Android - Cards de Referência

Menus

Bem vindo ao nosso sexto card sobre Android! :-) Desta vez, vamos conversar sobre os menus. Temos dois tipos de menus, conhecidos como ContextMenu e o OptionsMenu. Vamos fazer alguns exemplos e aprender a usar eles.

MENUS

Vamos ver como e quando usar os menus, sejam eles de contexto ou de opções. Os menus passaram por uma reformulação a partir da versão 3. Para quem já tem os dispositivos com Android mais novos, já percebeu que sumiu aquele botão físico para o menu. Reparou?

Agora, nós temos um novo conceito, chamado de **ActionBar**. As opções de menu aparecem na action bar e para os novos dispositivos, não é necessário ter um botão de menu. Vamos entender isso com exemplos?

OPTIONS MENU

Vamos começar nossa conversa com o **OptionsMenu**. É um tipo de menu com um contexto mais global. Cada Activity pode ter um menu associado a ela, contendo opções como "Preferências", "Novo" e por aí vai.

Aqui está a principal diferença do Android 3.0+ com relação às versões anteriores. Nas versões anteriores, para você ver esse menu era necessário apertar um botão físico. A partir da 3.0, as opções do menu aparecem na **ActionBar** e, apenas no caso de todas as opções não caberem na **ActionBar**, serão exibidos "três quadradinhos" no final, que ao ser clicado exibirá as demais opções.

Dá uma olhada nas duas imagens, uma acima (Android 2.X) e a outra logo abaixo (Android 3+), que você vai entender!

O primeiro passo que temos a fazer é criar um arquivo XML e definir quais as opções de menu que queremos. Criaremos, caso ainda não exista, uma pasta **/res/menu** e colocaremos nesta pasta um arquivo **menu_activity_main.xml**.

```
<menu>
   <item android:id="@+id/menu_settings"
      android:title="@string/menu_settings"
      android:orderInCategory="100"
      android:showAsAction="never" />
</menu>
```

Iniciamos esse arquivo com uma tag <menu>. Dentro desta tag, teremos várias tags <item> definindo cada opção do menu. Esta última tag pode ter um monte de atributos:

android:id - Identificador da opção de menu.
android:title - Título do menu. É o que o usuário verá.
android:titleCondensed - Caso o título do item seja grande e possa não aparecer em telas menores, esta opção será usada.
android:icon - Ícone que será exibido no item.
android:showAsAction - Define quando e como este item será exibido na action bar.

Opção	Dscrição
never	Nunca entrará na action bar.
withText	Exibir o título do item.
ifRoom	Aparece na action bar se tiver espaço para ele.
always	Sempre exibe na action bar.

android:alphabeticShortcut - Uma letra que servirá como atalho.
android:numericShortcut - Um número que servirá como atalho.
android:checkable - True se o item é "checável"(checkbox).
android:visible - True se a opção deve estar visível.
android:enabled - True se a opção deve estar habilitada.

android:menuCategory - Define uma categoria para o item. Pode assumir os valores container, system, secondary e alternative. Você usará mais as opções "default" e secondary. as demais são pouco explicadas na documentação do Android, infelizmente. :-(
android:orderInCategory - A ordem de importância do item dentro da categoria. Quanto menor, mais cedo o item será apresentado.

Agora é hora de colocar este menu para ser exibido de fato. Até agora, só definimos quais serão as opções. Precisamos associar este menu a uma Activity.

```
public class MainActivity extends Activity {

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

}
```

Nesta listagem, o que de fato nos importa é o método **onCreateOptionsMenu(Menu)**. Ele será chamado todas vez que a Activity for exibida, perguntando se há um menu para criar. Você deve, então, obter um MenuInflater e "inflar" o menu que quer usar. Observe no uso do **R.menu.menu_activity_main** também.

