PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

Depurando aplicações

Professor: Danilo Giacobo

OBJETIVOS DA AULA

- Aprender dicas e técnicas para encontrar erros em aplicações Android.
- Conhecer os recursos que auxiliam nos testes dos aplicativos Android.
- Desenvolver o aplicativo do cálculo do IMC.
- Executar um aplicativo Android em um dispositivo real.

INTRODUÇÃO

- Programar dispositivos móveis é algo relativamente complexo.
- Montagem do ambiente de desenvolvimento Android.
- Software Livre, Android SDK, IDE Eclipse, Plug-in ADT.
- Escolher a plataforma/versão do Android na qual o aplicativo será desenvolvido.
- Criação de configuração de uma Android Virtual Machine (AVD).
- Codificação da interface gráfica (XML) e mapeamento com código Java (Activity).
- Erros de programação, de sintaxe, de lógica, de atenção, de planejamento...

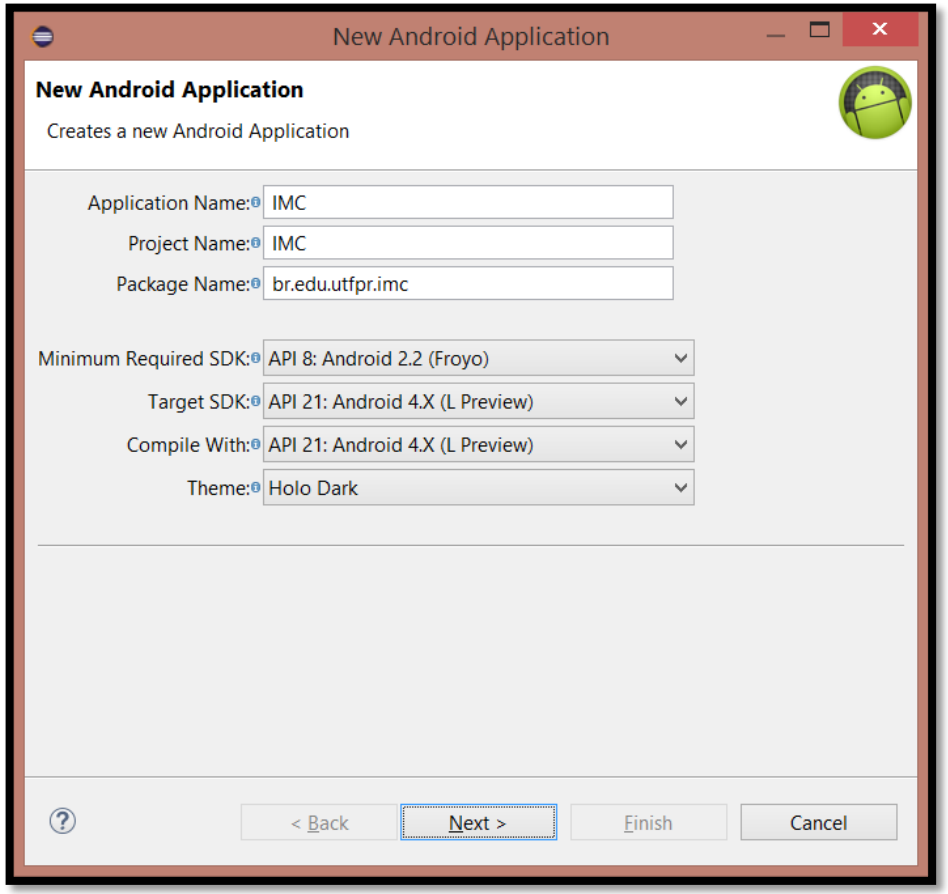
INTRODUÇÃO

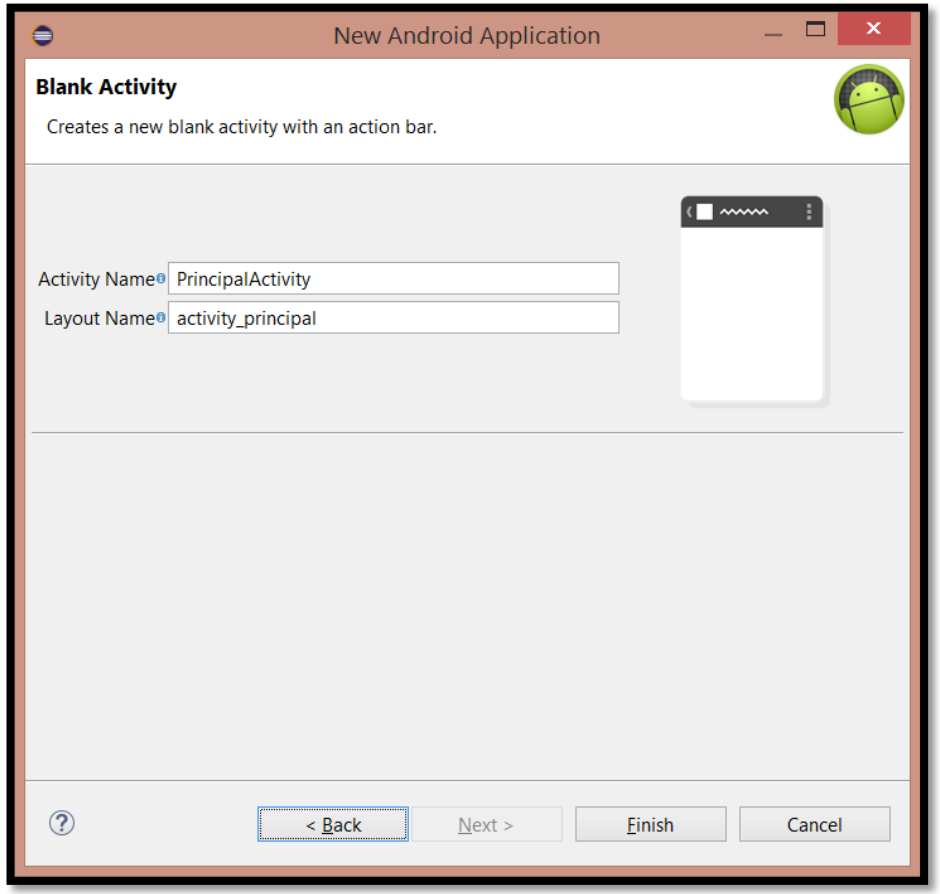
- O termo debugar é derivado da palavra debug, que, em inglês, significa retirar erros de programação (lembrando que os programadores costumam chamar de bugs os erros); esse termo foi aportuguesado para depurar.
- As ferramentas de desenvolvimento de software costumam trazer ferramentas que permitem a depuração dos programas, tais como, recursos como Breakpoint e Watchers (visualização do conteúdo de variáveis).
- Nesta aula serão apresentadas maneiras de minimizar os erros de programação para a plataforma Android, bem como ferramentas para encontrar esses erros, caso eles ocorram e, com certeza ocorrerão.
- Para exemplificar iremos construir o nosso projeto de IMC a partir do "zero" mas agora com o processo de depuração funcionando.

ESTUDO DE CASO - CALCULAR O IMC

- Crie um projeto chamado IMC.
- Ele deve possuir um arquivo de layout (activity_principal.xml).
- Uma *Activity* principal também deverá estar presente (PrincipalActivity.java).
- Use o pacote br.edu.utfpr.imc para este projeto.
- Você poderá escolher a versão do Android que lhe convier pois o recurso de depuração não interfere no processo.

