UNIDADE 4 – CONSTRUINDO APLICATIVOS ANDROID

MÓDULO 1 – PRIMEIRO APLICATIVO ANDROID

01

1 - PRIMEIRO EXEMPLO – HELLO WORLD!

Esta unidade irá utilizar-se de uma linguagem menos formal e uma metodologia de escrita mais flexível com o objetivo de facilitar a leitura e o desenvolvimento de um aplicativo Android passo a passo. Portanto, esta unidade será um guia (passo a passo) de como construir um aplicativo Android utilizando as ferramentas de desenvolvimento adequadas.

Um ponto importante a ser observado é que as ferramentas mudam de versão constantemente e, com certeza, irá alterar as interfaces e telas apresentadas ao longo desta unidade.

As telas abaixo foram feitas utilizando-se as versões mais recentes de cada ferramenta neste momento do tempo, a saber:

- JDK em sua versão 1.8_73
- SDK Android em sua versão r24.4.1
- Eclipse em sua versão Mars.1
- o plugin ADT em sua versão 23.0.7.2120684

Portanto fique atento para adaptar, sempre que for necessário, ao novo modus operandi de como fazer as coisas. Não fique ansioso caso seja essa a sua situação. Fique tranquilo porque essas mudanças de versão não costumam realizar grandes alterações nos procedimentos que serão descritos. Ao contrário, essas modificações são pequenas e fáceis de identificar, caso ocorram. Qualquer dúvida, o fórum da disciplina deve ser utilizado na tentativa de sanar qualquer dificuldade que apareça.

Para criarmos o nosso primeiro aplicativo, é necessário que o ambiente de programação esteja devidamente instalado e configurado. Para isso, iremos utilizar os softwares listados anteriormente.

02

Cada uma dessas ferramentas deve ser baixada e configurada adequadamente. Existem vários artigos disponíveis na internet que auxiliam esse procedimento.

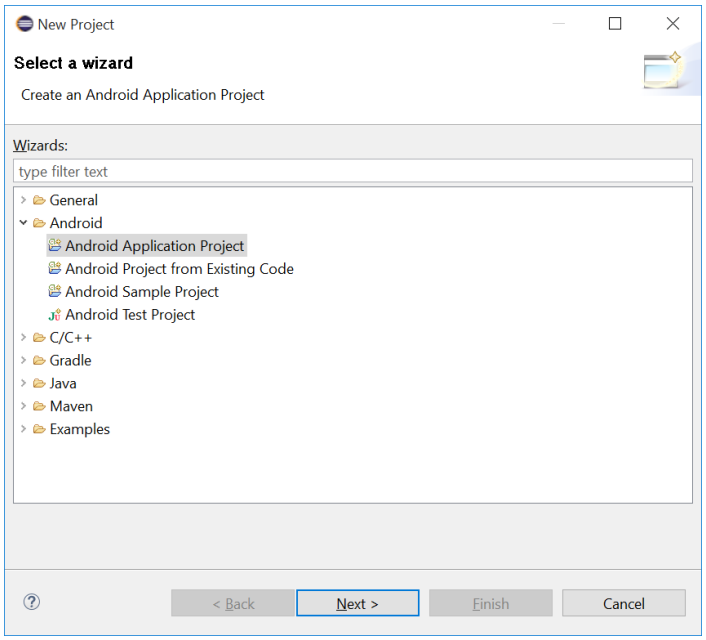
Contudo, atenção: após realizar o download do SDK do Android faz-se necessário executá-lo e realizar sua atualização. **Não atualize todo o SDK.** Isso, com certeza, levará muitíssimo tempo. Deste modo, apenas atualize o SDK com as versões mais recentes do conjunto mínimo de funcionalidades já instalado, que no caso da r24.4.1 que estamos instalando é a versão 6.0 (API 23).

No artigo do link abaixo, o autor utiliza a versão 2.2 (API 8). Para fins de instalação, atente-se para a versão que está sendo atualizada.

Como forma de auxiliá-lo, segue um dos possíveis links:
http://www.programarandroid.com.br/2014/11/como-instalar-o-eclipse-ide-e.html

Após a instalação e configuração podemos começar a programar!

Agora que temos o ambiente de desenvolvimento preparado, estamos prontos para escrever nossa primeira aplicação Android! Além de testarmos se o ambiente foi devidamente instalado e configurado, esse exemplo será a nossa primeira experiência em uma nova plataforma. Para começar, vamos desenvolver um clássico “Hello World”. No Eclipse, devemos criar um novo projeto Android, através do menu File > New > Project.... Na nova janela que será apresentada, selecione a opção “Android Application Project”, como mostra a figura abaixo.