Uma vez apresentado o menu, precisamos tratar a seleção de uma opção dele. Precisamos implementar mais um método da classe pai. Agora, será o método **onOptionsItemSelected**.

```
public boolean onOptionsItemSelected(MenuItem item) {
        boolean result = false;

        if(item.getItemId() == R.id.menu_settings) {
        // Faz algo! Chamar uma nova Activity, por exemplo.
                result = true;
        }

        return result;
}
```

Você terá como parâmetro qual o item do menu selecionado. Para você descobrir qual item foi selecionado, basta comparar o **itemId** do item com o ID do recurso na classe **R.java**. Daqui, você pode iniciar uma nova Activity, por exemplo. Lembre-se de retornar true para informar que o item do menu foi tratado.

Também é possível tratar a seleção de um item usando um **OnMenuItemClickListener**, conforme o exemplo a seguir. Observe que precisamos ter acesso ao MenuItem para definir quem será o Listener do clique neste item. Usamos o método **setOnMenuItemClickListener**. Podemos usar um mesmo listener para todas as opções e diferenciar isto no Listener.

```
public boolean onCreateOptionsMenu(Menu menu) {
   MenuItem menuItem = menu.add(1, 1, 1, "Opção 1");
   OnMenuItemClickListener listener = new
OnMenuItemClickListener() {
        public boolean onMenuItemClick(MenuItem item) {
                return true;
        }
   };
   menuItem.setOnMenuItemClickListener(listener);
   return true;
}
```

Eu posso criar um menu dinamicamente? Claro que pode. Você pode deixar de lado a criação do menu pelo arquivo XML e fazer isso na "mão grande".

```
public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0 // Grupo
                , 1 // item id
                , 0 // ordem
                , "Menu 1");
        super.onCreateOptionsMenu(menu);
        return true;
}
```

O código acima demonstra este caso. Você está adicionando manualmente cada opção. É interessante que você faça a chamada para o método da classe pai, passando esse menu. Existe uma categoria de menu chamada SYSTEM, que pode ser adicionada automaticamente pelo sistema.

Outro detalhe importante: retorne **true** neste método se você que o menu seja apresentado mesmo.

GRUPOS

Não. Grupos não é para você agrupar visualmente as opções do menu. :-) Eles permitem a você agrupar um conjunto de itens para que eles compartilhem atributos em comum. Só isso.

```
<menu>

   <group android:id="@+id/group" android:enabled="false" >

      <item android:id="@+id/group_item1"

         android:title="@string/group_item1" />

      <item android:id="@+id/group_item2"

         android:title="@string/group_item2"/>

   </group>

</menu>
```

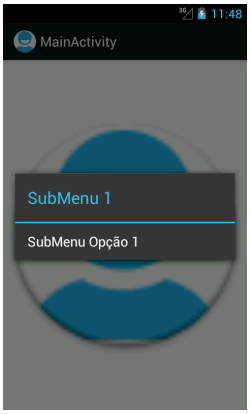
No exemplo acima definimos um grupo e uma característica em comum para todos os itens deste grupo: todos estão desabilitados.

A tag <group> pode ter os seguintes atributos:

android:enabled - True se os itens deve estar habilitados.
android:visible - True se os itens devem estar visíveis.
android:menuCategory - Define uma categoria para os itens. Pode assumir os valores container, system, secondary e alternative. Você usará mais as opções "default" e secondary. as demais são pouco explicadas na documentação do Android, infelizmente. :-(
android:orderInCategory - A ordem de importância da categoria.

SUBMENUS

Bom, se os grupos não resolvem a questão de agrupar opções, então como fazer isso? Usando submenus! Existe uma classe para isso, chamado SubMenu. Como você poderia esperar, ela herda de Menu.



A imagem acima é um exemplo de como os submenus são exibidos. Um detalhe interessante: embora a documentação afirme não ser possível ter um "subsubmenu", testes com a versão 4 do Android demonstram que isto é possível, sim.

Em todo caso, se a documentação diz que não é possível, é melhor evitar, uma vez que versões futuras podem "quebrar" sua aplicação se mudar esse comportamento. Existem duas formas para você adicionar um submenu: pelo XML ou em código Java. Vamos ver primeiro o XML.

```
<menu>
  <item android:id="@+id/menu_settings"
    android:title="@string/menu_settings"
    android:orderInCategory="100"
    android:showAsAction="never">
    <menu>
        <item android:id="@+id/menu_settings_2"
            android:title="@string/menu_settings_2"/>
    </menu>
  </item>
</menu>
```

Para conseguir isto, basta adicionar uma tag <menu> dentro da tag <item>. Simples, não é?