ESTUDO DE CASO - CALCULAR O IMC





ESTUDO DE CASO - CALCULAR O IMC

- O arquivo activity_principal.xml possui o layout apresentado abaixo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    tools:context="br.edu.utfpr.imc.PrincipalActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Peso: " />

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Altura: " />

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="IMC: " />

    <TextView
        android:id="@+id/tvResult"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0,0" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btCalcular"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Calcular" />

        <Button
            android:id="@+id/btLimpar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Limpar" />

    </LinearLayout>

</LinearLayout>
```

ESTUDO DE CASO - CALCULAR O IMC

- A *activity*, a classe Java responsável pelo funcionamento do aplicativo, é mostrada abaixo:

```java
package br.edu.utfpr.imc;

import java.text.DecimalFormat;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class PrincipalActivity extends ActionBarActivity {

    private EditText etPeso;
    private EditText etAltura;
    private TextView tvResult;
    private Button btCalcular;
    private Button btLimpar;

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

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

        etPeso = (EditText) findViewById(R.id.etPeso);
        etAltura = (EditText) findViewById(R.id.etAltura);
        tvResult = (TextView) findViewById(R.id.tvResult);
        btCalcular = (Button) findViewById(R.id.btCalcular);
        btLimpar = (Button) findViewById(R.id.btLimpar);

        btLimpar.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                btLimparOnClick();
            }
        });

        btCalcular.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                btCalcularOnClick();
            }
        });
    }
```


ESTUDO DE CASO - CALCULAR O IMC

- A *activity*, a classe Java responsável pelo funcionamento do aplicativo, é mostrada abaixo:

```
    private void btLimparOnClick() {
        etPeso.setText("");
        etAltura.setText("");
        tvResult.setText("0,0");
        etPeso.requestFocus();
    }

    private void btCalcularOnClick() {
        if(etPeso.getText().toString().equals("")) {
            Toast.makeText(this, "Campo Peso deve ser preenchido", Toast.LENGTH_LONG).show();
            etPeso.requestFocus();
            return;
        }

        if(etAltura.getText().toString().equals("")) {
            Toast.makeText(this, "Campo Altura deve ser preenchido", Toast.LENGTH_LONG).show();
            etAltura.requestFocus();
            return;
        }

        double peso = Double.parseDouble(etPeso.getText().toString());
        double altura = Double.parseDouble(etAltura.getText().toString());
        double imc = peso / Math.pow(altura, 2);
        tvResult.setText(new DecimalFormat("0.00").format(imc));
    }
}
```

ERROS DE DIGITAÇÃO NO XML E JAVA

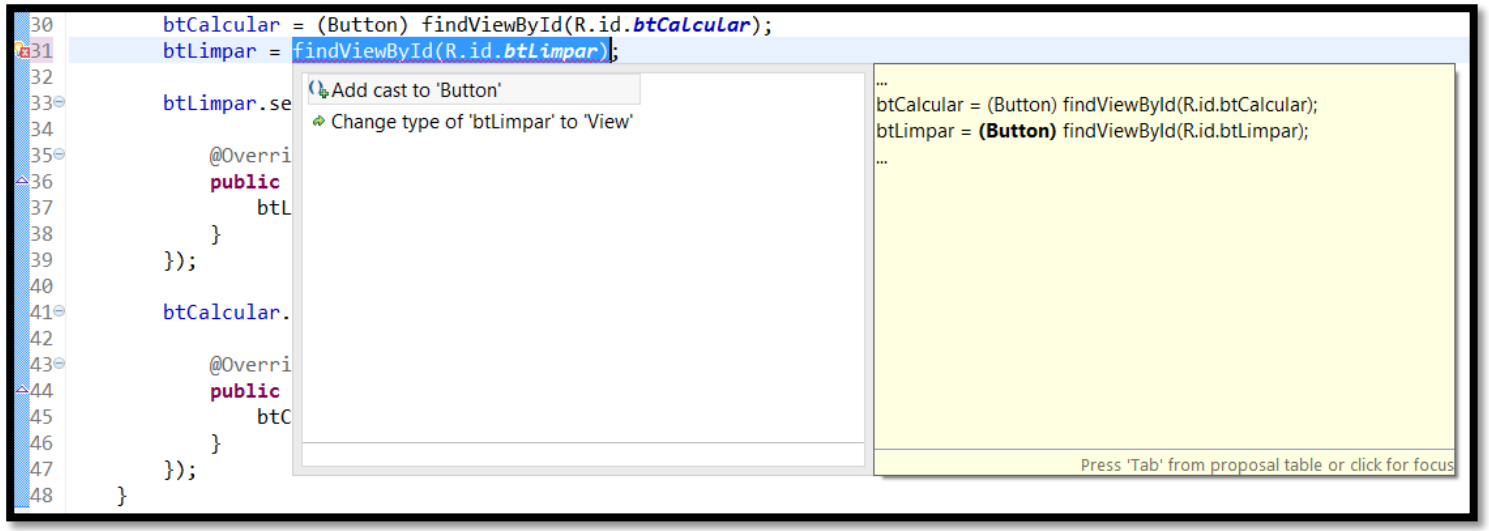
- Basicamente, os erros de programação se dividem em duas categorias: erros de digitação e erros de lógica.
- Os erros de digitação são muitas vezes fáceis de identificar, sendo que o próprio ambiente de desenvolvimento apresenta-os.

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

    etPeso = (EditText) findViewById(R.id.etPeso);
    etAltura = (EditText) findViewById(R.id.etAltura);
    tvResult = (TextView) findViewById(R.id.tvResult);
    btCalcular = (Button) findViewById(R.id.btCalcular);
    btLimpar = findViewById(R.id.btLimpar);
```

Exemplo de erro de código

ERROS DE DIGITAÇÃO NO XML E JAVA

- Os erros de código Java são facilmente identificados, uma vez que a linha onde o erro se encontra fica sublinhada de vermelho. Colocando o mouse sobre a linha com erro, uma mensagem informativa é apresentada.
- Em algumas situações, o próprio IDE de desenvolvimento sugere a correção do erro.



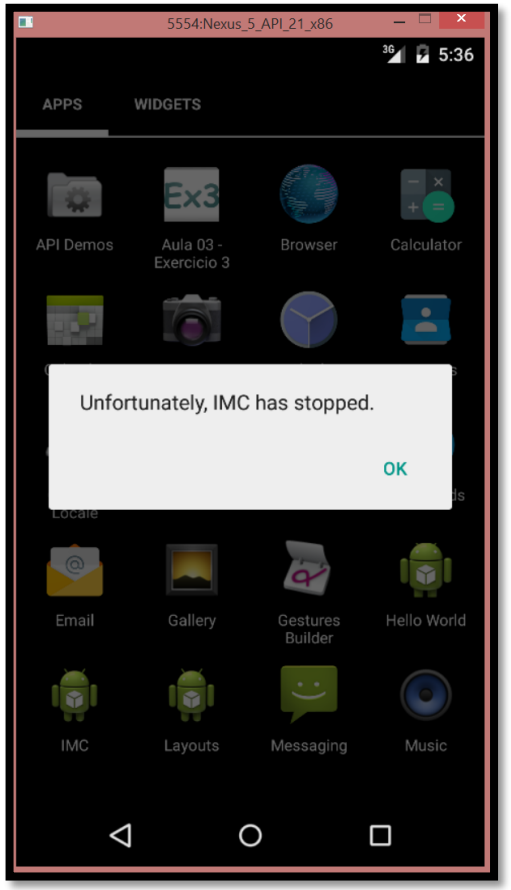
ERROS DE DIGITAÇÃO NO XML E JAVA

- Nem sempre são apresentadas sugestões para a correção dos erros, outras vezes, são apresentadas sugestões que não condizem com a solução do problema., assim, esse recurso deve ser utilizado com cuidado.
- Os erros de codificação no arquivo XML são mais difíceis de ser encontrados.

