PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

Tratamento de Eventos Sofisticados

Professor: Danilo Giacobo

OBJETIVOS DA AULA

- Conhecer e usar o evento de toque
- Conhecer e usar o evento de foco
- Conhecer e usar o menu de contexto
- Como usar um menu simples
- Como utilizar uma ActionBar
- Utilizar os eventos de seleção de itens de uma lista

INTRODUÇÃO

- Na aula anterior sobre eventos, foram apresentados apenas os eventos básicos de clique em botões, sendo que estes correspondem à maioria das interações que acontecem nos aplicativos móveis.
- A variedade de eventos existentes na plataforma Android é muito grande, contemplando eventos de toque na tela, teclas pressionadas, troca de foco entre componentes, avançando para eventos mais complexos, como o uso de menus simples e de contexto e até mesmo eventos automáticos gerados automaticamente de tempo em tempo, como os usados em Timers.
- Para os testes dos eventos propostos nesta aula, criaremos um novo projeto Android chamado UsandoEventosAvancados, que terá um arquivo de interface gráfica (activity principal.xml) e uma classe *Activity* (PrincipalActivity.java). Os mesmos serão implementados ao longo da aula.

EVENTO DE TOQUE

- O evento de toque acontece quando o usuário toca em algum componente visual do aplicativo, seja com uma caneta *stylus*, seja com os próprios dedos. O importante é não confundir este evento com o de clique.
- No evento de toque podemos saber em qual região da *View* o contato aconteceu, característica útil para os desenvolvedores de jogos.
- Na maioria das vezes, esse evento é utilizado para capturar um toque na janela do aplicativo e nessa situação, o *listener* deve ser atribuído ao componente *ViewGroup* que corresponde à tela toda, como, por exemplo, um *LinearLayout*.
- Entretanto é importante lembrar de nomear o componente *LinearLayout* no arquivo XML, adicionando a ele a propriedade android:id.
- Nos slide seguinte é apresentada a interface visual inicial do aplicativo (apenas o layout).

EVENTO DE TOQUE

- O código apresentado não possui componentes, apenas um *LinearLayout*. Para este exemplo, os componentes *TextView* para apresentação dos dados de toque serão criados dinamicamente, conforme código disponível no slide seguinte.

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:id="@+id/tela"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:orientation="vertical"
7     tools:context="pm25s.aula08.usandoeventosavancados.PrincipalActivity" >
8
9 </LinearLayout>
```

EVENTO DE TOQUE

```java
package pm25s.aula08.usandoeventosavancados;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends Activity {
    private LinearLayout tela;

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

        tela = (LinearLayout) findViewById(R.id.tela);
```

```java
        tela.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                TextView texto = new TextView(getApplicationContext());
                String msg = "";

                if(event.getAction() == MotionEvent.ACTION_DOWN) {
                    msg = "pressionou na tela.";
                } else if(event.getAction() == MotionEvent.ACTION_MOVE) {
                    msg = "moveu na tela.";
                } else if(event.getAction() == MotionEvent.ACTION_UP) {
                    msg = "soltou na tela.";
                }

                msg += " x: " + event.getRawX() + " e y: " + event.getRawY();
                texto.setText(msg);
                texto.setTextColor(Color.BLUE);
                tela.addView(texto);

                return true;
            }
        });
    }
```


EVENTO DE TOQUE

Dica: Retorno do método onTouch()

O método onTouch() possui um retorno booleano, como pode ser observado na linha 23. Esse retorno define se os eventos subsequentes ao atual também serão tratados ou não. Sabemos que ao clicar e arrastar na tela, três tipos de eventos ocorreram em onTouch(): evento de toque, um ou mais eventos de movimentação e um evento de soltura. Assim, ao receber um evento do tipo de toque, se o retorno for *false* os eventos seguintes serão ignorados; retornando *true*, os eventos seguintes serão tratados também.

EVENTO DE FOCO

- O evento de mudança de foco é executado toda vez que um componente ganha ou perde o foco, semelhante ao que acontece em um programa *desktop* quando se pressiona a tecla [TAB] em um componente visual.
- Para exemplificar o uso do evento de toque, foram adicionados dois componentes EditText à interface gráfica do aplicativo conforme apresentado no slide seguinte.
- A interface apresentada apenas declara na tela dois componentes EditText, com os nomes etDados1 e etDados2, respectivamente.

EVENTO DE FOCO

```xml
<?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/tela"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="pm25s.aula08.usandoeventosavancados.PrincipalActivity" >

    <EditText
        android:id="@+id/etDados1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="text" />

    <EditText
        android:id="@+id/etDados2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="text" />

</LinearLayout>
```

EVENTO DE FOCO

```java
package pm25s.aula08.usandoeventosavancados;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class FocoActivity extends Activity {
    private EditText etDados1;
    private EditText etDados2;