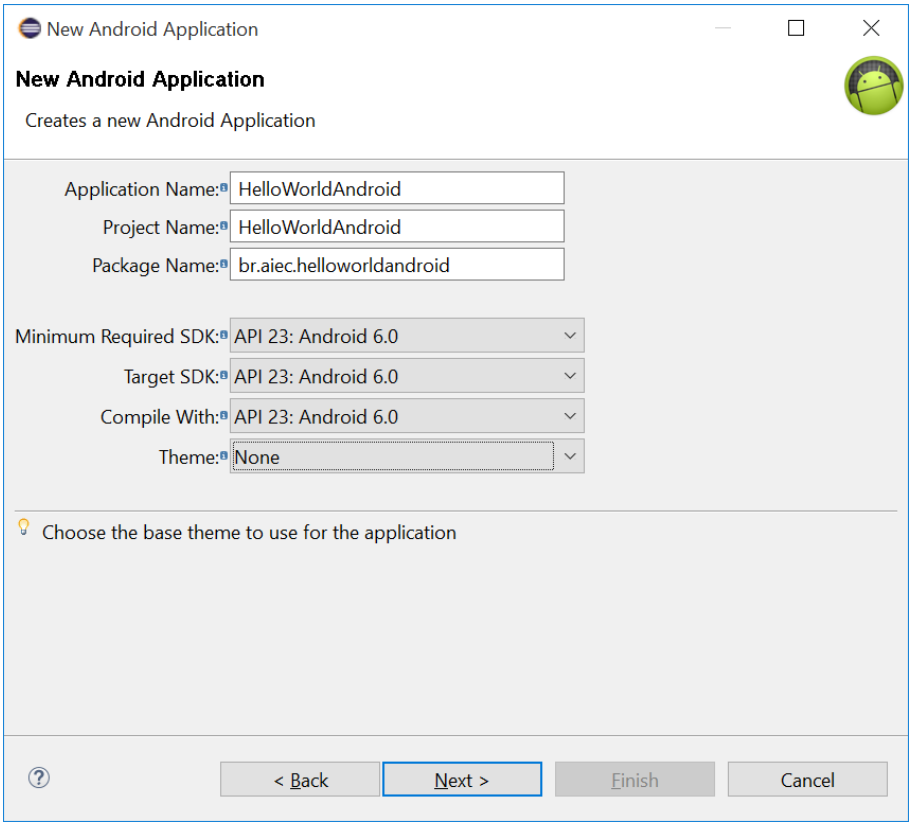
03

Na tela seguinte, escolha um nome para o seu projeto, no nosso caso, vamos chamar de **HelloWorldAndroid**.

É importante escolher o nome da aplicação e também do pacote com cautela, pois esses dois nomes serão utilizados para identificar sua aplicação, por exemplo, quando for feita uma publicação no Google Play.

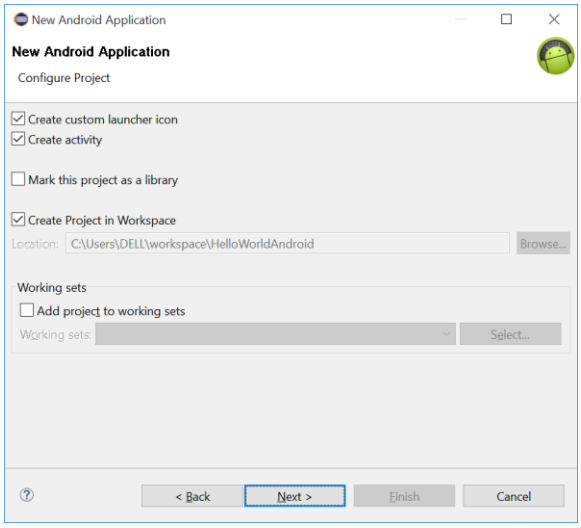
Para nossa aplicação, escolhemos o nome HelloWorldAndroid (no caso, o mesmo nome do projeto) e o pacote br.aiec.helloworldandroid. A opção Build SDK permite selecionar qual é a versão do Android que será utilizada por nossa aplicação. Como apenas a versão 6.0 (API 23) foi instalada, está é a única opção

disponível no momento. A opção Minimum Required SDK indica qual é a versão mínima exigida para executar o aplicativo. A imagem abaixo mostra como ficou a configuração do projeto.



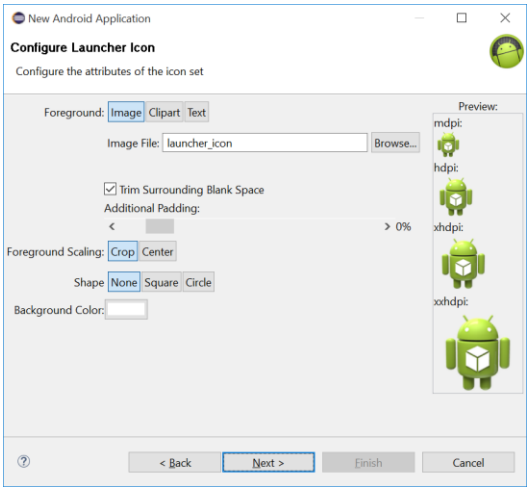
04

A próxima tela, apenas exibirá as opções padrão já marcadas. As opções “Create custom launcher icon” e “Create Activity” devem, obrigatoriamente, estarem marcadas neste nosso primeiro exemplo.