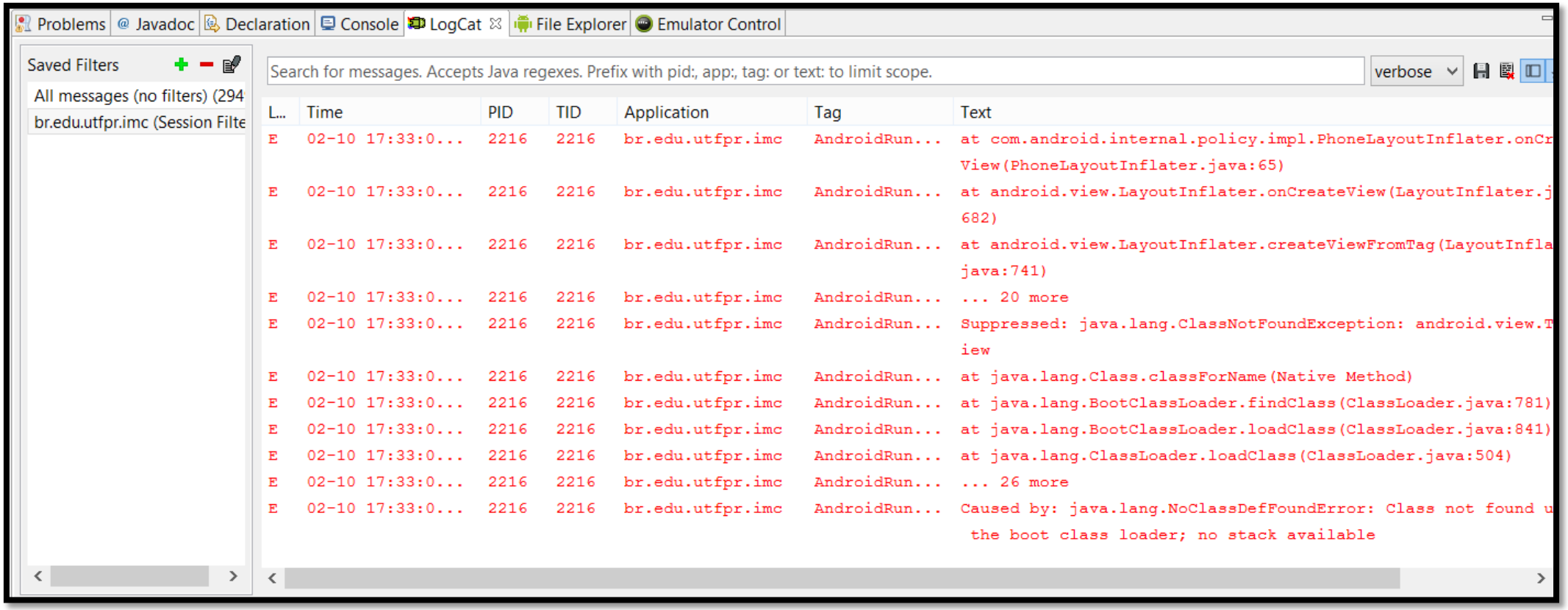
```
<Textview
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="IMC: " />

<TextView
    android:id="@+id/tvResult"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0,0" />

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
```

ERROS DE DIGITAÇÃO NO XML E JAVA

- Como observado no código XML anterior, foi digitado Textview com apenas a primeira letra maiúscula, sendo que o correto é TextView. Como a plataforma Android é *case-sensitive*, esse erro comprometerá a execução do programa (veja imagem ao lado).
- Esse erro pode não ser resolvido facilmente, já que no editor do XML, o código não é apresentando com um sublinhado vermelho.
- A descrição do erro XML é apresentada apenas na janela de LogCat, sendo que está pode ser exibida acessando o menu **Window > Show View > Other** e na janela apresentada, escolhe-se a categoria **Android > LogCat**.



ERROS DE DIGITAÇÃO NO XML E JAVA



Mensagem de erro apresentada via LogCat

ERROS DE DIGITAÇÃO NO XML E JAVA

- O LogCat costuma apresentar em vermelho os erros de execução, detalhando estes e, em alguns casos, apresentando até o arquivo e a linha onde o erro se encontra.
- Os erros do tipo InflateException costumam corresponder aos erros de sintaxe no arquivo XML.
- Se analisarmos a mensagem, a solução do erro se tornará mais simples. Veja a mensagem:

```
FATAL EXCEPTION: main
Process: br.edu.utfpr.imc, PID: 2216
java.lang.RuntimeException: Unable to start activity ComponentInf
o{br.edu.utfpr.imc/br.edu.utfpr.imc.PrincipalActivity}: android.v
iew.InflateException: Binary XML file line #32 Error inflating c
lass Textview
```

ERROS DE DIGITAÇÃO NO XML E JAVA

- Outros erros, entretanto, são apresentados com um círculo vermelho à esquerda da declaração do componente, a declaração com propriedades erradas. Ex.: android:test no lugar de android:text.
- Embora seja apresentado o componente com erro a partir de um círculo vermelho à esquerda, não é possível identificar a linha do erro automaticamente.
- Erros como a digitação de widht e heigth também são comuns.
- Nesses casos, o emulador não permite a execução do aplicativo até que o erro seja consertado.

```
32      <TextView
33          android:layout_width="wrap_content"
34          android:layout_height="wrap_content"
35          android:test="IMC: " />
36
```

Erro na declaração de propriedades

Dica: Correção de erros por meio da omissão do código.

ERROS DE CODIFICAÇÃO

- Os erros de codificação, em geral, são mais difíceis de ser corrigidos, uma vez que podem envolver declarações erradas, erros de lógica, erros de fórmulas ou erros alheios, como, por exemplo, um *firewall* no computador que não deixa uma requisição HTTP sair do computador.
- Muitas vezes, é muito difícil (ou impossível) encontrar erros usando apenas o emulador, sendo necessária a utilização de um aparelho real, como nas situações em que se trabalha com Bluetooth ou sensores, recursos estes que dificilmente podem ser simulados em um emulador.
- De qualquer maneira, quando nos deparamos com um erro de codificação, nada melhor do que executar o programa passo a passo para identificar em que momento ocorreu o erro no programa e para isso, é utilizado um recurso chamado Breakpoint.

ERROS DE CODIFICAÇÃO

- Para testarmos os erros de código, uma linha importante do programa foi alterada, simulando um **CTRL+C** e **CTRL+V**, o componente btCalcular é mapeado duas vezes e o componente btLimpar não foi mapeado, conforme apresentado abaixo.

```
private EditText etPeso;
private EditText etAltura;
private TextView tvResult;
private Button btCalcular;
private Button btLimpar;

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

    etPeso = (EditText) findViewById(R.id.etPeso);
    etAltura = (EditText) findViewById(R.id.etAltura);
    tvResult = (TextView) findViewById(R.id.tvResult);
    btCalcular = (Button) findViewById(R.id.btCalcular);
    btCalcular = (Button) findViewById(R.id.btLimpar);
```