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

        etDados1 = (EditText) findViewById(R.id.etDados1);
        etDados2 = (EditText) findViewById(R.id.etDados2);

        etDados1.setOnFocusChangeListener(new View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View v, boolean hasFocus) {
                etDados1OnFocus(hasFocus);
            }
        });
        etDados2.setOnFocusChangeListener(new View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View v, boolean hasFocus) {
                etDados2OnFocus(hasFocus);
            }
        });
    }

    protected void etDados1OnFocus(boolean hasFocus) {
        if(hasFocus) {
            Toast.makeText(getApplicationContext(), "[Dados 1] ganhou o foco.", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(getApplicationContext(), "[Dados 1] perdeu o foco.", Toast.LENGTH_SHORT).show();
        }
    }

    protected void etDados2OnFocus(boolean hasFocus) {
        if(hasFocus) {
            Toast.makeText(getApplicationContext(), "[Dados 2] ganhou o foco.", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(getApplicationContext(), "[Dados 2] perdeu o foco.", Toast.LENGTH_SHORT).show();
        }
    }
}
```

EVENTO DE FOCO

EVENTO DE FOCO

- Por padrão, a ordem da mudança de foco obedece as normas do gerenciador de layout e, no exemplo apresentado, como está sendo utilizado o *LinearLayout*, o foco navega do primeiro componente para o último, na ordem vertical.
- É possível mudar este comportamento padrão alterando o arquivo XML principal conforme imagem abaixo:

```
<EditText
    android:id="@+id/etDados1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:nextFocusDown="@+id/etDados2"
    android:inputType="text" />

<EditText
    android:id="@+id/etDados2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:nextFocusDown="@+id/etDados1"
    android:inputType="text" />
```

EVENTO DE TECLA

- O evento de tecla serve para capturar qualquer tecla pressionada em um componente quando este está com o foco.
- Esse evento pode ser útil para validar entradas de textos muito específicas, tais como, um CPF, CNPJ, entre outras.
- Para testar esse evento, será utilizada a mesma interface do exemplo anterior, sendo capturadas as teclas pressionadas em etDados1.
- O código correspondente é apresentado no slide seguinte.
- O código apresentado permite apenas que o campo etDados1 aceite letras minúsculas ou maiúsculas.

EVENTO DE TECLA

```
1  package pm25s.aula08.usandoeventosavancados;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.view.KeyEvent;
6  import android.view.View;
7  import android.widget.EditText;
8
9  public class TeclaActivity extends Activity {
10     private EditText etDados1;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_tecla);
16
17         etDados1 = (EditText) findViewById(R.id.etDados1);
18
19         etDados1.setOnKeyListener(new View.OnKeyListener() {
20             @Override
21             public boolean onKey(View v, int keyCode, KeyEvent event) {
22                 if(keyCode >= KeyEvent.KEYCODE_A && keyCode <= KeyEvent.KEYCODE_Z) {
23                     return false;
24                 } else {
25                     return true;
26                 }
27             }
28         });
29     }
30 }
```

EVENTO DE TECLA

- A relação abaixo mostra alguns dos vários métodos interessantes de KeyEvent:
  - *getUnicodeChar*: retorna o código Unicode que a tecla deve produzir.
  - *getRepeatedCount*: retorna o número de repetições do evento.
  - *isAltPressed*: responde se a tecla ALT estava pressionada no momento do evento.
  - *isModifierKey*: retorna se a tecla é modificadora.
  - *isShiftPressed*: retorna *true* se a tecla SHIFT está pressionada.

Dica: Utilizando a propriedade digits()

O componente EditText possui uma propriedade android:digits, que pode ser usada na declaração deste no XML, o que permite definir os caracteres que serão aceitos pelo componente, simplificando o processo para aceitar apenas letras, por exemplo, não havendo a necessidade de tratar um evento de tecla.

ADICIONANDO UM MENU DE CONTEXTO

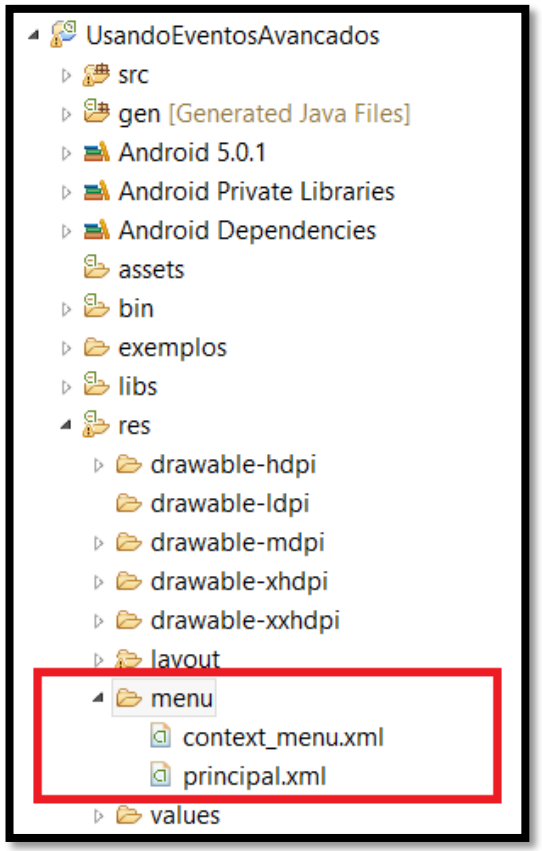
- Menu de contexto é um componente visual da plataforma Android. Esta *View* é adicionada no momento em que um clique longo acontece em um componente. Esse menu pode ter várias opções, que são programadas pelo desenvolvedor.
- Para o exemplo, será adicionado um mesmo menu de contexto para os dois componentes EditText da aplicação, sendo apresentadas para o usuário as opções de Ajuda e Histórico.

Dica: Menus de contexto

Os menus de contexto são componentes associados a uma *View* específica da tela, entretanto, um mesmo menu de contexto pode ser compartilhado entre diferentes *Views*.

ADICIONANDO UM MENU DE CONTEXTO

- O primeiro passo para criar um menu de contexto é definir um arquivo XML que possuirá as opções do menu, neste exemplo, as opções de Ajuda e Histórico.
- Para a tarefa descrita acima, procure o diretório *res*, na pasta *menu*, e crie um arquivo chamado *context_menu.xml*. Use as opções **New > Android XML File** clicando com o botão direito sobre a pasta *menu*.
- A estrutura de diretórios do projeto posterior à criação do arquivo XML ficará semelhante ao da figura do lado.
- O conteúdo do arquivo XML criado assim como o código Java modificado são apresentados no slide seguinte.



ADICIONANDO UM MENU DE CONTEXTO

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/ajuda"
          android:title="@string/ajuda" />
    <item android:id="@+id/historico"
          android:title="@string/ajuda" />
</menu>
```

```
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.MenuInflater;
```

```
etDados1 = (EditText) findViewById(R.id.etDados1);
etDados2 = (EditText) findViewById(R.id.etDados2);

registerForContextMenu(etDados1);
registerForContextMenu(etDados2);
```

