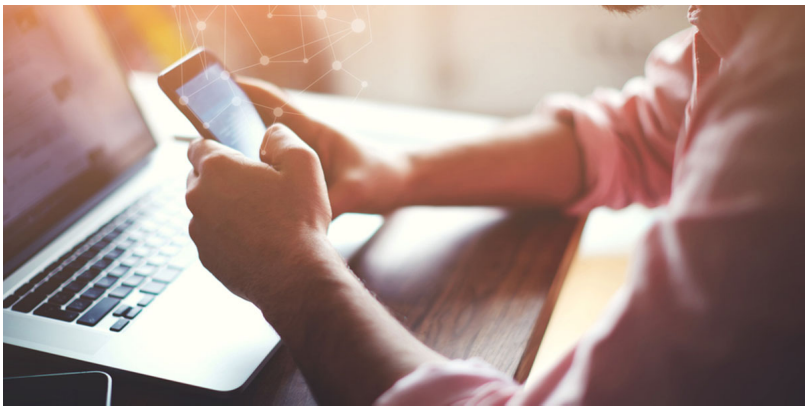
Programação para dispositivos móveis

Aula 5: Menus e Fragmentos

INTRODUÇÃO



Esta aula apresentará as classes referentes à implementação de menus e fragmentos em Android.

OBJETIVOS

Ilustrar o desenvolvimento de telas gráficas compostas por menus.

Explicar o emprego dos principais conceitos referentes a fragmentos.

MENUS

Fonte da Imagem:

Em aplicativos Android, um recurso extremamente utilizado é o de menu. Basta você baixar alguns aplicativos na Google Play Store que os encontrará facilmente.

Embora sejam de fácil implementação, devemos reforçar os cuidados em relação à usabilidade e aparência dos mesmos. É muito comum encontrarmos aplicativos sem nenhum menu, menus sem ícones ou até mesmo com títulos inapropriados.

Por isso, é importante lembrarmos de que um bom apelo visual e/ou textual facilitará muito a interação do usuário com o aplicativo.

Podemos trabalhar com três tipos de menus em Android:

Menu de opção e barra de opção

• É o menu default das aplicações Android;
• Normalmente encontramos nesse menu as principais opções.

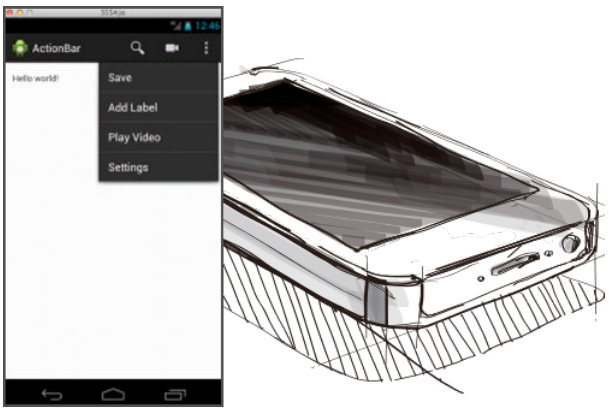
Existem dois modelos:

I. Ícone (icon menu):
• Disponível parte inferior da tela;
• Suporta até seis itens de menu;
• Suporta ícones;
• Não suporta caixa de seleção;
• Não suporta botões de rádio;
• Android 2.3.x – API Level 10 ou Inferior.

Fonte: https://developer.android.com/images/options_menu.png?hl=pt-br

II. Expandido (expanded menu):
• Suporta mais de seis itens de menu;

• Apresentado automaticamente na opção Mais (More) quando possuir mais de seis itens de menu;
• Android 3.0 – API Level 11 ou superior.



Fonte: https://dab1nmslvvntp.cloudfront.net/wp-content/uploads/2014/04/1396465832ab_with_action_buttons-181x300.jpg

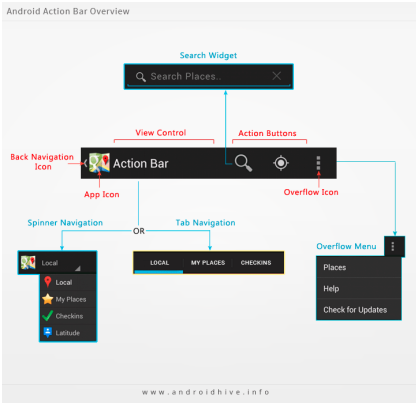
Menu de contexto

• É exibido quando o usuário clica e segura, por mais de 2 segundos, um componente visual;
• Não suporta atalhos, ícones ou até mesmo submenus;
• Pode ser compartilhado entre diferentes Views.