ERROS DE CODIFICAÇÃO

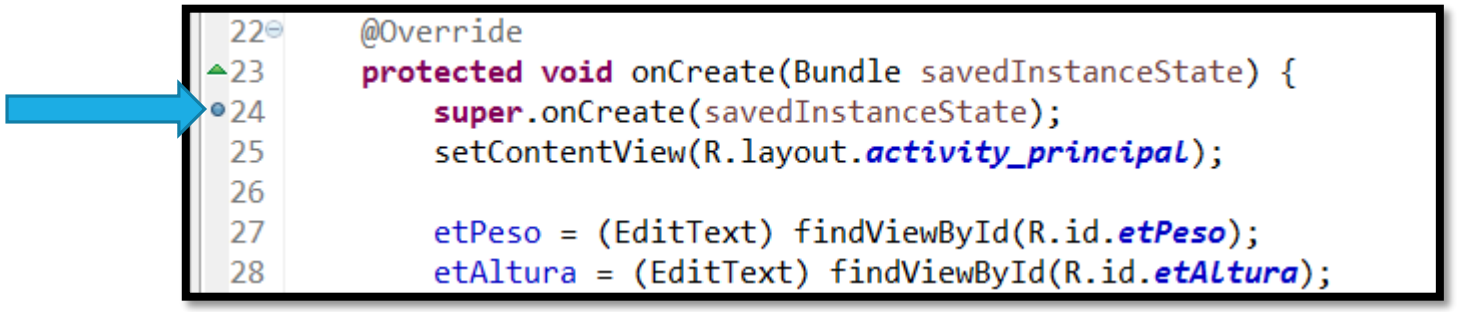
- Ao executar o aplicativo, a tela do emulador apresenta uma imagem idêntica a do slide 13, informando que infelizmente o aplicativo foi encerrado.
- Já a tela de **LogCat** apresenta uma mensagem de erro NullPointerException, muito conhecida dos programadores Java tradicionais, o que significa que um componente foi declarado, porém, não foi criado.

```
FATAL EXCEPTION: main
Process: br.edu.utfpr.imc, PID: 2420
java.lang.RuntimeException: Unable to start activity ComponentInf
o{br.edu.utfpr.imc/br.edu.utfpr.imc.PrincipalActivity}: java.lang
.NullPointerException: Attempt to invoke virtual method 'void and
roid.widget.Button.setOnClickListener(android.view.View$OnClickLi
stener)' on a null object reference
```

- Os erros NullPointerException são difíceis, principalmente para os usuários que estão iniciando com a tecnologia, de ser identificados, então, uma das alternativas é executar o programa passo a passo, verificando em que linha o erro ocorreu.

UTILIZANDO UM BREAKPOINT

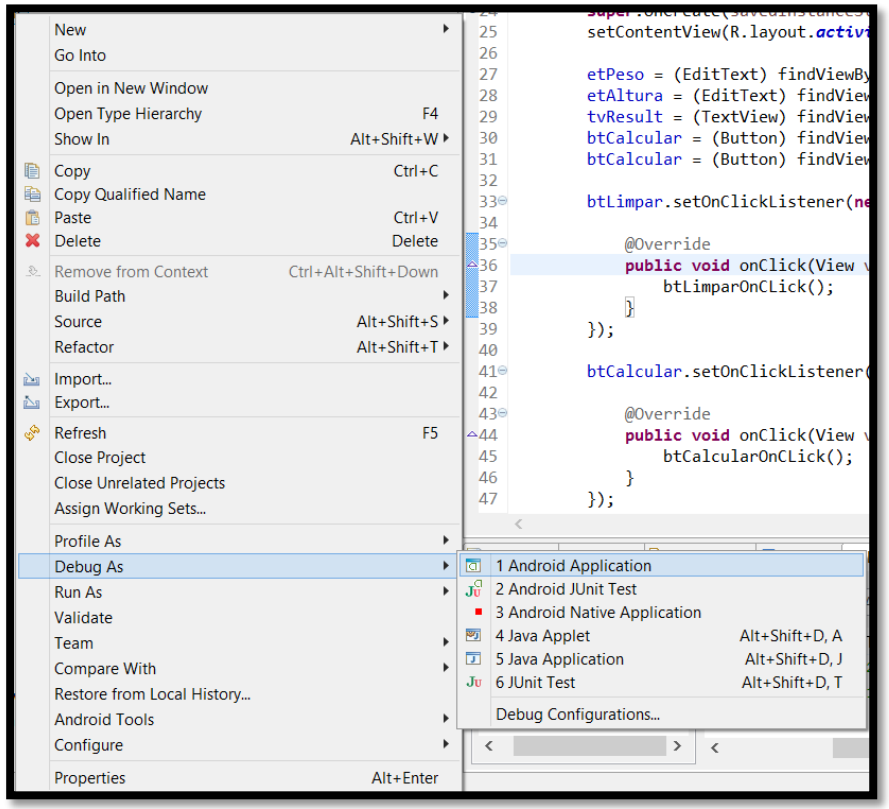
- Todo programa Android é iniciado a partir do método onCreate(), ou seja, se o erro aconteceu antes da apresentação da primeira tela do aplicativo no emulador, será muito provável que o erro esteja nesse método.
- Desta forma, visando encontrar o erro, iniciaremos colocando um **Breakpoint** na primeira linha deste método.
- Se ocorreu tudo certo, o **Breakpoint** será representado por um círculo azul.



Dica: Para adicionar um breakpoint, basta dar dois cliques na margem esquerda do editor.

UTILIZANDO UM BREAKPOINT

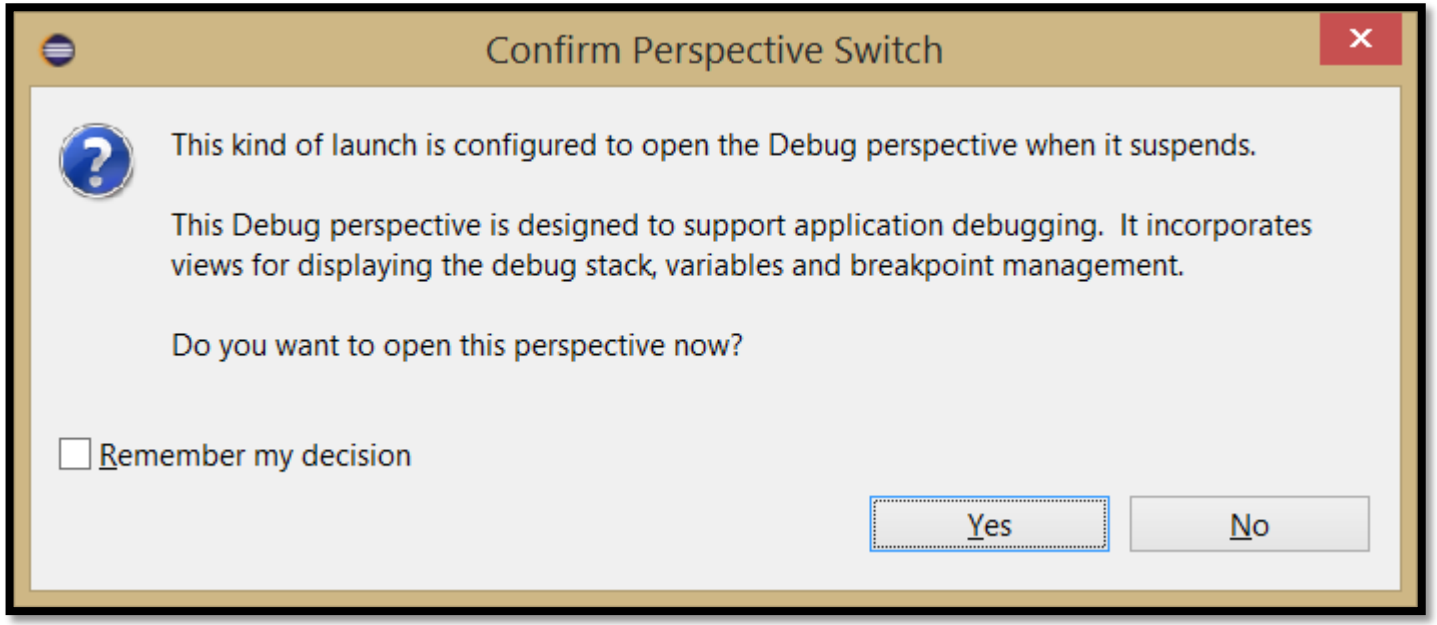
- Após o breakpoint, devemos executar o projeto no modo Debug, clicando com o botão direito no projeto e escolhendo a opção **Debug As > Android Application**.