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

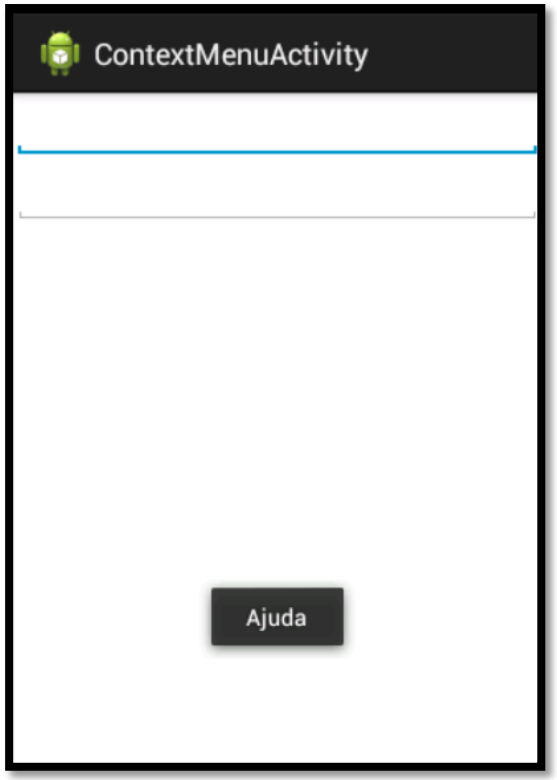
ADICIONANDO UM MENU DE CONTEXTO

- Realize as alterações contidas no slide anterior e execute novamente o aplicativo.
- Clique com o mouse no primeiro campo de texto e mantenha pressionado por alguns segundos. Será apresentado o menu de contexto conforme imagem ao lado.
- Porém, ao clicar em uma das opções do menu, nenhuma ação é executada. Para tanto é preciso sobrescrever o método *onContextItemSelected*. Veja o slide seguinte.

ADICIONANDO UM MENU DE CONTEXTO

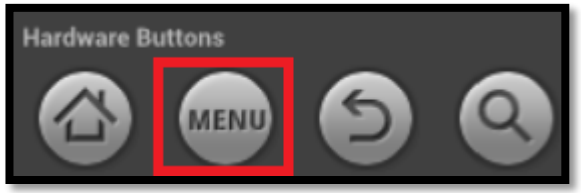
- Será necessário importar a classe *android.view.MenuItem* também. Teste o aplicativo novamente.

```
public boolean onContextItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case R.id.ajuda:
            Toast.makeText(this, "Ajuda", Toast.LENGTH_LONG).show();
            return true;
        case R.id.historico:
            Toast.makeText(this, "Histórico", Toast.LENGTH_LONG).show();
            return true;
        default: return super.onContextItemSelected(item);
    }
}
```



UTILIZANDO MENUS

- Outro modelo de interação com aplicações Android é a partir de menus.
- Um menu é algo semelhante ao menu de contexto; a diferença está na chamada e onde o menu está associado.
- O menu de contexto pode ser associado a qualquer componente visual e a chamada do mesmo é realizada por um clique longo no componente que o possui.
- Os menus tradicionais diferem pela quantidade por tela e pelo modo de chamada. É possível adicionar apenas um menu por tela e sua chamada é feita por meio do botão de menu existente nos dispositivos Android.