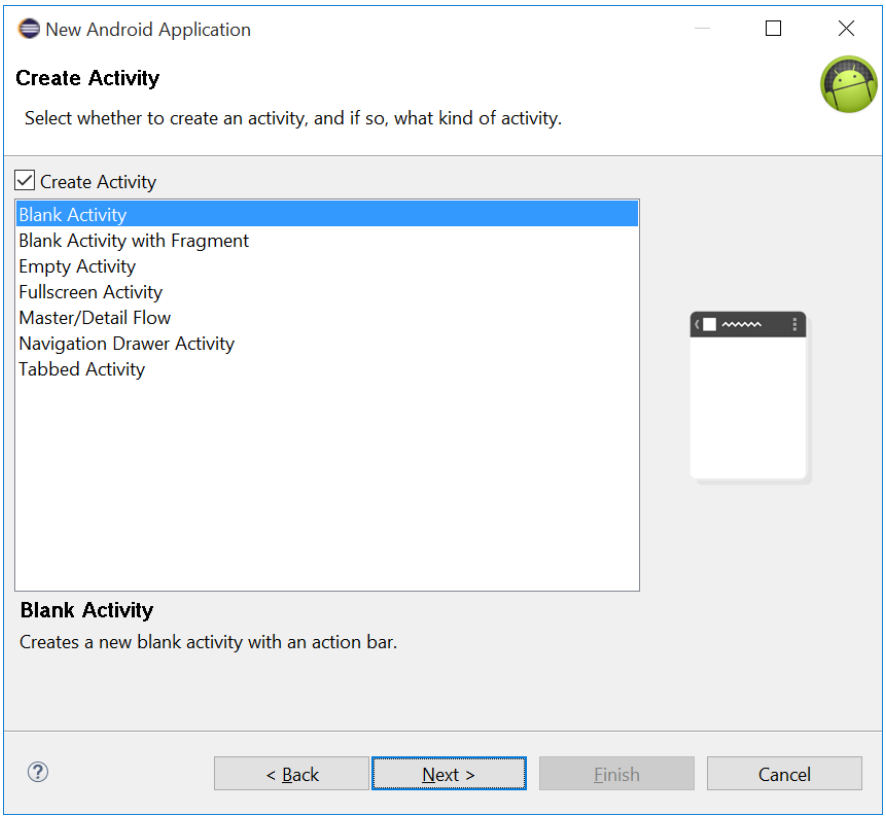
05

Na tela seguinte, é possível personalizar o ícone do seu aplicativo. Caso tenha alguma imagem personalizada, poderá utilizá-la nesta tela. Caso contrário, prossiga para a próxima tela.



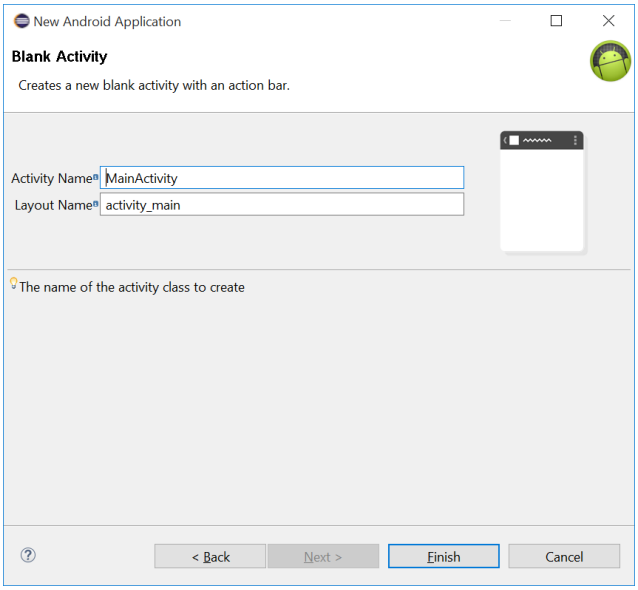
06

Na próxima tela, escolha as opções conforme a figura a seguir. Uma “Blank Activity” é suficiente para o nosso propósito neste momento.



07

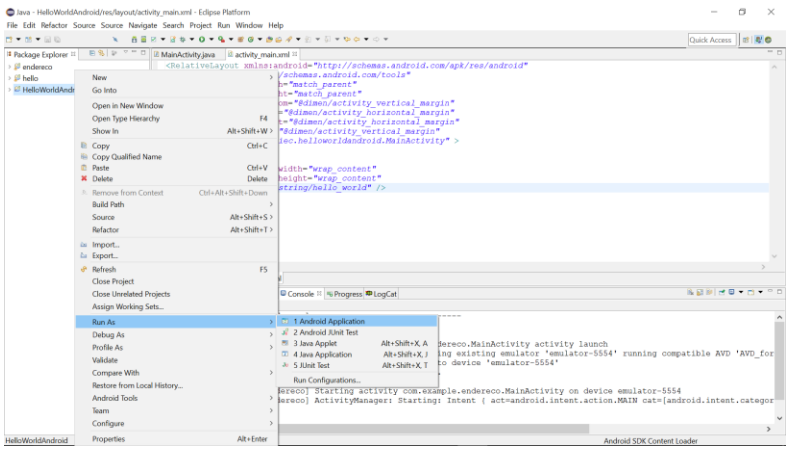
A tela seguinte, mostrada na figura abaixo, é a última. Escolha o nome da única Activity do aplicativo como sendo “MainActivity” e do layout como sendo “activity_main”. Não se esqueça desses nomes, pois precisaremos deles logo mais adiante.