Dica: Após a primeira execução passo a passo do projeto, é possível executar no modo Debug também clicando no notão Debug e escolhendo o nome do projeto na lista apresentada.

UTILIZANDO UM BREAKPOINT

- No IDE, uma mensagem é apresentada. Nela, é perguntado se o programador deseja executar o programa abrindo a perspectiva Debug, com novas telas e ferramentas que auxiliam na correção do erro. É aconselhável a execução nessa nova tela, escolhendo a opção Yes.



UTILIZANDO UM BREAKPOINT

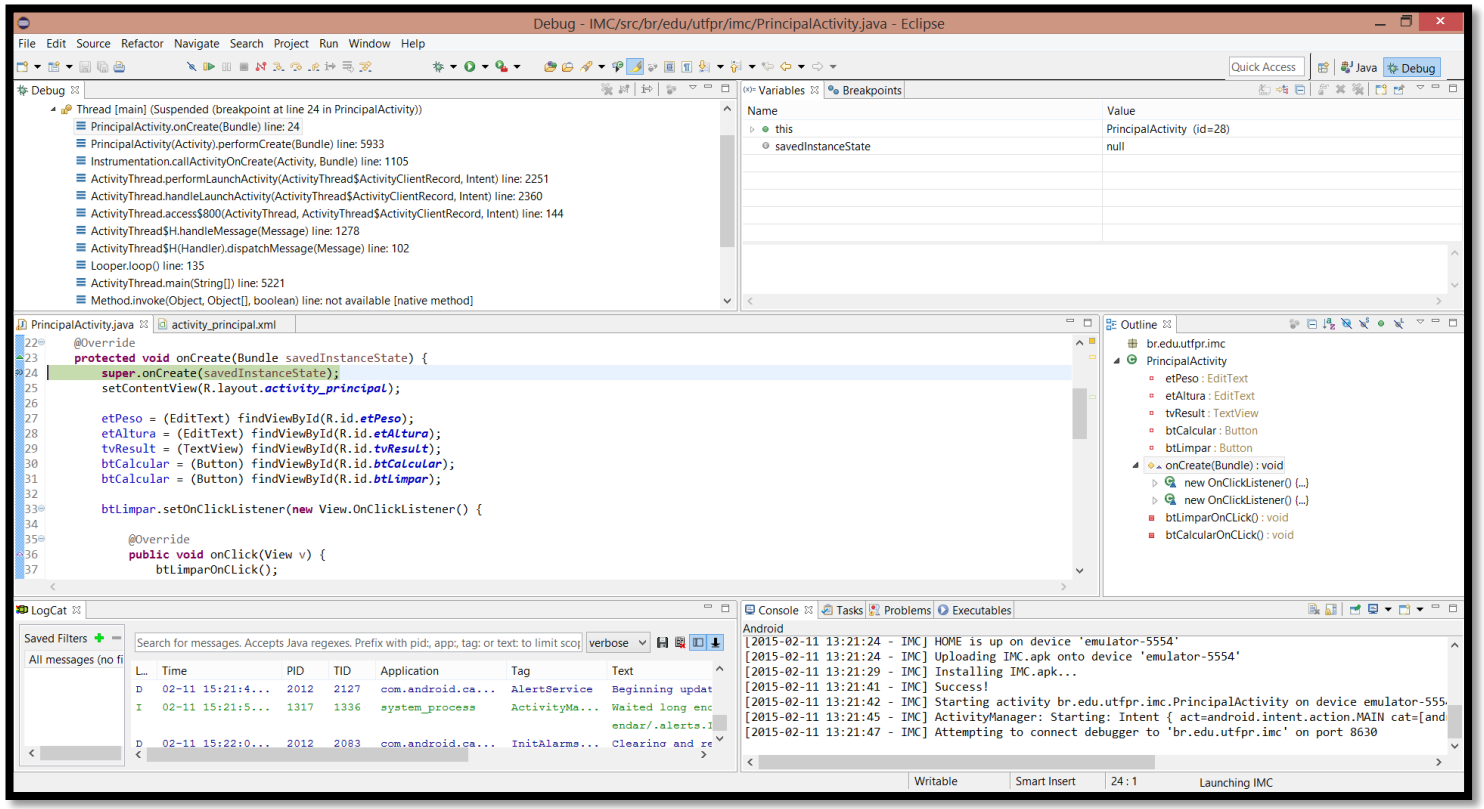
- Na imagem abaixo é possível observar uma linha verde na linha onde foi adicionado um Breakpoint, bem como uma seta à esquerda. Ela indica a linha que será executada pelo emulador. Passando pela linha, significa que ela foi executada com sucesso e a próxima linha será executada, e assim por diante.

PrincipalActivity.java | activity_principal.xml

```
22      @Override
23      protected void onCreate(Bundle savedInstanceState) {
24          super.onCreate(savedInstanceState);
25          setContentView(R.layout.activity_principal);
26
27          etPeso = (EditText) findViewById(R.id.etPeso);
28          etAltura = (EditText) findViewById(R.id.etAltura);
29          tvResult = (TextView) findViewById(R.id.tvResult);
30          btCalcular = (Button) findViewById(R.id.btCalcular);
31          btCalcular = (Button) findViewById(R.id.btLimpar);
32
33          btLimpar.setOnClickListener(new View.OnClickListener() {
34
35              @Override
36              public void onClick(View v) {
37                  btLimparOnClick();
```

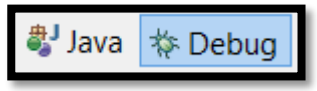
UTILIZANDO UM BREAKPOINT

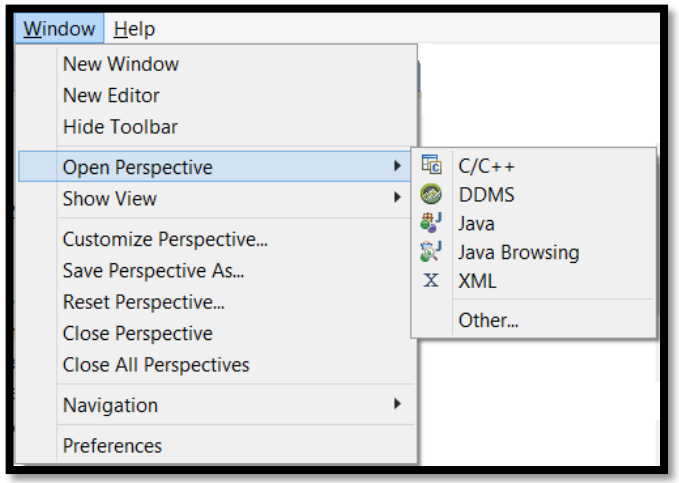
- A figura abaixo apresenta a nova perspectiva de execução, chamada de perspectiva Debug.