UTILIZANDO MENUS

- Para criar um menu na plataforma Android, assim como no menu de contexto, o primeiro passo é criar um arquivo XML contendo as opções do menu. Esse arquivo costuma ficar na pasta *menu*, dentro de *res*, e costuma ter o nome da tela do aplicativo que irá apresentá-lo.
- Porém, no Eclipse, ao entrar na pasta *menu*, você observará que já existe um arquivo com o nome da interface gráfica principal do aplicativo. Isso acontece porque o Eclipse já traz automaticamente um menu quando cria uma *Activity*, contendo o texto *Settings*.
- Assim, iremos editar o arquivo XML da pasta menu, adicionando uma nova opção chamada *Clear*, que limpará o conteúdo existente nos EditTexts da tela.
- Após a mudança, o arquivo XML da pasta *menu* deverá ficar parecido com o código apresentado no slide seguinte.

UTILIZANDO MENUS

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="pm25s.aula08.usandoeventosavancados.PrincipalActivity" >

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings" />

    <item
        android:id="@+id/action_clear"
        android:orderInCategory="200"
        android:title="@string/action_clear" />

</menu>
```

UTILIZANDO MENUS

- Após criar os itens, o passo seguinte é adicionar a tela da Activity e para isso, deve-se adicionar o método **onCreateOptionMenu** na **ActivityPrinicipal.Java**, conforme apresentado abaixo. Será necessário importar a classe **Menu** também (import android.view.Menu).

```
public class PrincipalActivity extends ActionBarActivity {

    private EditText etDados1;
    private EditText etDados2;

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

        etDados1 = (EditText) findViewById(R.id.etDados1);
        etDados2 = (EditText) findViewById(R.id.etDados2);

        // restante do código do método onCreate()
    }

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

    // restante do código da classe
}
```

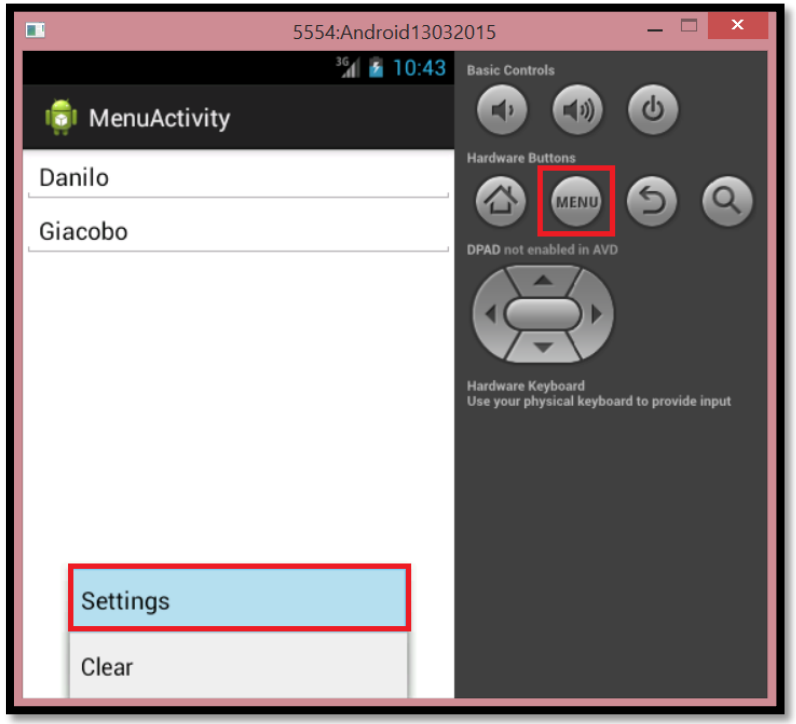
UTILIZANDO MENUS

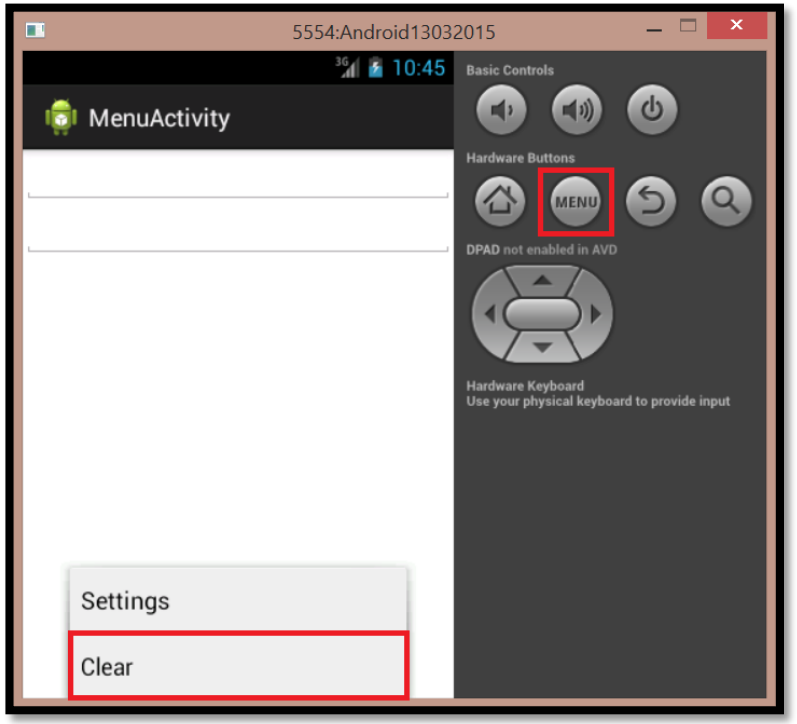
- O próximo passo é tratar o evento de clique nos menus e para isso, é dado o método **onOptionsItemSelected**, conforme apresentado abaixo:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if(item.getItemId() == R.id.action_settings) {
        Toast.makeText(getApplicationContext(), "Opção não disponível.", Toast.LENGTH_LONG).show();
    } else if(item.getItemId() == R.id.action_clear) {
        etDados1.setText("");
        etDados2.setText("");
    }

    return true;
}
```