Menu Pop-up

• Abre quando tocamos no item de menu Options (Opções) ou em menu contextual;
• Não suporta ícones;
• Não suporta submenus aninhados;

ACTION BAR



App Icon

- Exibe o ícone do projeto ou logo customizado;
- *Back Navigation Icon* - permite a navegação para cima na hierarquia de telas.

View Control

• Exibe o título do aplicativo ou a tela em que o usuário se encontra;
• Exibe o controle de Navegação *(Drop-down ou Tabs)*.

Action Buttons

• Exibem as ações mais comuns em seu aplicativo;
• Os ícones que não couberem nesse espaço serão inseridos automaticamente no *Action OverFlow*.

Action OverFlow

• Exibe as ações não utilizadas frequentemente de seu aplicativo.

Podemos desenvolver menus Android através de:

Nossa sugestão é que você defina em um arquivo XML, pois isso facilita a visualização da estrutura do menu, bem como nos ajuda a implementar para diferentes versões do Android, tamanhos de tela etc.

Gerenciar as interações efetuadas pelos usuários ficou ainda mais fácil. Para isso, implementamos os métodos onOptionsItemSelected() e onContextItemSelected().

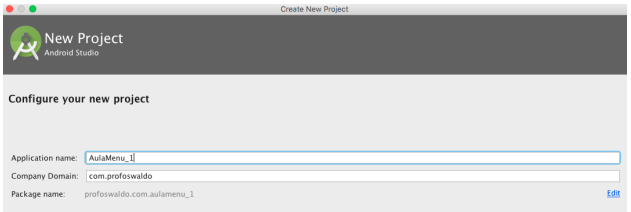
Vamos desenvolver exemplos para visualizarmos a implementação de menus na prática?

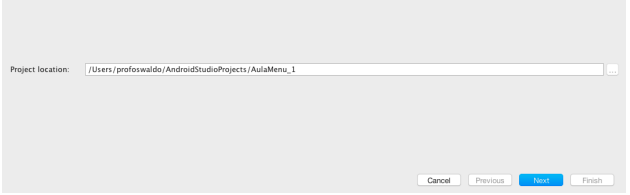
Exemplo:

Crie um novo projeto Android.

Em nosso exemplo, o chamaremos de AulaMenu_1.

Na sua tela, deverá ser similar ao exibido:



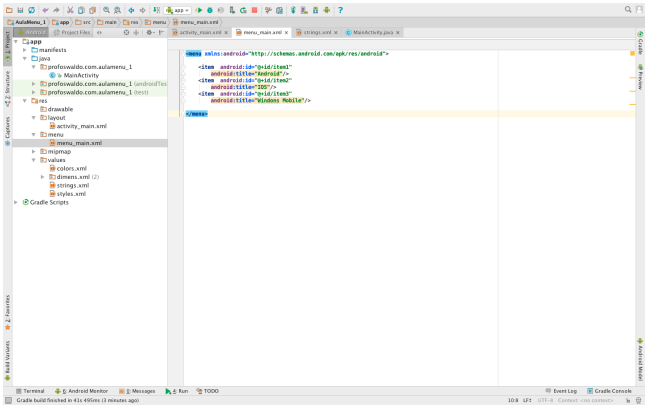


Provavelmente, não existe a pasta menu em sua árvore de recursos. Por isso, clique com o botão direito na pasta res e crie o diretório menu.

Agora crie o arquivo xml, chamado menu_main.

Nesse arquivo, serão definidos os itens que comporão nosso menu.

Veja a tela:



Se você olhar mais atentamente o código, perceberá que cada tag item é referente a uma opção do menu.