08

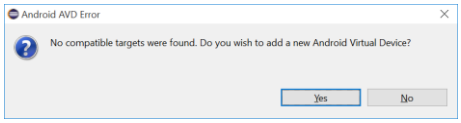
Com a ajuda do ADT, um novo projeto foi criado com as configurações escolhidas. Esse projeto já está com tudo que é necessário para ser executado.

Esse aplicativo de exemplo já possui toda a estrutura mínima necessária para ser executado no ambiente Android. Entraremos em mais detalhes adiante. Por ora, seu aplicativo já pode ser executado. Para isso selecione o projeto e clique com o botão direito do mouse em Run As > ***Android Application*** como na figura a seguir.



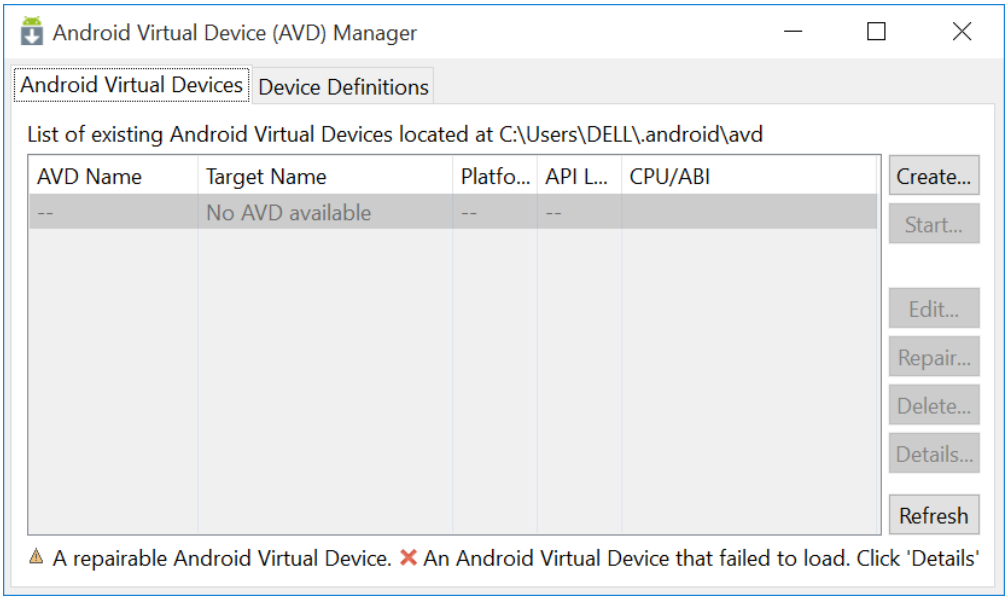
09

Ops! Algum problema aconteceu. Surgiu uma caixa de diálogo dizendo que não temos nenhum dispositivo compatível instalado, conforme a figura abaixo.



De fato, não criamos nenhum dispositivo virtual (Android Virtual Device, AVD) para rodar nossa aplicação! Vamos aproveitar e fazer isso agora.

Após mais essa configuração, seu ambiente estará pronto para executar outros aplicativos no futuro. A caixa de diálogo é fechada e o aplicativo Android Virtual Device Manager será iniciado conforme figura a seguir.

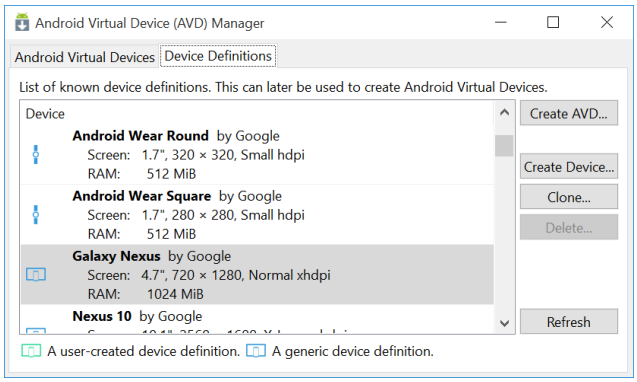


Na tela apresentada, existem duas abas disponíveis. Na aba ‘’Android Virtual devices” é possível criar um AVD personalizado por meio do botão “create” ou utilizar um AVD já previamente configurado por meio da aba “Device Definitions”.

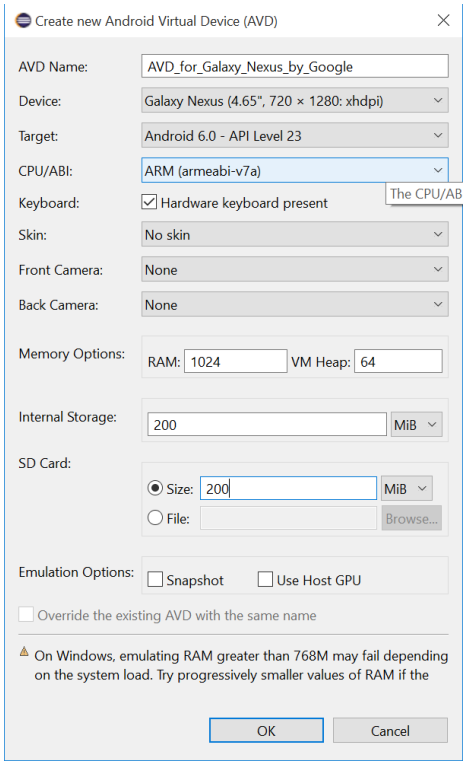
10

Como estamos procurando facilitar esse procedimento, iremos optar pela opção 2 (AVD já previamente configurado) clicando na aba “Device Definitions”, conforme a figura abaixo.