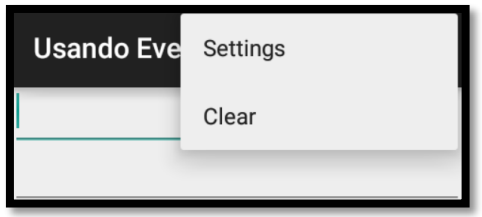
UTILIZANDO MENUS





UTILIZANDO O ACTIONBAR

- A utilização de menus é uma técnica muito interessante no desenvolvimento de aplicações móveis, pois são elementos de ação, que ficam ocultos e não ocupam espaço na tela.
- Um menu possui uma desvantagem também: o mesmo não apresenta nenhum identificador visual na tela indicando que esta possui um menu.
- Ao longo da vida da plataforma Android alguns desenvolvedores resolveram explorar a barra de título, que a princípio não tinha muita utilidade, e assim milhares de aplicativos começaram a surgir dando alguma funcionalidade a ela.
- Depois de algum tempo, chegaram a um resultado, a então conhecida ActionBar, que ficou tão popular que a Google incluiu a mesma na API do Honeycomb (Fevereiro de 2011).



UTILIZANDO O ACTIONBAR

- O componente é um *widget* para atividades que substitui a barra de títulos tradicional no topo da tela. Por padrão, inclui-se o logo da aplicação à esquerda, seguido pelo título da atividade e quaisquer outros itens disponíveis do menu de opções no lado direito. Ela oferece muitas características úteis, incluindo a habilidade de:
  - Mostrar itens do menu de opções diretamente na ActionBar, como itens de ação - *ActionItens* - provendo acesso instantâneo às ações do usuário.
  - Os itens de menu que não aparecem como itens de ação são colocados num menu flutuante, que é revelado por uma lista *drop-down*.
  - Provê tabulação para navegar entre os fragmentos.
  - Provê uma lista *drop-down* para a navegação.
  - Provê *Action Views* no local dos itens da ação.

UTILIZANDO O ACTIONBAR

- Para o uso da **ActionBar**, nas versões do Android posteriores a 3.0, o processo é simples e muito próximo da criação de um menu, inclusive os passos adotados no exemplo anterior, sobre menus, também serão adotados aqui, com apenas algumas mudanças.
- As ActionBars são baseadas em menus e por este motivo, o primeiro passo é criar um arquivo XML com os itens desse menu, como feito anteriormente. A diferença está na inclusão da propriedade android:showAsAction.