No nosso exemplo, os parâmetros são:

• android:id - Esse id é exclusivo para cada item. É através dele que podemos identificar o item de menu;
• android:title - É o texto título de nosso item de menu;
• aandroid:icon - Embora não tenha sido usado nesse exemplo, poderia definir um ícone para o nosso menu. O valor aqui é a referência a um drawable;
• android:showAsAction - Define a forma de exibição do componente.

As constantes que devemos empregar são:

always

- O componente sempre fica visível;
- Recomendado para ações mais comuns do aplicativo.

ifRoom

- O componente é exibido na action bar, se existir espaço;
- Adequado para manter compatibilidade com diversos tipos de dispositivos e também com telas na vertical ou horizontal.

withText

- O componente exibe o seu título ao lado do ícone, caso tenha espaço disponível.

nerver

- Não exibe o componente na action bar.

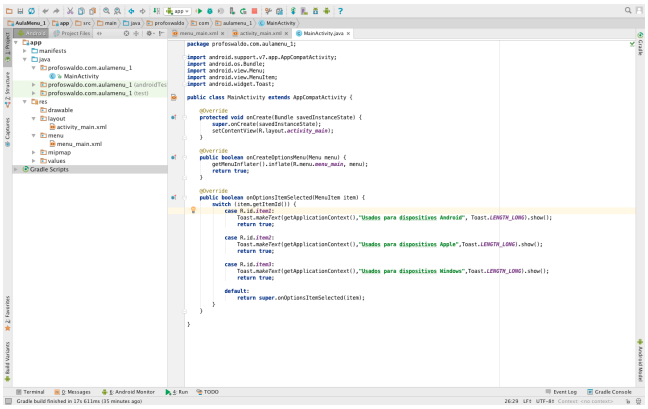
collapseActionView

- Quando a view é grande, deve ser contraída para exibir apenas um botão.

É oportuno saber que também podemos combinar as constantes com separadores, como, por exemplo, ifRoom|withText.

Faça um teste em nossos exemplos. Você vai se surpreender com essas combinações.

Nesta próxima tela, temos algumas novidades.



Quando o botão físico do menu for acionado, o método onCreateOptionsMenu (Menu menu) de nossa atividade é invocado e, em nosso exemplo, faz uso do MenuInflater para criar o menu definido no arquivo menu_main.xml.

Isso pode ser verificado no método abaixo:

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

Ao clicar em uma das opções do menu, o onOptionsItemSelected(MenuItem item) entra em ação.

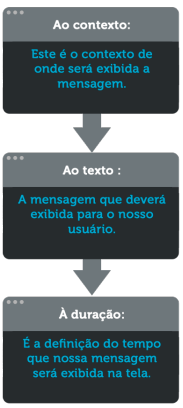
Mas não para por aí, a classe Toast(import android.widget.Toast) é outra grande novidade. Ela é bastante similar ao nosso velho conhecido JOptionPane. Seu objetivo é exibir, por alguns segundos, uma pequena e breve mensagem de alerta para nosso usuário, sobre a tela vigente de nossa aplicação.

Podemos definir sua posição e até mesmo personalizar seu layout, mas esta nunca receberá o foco.

Sua sintaxe é bastante simples de entender. Veja o exemplo abaixo:

```
Toast toast = Toast.makeText(contexto, texto, duracao);
            toast.show();
```

Os parâmetros dessa classe correspondem:

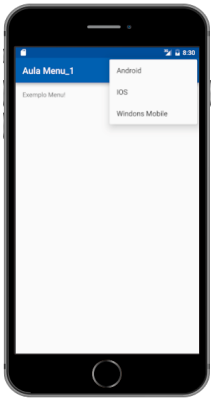


Podemos configurar a partir das seguintes constantes:

- Toast.LENGTH_LONG - 4 segundos;
- Toast.LENGTH_SHORT - 2 segundos.

Após configurarmos nossa mensagem, é necessário executar o método .show() para que possa ser exibida.

Pronto. Excute sua aplicação para ver a tela abaixo.



Fonte da Imagem:

Selecione uma das opções de nosso menu.

Observe que a mensagem foi exibida na parte inferior da tela.

Podemos alterar isso. Basta definirmos o método .setGravity().

Sua sintaxe é *toast.setGravity(constante, valor_x, valor_y)*, onde:

• Constante:
Constante Gravity (glossário). Existem várias constantes que você pode empregar e até mesmo combinar.