Escolha uma opção e clique no botão “Create AVD”.



Termine de preencher os campos conforme figura a seguir e clique em OK para finalizar.



11

Neste caso, foi necessário complementar as informações com o target (Android 6.0 – API level 23) e CPU/ABI (ARM – armeabi-v7a). Para aqueles que não conhecem, o ARM (Advanced Risc Management) é uma arquitetura de processadores RISC (Reduced instruction Set Computing) muito utilizada em sistemas embarcados como, por exemplo, smartphones modernos, calculadoras, tablets, dentre outros. Essa arquitetura é extremamente eficiente no que diz respeito ao baixo consumo de energia e à grande quantidade de instruções executadas por ciclo de clock.

Diferentemente do CISC (Complex Instruction Set Computer), o RISC possui algumas características significativas como:

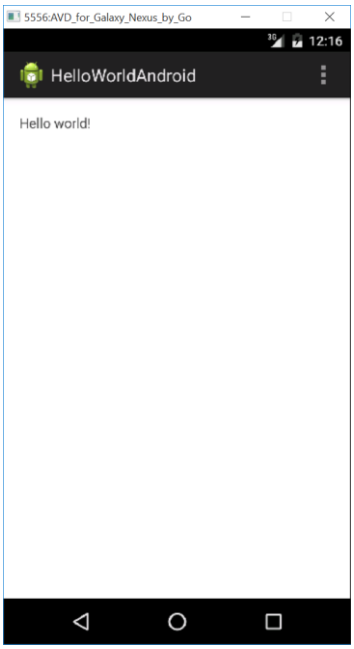
- o acesso à memória é limitado a instruções de carga e armazenamento (load e store),
- o formato de qualquer instrução é facilmente decodificável e de tamanho fixo
- a execução das instruções se dá em um único ciclo de clock o que, consequentemente, otimiza o uso da energia do processador.

Por esses e outros motivos, o ARM é largamente utilizado neste segmento de tecnologia móvel que possui sérias limitações, por exemplo, tamanho e durabilidade, de seus recursos energéticos. O ARMv7-A, que estamos usando no simulador, inclui em sua arquitetura, em relação às versões anteriores, um FPU (Float-Point Unit) que nada mais é que um coprocessador matemático desenhado para cálculos com números de ponto flutuante.

Obviamente que no simulador AVD disponibilizado pelo SDK Android os ganhos com o uso deste hardware podem não ser percebidos. Contudo, nos dispositivos reais essa melhora no desempenho é significativa, uma vez que os cálculos deixam de ser feitos por softwares e passam a ser feitos pelo próprio hardware.

12

Agora e só executar o projeto novamente. Como já existe um AVD criado para a plataforma alvo do nosso aplicativo, o emulador será iniciado automaticamente. A inicialização do emulador pode demorar um pouco, portanto, não se irrite com a demora. Aproveite para tomar um suco! Após a inicialização, a imagem abaixo será exibida.



2 - CONHEÇA A ESTRUTURA DO PROJETO

Já temos a primeira versão do nosso “Hello World”. Pode parecer estranho, mas ainda não fizemos nenhuma codificação para criar o nosso primeiro aplicativo. Contudo não se esqueça de que o ADT fez todo o trabalho por nós até agora.

Então vamos analisar a estrutura do projeto e tudo aquilo que foi gerado automaticamente para que essa “mágica” acontecesse. Na figura abaixo podemos identificar seis pastas principais (src, gen, assets, bin, libs e res), além de uma referência para a biblioteca Android 6.0 que está sendo utilizada:

1) src

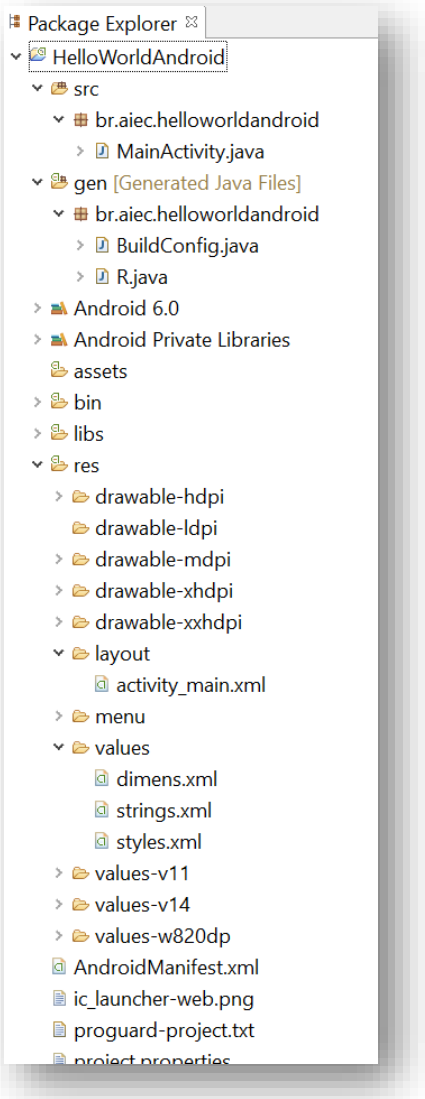
2) res

3) assets

4) gen

5) libs

6) bin



src Pasta dedicada aos armazenamentos dos códigos-fonte do projeto e será onde colocaremos as classes Java que criaremos em nossa aplicação. Repare que já existe uma MainActivity.java que foi criada automaticamente pelo plugin ADT quando criamos o projeto;
res Dedicado ao armazenamento de recursos (arquivos de layout, imagens, animações e xml contendo valores como strings, arrays etc.), acessíveis através da classe R;
assets Diretório para o armazenamento de arquivos diversos utilizados por sua aplicação. Diferentemente dos recursos armazenados na pasta res, estes são acessíveis apenas programaticamente, ou seja, é necessário escrever código de forma direta para ter acesso aos recursos armazenados neste diretório;
gen Armazena códigos gerados automaticamente por meio do SDK Android (aapt), como a classe R que mantém referencias para diversos tipos de recursos utilizados na aplicação;
libs Pasta para armazenar bibliotecas de terceiros que serão utilizadas pela aplicação;
bin Local utilizado pelos processos de compilação e empacotamento para manter arquivos temporários e códigos compilados.

14

No diretório raiz do projeto também existem alguns arquivos, sendo um deles, o **AndroidManifest.xml**, obrigatório para toda aplicação Android. Esse arquivo contém informações essenciais sobre a sua aplicação e sobre o que é necessário para executá-la, incluindo a versão mínima do Android.

O nome do pacote escolhido durante a criação do projeto, por exemplo, é armazenado lá para servir como identificador único da sua aplicação. Lembre-se que o manifesto também descreve, dentre várias outras coisas, os componentes (Activities, Services, ContentProviders e BroadcastReceivers) que fazem parte da aplicação, possibilitando que o sistema operacional Android seja capaz de identificá-los e determinar quando serão executados.

No código a seguir, a activity MainActivity é quem representa o componente que é iniciado quando executamos a aplicação.

HelloWorldAndroid Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.aiec.helloworldandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="23"
        android:targetSdkVersion="23" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

15

Lembre-se que já falamos um pouco sobre as activities anteriormente. Agora vamos aprender um pouco mais sobre elas.

As activities são componentes da plataforma Android, capazes de apresentar uma tela para interagir com os usuários.

Através delas podemos tirar uma foto, enviar um e-mail, visualizar uma imagem e navegar na Internet. Geralmente uma aplicação é composta por várias activities, sendo uma delas a activity principal que é executada quando a iniciamos.

O código a seguir demonstra a activity principal (**MainActivity.java**) do nosso projeto HelloWorldAndroid que foi criada automaticamente pelo ADT plugin:

MainActivity.java

```
1  package br.aiec.helloworldandroid;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.view.Menu;
6  import android.view.MenuItem;
7
8  public class MainActivity extends Activity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     }
15
16     @Override
17     public boolean onCreateOptionsMenu(Menu menu) {
18         // Inflate the menu; this adds items to the action bar if it is present.
19         getMenuInflater().inflate(R.menu.main, menu);
20         return true;
21     }
22
23     @Override
24     public boolean onOptionsItemSelected(MenuItem item) {
25         // Handle action bar item clicks here. The action bar will
26         // automatically handle clicks on the Home/Up button, so long
27         // as you specify a parent activity in AndroidManifest.xml.
28         int id = item.getItemId();
29         if (id == R.id.action_settings) {
30             return true;
31         }
32         return super.onOptionsItemSelected(item);
33     }
34 }
```

16

Para criar uma activity, basta fazer com que nossa classe estenda a classe Activity do Android, como pode ser visto na linha 8.

Já na linha 13, passamos para o método setContentView o identificador do layout, R.layout.activity_main, que deve ser carregado para construir a interface gráfica da nossa Activity. E bastante comum e também recomendado que as informações referentes a layouts e interfaces gráficas estejam externalizadas em arquivos XML, separados do código da aplicação. Isso é para evitar o famoso problema do **HARD CODE** e do engessamento da aplicação.

