

# Introdução ao Desenvolvimento Android

Vitor Freitas

[vitor@freitas.com](mailto:vitor@freitas.com)

<http://vitorfs.com>



# Site do Mini-Curso

<http://vitorfs.com/android/>

- Material didático
- Exemplos

# Conteúdo

- Android
- Criando Um Projeto Android
- Executando Sua App
- Criando Uma Interface de Usuário
- Iniciando Uma Nova Activity
- Ciclo de Vida Activity
- Exemplo Conversor
- Criando Menus
- AsyncTask
- Gerando uma APK

# Android

- Sistema operacional para dispositivos móveis, baseado em Linux
- Plataforma mais popular para dispositivos móveis

# Android

## Suporte a Linguagens

- SDK – Java
- NDK – C++

# Android

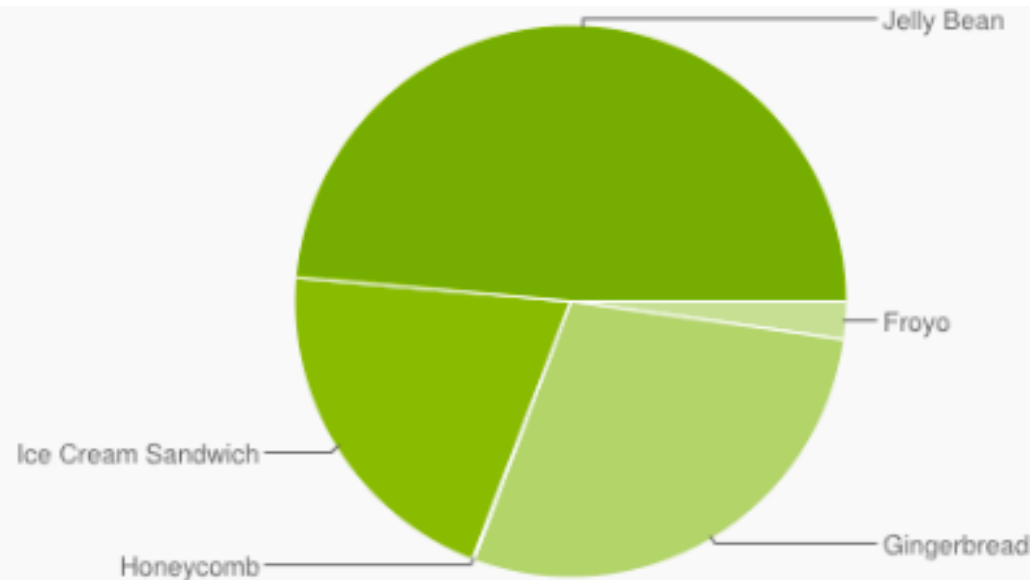
## Apps

- Normalmente escritos em linguagem Java compilados pela SDK do Android
- Não possui suporte à bibliotecas AWT e Swing
- IDE Eclipse

# Android

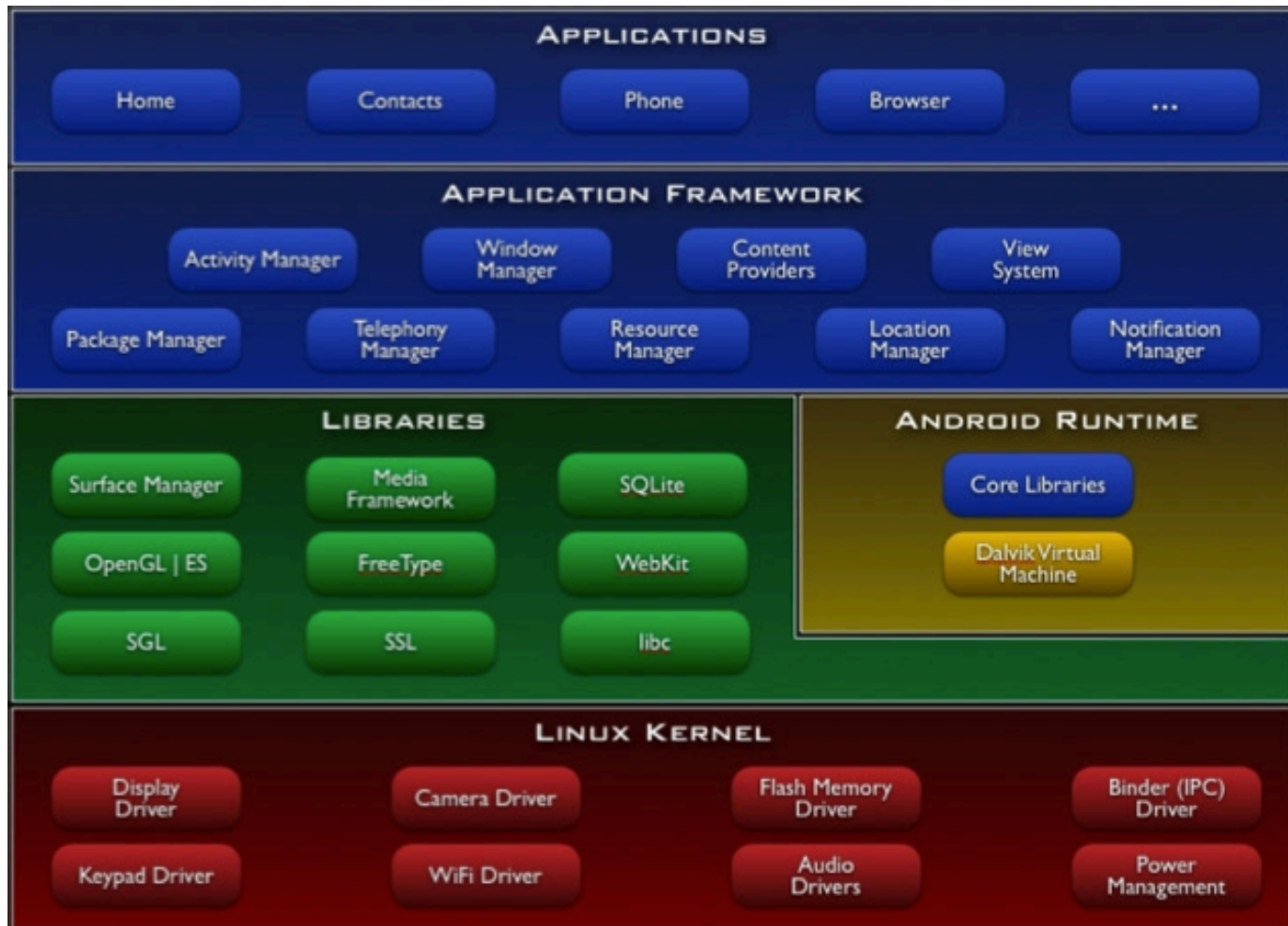
## Versões

Version	Codename	API	Distribution
2.2	Froyo	8	2.2%
2.3.3 - 2.3.7	Gingerbread	10	28.5%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	20.6%
4.1.x	Jelly Bean	16	36.5%
4.2.x		17	10.6%
4.3		18	1.5%