DICA: ALTERNANDO ENTRE DIFERENTES PERSPECTIVAS

- O uso de perspectivas permite alternar rapidamente entre os modos de visualização do projeto, por exemplo, entre o modo Java (utilizado normalmente para a codificação) e o modo Debug (utilizado para encontrar erros de códigos na execução passo a passo).
- Para alternar entre as perspectivas, permitindo voltar à perspectiva utilizada para o desenvolvimento (Java), pode-se escolher o menu **Window > Open Perspective > Java** ou, ainda, é possível alternar clicando no botão Java, apresentado no canto superior esquerdo do IDE.





PERSPECTIVA DEBUG (01/02)

- Das ferramentas apresentadas na perspectiva **Debug**, uma das principais é a que permite a execução passo a passo, apresentada na imagem abaixo:

Skip all Breakpoints: esta opção "desativa" todos os Breakpoints do programa, desta forma, a execução acontece como se não houve Breakpoints.

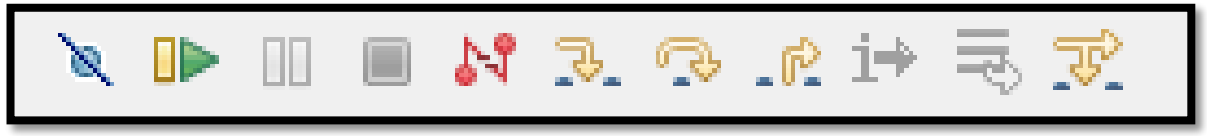
Resume: sai do passo a passo, executando o programa até encontrar outro Breakpoint [F8].

Suspend: Pausa uma execução passo a passo.

Terminate: Finaliza uma execução passo a passo, encerrando o programa [CTRL + F12].

Disconnect: Desconecta o Eclipse da ferramenta que executa o aplicativo passo a passo.

PERSPECTIVA DEBUG (02/02)



Step Into: esta opção permite entrar em um método, verificando passo a passo seu conteúdo [F5].

Step Over: executa o método sem entrar em seu código. Opção mais utilizada no passo a passo [F6].

Step Return: permite voltar ao "nível anterior", ou seja, se um método chamador chamou outro, retornará o código do método chamador [F7].

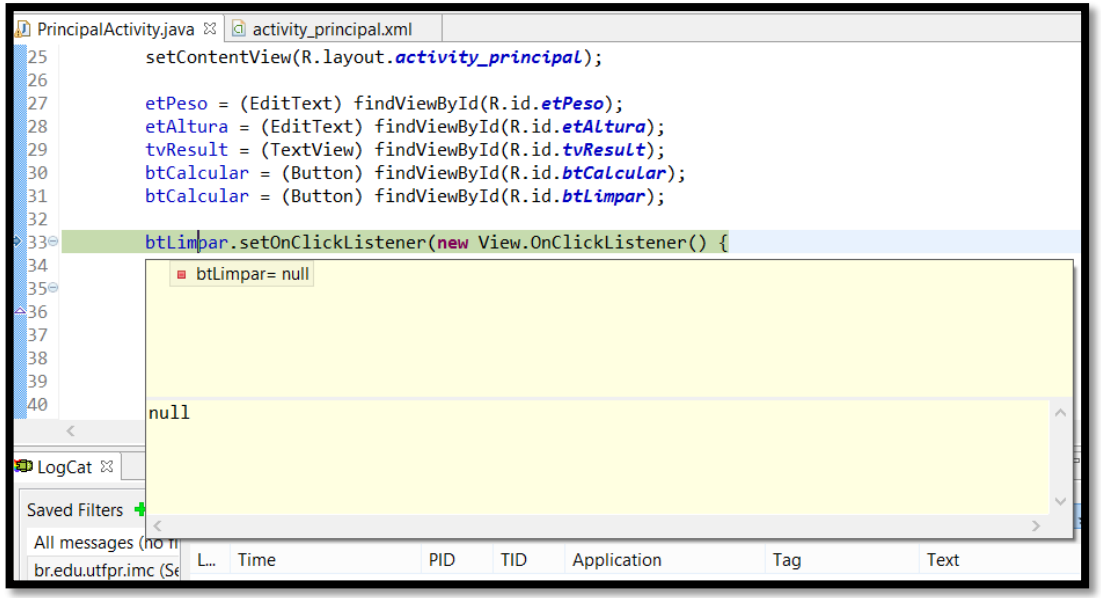
Instruction Stepping Mode: habilita a visualização do "Disassembly" do seu código.

Drop to Frame: após modificar o conteúdo de algumas variáveis no debug, esta opção permite voltar ao estado original.

Use Step Filters: esta opção permite omitir as classes durante a execução do passo a passo [Shift + F5].

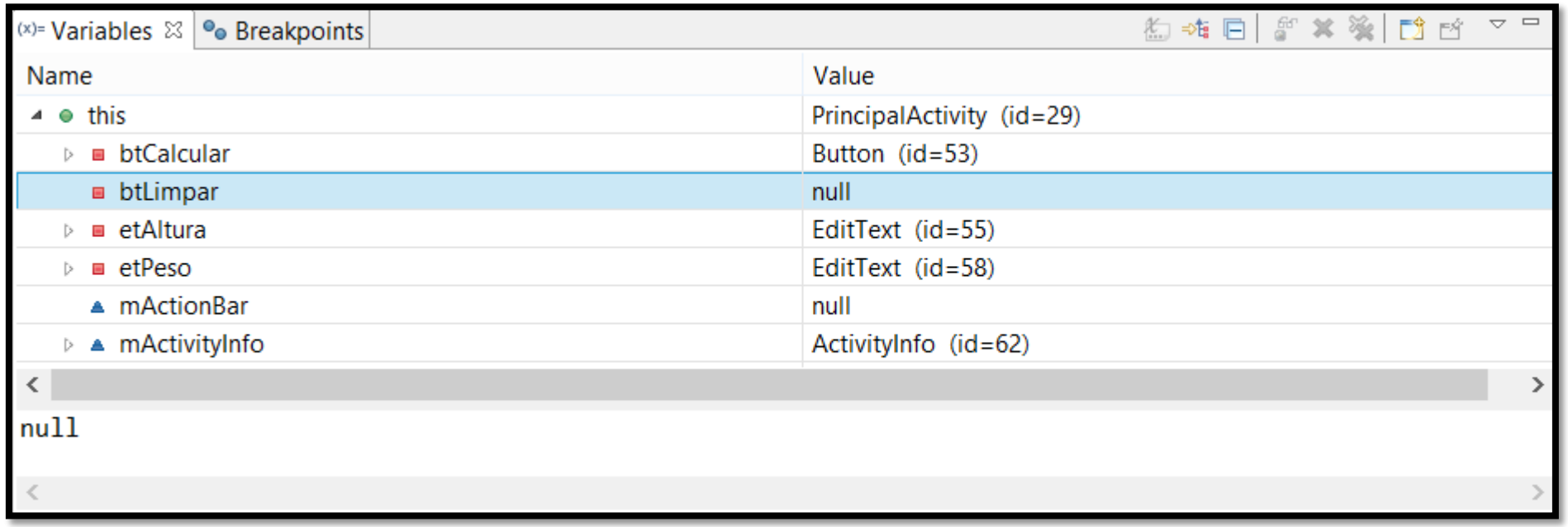
JANELA DE VISUALIZAÇÃO DE VARIÁVEIS

- Durante a execução passo a passo de um programa, é possível, a qualquer momento, verificar o conteúdo das variáveis e objetos, bastando passar o mouse sobre o objeto desejado.
- No nosso exemplo anterior, colocando o mouse sobre o objeto btLimpar, é possível identificar que este não foi criado, possuindo um valor igual a null. No Java, sempre que tentamos utilizar um objeto nulo, é retornado um erro do tipo NullPointerException, sendo esta a causa do erro.



