tipo de layout organiza seus componentes de forma a implementar um plano cartesiano, no qual as suas posições x e y devem ser definidas para que estes possam ser posicionados de forma absoluta;
- Caso os componentes não declarem as suas posições explicitamente, eles serão dispostos na posição (0,0);
- ***Cuidado ao utilizar este layout para evitar comportamentos inesperados e sobreposição de componentes***.

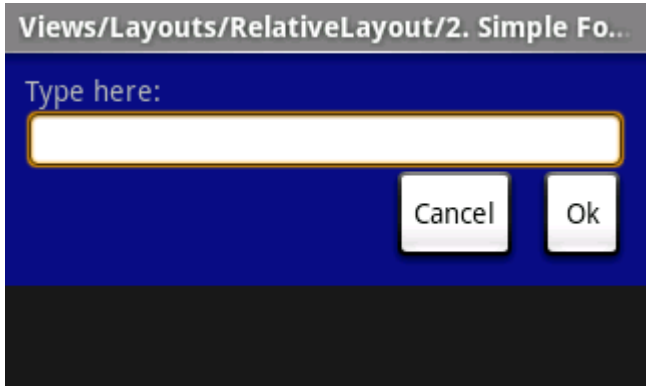
LAYOUTS – ABSOLUTELAYOUT



LAYOUTS – RELATIVELAYOUT

- Neste layout, os componentes são ajustados através de relacionamentos entre si ou ao seu pai

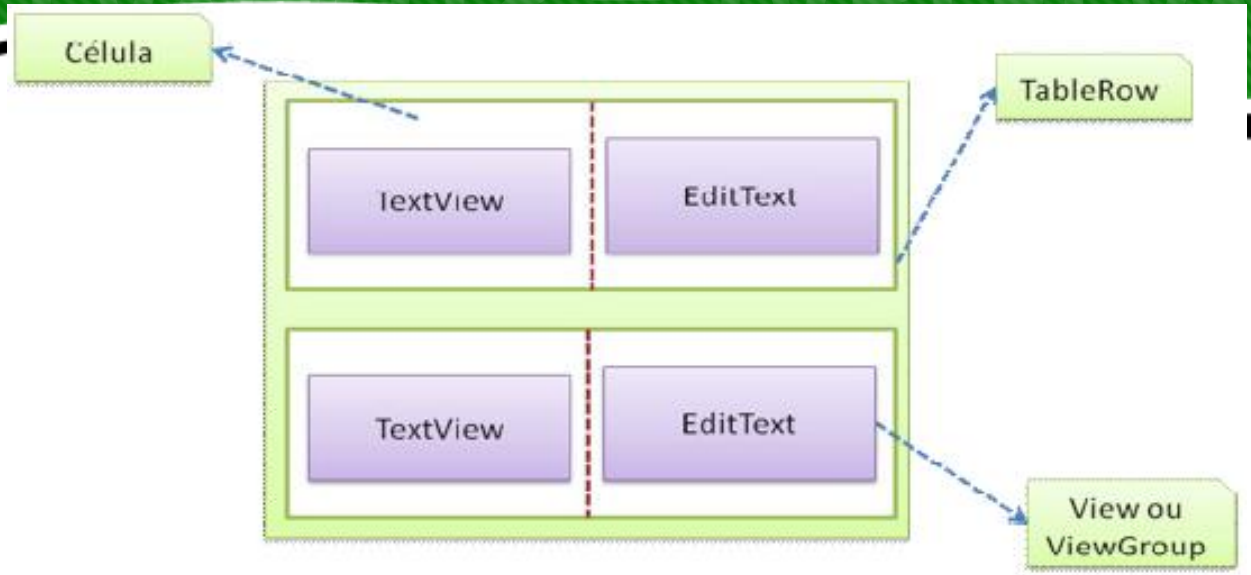
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:background="@drawable/blue"
                android:padding="10px" >

    <TextView android:id="@+id/label"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Type here:" />

    <EditText android:id="@+id/entry"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:background="@android:drawable/editbox_background"
              android:layout_below="@id/label" />

    <Button android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/entry"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10px"
            android:text="OK" />