# Android

## Arquitetura





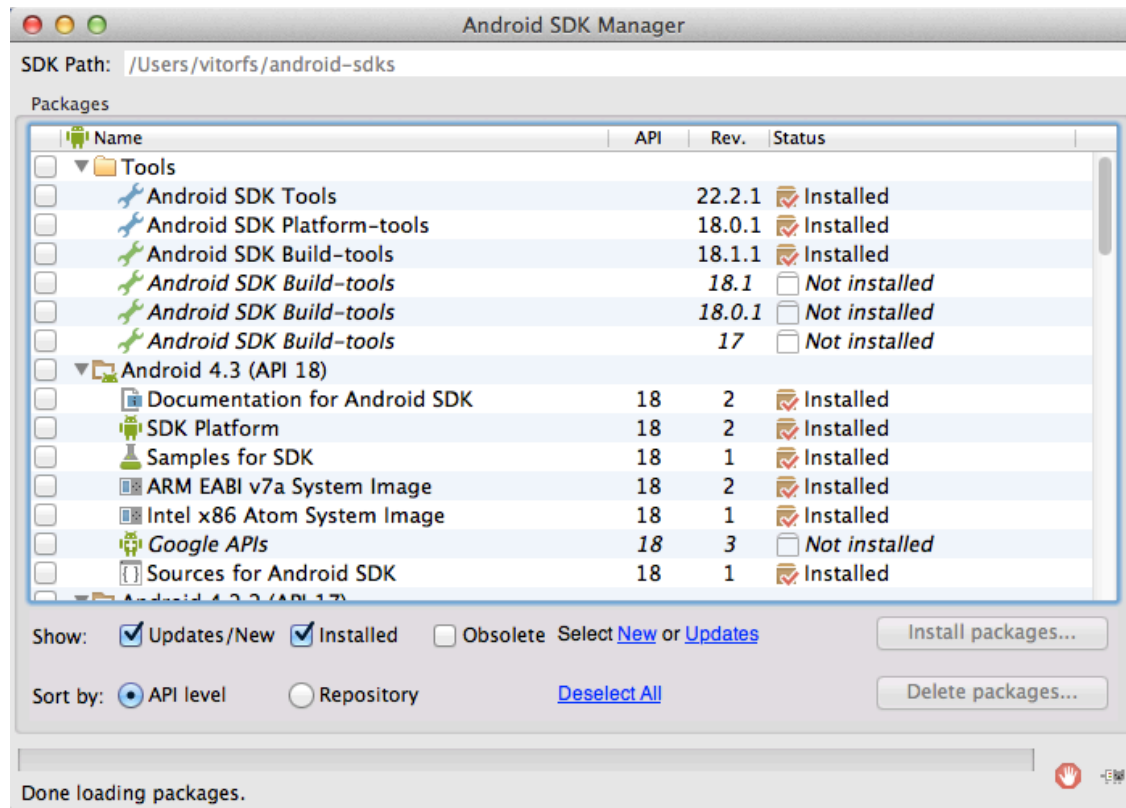
# Android

- Download Eclipse + SDK

<http://developer.android.com/sdk/index.html>

# Android


- Atualize o SDK e baixe outras versões do framework pelo Android SDK Manager



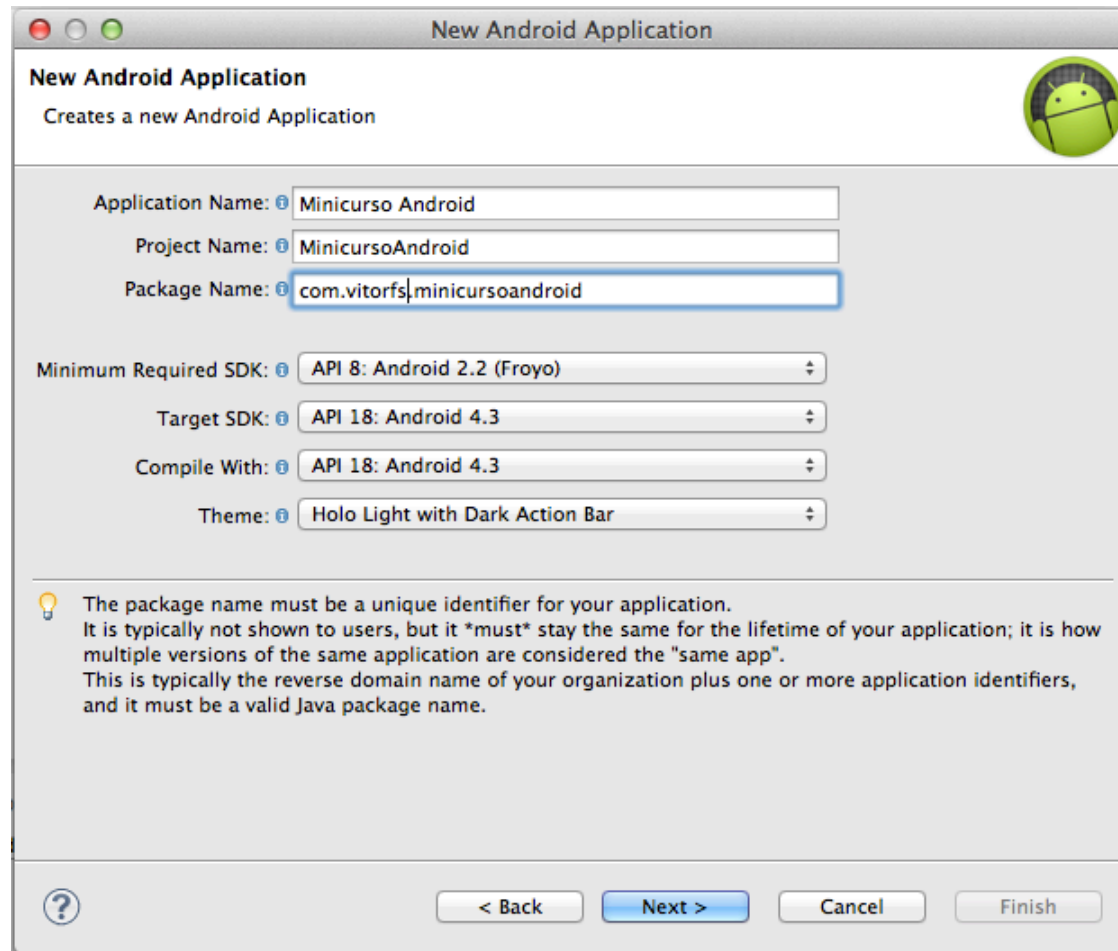
# Criando Um Projeto Android

- Um projeto Android contém todos os arquivos que compõem o código fonte da sua app
- As ferramentas do Android SDK facilitam a criação da estrutura padrão de diretórios e arquivos de um projeto

# Criando Um Projeto Android

1. Clique em **New**  na barra de tarefas
2. Na janela que aparecer, dentro do diretório **Android**, selecione **Android Application Project**, e clique em **Next**
3. Preencha o formulário exibido

# Criando Um Projeto Android



**New Android Application**  
Creates a new Android Application

Application Name:

Project Name:


Package Name:


Minimum Required SDK:

Target SDK:

Compile With:

Theme:

 The package name must be a unique identifier for your application. It is typically not shown to users, but it *must* stay the same for the lifetime of your application; it is how multiple versions of the same application are considered the "same app". This is typically the reverse domain name of your organization plus one or more application identifiers, and it must be a valid Java package name.



# Criando Um Projeto Android

- **Application Name:** nome da app que será exibido ao usuário
- **Project Name:** nome do diretório do projeto e o nome que ficará visível no Eclipse
- **Package Name:** define o package namespace da sua app (seguindo as mesmas regras de nome de pacote do Java). O nome do seu pacote deve ser único no contexto dos pacotes instalados no sistema Android

# Criando Um Projeto Android

- **Minimum Required SDK:** versão mais antiga do Android que a sua app deverá suportar
- **Target SDK:** indica a maior versão que você testou a sua app
- **Compile With:** determina para qual versão você compilará a sua app. Por padrão, é definido a versão mais recente do Android
- **Theme:** especifica o Android UI que será aplicado na sua app

# Criando Um Projeto Android

4. Na tela seguinte, **Configure Project**, mantenha as configurações padrão e clique em **Next**
5. A tela seguinte permite a alteração do ícone da sua app. A ferramenta gera um ícone para cada densidade de tela. Clique em **Next**
6. Selecione uma activity template para iniciarmos a construção da app. Para este projeto, selecione `BlankActivity` e clique em **Next**
7. Mantenha as configurações padrão da activity e clique em **Finish**



# Executando Sua App

- `AndroidManifest.xml`
  - Este arquivo descreve características fundamentais da app e define cada um de seus componentes
  - Um dos elementos mais importantes que seu **manifest** deve incluir é o elemento `<uses-sdk>`. Este elemento determina a compatibilidade da sua app com as diferentes versões do Android

# Executando Sua App

- `src/`
  - Diretório de código fonte da sua app
- `res/`
  - Composto por diversos subdiretórios que contém os recursos da app
  - `drawable-hdpi/`
    - Diretório para arquivos de imagem como bitmaps que foram desenhados para telas de grande densidade (hdpi)

# Executando Sua App

- `layout/`

- Diretório de arquivos que definem a interface de usuário da sua app

- `values/`


- Diretório para arquivos XML variados, que contém uma coleção de recursos, como strings e definições de cores