Essas e outras boas práticas são fundamentais para se construir aplicativos robustos. Vamos prosseguir verificando o arquivo activity_main.xml, representado na imagem abaixo, que se encontra no diretório res/layout/ do projeto.

activity_main.xml

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      tools:context="br.aiec.helloworldandroid.MainActivity" >
10
11     <TextView
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="@string/hello_world" />
15
16 </RelativeLayout>
17
```

17

Neste arquivo de definição de layout temos dois elementos declarados, o RelativeLayout e o TextView, com seus respectivos atributos. Como o nome sugere, o RelativeLayout (linha 1) é um elemento para organização do layout da tela, permitindo configurar a sua altura e largura. Já o TextView é um widget utilizado para apresentar na tela uma informação textual.

O valor a ser exibido por este elemento está especificado através do atributo “text”. Repare que na linha 14 o valor que TextView deve exibir e @string/hello_world. Aqui temos novamente um caso no qual a externalização é recomendada principalmente para facilitar a internacionalização da aplicação, suportando vários idiomas diferentes e para até mesmo reaproveitar mensagens.

O valor que será utilizado no TextView será na verdade o conteúdo da string que possui o identificador hello_world. No arquivo res/values/strings.xml é possível observar como isso foi definido:

strings.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">HelloWorldAndroid</string>
5      <string name="hello_world">Hello world!</string>
6      <string name="action_settings">Settings</string>
7
8  </resources>
9
```

Na linha 5, declaramos uma string com o nome hello_world cujo valor e “Hello World!”. Por convenção, o arquivo strings.xml é onde definimos recursos do tipo string, ou seja, textos que queremos exibir de alguma maneira em nossa aplicação.

3 – Resumo

O objetivo deste módulo foi apresentar o passo a passo de como construir um esqueleto de um aplicativo Android que executa o clássico Hello World. Esse esqueleto foi construído com o auxílio da IDE Eclipse, SDK Android, JDK e do plugin ADT (Android Development Tools). Além disso, foi apresentada a estrutura (classes, recursos e configurações) comum, a qual é frequentemente utilizada por diversos aplicativos Android.

Unidade 4 – Construindo aplicativos Android

Módulo 2 – Segundo aplicativo Android

01

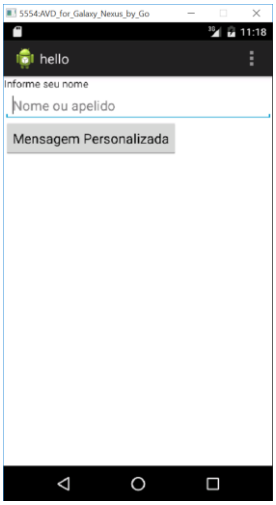
1 - Componentes visuais Edit Text e Button

Para melhorar a nossa aplicação HelloWorld 1.0 iniciada no módulo anterior, iremos incluir mais algumas coisas e aproveitar para entender alguns pontos fundamentais do desenvolvimento Android.

Em nossa versão melhorada do “HelloWorld”, a versão 2.0, o usuário:

- informará seu nome em uma caixa de texto,
- pressionará um botão e
- a aplicação apresentará uma mensagem de boas-vindas personalizada.

A aplicação ficará com a aparência da figura abaixo.



Não se esqueça que para cada Activity criada existirá um arquivo de layout XML para ela. Apesar de ser possível definir o layout diretamente no código Java, essa prática é desaconselhada e a forma recomendada pelo framework Android é de que o layout seja feito por meio dos respectivos arquivos XML de modo a tornar a sua aplicação mais flexível, universal e até mesmo customizável.

02

Podemos modificar o layout activity_main.xml já existente, para incluir um campo onde o usuário irá informar o seu nome. Esse campo pode ser criado utilizando um widget do tipo EditText, no qual podemos inclusive indicar que o mesmo receberá o foco da aplicação, conforme as instruções abaixo:

```
<EditText
        android:id="@+id/nomeEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:hint="@string/nome_hint">

        <requestFocus />

    </EditText>
```

Esses widgets que estamos apresentando neste módulo nada mais são que classes Java pertencentes a API do Android.

Observe que o elemento em comento possui vários atributos declarados. Por exemplo, o atributo “android:inputType” é responsável por determinar qual o tipo de teclado (alfa-numérico, numérico, calendário, senha, e-mail, dentre outros) deverá ser exibido para o usuário quando o componente receber o foco da navegação do usuário. Além disso, esse atributo pode também controlar alguns comportamentos do teclado com o objetivo de melhorar a experiência de uso do usuário.

Portanto, o “android:inputType” determina alguns comportamentos e quais os tipos de caracteres permitidos no referido campo, além de também exibir o teclado virtual mais adequado ao tipo pré-determinado, de modo a otimizar a experiência do usuário por meio da digitação em função dos caracteres permitidos e, consequentemente, mais utilizados.

03

Abaixo, uma tabela com alguns possíveis valores para o atributo **android:inputType**:

Valor	Descrição
“text”	Teclado de texto
“textEmailAddress”	Teclado de texto com o caractere de @

“textUri”	Teclado de texto com o caractere de /
“number”	Teclado numérico
“phone”	Teclado telefônico
“textCapSentences”	Teclado de texto que coloca em maiúsculo a primeira letra de cada sentença/frase.
“textCapWords”	Teclado de texto que coloca em maiúsculo a primeira letra de cada palavra.
“textAutoCorrect”	Teclado de texto que corrige palavras com erros ortográficos comumente conhecidos.
“textPassword”	Teclado de texto, porém os caracteres introduzidos se transformam visualmente em pontos.
“textMultiLine”	Teclado de texto que permite aos usuários digitarem cadeias longas de caracteres que incluem quebras de linha (\n) e retorno de carro (\r).