    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@id/ok"
            android:layout_alignTop="@id/ok"
            android:text="Cancel" />
</RelativeLayout>
```



LAYOUTS – TABLELAYOUT

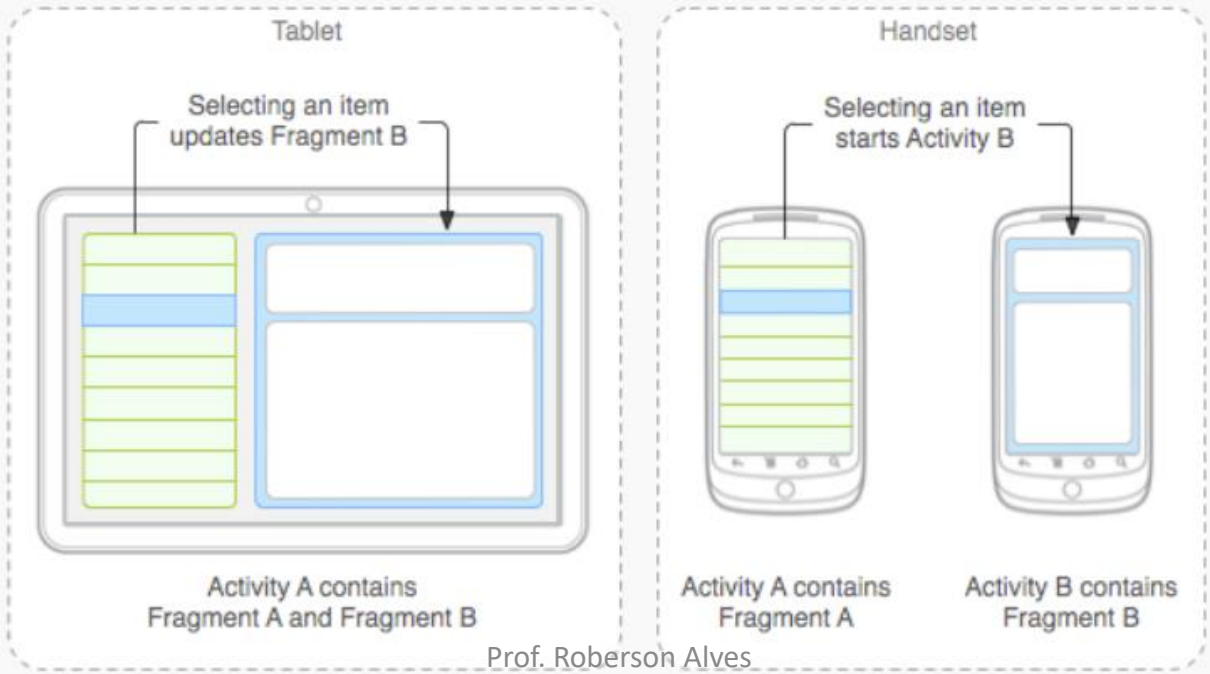
- O TableLayout organiza seus filhos em linhas e colunas
- Cada filho é representado pelo componente **TableRow** (que também é uma espécie de LinearLayout restrito na direção horizontal) que permite que uma ou mais células sejam adicionados horizontalmente, sendo que cada célula pode conter somente um único View;
- Caso o atributo ***android:layout_width*** e ***android:layout_height*** não sejam declarados, este layout irá forçar para que a largura de cada componente seja automaticamente **FILL_PARENT** e a altura **WRAP_CONTENT.**

LAYOUTS – TABLELAYOUT



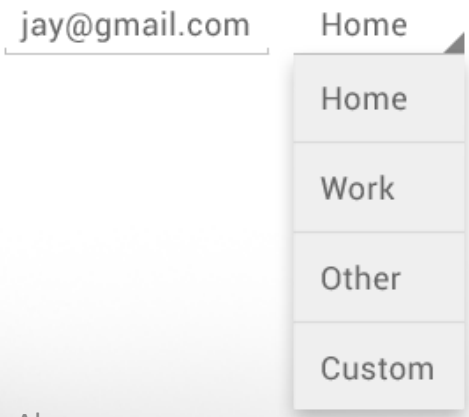
FRAGMENTS

- **Recurso introduzido no Android 3.0(Honeycomb);**
- **Pode representar comportamento ou uma porção da interface do usuário;**
- **Funciona como uma seção modular dentro da *Activity*;**
- **Permite construir interfaces gráficas mais dinâmicas e flexíveis.**



SPINNER

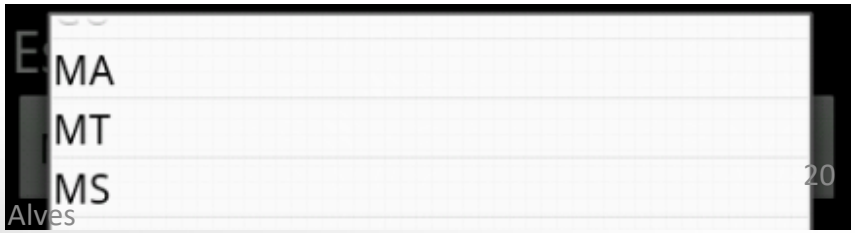
- O Spinner é um tipo de view que permite o usuário selecionar um valor de uma lista de valores, muito parecido com uma lista de seleção(dropdown);
- Para preencher um objeto do tipo spinner é necessário utilizar um adaptador, indicando a origem dos dados;
- Os adapters podem ser: SpinnerAdapter, ArrayAdapter ou CursorAdapter;



SPINNER – EXEMPLO 01

- Alimentando as opções de seleção através de um vetor(array) de Strings