Com código em Java, na Activity, basta criarmos um SubMenu dentro do método **onCreateOptionsMenu**. Usamos, para isto, o método **addSubMenu**, existente na classe Menu.

```
public boolean onCreateOptionsMenu(Menu menu) {
        SubMenu subMenu = menu.addSubMenu("SubMenu 1");
        subMenu.add("SubMenu Opção 1");
        MenuItem menuItem = subMenu.add(1, 1, 1, "Opção 1");
}
```

Um último detalhe, observe que a classe **MenuItem** tem o método **setIntent(intent)**. Caso você defina um Intent, ele será chamado usando método **startActivity(intent)**. Ou seja, você pode, dessa forma, facilitar a chamada de uma nova Activity a partir de um menu.

```
public boolean onCreateOptionsMenu(Menu menu) {
        MenuItem menuItem = menu.add(1, 1, 1, "Opção 1");
        menuItem.setIntent(...);
}
```

MENUS DE CONTEXTO

O nome já diz tudo: este tipo de menu é para ser usado em contextos específicos, como um menu que aparece ao se selecionar um item de uma **ListView**. Aliás, esta é a forma mais comum de se usar os menus de contexto.

A declaração destes menus é idêntica a como fizemos para os menus de opção. A única diferença é o QUANDO e COMO vamos mostrar esse menu. Pode ser no clique de um botão, na seleção de um item de um ListView e etc.

Uma diferença é que menus de contexto não suportam ícones, atalhos e submenus. Inicialmente, precisamos registrar uma View para responder pelo menu de contexto, usando o método **registerForContextMenu(view)** da classe Activity. Em seguida, implementar o método **onCreateContextMenu**, também da Activity. Finalizando, basta tratar as seleções do menu no método **onContextItemSelected**.

Vamos aos exemplos. Vamos colocar um botão para exibir um menu ao receber um **long-click**? O long-click é aquele que você aperta o componente e mantém pressionado por um tempo maior, tipo uns 2 segundos. Vamos usar o mesmo arquivo de menu que fizemos anteriormente.

```
<RelativeLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <Button
    android:id="@+id/botao"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Botão" />
</RelativeLayout>
```

O código anterior é a definição do nosso layout de tela, que contém apenas um botão. Agora precisamos implementar os métodos na classe Activity.

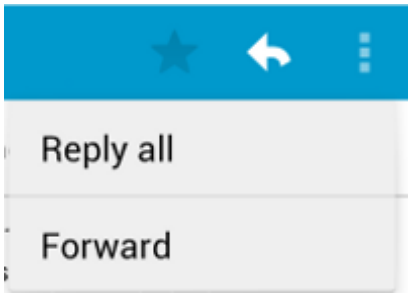
```
public class MainActivity extends Activity {
        private Button button;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                button = (Button) findViewById(R.id.botao);

                registerForContextMenu(button);
        }

        public void onCreateContextMenu(ContextMenu menu, View
v, ContextMenuInfo menuInfo) {
                MenuInflater menuInflater = getMenuInflater();
                menuInflater.inflate(R.menu.activity_main,
menu);
        }

        public boolean onContextItemSelected(MenuItem item) {
                return true;
        }
}
```

Observe os parâmetros que **onCreateContextMenu** tem. Podemos usar eles para identificar de qual View partiu o long-click. Afinal de contas, podemos ter mais de uma View respondendo por menus de contexto, não é? Agora você já pode executar o projeto. Observe que não adianta dar um clique normal no botão, você precisa pressionar e manter assim por um tempinho.

Não é isso que você queria? Gostaria que aparecesse o menu logo no primeiro clique? Então, o que você quer é um **Popup Menu**.