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:app="http://schemas.android.com/apk/res-auto"
3      xmlns:tools="http://schemas.android.com/tools"
4      tools:context="pm25s.aula08.usandoeventosavancados.PrincipalActivity" >
5
6      <item
7          android:id="@+id/action_settings"
8          android:orderInCategory="100"
9          android:title="@string/action_settings"
10         app:showAsAction="always"/>
11
12     <item
13         android:id="@+id/action_clear"
14         android:orderInCategory="200"
15         android:title="@string/action_clear"
16         app:showAsAction="always"/>
17
18 </menu>
```

UTILIZANDO O ACTIONBAR

- O código apresentado fica no mesmo arquivo de menus, pois são excludentes: o usuário utiliza um menu ou utiliza a ActionBar. Assim na pasta *menu*, no arquivo XML, foram alterados os dois itens de menu criados no exemplo anterior, adicionando a propriedade android:showAsAction, que pode ter quatro valores:
  - never: o item de menu nunca é apresentado na ActionBar.
  - ifRoom: o item de menu só será apresentado diretamente ao usuário se houver espaço na tela; caso contrário, ficará no menu suspenso ao lado.
  - always: o item de menu sempre será apresentado na ActionBar.
  - withText: caso o item de menu tenha ícone (propriedade android:icon), fica esteticamente mais bonito a apresentação apenas do ícone. Se o usuário deseja que o texto (propriedade title) seja apresentado de qualquer maneira, utiliza-se este atributo.
- Também é possível integrar mais de uma opção usando o caractere "|", como, por exemplo, android:showAsAction="always|withText".

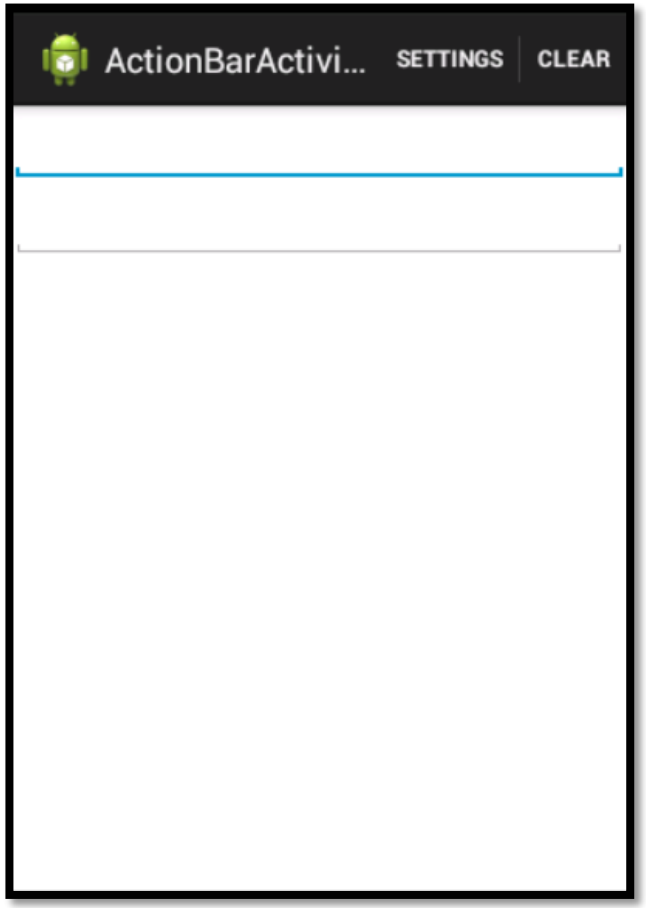
UTILIZANDO O ACTIONBAR

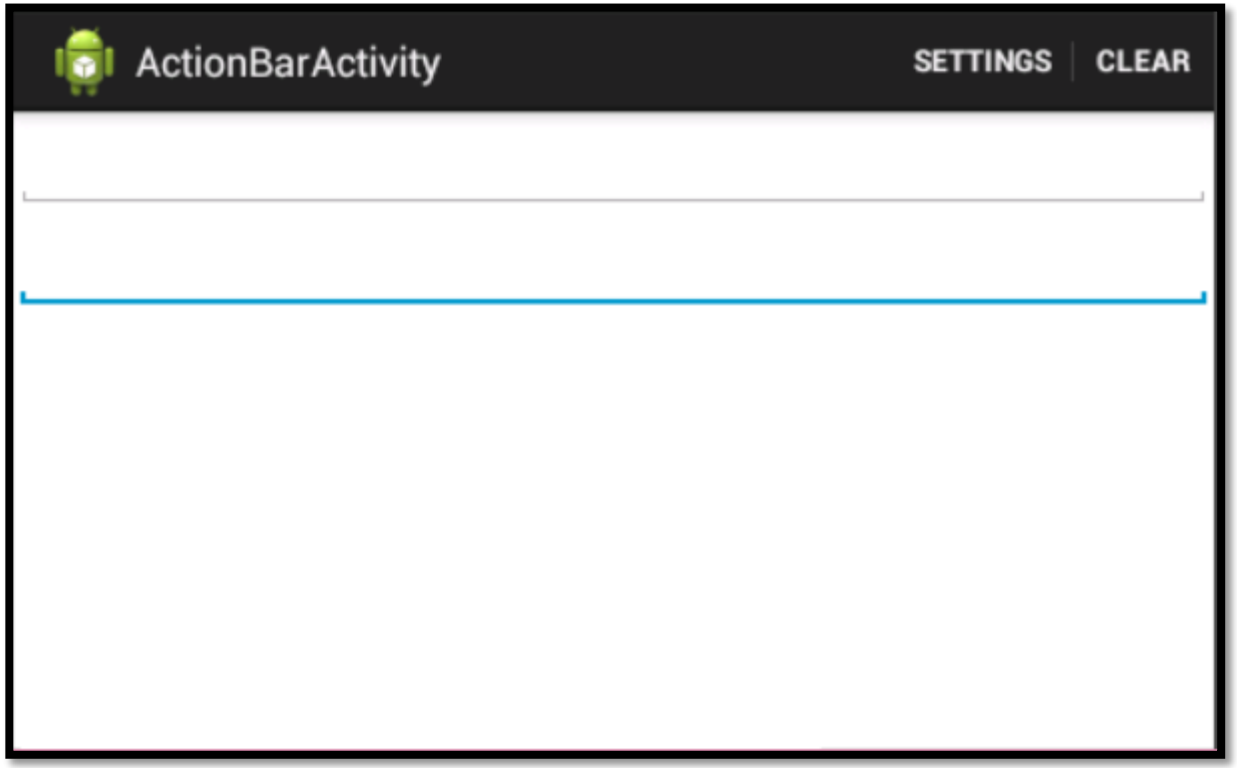
- A próxima mudança está no tema da aplicação, outro recurso incluído a partir do Android 3.0. As ActionBar estão presentes no tema Holo, assim, é necessário acessar o arquivo AndroidManifest.xml, alterando android:theme, conforme imagem abaixo:

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/Theme.AppCompat.Light" >
```