# Executando Sua App

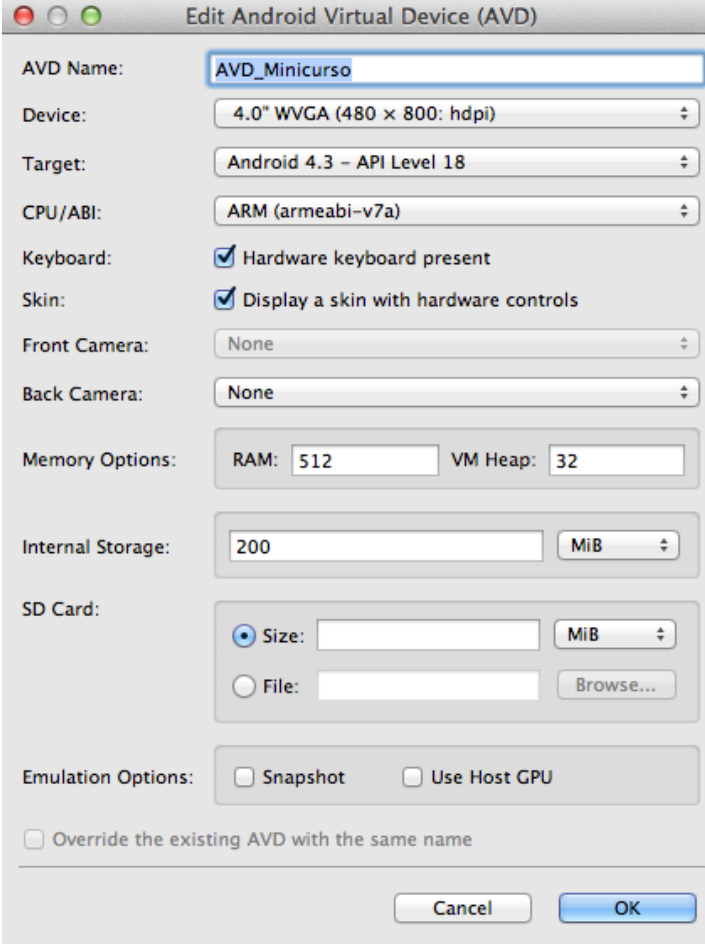
- Antes de executar sua primeira app Android, é necessário criar uma **Android Virtual Machine (AVD)**
- Uma AVD representa a configuração de um dispositivo Android, permitindo assim testar sua aplicação em diferentes versões e modelos

# Executando Sua App

Para criar uma nova AVD:

1. Abra o Android Virtual Device Manager 
2. Na janela do Manager, clique em **New**
3. Preencha os detalhes do AVD. Dê um nome, plataforma alvo, tamanho de SD Card e uma skin
4. Clique em **Create AVD**
5. Selecione a AVD criada e clique em **Start**
6. Depois que o emulador iniciar, desbloqueie a tela do dispositivo

# Executando Sua App



AVD Name: AVD\_Minicurso

Device: 4.0" WVGA (480 × 800: hdpi)

Target: Android 4.3 – API Level 18

CPU/ABI: ARM (armeabi-v7a)

Keyboard: ☒ Hardware keyboard present

Skin: ☒ Display a skin with hardware controls

Front Camera: None

Back Camera: None

Memory Options: RAM: 512 VM Heap: 32

Internal Storage: 200 MiB

SD Card: ☒ Size: MiB ☐ File: Browse...

Emulation Options: ☐ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

Cancel OK

# Executando Sua App

Para executar sua App:

1. Abra um dos arquivos de código fonte do seu projeto e clique em **Run**
2. Na janela **Run as** que aparecer, selecione **Android Application** e clique em **OK**

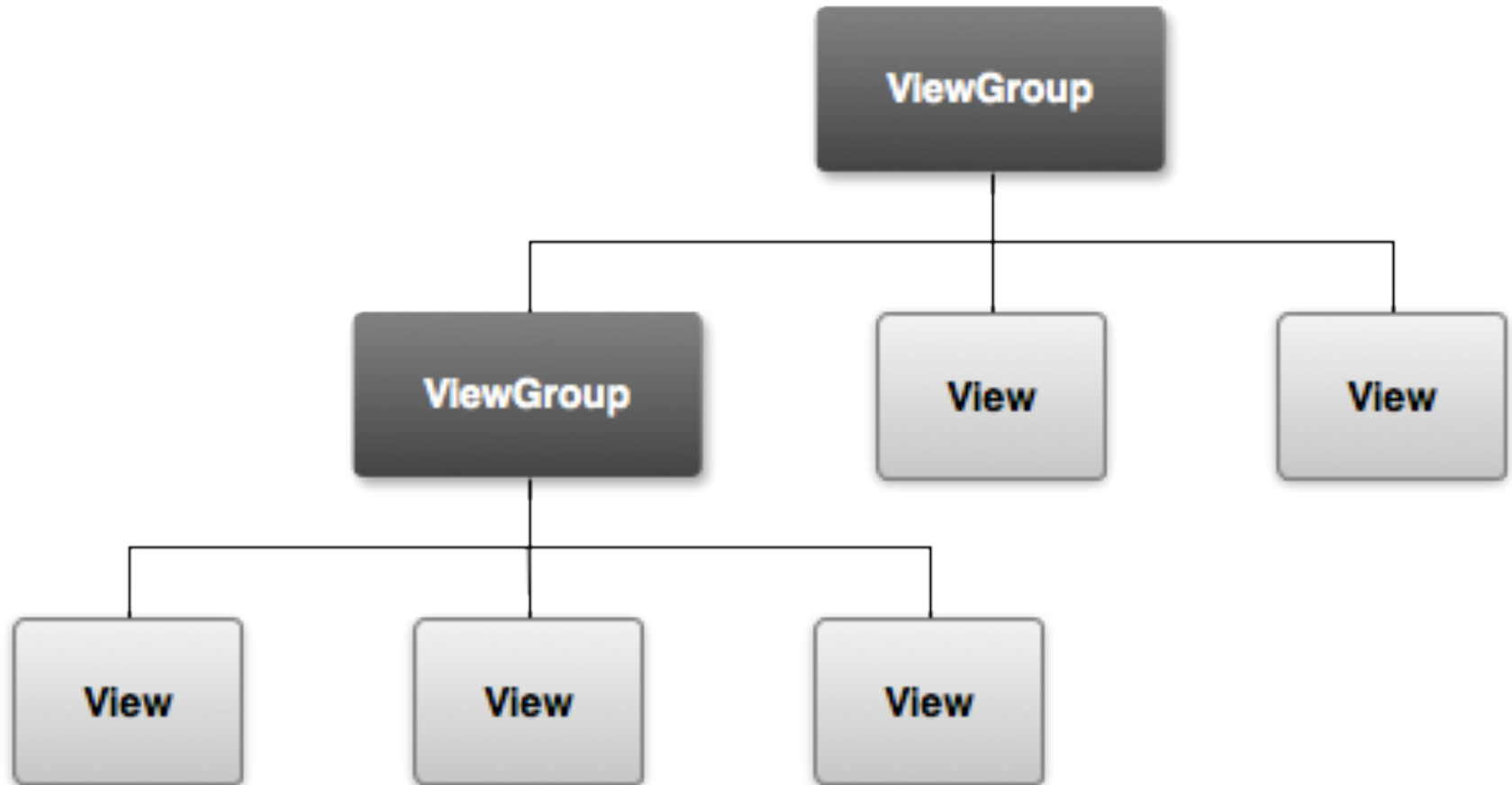
Importante: sempre mantenha a AVD aberta, para agilizar os testes de execução durante o desenvolvimento. Para testar alterações no código não é necessário inicializar o AVD novamente.

# Criando Uma Interface de Usuário

- As interfaces gráficas do Android são construídas em uma hierarquia de objetos **View** e **ViewGroup**
- Um objeto **View** são *UI widgets* como botões ou campos de texto
- **ViewGroup** são objetos invisíveis que se comportam como *containers* que definem como os objetos **View** serão dispostos na tela



# Criando Uma Interface de Usuário



# Criando Uma Interface de Usuário

## Construindo um Layout Linear

- Abra o arquivo `activity_main.xml` do diretório `res/layout/`
- Remova o `TextView`
- Altere o elemento `RelativeLayout` para `LinearLayout` e adicione um atributo `android:orientation="vertical"`

# Criando Uma Interface de Usuário

- LinearLayout é uma subclasse de ViewGroup que comporta elementos filhos na horizontal ou na vertical
- Cada filho de LinearLayout aparece na tela na ordem de aparição do XML
- Os outros dois atributos, `android:layout_width` e `android:layout_height`, são obrigatórios para todas as Views para especificar seus respectivos tamanhos

# Criando Uma Interface de Usuário

- Como o nosso `LinearLayout` é uma view root, ela deve preencher toda área disponível da tela, definindo altura e largura como `match_parent`
- Este valor define que o elemento deve ter exatamente a altura e a largura do elemento pai

# Criando Uma Interface de Usuário

```
<LinearLayout xmlns:android="http://  
schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity"  
    android:orientation="vertical">  
  
</LinearLayout>
```

# Criando Uma Interface de Usuário

## Adicionando um Text Field

- Adicione um elemento `<EditText>` dentro do `<LinearLayout>`
- Como qualquer outro objeto View, você deve definir alguns atributos XML para especificar as propriedades do objeto

# Criando Uma Interface de Usuário

```
<EditText android:id="@+id/edit_message"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:hint="@string/edit_message" />
```

# Criando Uma Interface de Usuário

- android:id
  - Identificador único da view, utilizado para referenciar o objeto a partir do código da app
  - A arroba (@) é necessária sempre que estiver referenciando algum objeto de recurso a partir do XML. A arroba é seguida pelo tipo de recurso (id, neste caso), uma barra e o nome do recurso (edit\_message)
  - O símbolo de soma (+) antes do tipo de recurso somente é necessário quando você está definindo o id do recurso pela primeira vez. Quando o projeto é compilado, a SDK cria um novo recurso no arquivo gen/R.java