POPUP MENU

Você usa os popups com menu para exibir opções extras próximas a uma View. Por exemplo, podemos ter um botão que ao ser clicado exibe outras opções, próximas a ele. Conhece o aplicativo Gmail? Temos um exemplo de Popup Menu nele. Veja a imagem abaixo, tirada da documentação do Android.




```
public class MainActivity extends Activity
        implements PopupMenu.OnMenuItemClickListener {
    private Button button;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = (Button) findViewById(R.id.botao);
        button.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {

                PopupMenu popupMenu =
                    new PopupMenu(MainActivity.this, button);

                MenuInflater menuInflater =
                    popupMenu.getMenuInflater();

                menuInflater.inflate(R.menu.activity_main,
                    popupMenu.getMenu());

                popupMenu.
                    setOnMenuItemClickListener(MainActivity.this);

                popupMenu.show();
            }
        });
    }

    public boolean onMenuItemClick(MenuItem item) {
        return true;
    }
}
```

No código acima, temos alguns pontos a destacar. Primeiro, obtemos uma referência para o botão e definimos um Listener para o clique nele. Neste listener, criamos um **PopupMenu**. Fazemos um inflate do menu que já usamos anteriormente. Depois, precisamos exibir ele e tratar os cliques nos itens. Para o tratamento do clique, implementamos a interface PopupMenu.OnMenuItemClickListener na própria classe de Activity.

Desta forma, precisamos implementar o método **onMenuItemClick(MenuItem)**. Finalizando, é só informar para o popupMenu quem é o objeto responsável em tratar os cliques.

MENU DE CONTEXTO - ACTIONMODE

Agora, vamos supor que você tenha uma ListView no qual os itens tenham um checkbox do lado deles. O usuário seleciona vários itens e deseja realizar uma ação que abrange todos eles. Como fazer isso? Usando um tipo especial de menus de contexto.

A imagem acima é um exemplo de uso desse tipo de menu. Toda vez que algum item é selecionado, é exibido um menu no topo da aplicação. Para este menu desaparecer, o usuário deve não ter itens selecionados, apertar o botão BACK ou clicar no primeiro ícone (DONE, uma imagem de um check).

O primeiro passo é criar uma instância de **ActionMode.Callback**. Trata-se de uma interface com quatro métodos que vão tratar a criação, destruição e clique nos itens.

```
private ActionMode.Callback callback = new Callback() {
        public boolean onPrepareActionMode(ActionMode mode,
                Menu menu) {
                return false;
        }

        public void onDestroyActionMode(ActionMode mode) {
        }

        public boolean onCreateActionMode(ActionMode mode,
                Menu menu) {
                menu.add("Ação 1");
                menu.add("Ação 2");
                return true;
        }

        public boolean onActionItemClicked(ActionMode mode,
                MenuItem item) {
                mode.finish();
                return false;
        }
};
```

O método **onCreateActionMode** será chamado quando o menu precisar ser criado (normalmente, chamado apenas uma vez). É aqui que montamos o menu que queremos exibir. Podemos fazer um inflate ou adicionar as opções no menu de forma manual, no código Java, da mesma forma como já fizemos com os demais tipos de menus.

Depois, quando alguém selecionar um item, será chamado o método **onActionItemClicked**. O método **onDestroyActionMode** quando o usuário deixa o ActionMode. Por fim, o método **onPrepareActionMode** é chamado todas as vezes que o menu é exibido. Pode ser chamado muitas vezes, diferente de **onCreateActionMode**.

Agora, vamos colocar uma ListView no nosso layout. Quando o usuário fizer um long-click, vamos exibir nosso menu. Vamos, então, criar um String Array com os dados que serão exibidos nesta ListView. No nosso arquivo /res/values/strings.xml, teremos o array da listagem abaixo.

```
<resources>
        <string-array name="teste">
          <item>Item 1</item>
          <item>Item 2</item>
          <item>Item 3</item>
          <item>Item 4</item>
          <item>Item 5</item>
          <item>Item 6</item>
        </string-array>
</resources>
```

Agora, precisamos colocar a ListView no layout e popular ela com esse Array.

```
<RelativeLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <ListView
    android:id="@+id/listView"
    android:layout_width="fill_parent"
    android:entries="@array/teste"
    android:layout_height="fill_parent" />
</RelativeLayout>
```

Vamos partir agora para a Activity, pois precisamos informar em qual momento o menu será exibido. No nosso caso, vamos colocar no long-click de um item da **ListView**. Lembre-se que você precisa primeiro obter a referência para esta ListView em sua Activity.

```
listView.setOnItemLongClickListener(
        new OnItemLongClickListener() {

        public boolean onItemLongClick(AdapterView<?> arg0,
                View arg1, int arg2, long arg3) {
                MainActivity.this.startActionMode(callback);
                return true;
        }
});
```

Observe que quando você der um long-click, vamos chamar o método **startActionMode** passando o callback que criamos anteriormente. Neste mesmo callback, no método **onActionItemClicked**, chamamos **mode.finish()** para que o menu seja fechado.