```
// criando vetor de estados
final String[] estados = new String[] {
        "AC", "AL", "AP", "AM", "BA", "CE", "DF", "ES", "GO", "MA",
        "MT", "MS", "MG", "PA", "PB", "PR", "PE", "PI", "RJ", "RN", "RS",
        "RO", "RR", "SC", "SP", "SE", "TO"
};

// recuperando widget spinner do layout xml
Spinner spinner1 = (Spinner)findViewById(R.id.spinner1);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, estados);
spinner1.setAdapter(adapter);
spinner1.setSelection(1);
```



SPINNER – EXEMPLO 02

- Alimentando as opções a partir de um arquivo XML: **/res/values/strings.xml**

```
<string-array name="paises">
    <item>Brasil</item>
    <item>Estados Unidos</item>
    <item>Argentina</item>
    <item>Espanha</item>
</string-array>
```


```
// recuperando widget spinner do layout xml
Spinner spinner2 = (Spinner)findViewById(R.id.spinner2);
ArrayAdapter<CharSequence> adapter1 = ArrayAdapter.createFromResource(
        this, R.array.paises, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner2.setAdapter(adapter1);
spinner2.setSelection(1);
```

SPINNER

- O valor ***simple_spinner_item***, é um layout fornecido pela plataforma. É o layout padrão para exibição da lista de valores do spinner;
- É necessário chamar o método ***setDropDownViewResource(int)*** passando o parâmetro ***simple_spinner_dropdown_item*** para exibir a lista em forma de um dropdown choice. Este layout também é fornecido pela plataforma;
- O layout de exibição dos valores no spinner pode ser customizado pelo desenvolvedor.

SPINNER – ALGUNS MÉTODOS

- **Métodos:**
  - Object getSelectedItem()
    - **retorna o item selecionado**
  - long getSelectedItemId()
    - **retorna o id do item selecionado**
  - int getSelectedItemPosition()
    - **retorna a posição do item selecionado no array fornecido para ArrayAdapter**
  - void setPrompt(String)
    - **determina o prompt da lista de opções**
  - void setPromptId(int resId)
    - **determina o prompt a partir de um R.string.id_string**
  - void setSelection(int position)
    - **determina o item atualmente selecionado**

SPINNER – ONITEMSELECTEDLISTENER

- Para responder as ações (eventos) do usuário deve implementar a interface **AdapterView.OnItemSelectedListener**;
- Ao implementar a interface temos disponíveis os métodos:
  - public void onItemSelected(AdapterView<?> parent, View view, int pos, long id)
    - **Método é chamado quando um item é selecionado**
  - public void onNothingSelected(AdapterView<?> parent)
    - **Método chamado quando nenhum item é selecionado**
- Vincula-se ao spinner a implementação da interface AdapterView.OnItemSelectedListener através do método setOnItemSelectedListener

SPINNER – ONITEMSELECTEDLISTENER

Declarando o listener

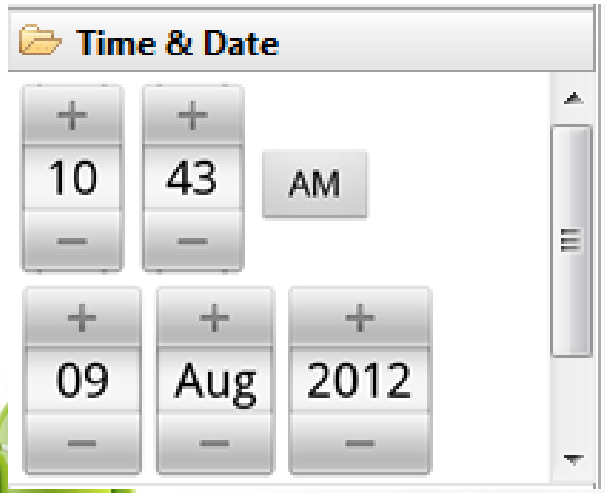
```
public class SpinnerActivity extends Activity implements OnItemSelectedListener {
    ...