# Criando Uma Interface de Usuário

- `android:layout_width` e `android:layout_height`
  - Ao invés de utilizar um tamanho específico, utilizamos o valor `wrap_content`, definindo que a View deve ter todo o tamanho necessário para visualização
- `android:hint`
  - Texto padrão que será exibido quando o campo estiver vazio
  - Este atributo pode receber um texto como parâmetro. Mas é uma boa prática definir todas constantes no resource string

# Criando Uma Interface de Usuário

## Adicionando Um String Resource

- Por default, seu projeto Android possui um string resource no caminho `res/values/strings.xml`
- Adicione uma nova string com o nome “edit\_message” e defina seu valor como “Informe uma mensagem”
- Adicione outra string com nome “button\_send” e defina seu valor como “Enviar”

# Criando Uma Interface de Usuário

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Minicurso Android</string>
    <string name="action_settings">Settings</string>
    <string name="edit_message">Informe uma mensagem</string>
    <string name="button_send">Enviar</string>
</resources>
```

# Criando Uma Interface de Usuário

## **Adicionando um Botão**

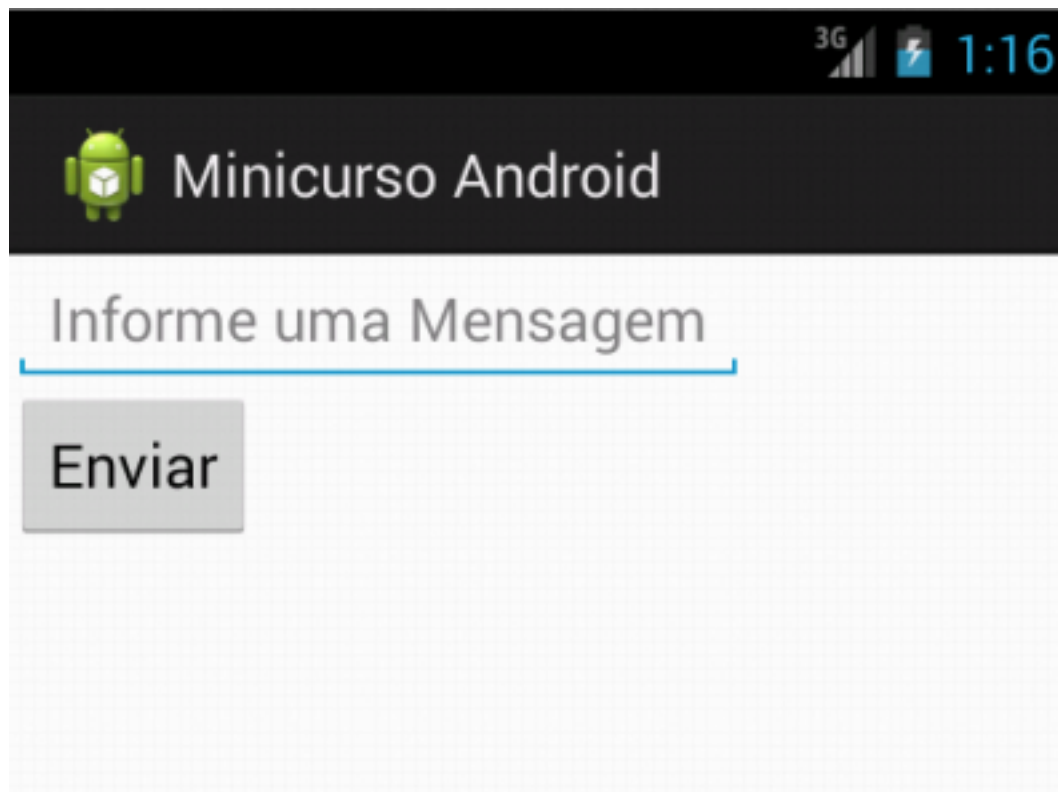
- Adicione um elemento `<Button>`
- Defina a largura e altura como `wrap_content`, para que o botão ocupe somente o espaço necessário de visualização

# Criando Uma Interface de Usuário

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send" />
```

# Criando Uma Interface de Usuário

O que temos até agora...



# Criando Uma Interface de Usuário

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />
</LinearLayout>
```

# Iniciando Uma Nova Activity

- Para responder ao evento de clique do botão que adicionamos à nossa interface, abra o arquivo de layout `activity_main.xml` e adicione o atributo `android:onClick`

`<Button`

```
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```



# Iniciando Uma Nova Activity

- O valor que adicionamos, “sendMessage” é o nome do método na sua activity
- Abra a classe MainActivity (no diretório src/) e adicione o seguinte método:

```
public void sendMessage(View view) {  
    // código...  
}
```

# Iniciando Uma Nova Activity

- Importe a classe View

```
import android.view.View;
```

Dica: Ctrl + Shift + O para importar as classes que estão faltando no arquivo

# Iniciando Uma Nova Activity

## Construindo Intents

- Objetos **Intent** são utilizados geralmente para inicializar uma nova **Activity**
- Dentro do método `sendMessage()`, crie uma Intent para inicializar uma Activity denominada `DisplayMessageActivity`

# Iniciando Uma Nova Activity

```
Intent intent = new Intent(this,  
DisplayMessageActivity.class);
```

# Iniciando Uma Nova Activity

- O constructor de Intent recebe dois parâmetros:
  - Um Context como primeiro parâmetro
  - Uma classe da app que o sistema deve entregar à Intent (neste caso, a Activity que gostaríamos de inicializar)

# Iniciando Uma Nova Activity

- Uma Intent permite trafegar dados entre as Activities
- Utilize o método `findViewById()` para recuperar o elemento `EditText` definido no `activity_main.xml`
- Adicione o texto recuperado do campo, dentro da Intent

# Iniciando Uma Nova Activity

```
public final static String EXTRA_MESSAGE =  
    "com.vitorfs.minicursoandroid.MESSAGE";  
  
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
    EditText editText = (EditText) findViewById(R.id.edit_message);  
    String message = editText.getText().toString();  
    intent.putExtra(EXTRA_MESSAGE, message);  
    startActivity(intent);  
}
```

# Iniciando Uma Nova Activity

- `intent.putExtra()`
  - O método `putExtra()` permite trafegar dados entre as Activities
  - Este método trabalha com um conjunto de Chave e Valor
  - O primeiro parâmetro é a Chave, que deve ser um identificador único no contexto da app
  - O segundo parâmetro é o valor que será passado

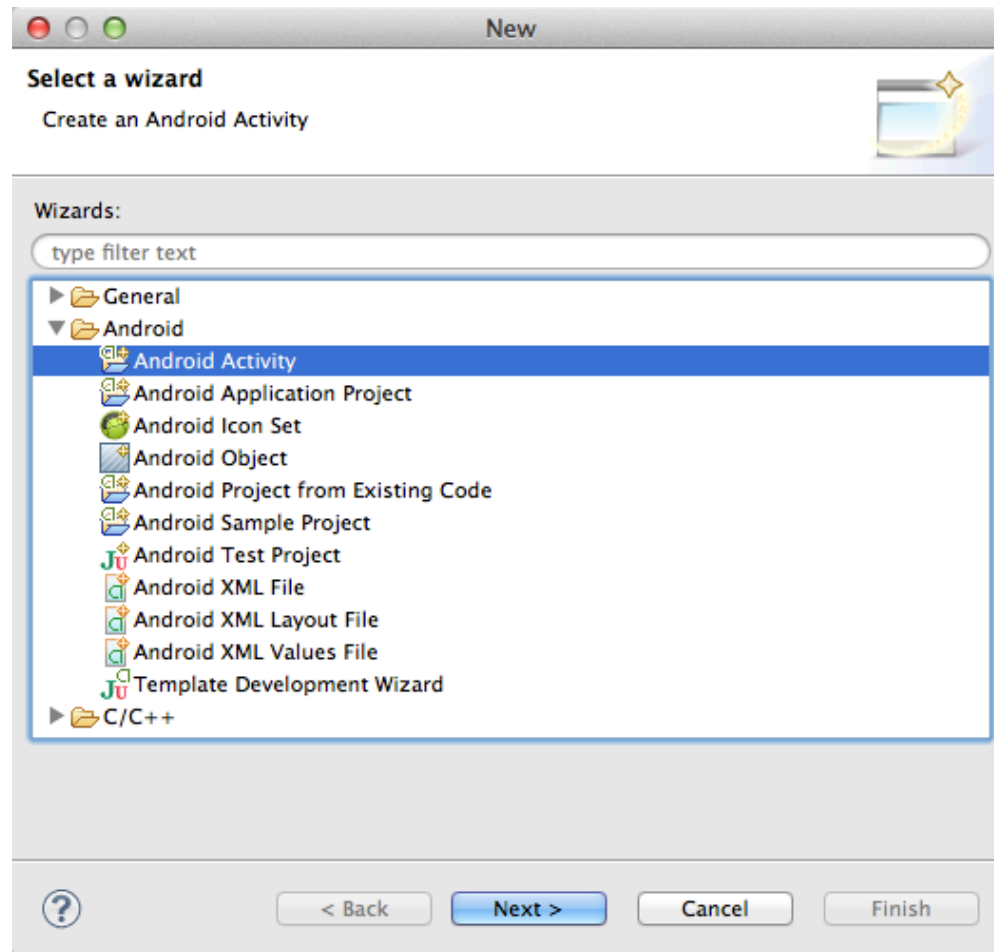
Dica: é uma boa prática definir uma constante, com o prefixo da app, de modo a garantir que o identificador seja único