- Por fim, os demais tratamentos para a ActionBar são feitos da mesma maneira que os menus, deve-se codificar na *Activity* o método **onCreateOptionMenu()** e o tratamento de seleção deste acontece no método **onOptionsItemSelected()**, exatamente da mesma forma como nos menus.

UTILIZANDO O ACTIONBAR





EVENTO DE ITEM SELECIONADO

- Existem algumas *Views* que têm tratamento de eventos únicos ou, no máximo, compartilham o mesmo comportamento com outros componentes semelhantes. Este é o caso de ListView e Spinner, por exemplo.
- Para o nossos exemplo, iremos incluir para uma caixa de seleção com algumas cidades da região sudoeste do Paraná, onde o usuário poderá escolher, por exemplo, qual o melhor lugar para passar o final de semana.
- Assim, o arquivo activity_principal.xml da pasta layout sofrerá algumas alterações, conforme apresentado no slide a seguir.

EVENTO DE ITEM SELECIONADO

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical"
6      tools:context=".EventoItemSelecionadoActivity" >
7
8      <TextView
9          android:layout_width="fill_parent"
10         android:layout_height="wrap_content"
11         android:text="@string/cidade" />
12
13     <Spinner
14         android:id="@+id/spCidade"
15         android:layout_width="fill_parent"
16         android:layout_height="wrap_content" />
17
18 </LinearLayout>
```

EVENTO DE ITEM SELECIONADO

```java
package pm25s.aula08.usandoeventosavancados;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

public class EventoItemSelecionadoActivity extends Activity {

    private Spinner spCidade;
    private String[] cidades = new String[] {"Pato Branco", "Francisco Beltrão", "Marmeleiro",
                                             "Coronel Vivida", "Vitorino", "Mariópolis", "Mangueirinha"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_evento_item_selecionado);

        spCidade = (Spinner) findViewById(R.id.spCidade);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, cidades);
        spCidade.setAdapter(adapter);
    }
}
```

EVENTO DE ITEM SELECIONADO

- O próximo passo é tratar um evento específico dos componentes do tipo *seleção*, como o *Spinner* e o *ListView*. Esse evento é o item selecionado e seu código é apresentado abaixo. Deve-se colocar este código ao final do método *onCreate()*.

```
spCidade.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view,
            int position, long id) {

    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {

    }
});
```

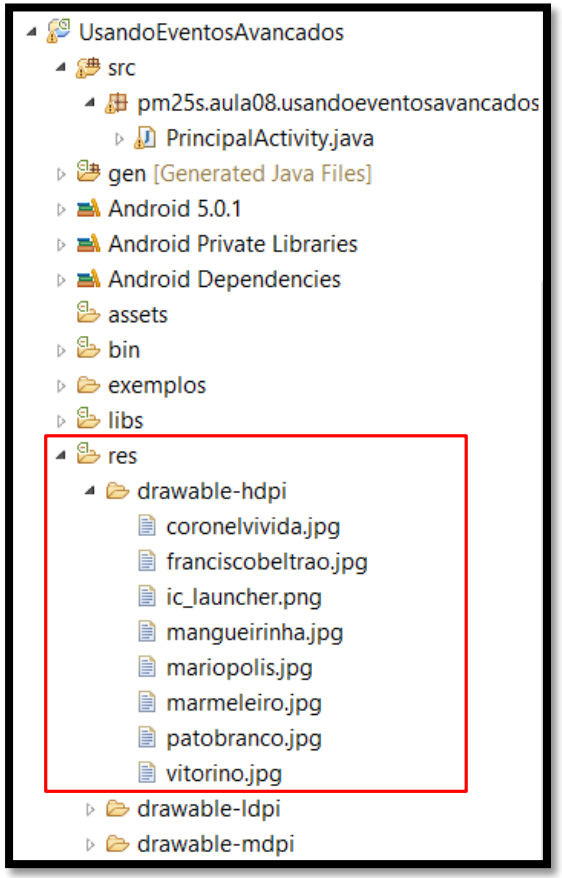
EVENTO DE ITEM SELECIONADO

- Para enriquecer o estudo de caso dessa aula, será adicionado um tratamento especial para o componente *Spinner*: quando o usuário selecionar qualquer cidade, será apresentada uma imagem com a região selecionada.
- Para isso, deve-se adicionar uma *View* de imagem na tela do aplicativo, que inicialmente não mostrará nada.