    public void onItemSelected(AdapterView<?> parent, View view,
            int pos, long id) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }

    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

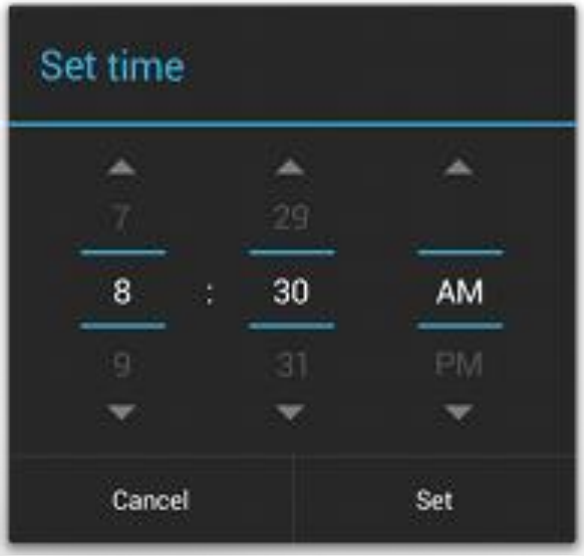
Vinculando o listener ao spinner

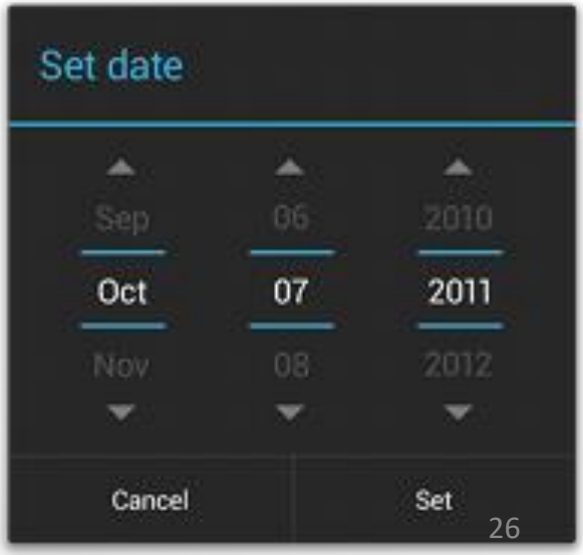
```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

PICKERS – RESPONDENDO AO USUÁRIO

- Pickers são caixas de diálogo *ready-to-use*(prontas para usar) que o Android disponibiliza;
- Permitem a seleção/indicação de data e hora;







PICKERS - RESPONDENDO AO USUÁRIO

- Para responder a ações do usuário de alteração de data ou hora deve-se implementar as interfaces: ***DatePickerDialog.OnDateSetListener*** ou ***TimePickerDialog.OnTimeSetListener***;
- Os métodos que devem ser implementados são:
  - onTimeSet(TimePicker view, int hourOfDay, int minute)
    - **Acionado quando ocorre a alteração de hora**
  - onDateSet(DatePicker view, int year, int month, int day)
    - **Acionado quando ocorre a alteração de data**

PICKERS – RECUPERANDO OS VALORES

- **DatePicker**

```
DatePicker dpdata = (DatePicker) findViewById(R.id.dpdata);
Toast.makeText(
        this,
            dpdata.getDayOfMonth() + "/" +
            dpdata.getMonth() + "/" +
            dpdata.getYear(), Toast.LENGTH_LONG).show();
```

- **TimePicker**