Da forma como fizemos, você faz um long-click em apenas um item. Mas o mais interessante é poder selecionar vários itens e excluir eles, certo? Vamos mudar um pouco as coisas, então. Primeiro, é preciso definir que o ListView aceita a seleção múltipla, chamando o método **setChoiceMode(Mode)** dele. Nossa classe de Activity ficará igual a próxima listagem.

```
public class MainActivity extends Activity {
    private ListView listView;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listView = (ListView) findViewById(R.id.listView);
        listView.setChoiceMode(
        ListView.CHOICE_MODE_MULTIPLE_MODAL);
        listView.setMultiChoiceModeListener(
            new MultiChoiceModeListener() {

                public void onItemCheckedStateChanged
                        (ActionMode mode,
                        int position,
                        long id,
                        boolean checked) { }

                public boolean onActionItemClicked
                        (ActionMode mode, MenuItem item) {
                        mode.finish();
                        return true;
                }

                public boolean onCreateActionMode
                        (ActionMode mode, Menu menu) {
                        menu.add("Ação 1");
                        menu.add("Ação 2");
                        return true;
                }

                public void onDestroyActionMode
                        (ActionMode mode) { }

                public boolean onPrepareActionMode
                            (ActionMode mode, Menu menu) {
                            return false;
                }
        });
    }
}
```

O que mudou? Observe que não precisamos mais criar uma instância de **ActionMode.Callback.** Agora precisamos de uma instância de **MultiChoiceModeListener** que é passada como parâmetro para o método **setMultiChoiceModeListener** da ListView. Com isto, você pode fazer um long-click em um item e depois ir selecionando outros da ListView. A instância de

Mas, se você executar este projeto, perceberá que não há nenhuma indicação visual disto. É porque você que deve dar um feedback pro usuário. E faz isso implementando algo no método **onItemCheckedStateChanged**. Você tem como parâmetro a posição, id e se o item da ListView que foi selecionado. Agora, você pode fazer alguma coisa, como colocar um checkbox, mudar a cor...

Por exemplo, coloque um Toast para exibir uma mensagem! Observou também como os demais métodos são parecidos com aqueles que encontramos na interface **ActionMode.Callback**? Pois é. Então, não precisa explicar tudo novo, certo? :-)

Sobre o Autor

Marlon Silva Carvalho

É um programador de longa data. Já peregrinou por diversas linguagens e hoje se considera um programador agnóstico. Atualmente, está fascinado pelo mundo mobile e suas consequências para a humanidade. Está mais focado no desenvolvimento para Android, embora também goste de criar aplicativos para iOS.

Trabalha em projetos sobre mobilidade no SERPRO, tendo atuado no projeto Pessoa Física, para a Receita Federal do Brasil. Você pode encontrar ele no Twitter, no perfil @marlonscarvalho, em seu blog, no endereço http://marlon.silvacarvalho.net e através do e-mail marlon.carvalho@gmail.com.

Sobre os Cards

Caso você já tenha visto os excelentes RefCards da DZone, deve ter percebido a semelhança. E é proposital. A ideia surgiu após acompanhar estes RefCards por um bom tempo. Contudo, eles são mais gerais. O objetivo destes Cards é tratar assuntos mais específicos. Trazendo informações relevantes sobre eles.

Caso tenha gostado, continue nos acompanhando! Tentaremos trazer sempre novos cards sobre os mais variados assuntos. Inicialmente, focaremos no Android.

O ícone principal deste card é de autoria de Wallec e foi obtido em seu profile no DeviantArt.
http://wwalczyszyn.deviantart.com/