```
<ImageView
    android:id="@+id/imgCidade"
    android:contentDescription="@string/img"
    android:layout_width="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_height="wrap_content" />
```

EVENTO DE ITEM SELECIONADO

- Agora, devem-se colocar algumas imagens de cidade na pasta da aplicação. As imagens possuem um tamanho médio de 400x400 e foram copiadas para uma subpasta *drawable*, dentro da pasta *res* do projeto.
- A estrutura da aplicação, após a inclusão das imagens, é apresentada na imagem ao lado.
- Essas imagens foram coladas em todas as pastas *drawable* para um melhor tratamento de fragmentação das plataformas Android (dispositivos com diferentes tamanhos de tela).



EVENTO DE ITEM SELECIONADO

- O próximo passo no desenvolvimento do aplicativo é recuperar o objeto *ImageView* e tratar o evento *setOnItemSelectedListener*, conforme listagem abaixo.

```
import android.widget.ImageView;
```

```
int imagemID[] = {R.drawable.patobranco, R.drawable.franciscobeltrao, R.drawable.marmeleiro,
                  R.drawable.coronelvivida, R.drawable.vitorino, R.drawable.mariopolis, R.drawable.mangueirinha};
```

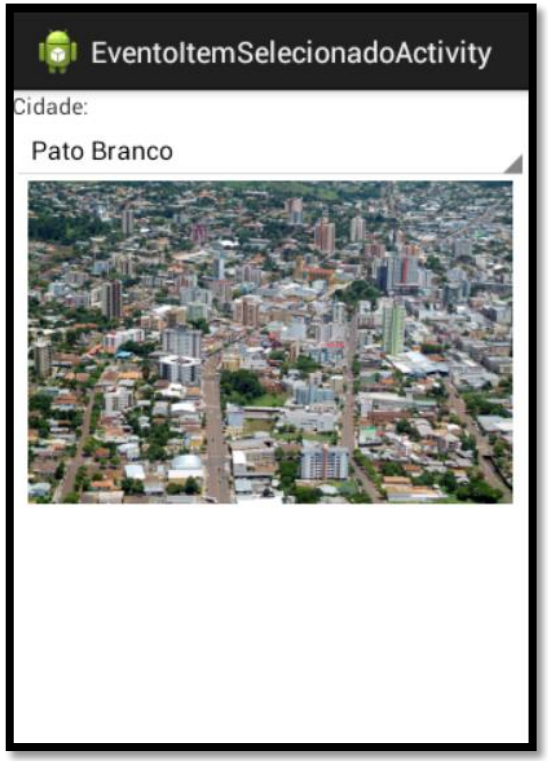
```
private ImageView imgCidade;
```

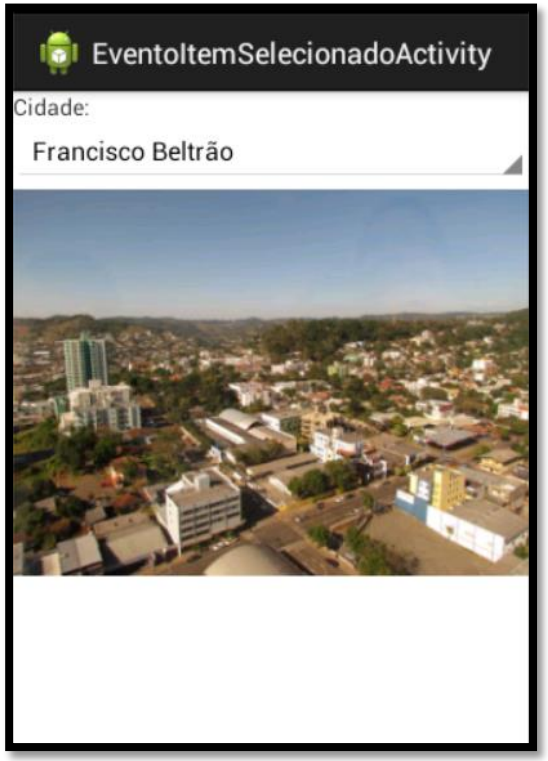
```
imgCidade = (ImageView) findViewById(R.id.imgCidade);
```

```
spCidade.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view,
            int position, long id) {
        imgCidade.setImageResource(imagemID[position]);
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        Toast.makeText(getApplicationContext(), "Selecione uma cidade", Toast.LENGTH_LONG).show();
    }
});
```

EVENTO DE ITEM SELECIONADO





Dica: Código para o tratamento dos itens incluídos ou excluídos da lista.

Também é possível trabalhar com os eventos de adicionar ou retirar elementos de uma lista. Para isso, existe um *listener* que "escuta" os eventos para adicionar ou remover os elementos de *View*, procedimento apresentado abaixo:

```
spCidade.setOnHierarchyChangeListener(new
   OnHierarchyChangeListener() {
       public void onChildViewAdded(View parent, View child) {}
       public void onChildViewRemoved(View parent, View child) {}
});
```