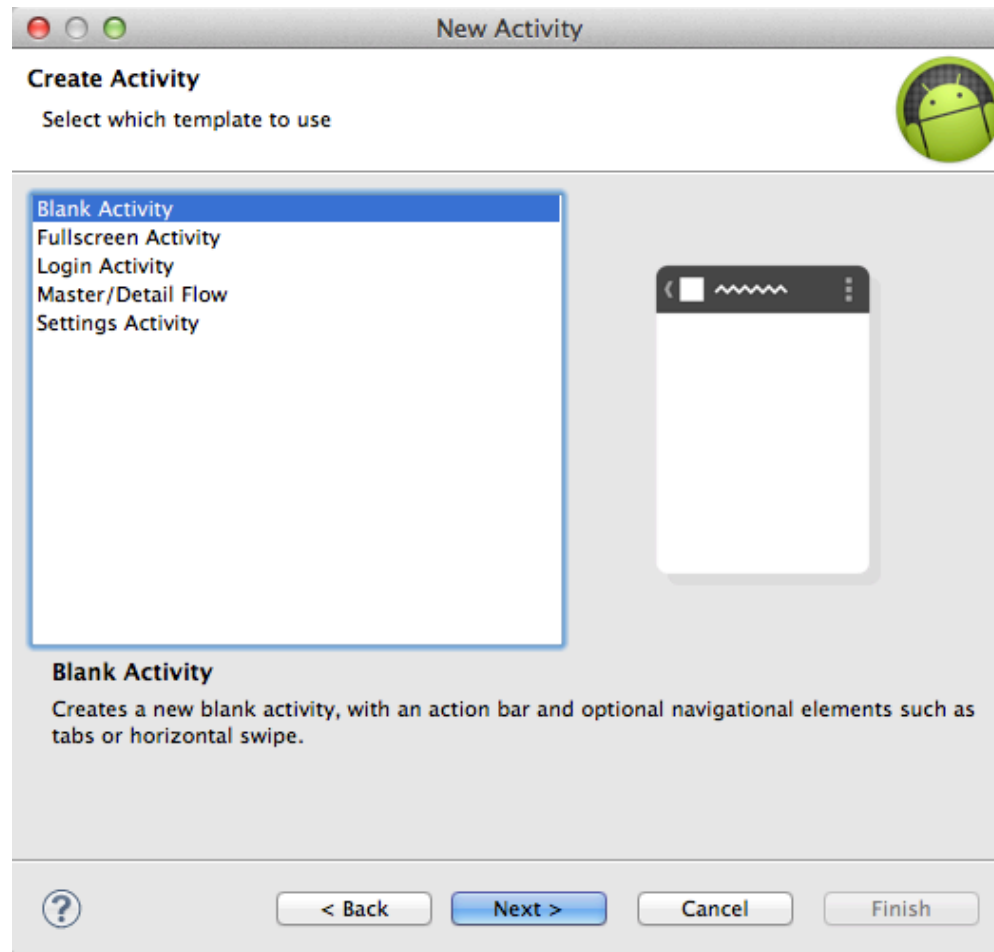
# Iniciando Uma Nova Activity

- Neste ponto é necessário criar uma nova Activity, chamada `DisplayMessageActivity`
  1. Clique em **New** na barra de ferramentas
  2. Abra o diretório **Android** e selecione **Android Activity**. Clique em **Next**
  3. Selecione o template **BlankActivity** e clique em **Next**
  4. Preecha os detalhes da Activity
  5. Clique em **Finish**

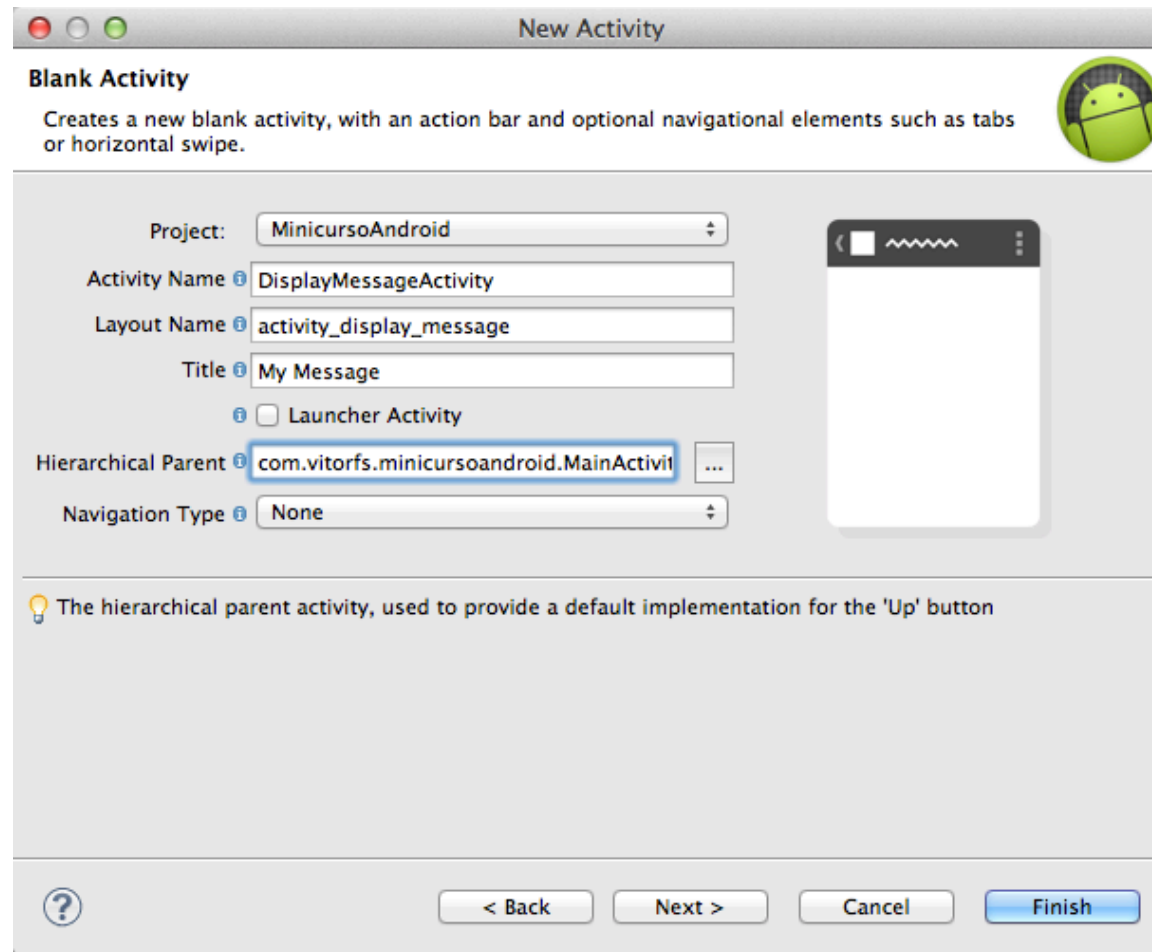
# Iniciando Uma Nova Activity



# Iniciando Uma Nova Activity



# Iniciando Uma Nova Activity



**New Activity**

**Blank Activity**

Creates a new blank activity, with an action bar and optional navigational elements such as tabs or horizontal swipe.