```
TimePicker tphora = (TimePicker) findViewById(R.id.tphora);
Toast.makeText(
        this,
            tphora.getCurrentHour() + ":" +
            tphora.getCurrentMinute(), Toast.LENGTH_LONG).show();
```

CAIXAS DE DIÁLOGO - DIALOG

- São pequenas janelas exibidas na frente da atividade(activity) corrente;
- A janela da atividade(activity) perde o foco e o usuário pode interagir com a caixa de diálogo;
- A classe base para definição de caixas de diálogos é **android.app.Dialog**;
- Alguns tipos de caixas de diálogo: AlertDialog, ProgressDialog, DatePickerDialog e TimePickerDialog.

CAIXAS DE DIÁLOGO - DIALOG

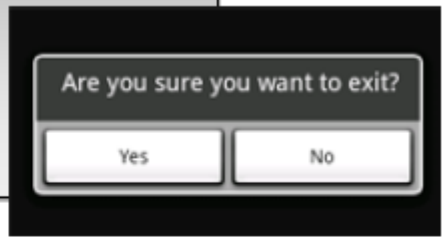
- Recomendado usar os métodos a seguir para que a Activity gerencie o ciclo de vida do Dialog
  - **boolean showDialog(int id, Bundle)**: Tenta mostrar o Dialog para o id chamando onCreateDialog() se necessário, e em seguida onPrepareDialog(). Retorna true em caso de sucesso.
  - **void dismissDialog(int id)**: Fecha o Dialog criado por showDialog() com o id especificado. Se nenhum Dialog com o id tiver sido mostrado lança IllegalArgumentException.
  - **void removeDialog(int id)**: Remove qualquer referência para o Dialog especificado. Caso esteja sendo exibido, ele é fechado antes.
  - **Dialog onCreateDialog (id, Bundle)**: Callback chamado quando um Dialog deve ser criado para o id especificado. Implementado pelo desenvolver.
  - **void onPrepareDialog(id, Dialog, Bundle)**: Callback que permite que o Dialog seja preparado antes de ser apresentado. Por exemplo, configurar alguma variável.

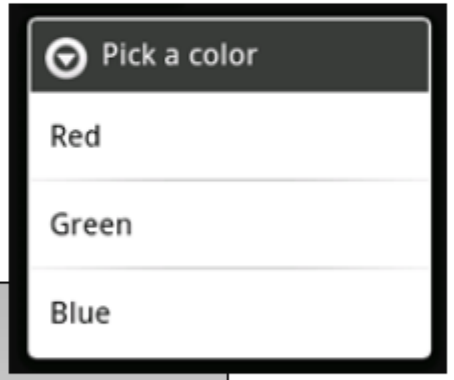
CAIXAS DE DIÁLOGO - ALERTDIALOG

- Conteúdo e botões de escolha personalizados
- Classe AlertDialog.Builder para construir alertas
  - **setIcon(int resId)**: Determina o ícone a ser mostrado
  - **setTitle(String)**: Determina o título do alerta
  - **setMessage(String)**: Mensagem a ser mostrada no interior do alerta
  - **setPositiveButton(String, listener)**: Texto do botão positivo (Sim, Yes, Ok, etc)
  - **setNegativeButton(String, listener)**: Texto do botão negativo (Não, No, etc)
  - **setItems(String [], listener)**: Items a serem mostrados para usuário selecionar
  - **setSingleChoiceItems(String [], int checkedItem, listener)**: Determina lista de RadioButtons a serem mostrados ao usuário
  - **setCancelable(boolean)**: Alerta cancelável ou não. Cancelável significa que usuário não pode fechar com botão voltar.
  - **show()**: Exibe o alerta para o usuário
  - **cancel()**: Cancela o alerta

CAIXAS DE DIÁLOGO – ALERTDIALOG - EXEMPLO

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?");
builder.setCancelable(false);
builder.setPositiveButton("Yes", ...);
builder.setNegativeButton("No", ...);
AlertDialog alert = builder.create();
```






```
CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items,
new DialogInterface.OnClickListener() { ... } );
builder.show();
```

ANDROID.WIDGET.ADAPTERVIEW

- Subclasse de ViewGroup onde as views internas são determinadas por um Adapter
- Útil quando queremos exibir dados armazenados(seja em arrays ou BD)
- Gallery, ListView e Spinner são exemplos de AdapterView
- Objetos deste tipo tem duas responsabilidades:
  - Preencher o layout com dados
  - Tratar as seleções do usuário