JANELA DE VISUALIZAÇÃO DE VARIÁVEIS

- Outra maneira de visualizar o conteúdo das variáveis/objetos utilizados no programa, é abrir a janela Variables, por meio do menu **Window > Show View > Variables.**
- Essa janela apresenta o conteúdo de todas as variáveis/objetos do programam podendo identificar quais são nulos.



EXECUTANDO O APLICATIVO EM UM DEVICE REAL

- A plataforma Android dá suporte a um recurso bem interessante, que é a execução de aplicativos diretamente nos dispositivos Android, podendo ser um *smartphone* ou *tablet*.
- Enquanto em um AVD, dependendo da velocidade do computador, o tempo para carregar e executar uma aplicação pode ser superior a um minuto; em dispositivos reais a execução do aplicativo é praticamente imediata.
- Na execução em dispositivos reais, recursos como, a execução passo a passo e a verificação de variáveis também funciona, além de poder testar de forma real recursos como Bluetooth, GPS, ligações, SMS e muitos outros.
- A ligação entre os computadores de desenvolvimento e o dispositivo Android acontece comumente por meio de uma conexão física, via cabo USB.

EXECUTANDO O APLICATIVO EM UM DEVICE REAL

- O processo para a execução dos aplicativos diretamente nos dispositivos Android é relativamente complexo, sendo este dividido em algumas etapas:

  Passo 1: conectar o dispositivo Android no computador de desenvolvimento, instalando seus drivers.

  Passo 2: configurar o dispositivo Android, habilitando o modo Debug deste.

  Passo 3: instalar o recurso USB Driver no Android SDK.

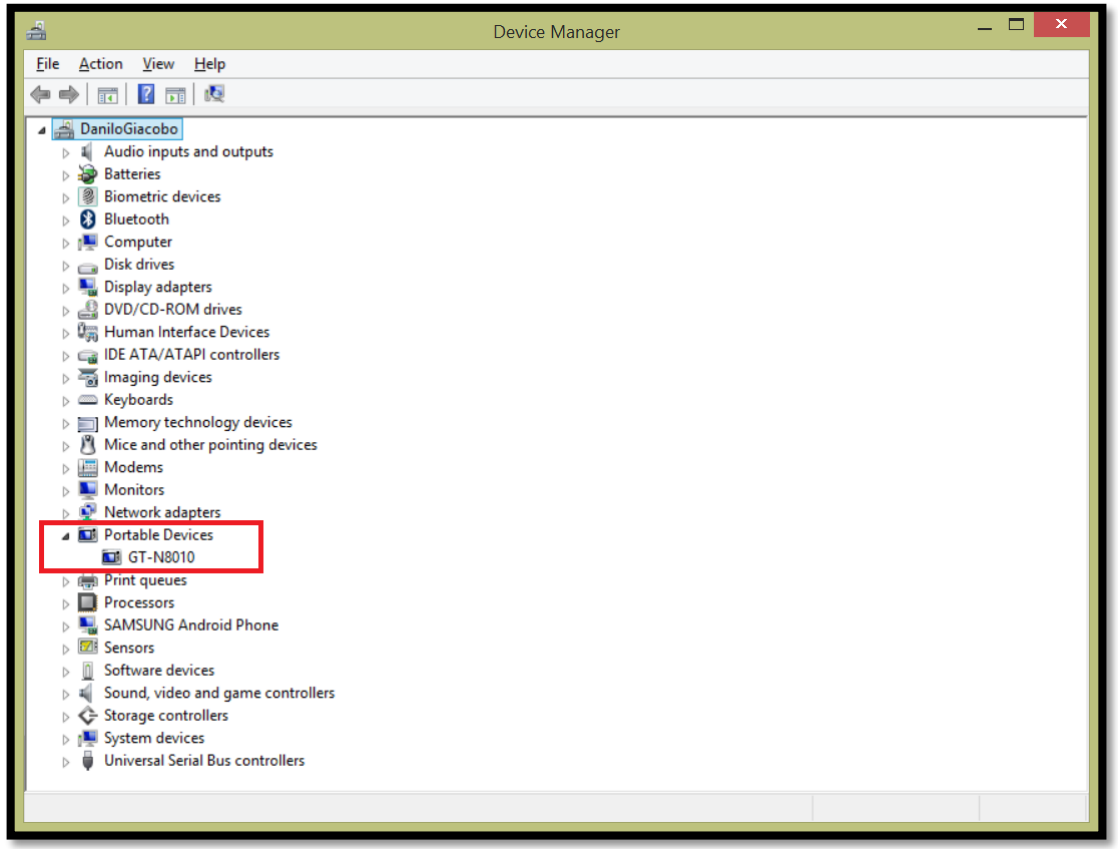
  Passo 4: ao executar o aplicativo, escolher o dispositivo Android.

- Após conectar o dispositivo pela primeira vez no computador, é normal que seja solicitada a instalação dos drivers para o mesmo, em especial no sistema operacional Windows.

- Se tudo ocorreu bem o dispositivo será apresentada no Gerenciador de Dispositivos (Windows). Use a tecla de atalho [Windows + Break] para acessar o mesmo.

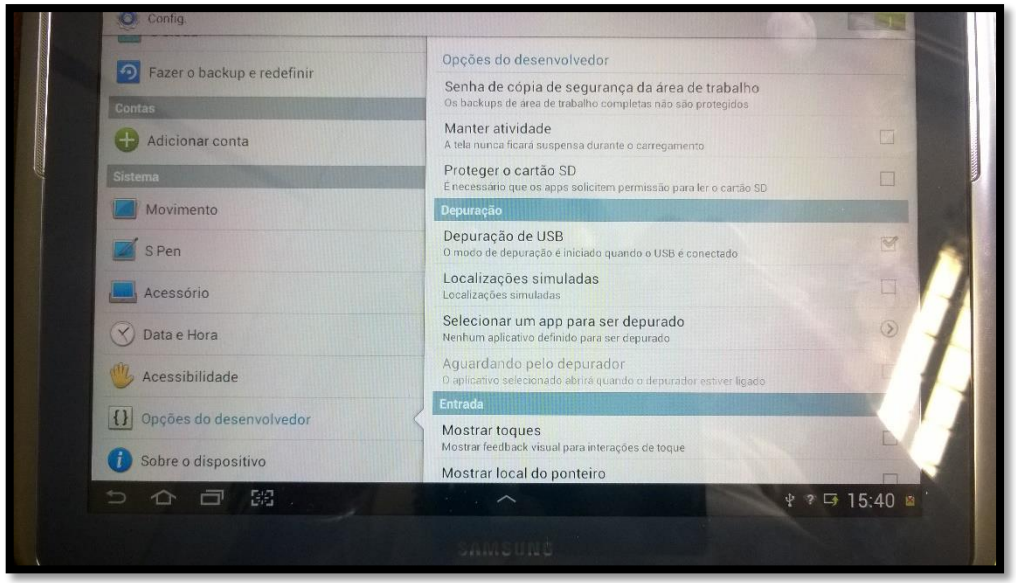
EXECUTANDO O APLICATIVO EM UM DEVICE REAL

- Em outros sistemas operacionais, como o Linux ou MacOS, não é comum a instalação de driver, como acontece no Windows. Na maioria das vezes, conectando o dispositivo Android no Linux ou MacOS, o mesmo será reconhecido automaticamente e já ficará pronto para o uso.