Project: MinicursoAndroid

Activity Name: DisplayMessageActivity


Layout Name: activity\_display\_message


Title: My Message


☐ Launcher Activity


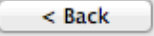
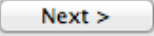

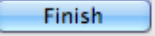
Hierarchical Parent: com.vitorfs.minicursoandroid.MainActivity

Navigation Type: None





 The hierarchical parent activity, used to provide a default implementation for the 'Up' button

# Iniciando Uma Nova Activity

- Project: MinicursoAndroid
- Activity Name: DisplayMessageActivity
- Layout Name: activity\_display\_message
- Title: My Message
- Hierarchical Parent:  
com.vitorfs.minicursoandroid.MainActivity
- Navigation Type: None

# Iniciando Uma Nova Activity

- Adicionando a nova Activity ao Manifest
- Todas activities devem ser declaradas no arquivo de manifest, utilizando o elemento `<activity>`
- Utilizando o Eclipse, a IDE faz esse trabalho para você

# Iniciando Uma Nova Activity

- O arquivo **AndroidManifest** deve ter as seguintes informações:

```
<activity
    android:name="com.vitorfs.minicursoandroid.DisplayMessageActivity"
    android:label="@string/title_activity_display_message"
    android:parentActivityName="com.vitorfs.minicursoandroid.MainActivity" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.vitorfs.minicursoandroid.MainActivity" />
</activity>
```

# Iniciando Uma Nova Activity

## Recebendo a Intent

- Toda Activity é chamada por uma Intent.
- Você pode acessar a Intent que chamou a sua Activity utilizando o método `getIntent()` e recuperar os dados que ela trouxe



# Iniciando Uma Nova Activity

- Dentro do método onCreate() da classe DisplayMessageActivity, acesse a Intent e recupere seus dados, enviados a partir da MainActivity

```
Intent intent = getIntent();  
String message =  
intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

# Iniciando Uma Nova Activity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    setContentView(R.layout.activity_display_message);
    // Show the Up button in the action bar.
    setupActionBar();
}
```

# Iniciando Uma Nova Activity

## Exibindo a Mensagem

- Para exibir a mensagem na tela, crie um `TextView` e defina o texto utilizando o método `setText()`
- Adicione o `TextView` como root view do layout da Activity, passando ele como parâmetro do método `setContentView()`

# Iniciando Uma Nova Activity

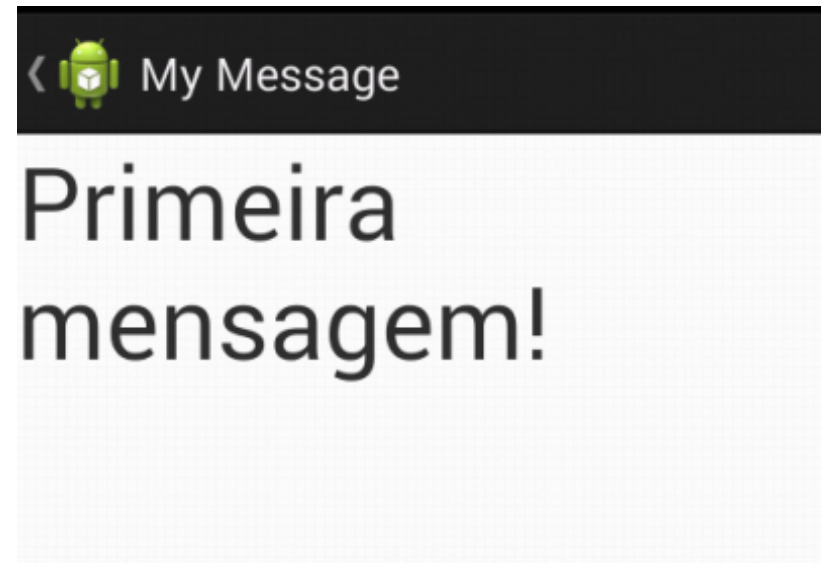
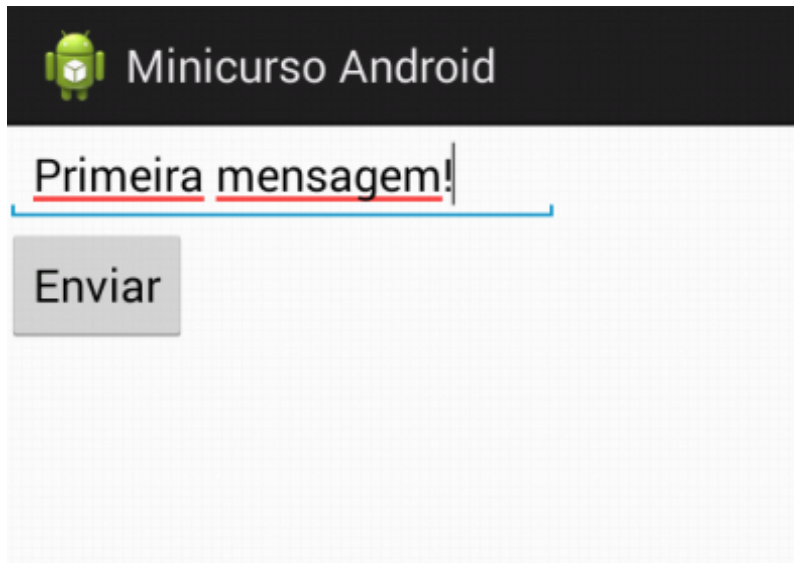
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    TextView textView = new TextView(this);
    textView.setTextSize(40);
    textView.setText(message);