ANDROID.WIDGET.ADAPTERVIEW

- Preenchendo o layout com dados(spinner):

```
// recupera o Spinner da tela para vincular ele a um ArrayAdapter
Spinner s1 = (Spinner) findViewById(R.id.spinner1);
// cria um adapter a partir de um resourse, neste caso strings.xml, R.array.colors
ArrayAdapter adapter = ArrayAdapter.createFromResource( this, R.array.colors, android.R.layout.simple_spinner_item);
// define o padrão de layout a ser apresentado no adapter
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// vincula o adapter ao spinner
s1.setAdapter(adapter);
```

ANDROID.WIDGET.ADAPTERVIEW

- Tratando as seleções do usuário

```
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int position, long id)
    {
        // Display a messagebox.
        Toast.makeText(mContext,"You've got an event",Toast.LENGTH_SHORT).show();
    }
};

// Now hook into our object and set its onItemClickListener member
// to our class handler object.
mHistoryView = (ListView)findViewById(R.id.history);
mHistoryView.setOnItemClickListener(mMessageClickedHandler);
```

ANDROID.WIDGET.LISTVIEW

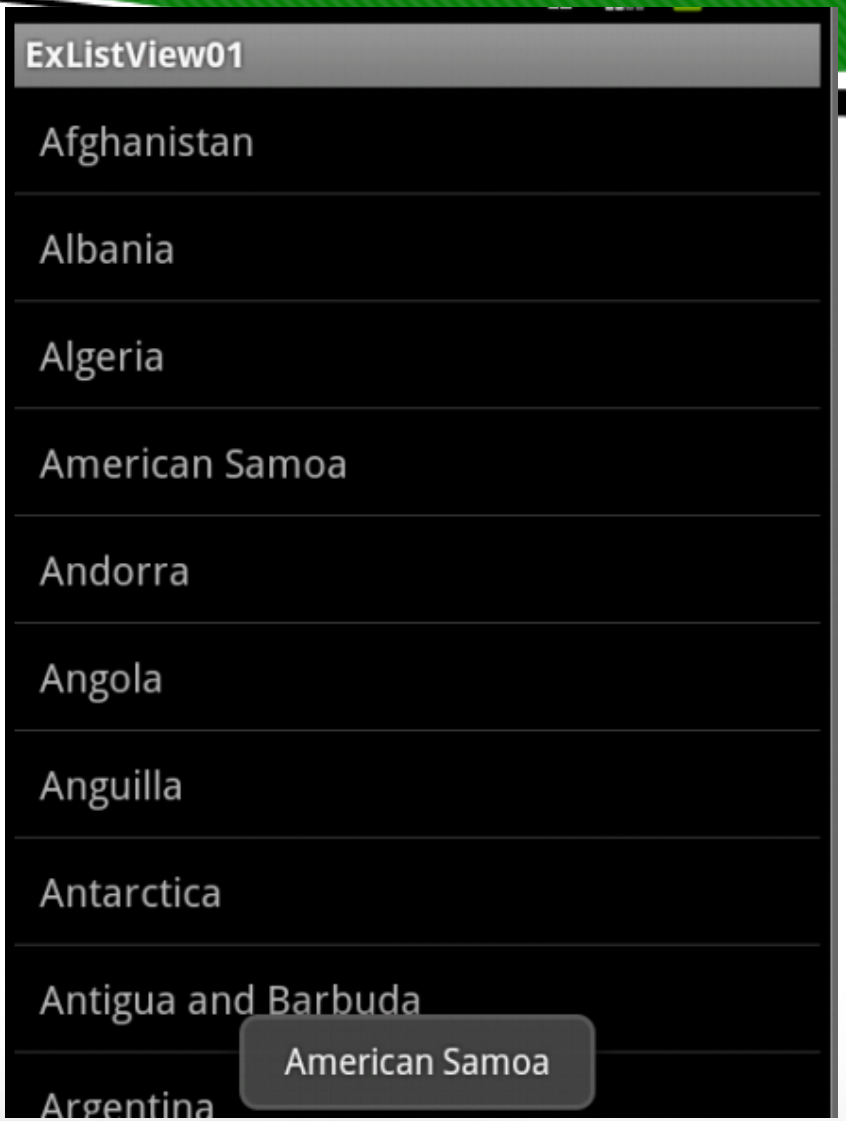
- É um tipo de ViewGroup que cria uma lista de itens
- Os itens são adicionados ao ListView através de um ListAdapter
- ListAdapter , estende Adapter, e é a ponte entre a lista e os dados exibidos
- **ListActivity** é uma tela para listar itens de um ListView – *já vem com um ListView pronto*
- Os itens estão amarrados a uma fonte de dados que pode ser um array ou um Cursor