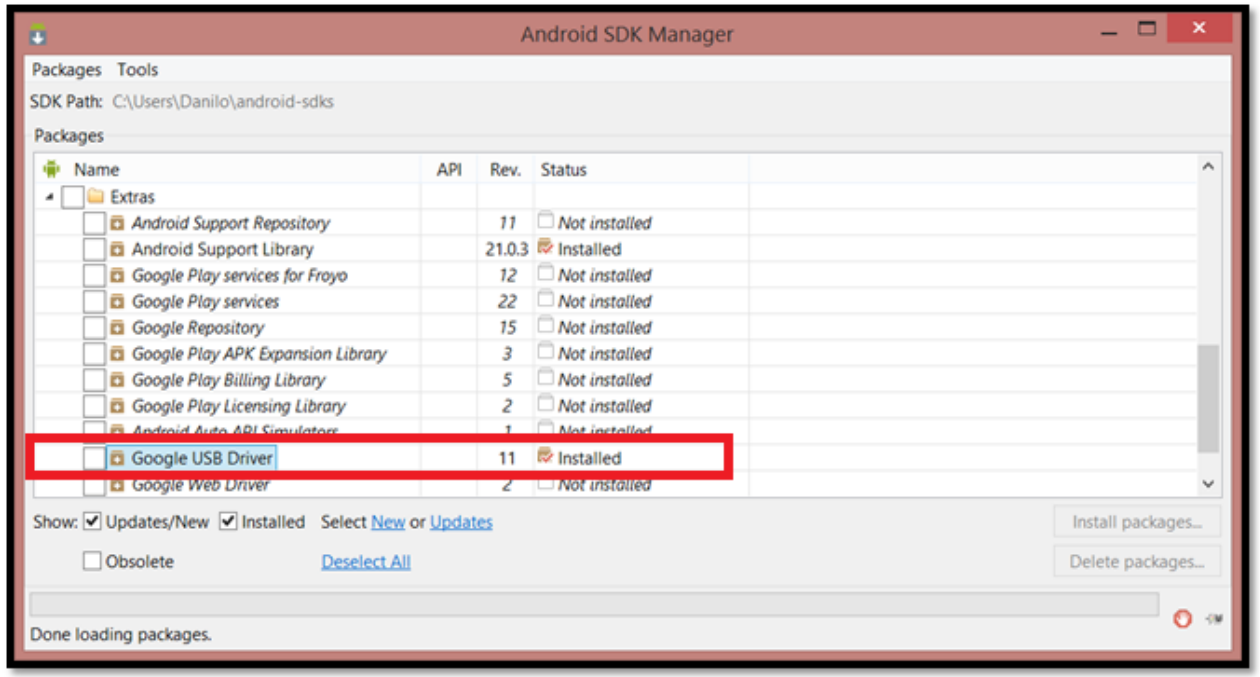
EXECUTANDO O APLICATIVO EM UM DEVICE REAL

- O passo seguinte consiste em habilitar o recurso de Debug no dispositivo Android. Este processo pode diferenciar um pouco, dependendo da versão do Android.
- Na versão 4.1.2 é possível habilitar tal recurso acessando o menu de configurações do dispositivo Android e escolhe a opção “Opções do desenvolvedor”. Depois basta marcar a opção “Depuração de USB”. A depuração USB destina-se apenas para efeito de programação.



EXECUTANDO O APLICATIVO EM UM DEVICE REAL

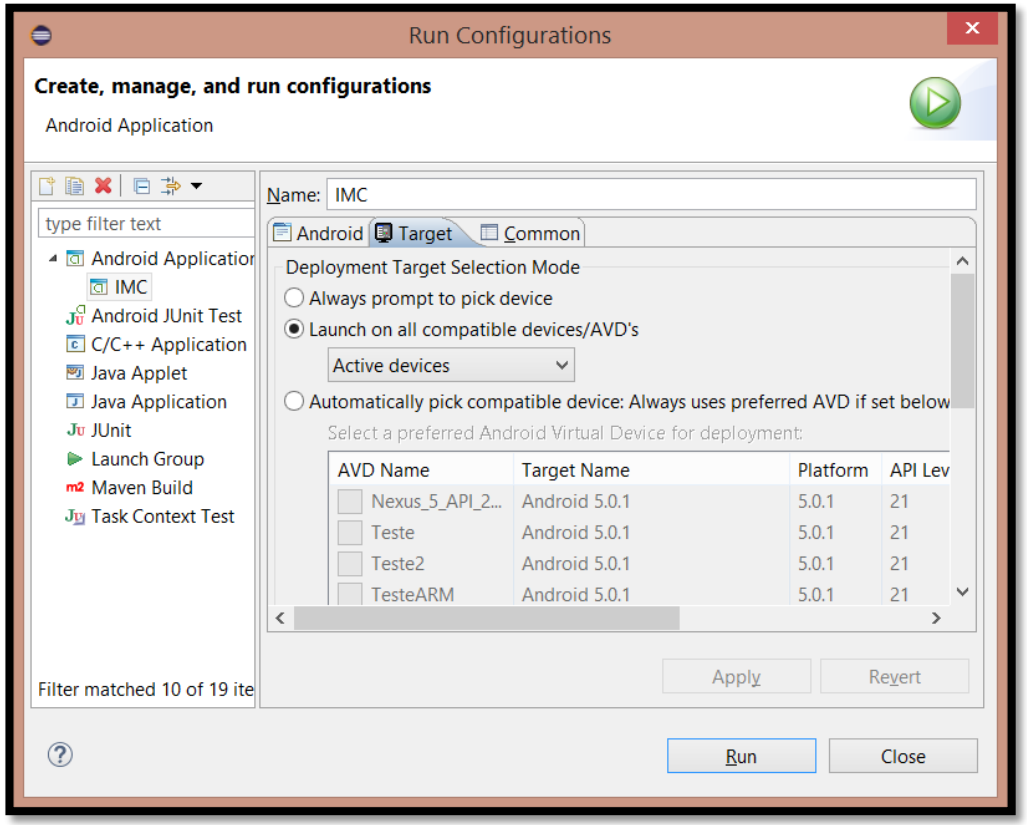
- O passo seguinte é instalar no computador o componente **USB Driver**. Este é necessário para o debug no aparelho. Para instalar, no IDE Eclipse, acesse o menu **Window > Android SDK Manager** e na janela apresentada, selecione a categoria **Extras** - selecionando o **USB Driver**. Após isso, a instalação deve ser realizada.



EXECUTANDO O APLICATIVO EM UM DEVICE REAL

- Com isso, a aplicação deve estar pronta para ser executada no dispositivo Android, desta forma, deve-se conectar o *device* via USB ao computador de desenvolvimento. Após, deve-se clicar com o botão direito no projeto, escolhendo a opção **Debug As > Debug Configurations** (esse é o procedimento para debugar o aplicativo em um *device* real. Para executar o aplicativo sem debug no *device*, deve-se clicar com o botão direito, escolhendo a opção **Run As > Run Configurations**).
- Na tela apresentada, deve-se acessar a categoria **Target** e na opção **Launch on all compatible devices/AVD's**, escolhe-se **Active devices**.
- Clicando em **Run**, será apresentada uma lista de todos os *devices* conectados ao computador, que deve ser escolhido, conforme apresentado abaixo.
- Assim o aplicativo é executado em um *device* Android real. No exemplo utilizado, o aplicativo é relativamente simples, mas é possível usar recursos como Breakpoint, mudar o conteúdo da variável e se houvesse recursos como Bluetooth no aplicativo, o mesmo poderia ser testado de forma real.

EXECUTANDO O APLICATIVO EM UM DEVICE REAL



EXECUTANDO O APLICATIVO EM UM DEVICE REAL