    setContentView(textView);
    // Show the Up button in the action bar.
    setupActionBar();
}
```

# Iniciando Uma Nova Activity



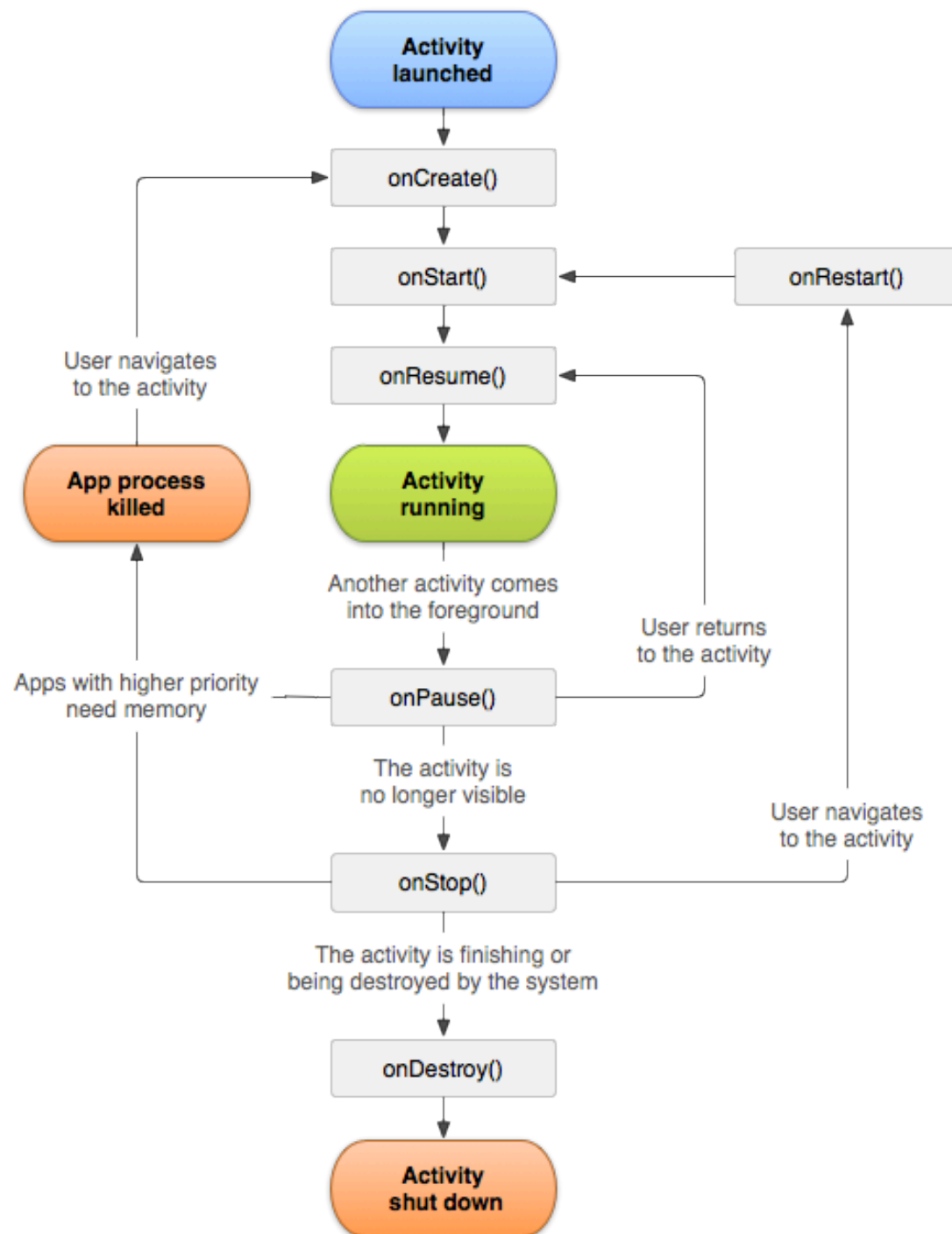
# Iniciando Uma Nova Activity

- Exemplo disponível para download em:

<http://vitorfs.com/android/downloads/MinicursoAndroid.zip>

# Ciclo de Vida Activity

- Activities possuem basicamente quatro estados:
  - Se ela estiver visível na tela (no topo do stack), ela está *active* ou *running*
  - Se a activity perdeu o foco mas ainda é visível (uma activity que não ocupa toda área da tela ou uma activity transparente recebe o foco por cima da sua activity), ela fica *paused*
  - Se a activity é completamente substituída por uma nova, ela entra em status de *stopped*
  - Se uma activity estiver *paused* ou *stopped*, o sistema pode excluí-la da memória quando precisar





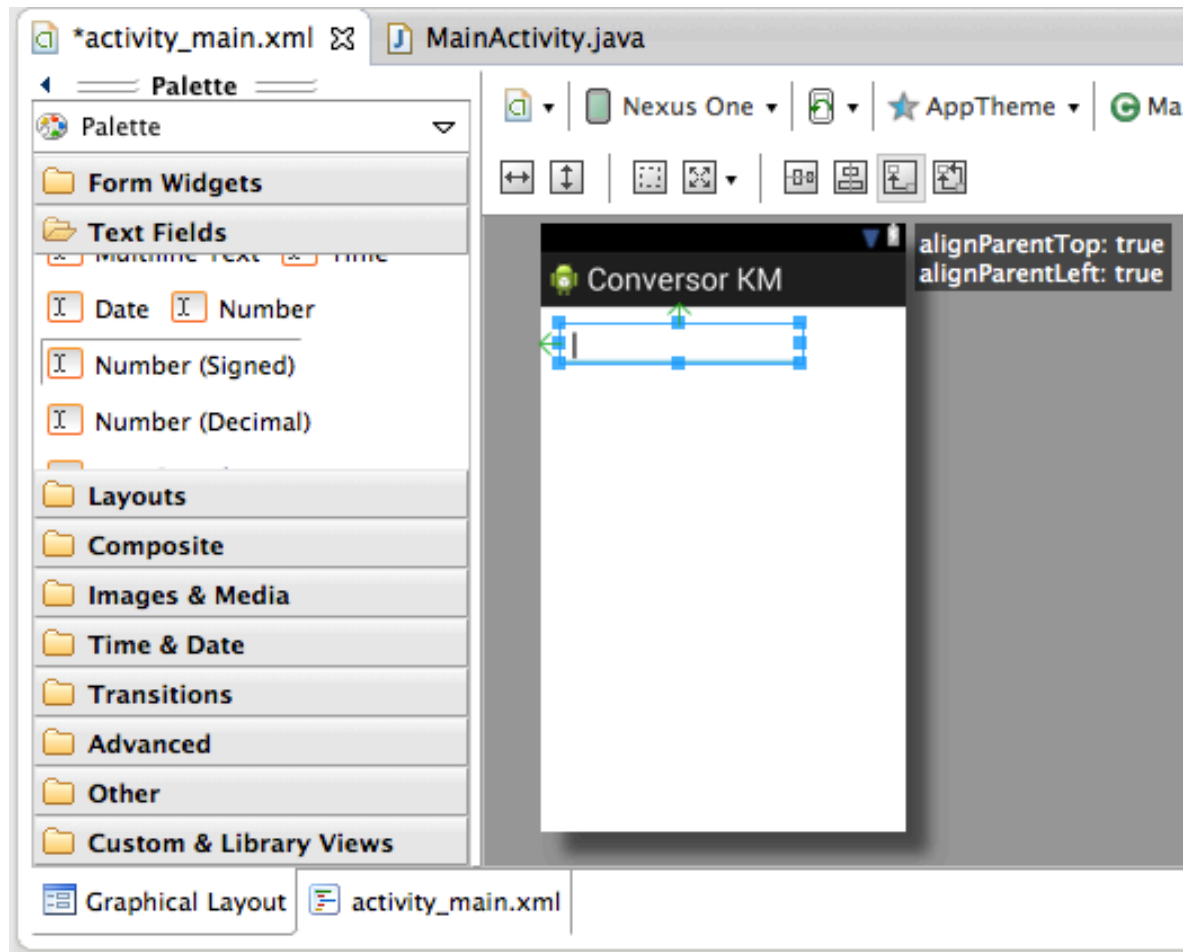
# Exemplo Conversor

- Criar uma app para converter quilômetros para milhas
- EditText para entrada de dados
- Button para converter
- TextView para exibir o resultado

# Exemplo Conversor

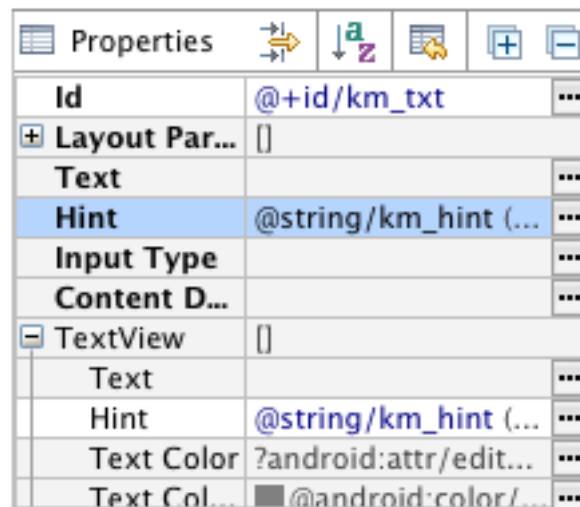
- New → Android Application Project
- Project Name: Conversor KM
- `res/layout/activity_main.xml`
- Graphical Layout
- Text Fields → Plain Text

# Exemplo Conversor



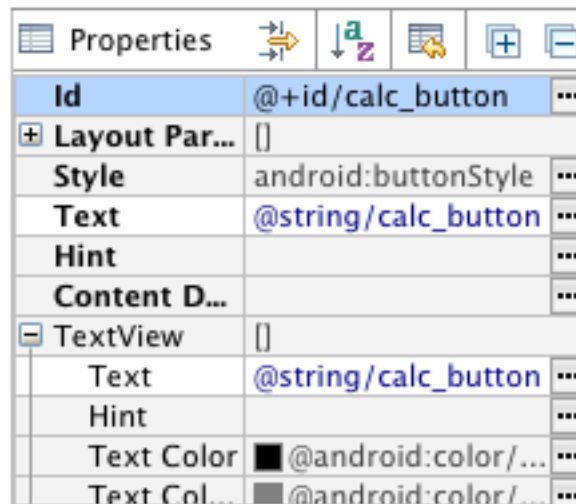
# Exemplo Conversor

- Altere o id para km\_txt
- Adicione um hint como “Quilômetros” (crie uma string nos resources para tal)



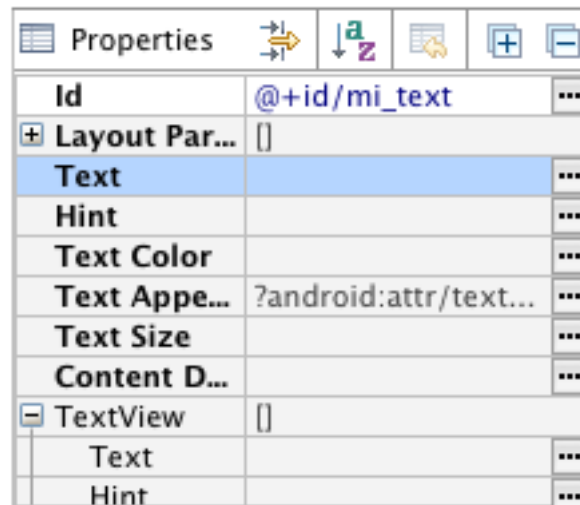
# Exemplo Conversor

- Form Widgets → Button
- Altere o ID e o Text (Crie uma String para o button também)



# Exemplo Conversor

- Form Widgets → TextView
- Altere o ID e remova o Text



# Exemplo Conversor

- O que temos até agora:



# Exemplo Conversor

- Na classe MainActivity, crie uma constante estática do tipo double, com o valor utilizado para criar a conversão:

```
private static double MILES = 0.62137;
```



# Exemplo Conversor

- Desta vez iremos adicionar o evento de clique através da Activity:

```
private OnClickListener calculateClick = new OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // evento...  
    }  
};
```

# Exemplo Conversor

- Dentro do evento onClick, podemos realizar a conversão da seguinte maneira:

```
TextView textView = (TextView) findViewById(R.id.mi_text);
try {
    Double km;
    Double mi;
    EditText editText = (EditText) findViewById(R.id.km_txt);
    km = Double.parseDouble(editText.getText().toString());
    mi = km * MILES;
    textView.setText("Resultado: " + mi.toString() + " milhas");
} catch (Exception e) {
    textView.setText("Valor informado inválido!");
}
```

# Exemplo Conversor

- Defina o evento de clique para o elemento Button que criamos, dentro do método onCreate