• Valor_x:
Deslocamento x da posição;

• Valor_y:
Deslocamento y da posição.

Exemplo 1: *toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);*

Vamos subir um pouco mais o nível de nosso menu?

Então, crie um novo projeto Android, veja abaixo.

FRAGMENTO

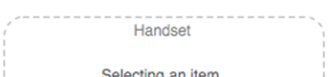
Bastante similar a uma Activity, um fragmento (*fragment*) consiste em uma pequena porção de Activty, que permite um projeto mais modular.

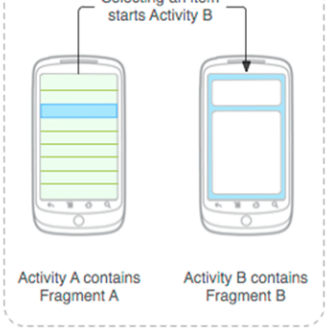
Não seria errado afirmarmos, assim, que um fragment é uma espécie de *subactivity*.

Esse conceito surgiu com o Android 3.0 (*Honeycomb*) devido à necessidade de customizá-lo para as interfaces dos aplicativos, em função da pluralidade de tipos e tamanhos de dispositivos, em especial os *tablets*.

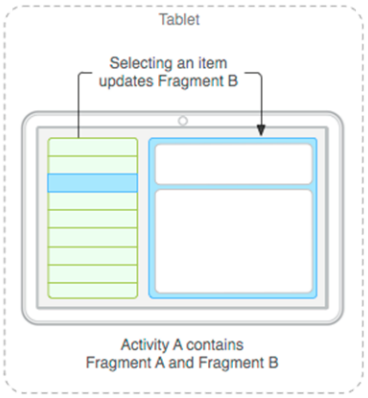
Veremos agora que um *fragment* é muito mais do que dividir a tela em duas ou mais.

Vamos analisar o exemplo da tela abaixo para compreender o conceito:





Podemos perceber que na do *SmartPhone*, ao selecionarmos uma opção da lista, é necessário apresentar outra tela, devido ao tamanho do dispositivo.

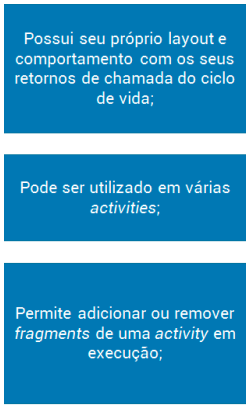


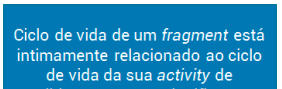
Já na figura do tablet a navegabilidade se dá na mesma tela.

É uma boa prática de desenvolvimento em *Android* sempre encapsular o código de uma *activity* em um *fragment*. Isso permite a reutilização deste código em outro contexto.

CARACTERÍSTICAS

Veja algumas características importantes desse:





acolhimento, o que significa que, quando a *activity* estiver em pausa, todos os *fragments* disponíveis na *activity* também serão interrompidos;

Permite combinar vários *fragments* em uma única *activity*;

Permite implementar um comportamento que não tem nenhum componente interface do usuário.

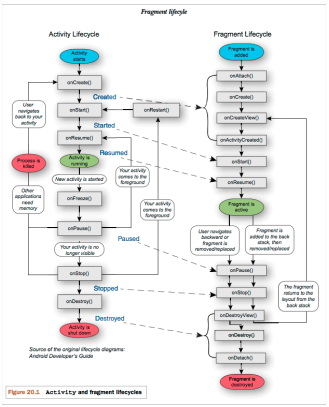
CARACTERÍSTICAS

Ciclo de vida de *Fragment*:

Embora o *Fragment* possua seu próprio ciclo de vida, que é bastante similar ao de uma *activity*, este não é uma entidade independente.

Na verdade, é parte de uma *activity* hospedeira, que orienta o seu ciclo de vida.