LISTVIEW E LISTACTIVITY

```
13 public class ExListView01Activity extends ListActivity {
14     static final String[] COUNTRIES = new String[] { "Afghanistan", "Albania",
15             "Algeria", "American Samoa", "Andorra", "Angola", "Anguilla",
16             "Antarctica", "Antigua and Barbuda", "Argentina", "Armenia",
17             "Aruba", "Australia", "Austria", "Azerbaijan", "Bahrain",
18             "Bangladesh", "Barbados", "Belarus", "Belgium", "Belize", "Benin",
19             "Bermuda", "Bhutan", "Bolivia", "Bosnia and Herzegovina",
```

```
74     public void onCreate(Bundle savedInstanceState) {
75         super.onCreate(savedInstanceState);
76
77         // carregando a lista estática de paises ...
78         // não carregamos um layout geral, o ListView preencherá a tela
79         setListAdapter(new ArrayAdapter<String>(this, R.layout.item_lista,
80                 COUNTRIES));
81
82         // String[] countries = getResources().getStringArray(R.array.countries_array);
83         // setListAdapter(new ArrayAdapter<String>(this, R.layout.item_lista, countries
84
85         ListView lv = getListView();
86         lv.setTextFilterEnabled(true);
87
88         // método para tratar o clique do usuário
89         lv.setOnItemClickListener(new OnItemClickListener() {
90             public void onItemClick(AdapterView<?> parent, View view,
91                     int position, long id) {
```

LISTVIEW E LISTACTIVITY



MENUS

- Menus são componentes comumente utilizado pelos usuários
- A API do Android suporta o conceito de menus
- O Android suporta os seguintes tipos de menus:
  - **Menu de opções e barra de ações:** É primeira opção de menu de uma Activity. Permite definir ações que tem impacto global na aplicação. Para acionar este menu o usuário deve pressionar o botão “Menu”. A exibição das opções de menu como barra de ações acontece a partir do Android 3.0.
  - **Menu de contexto:** É um tipo de menu flutuante que só aparece quando o usuário realizar um clique longo sobre um elemento de tela. Prove ações que afetarão o conteúdo selecionado.
  - **Menu popup:** Exibe uma lista de itens, ancorado na View que acionou o menu. Útil para prover ações customizadas para determinado conteúdo ou ações secundárias para um botão em tela.

MENUS – MENU DE OPÇÕES

- São exibidas opções relativas ao contexto da atividade(activity) em uso

MENUS – MENU DE OPÇÕES

- Para adicionar opções de menu deve-se sobrescrever o método:
  - **public boolean onCreateOptionsMenu(Menu menu)**
  - **Menu é um objeto do tipo android.view.Menu**
  - **Através do método public abstract MenuItem add (int groupId, int itemId, int order, CharSequence title)**
    - **groupId**: Define a qual grupo o menu deve pertencer.
    - **itemId**: Identificador único para o item. O valor Menu.NONE é utilizado caso não seja necessário um identificador.
    - **order**: Indica em qual ordem o item deve aparecer. O valor 0 (ZERO) caso não seja necessário ordenar.
    - **title**: Representa o texto que aparecer no item.

MENUS – MENU DE OPÇÕES

- Para responder ao clique em uma opção do menu é utilizado deve-se sobrescrever o método.
  - **public boolean onMenuItemSelected(int featureId, MenuItem item)**
- Adicionando submenus através do método *addSubMenu(CharSequence title)*
- Útil para adicionar opções a um menu existente
- O método **onPrepareOptionsMenu(Menu menu)** pode ser utilizado para exibir ou não opções do menu. O método é chamado toda vez que o menu é exibido.

MENUS – MENU DE OPÇÕES

```
10 public class Menu01Activity extends Activity {
11
12     private final int CRIAR = 0;
13     private final int PESQUISAR = 1;
14     private final int PROXIMO = 2;
15     private final int SOBRE = 3;
16
17     /** Called when the activity is first created. */
18     @Override
19     public void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.main);
22     }
```

Declarando constantes para o menu

Implementando criação do menu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // adicionando menu principal
    menu.add(0, CRIAR, 0, "Criar");
    menu.add(0, PESQUISAR, 0, "Pesquisar");
    menu.add(0, PROXIMO, 0, "Próximo");