```
calcular = (Button) findViewById(R.id.calc_button);  
calcular.setOnClickListener(calculateClick);
```

# Exemplo Conversor

- Exemplo disponível para download em:

<http://vitorfs.com/android/downloads/ConversorKM.zip>

# Criando Menus

- Crie um novo projeto
- Altere a classe MainActivity para herdar de ListActivity

```
public class MainActivity extends ListActivity
```

# Criando Menus

- Defina dois arrays de string para comportar o nome do Menu e a Activity de referência:

```
private String menu[] = {  
    "Inicial",  
    "Sobre",  
    "Sair"};
```

```
private String classes[] = {  
    "InicialActivity",  
    "SobreActivity",  
    "SairActivity" };
```

# Criando Menus

- Dentro do método onCreate(), remova a chamada do método setContentView e substitua pelo seguinte trecho de código:

```
setListAdapter(new ArrayAdapter<String> (  
    MainActivity.this,  
    android.R.layout.simple_list_item_1,  
    menu)  
);
```

# Criando Menus

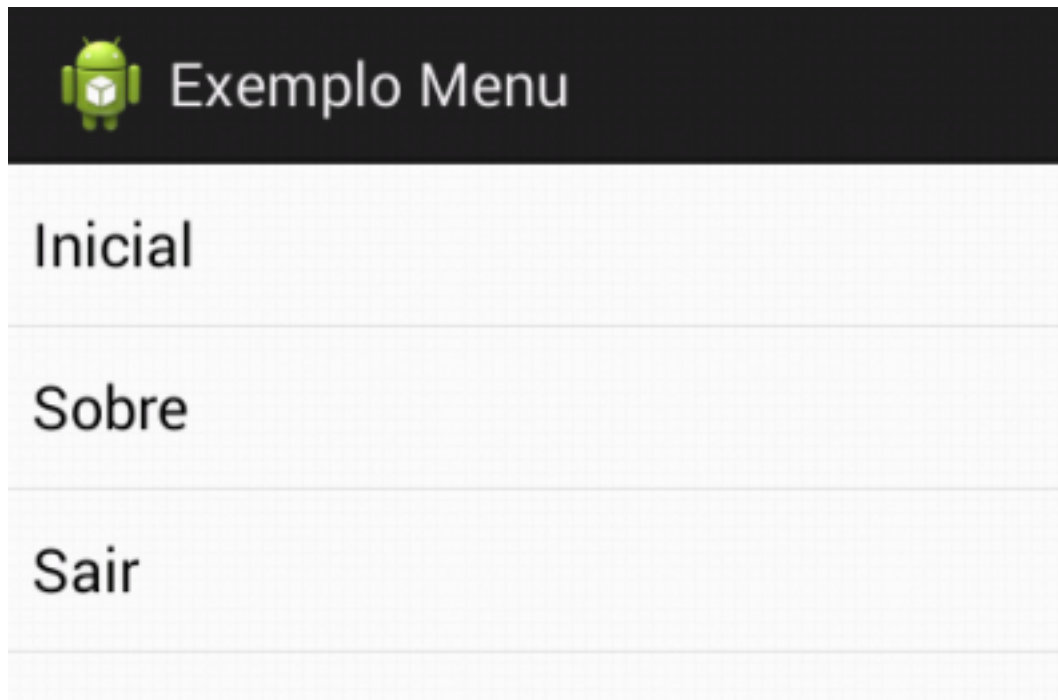
- Agora é necessário criar um listener para o evento de clique:

```
@Override
protected void onListItemClick(ListView l, View v, int
position, long id) {
    super.onListItemClick(l, v, position, id);
    try {
        Class c = Class.forName("com.vitorfs.exemplomenu." +
classes[position]);
        Intent i = new Intent(MainActivity.this, c);
        startActivity(i);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```



# Criando Menus

- Crie as respectivas Activities para testar o menu

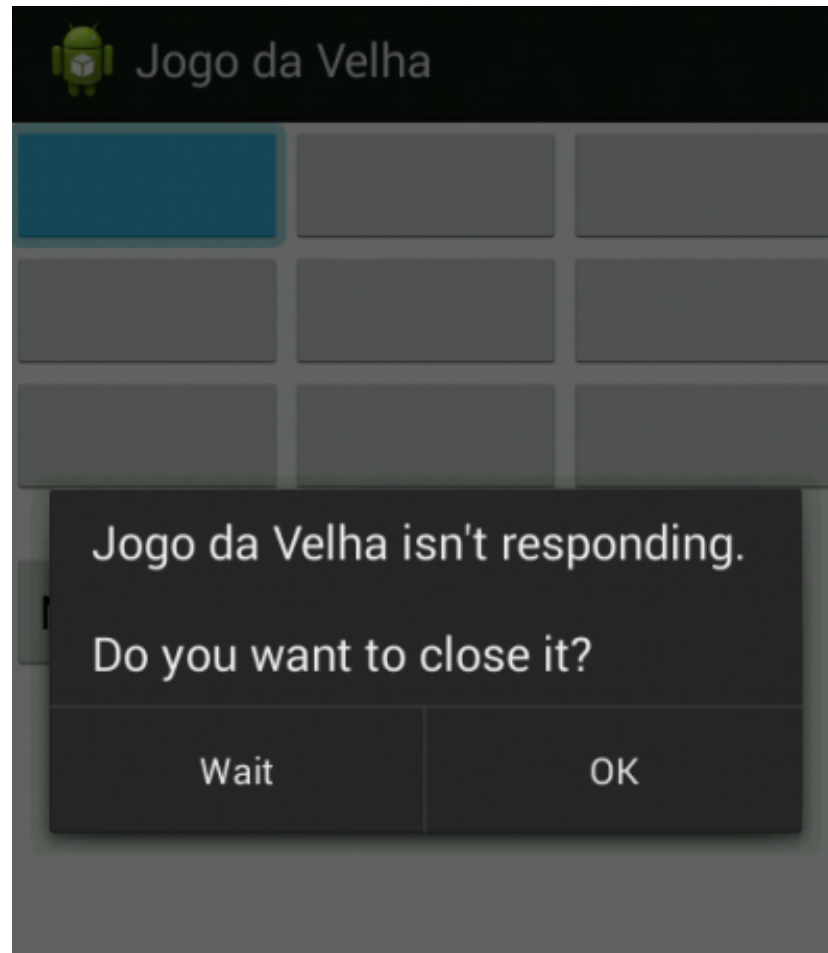


# Criando Menus

- Exemplo disponível para download em:

<http://vitorfs.com/android/downloads/ExemploMenu.zip>

# AsyncTask



# AsyncTask

- ANR
  - UI Thread não responde a um evento de input em até 5 segundos
  - BroadcastReceiver não responde em até 10 segundos

# AsyncTask

- Problemas de Lentidão
  - Operações matemáticas pesadas
  - Grande utilização de I/O
  - Leitura em SD Card
  - Utilização de Rede

# AsyncTask

- doInBackground
- onPostExecute

# AsyncTask

- Normalmente implementada como uma inner class

```
class AsyncTaskExample extends AsyncTask<String, Void, String> {  
  
    @Override  
    protected String doInBackground(String... params) {  
        return null;  
    }  
  
    @Override  
    protected void onPreExecute() {  
  
    }  
  
    @Override  
    protected void onPostExecute(String result) {  
  
    }  
}
```

# AsyncTask

- `doInBackground`: onde será inserido a execução pesada de código
- `onPreExecute`: método executado antes de executar o `doInBackground`
- `onPostExecute`: método executado após a execução do método `doInBackground`. Recebe como parâmetro o retorno do processamento



# AsyncTask

- Para executar o processamento, é necessário criar uma instância da implementação da `AsyncTask`, e chamar o método `execute`, com os parâmetros necessários

```
AsyncTaskExample task = new AsyncTaskExample();  
task.execute("");
```

# AsyncTask

- Dialog de processamento
- Objeto do tipo ProgressDialog
- Necessário informar o Context no constructor

```
ProgressDialog dialog;  
dialog = new ProgressDialog(MainActivity.this);
```

# AsyncTask

- A ideia é iniciar o dialog no método onPreExecute e removê-lo no método onPostExecute

```
@Override
protected void onPreExecute() {
    this.dialog.setMessage("Executando...");
    this.dialog.show();
}
```

```
@Override
protected void onPostExecute(String result) {
    if (this.dialog.isShowing())
        this.dialog.dismiss();
}
```

# Gerando uma APK

- Clique com botão direito no projeto, selecione a opção **Export**
- Na pasta **Android**, selecione a opção **Export Android Application** e clique em **Next**
- Confirme o nome do projeto e clique em **Next**
- Crie uma chave nova ou utilize uma existente, e clique em **Next**
- Escolha o diretório de destino e clique em **Finish**

Para rodar app em um dispositivo Android, basta abrir o arquivo no aparelho