Isso é demonstrado na figura abaixo:



https://docs.google.com/file/d/0BxbayAAcS8IiRld4WTVoZE5Bd0U/edit

MÉTODOS DO CICLO DE VIDA

Esta lista mostra os principais métodos do ciclo de vida:

Método	Descrição
public void onAttach(Activity activity)	Chamado quando a Activity está no estado Created depois que o fragment é associado a sua Activity
public void onCreate(Bundle savedInstanceState)	Chamado quando a Activity está no estado Created para fazer a criação inicial do fragmente
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)	Chamado quando a Activity está no estado Created para criar e retornar à hierarquia de views associada ao fragment
public void onActivityCreated(Bundle savedInstanceState)	Chamado quando a Activity está no estado Created para informar ao fragment de que sua Activity concluiu sua própria Activity onCreate()
public void onStart()	Chamado quando a Activity está no estado Started, indicando que o fragment está visível para o usuário
public void onResume()	Chamado quando a Activity está no estado Resumed, indicando que o fragment agora está interagindo com o usuário.
public void onPause()	Chamado quando Activity está no estado Paused, indicando que o fragment não está mais interagindo com o usuário porque sua Activity está sendo pausada ou uma operação de fragment o está modificando na Activity
public void onStop()	Chamado quando a Activity está no estado Stopped, indicando que o fragment não está mais visível para o usuário porque sua Activity está sendo interrompida
public void onDestroyView()	Chamado quando a Activity está no estado Destroyed para permitir que o fragment limpe os recursos associados a sua view
public void onDestroy()	Chamado quando a Activity está no estado Destroyed para permitir que o fragment faça a limpeza final do estado do fragment
public void onDetach()	Chamado quando a Activity está no estado Destroyed imediatamente anterior ao fragment deixar de estar associado a sua Activity

FRAGMENTS E SUAS SUBCLASSES

Para implementar um *fragment* é preciso estender uma das classes abaixo:

Fragments - Comportamento específico dentro de uma *Activity* (Ex: parte de interface ou operação);
DialogFragment - Exibir uma janela por cima da janela de sua *Activity*;

ListFragment - Exibe uma lista de itens de uma fonte de dados;
PreferenceFragment - Armazena e acessa dados de configuração de uma aplicação;
WebViewFragment - Exibe *WebView*.

API DE FRAGMENTS:

As principais classes da API Fragments são:

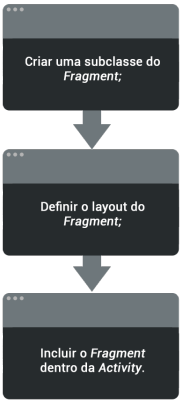
Fragment(android.app.Fragment)
• Classe que o fragment deve estender;
• É necessário sobrescrever o método onCreate (inflater, container, bundle) para criar a view.

Fragment(android.app.Fragment)
Classe que gerencia os fragments pela API;
• Possui métodos findFragmentById(id) e findFragmentByTag(tag) utilizados para encontrar os fragments no layout, de forma similar ao método findViewById(id) que uma activity utiliza para buscar uma view.

Fragment(android.app.FragmentTransaction)
• Classe utilizada para adicionar, remover ou substituir os fragments dinamicamente no layout.

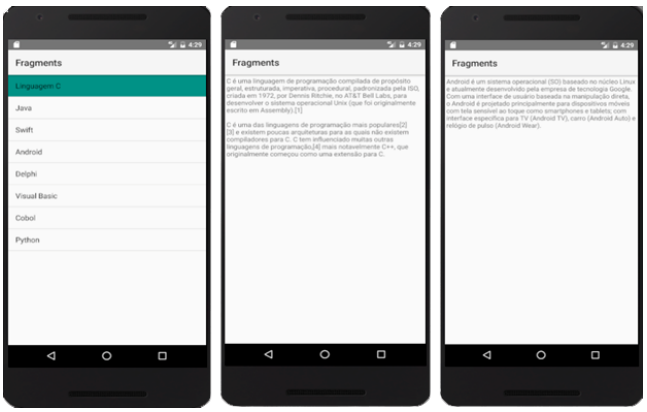
CRIANDO FRAGMENTS

Para implementar um Fragment, basta seguir os três passos básicos:



ATIVIDADE

Observe as imagens:



Desenvolva um pequeno aplicativo que, como demonstrado na tela abaixo, ao selecionarmos uma linguagem de programação, exiba uma breve descrição da mesma.

Resposta Correta

Glossário