    // adicionando um submenu
    SubMenu sub = menu.addSubMenu(0, SOBRE, 0, "Sobre");
    sub.add("Autor");
    sub.add("Informações");

    return true;
}
```

MENUS – MENU DE OPÇÕES

Respondendo ao clique do usuário

```
@Override
public boolean onMenuItemSelected(int featureId, MenuItem item) {
    switch (item.getItemId()) {
    case CRIAR:
        Toast.makeText(this, "Teste da Opção Criar", Toast.LENGTH_LONG)
                .show();
        break;
    case PESQUISAR:
        Toast.makeText(this, "Teste da Opção Pesquisar", Toast.LENGTH_LONG)
                .show();
        break;
    case PROXIMO:
        Toast.makeText(this, "Teste da Opção Próximo", Toast.LENGTH_LONG)
        .show();
        break;
    }
    return false;
}
```

MENUS – MENU DE CONTEXTO

- Para responder as opções selecionadas no menu de contexto implemente o método .
- Para prover um menu de contexto é necessário:

  1) Registrar a view que apresentará o menu de contexto com o método **registerForContextMenu**

```
final ListView listContatos = (ListView) findViewById(R.id.listContatos);
final ListView listAgendamentos = (ListView) findViewById(R.id.listAgendamentos);
registerForContextMenu(listAgendamentos);
registerForContextMenu(listContatos);
```

MENUS – MENU DE CONTEXTO

2) Implemente o método **onCreateContextMenu()**

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

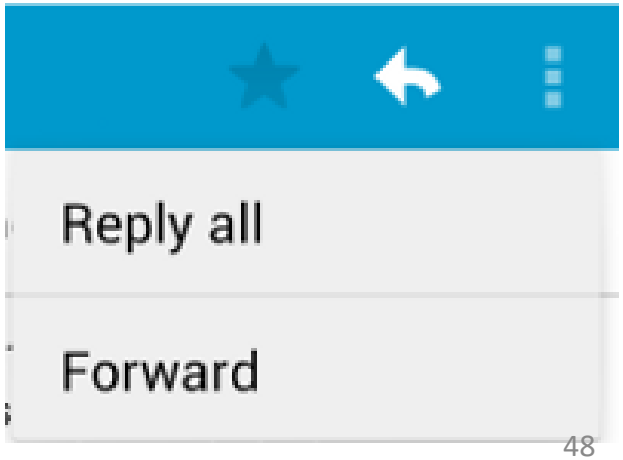
MENUS – MENU DE CONTEXTO

3) Para responder as opções selecionadas(pelo usuário) no menu de contexto implemente o método **onContextItemSelected()**

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

MENUS – MENU POPUP

- É um tipo de menu modal que fica ancorado na view;
- É uma espécie de menu dropdown;
- É útil para prover um menu relacionado a um conteúdo específico;
- Pode ser utilizado também para prover opções de comando secundárias ao menu principal;
- Disponível somente a partir da API 11.



MENUS – MENU INFLATER

- Recurso que permite inflar/carregar um menu a partir de um arquivo de layout XML;
- Isso permite uma maior customização do menu e maior flexibilidade na adição e remoção de itens;
- Permite também a internacionalização do menu.

MENUS – MENU INFLATER

1) Criar um objeto do tipo MenuInflater;
2) Indicar o resource que representa o menu(normalmente localizado no diretório menu).

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_principal, menu);
    return true;
}
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_lanca_aula"
        android:title="Lançar Aulas"
        android:orderInCategory="100" />
    <item android:id="@+id/menu_sobre_sistema"
        android:title="Sobre o Sistema"
        android:orderInCategory="101" />
    <item android:id="@+id/menu_sair"
        android:title="Sair"
        android:orderInCategory="102" />
</menu>
```

REFERÊNCIAS

- Android Developers. Layouts. 2012. Disponível em: <http://developer.android.com/guide/topics/ui/declaring-layout.html>. Acesso em: 04 jul. 2012.
- Android Developers. User Interface. 2012. Disponível em: < http://developer.android.com/guide/topics/ui/index.html>. Acesso em: 03 jul. 2012.
- LECHETA, Ricardo R. Google android: aprenda a criar aplicações para dispositivos móveis com o android SDK. 2. ed., rev. e ampl. São Paulo: Novatec, 2010. 608 p. ISBN 9788575222447.
- MURPHY, Mark L. Beginning android. New York, USA: Apress, 2009. xxii, 361 p. ISBN 9781430224198.