Além disso, o atributo “android:inputType” permite uma combinação **bitwise** de valores específicos como no exemplo resumido de código abaixo:

```
<EditText
      ....
        android:inputType="textPersonName|textCapWords"
      ....

</EditText>
```

Desta forma, o EditText assume os dois valores específicos e simultâneos bem como os seus respectivos comportamentos. Ou seja, no caso do exemplo resumido acima, o EditText apresentará um teclado de texto que auxilia a digitação do nome da pessoa bem como colocará a primeira letra de cada termo que compreende o nome e sobrenome, quando digitado, em maiúsculo.

04

É importante perceber que o uso do bitwise pode ser aplicado para mais de dois comportamentos específicos e simultâneos, ou seja, três, quatro, inúmeros, desde que a combinação dos comportamentos tenha algum significado real.

Combinar os comportamentos de forma irresponsável poderá dificultar a experiência do usuário. Para uma lista completa de todos os possíveis valores clique aqui.

Além disso, também teremos que incluir um **botão**, através do widget Button, conforme o trecho de código abaixo:

```
<Button
        android:id="@+id/mensagemPersonalizadaButton"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/mensagem_personalizada"

/>
```

Clique aqui

Para uma lista completa de todos os possíveis valores acesse http://developer.android.com/intl/pt-br/reference/android/widget/TextView.html#attr_android:inputType

05

Neste momento, para facilitar a criação do layout com estes novos elementos, iremos substituir o "RelativeLayout" por um "LinearLayout" conforme o código abaixo:

```
<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="com.example.hello.MainActivity"
    android:orientation="vertical" >

      <!—Campos/Widgets virão aqui -->

</LinearLayout>
```

O "***RelativeLayout***" é um dos mais poderosos e versáteis disponíveis na plataforma Android pois permite posicionar um elemento em um local da tela que seja relativo a outro componente. Contudo, neste exemplo não será necessário utilizá-lo em virtude da quantidade pequena de widgets (apenas 3 componentes) utilizados pelo exemplo. Saiba+ sobre *layouts.*

O "LinearLayout" permite a organização dos elementos de forma linear, posicionando itens um abaixo do outro, quando configurado com orientação vertical, ou um ao lado do outro, quando configurado com orientação horizontal. Às vezes, escolher a orientação certa, causa um pouco de confusão. Portanto, a dica é se lembrar de que a orientação diz respeito a direção na qual os itens serão incluídos na tela, ou seja, na orientação vertical, os itens serão incluídos no layout de cima para baixo e na orientação horizontal, da esquerda para a direita.

Saiba+
Para saber mais sobre layouts acesse http://developer.android.com/training/improving-layouts/index.html.

06

Para especificar um layout existem dois atributos fundamentais:

- o "***layout_width***", que indica a largura do elemento,
- e "***layout_height***", que indica a sua altura.

Há dois valores importantes para estes atributos que são o "match_parent" e o "wrap_content". O primeiro valor indica que o tamanho deve ser o mesmo que o do elemento-pai enquanto o segundo indica que o tamanho deve ser grande o suficiente para abrigar o conteúdo a ser exibido.

Esses mesmos dois atributos se aplicam a diversos outros layouts e widgets, como o EditText e o Button mostrados anteriormente. Desta forma, a tela do HelloWorld 2.0 ficará conforme o código abaixo:

```
<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="com.example.hello.MainActivity"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/titulo" />

    <EditText
        android:id="@+id/nomeEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName">

        <requestFocus />

    </EditText>

    <Button
        android:id="@+id/mensagemPersonalizadaButton "
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/mensagem_personalizada"

        />

    <TextView
        android:id="@+id/boasVindasTextView"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />

</LinearLayout>
```

Nos itens recém adicionados, repare que colocamos um atributo “android:id” que é importante, pois posteriormente precisaremos referenciar e manipular esses componentes visuais por meio da classe R.

07

Também é necessário criar as strings que serão utilizadas como o título, o rótulo do botão e também as boas_vindas. Nosso arquivo strings.xml deverá ficar conforme o código abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">hello</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="title_activity_main">Titulo da Activity Principal</string>
    <string name="titulo">Informe seu nome</string>
    <string name="mensagem_personalizada">Mensagem Personalizada</string>
    <string name="boas_vindas">seja bem vindo a disciplina de Programação
Mobile (AIEC).</string>
    <string name="nome_hint">Nome ou apelido</string>

</resources>
```

Ao executarmos a aplicação novamente já perceberemos as mudanças realizadas. Contudo, ainda não programamos nenhuma ação para o botão disponível na tela, que, caso seja pressionado, nada de diferente acontecerá.

Para obter o resultado esperado, criaremos um método na nossa “MainActivity”, que responderá a esta ação exibindo ao usuário uma mensagem personalizada.

08

Então vamos configurar nosso botão que já existe para que, quando ele for pressionado, um método seja invocado. Para isso, adicionaremos o atributo **onClick** no widget Button conforme código abaixo:

```
<Button
        android:id="@+id/mensagemPersonalizadaButton "
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="@string/mensagem_personalizada"
        android:onClick="mensagemPersonalizada"
 />
```

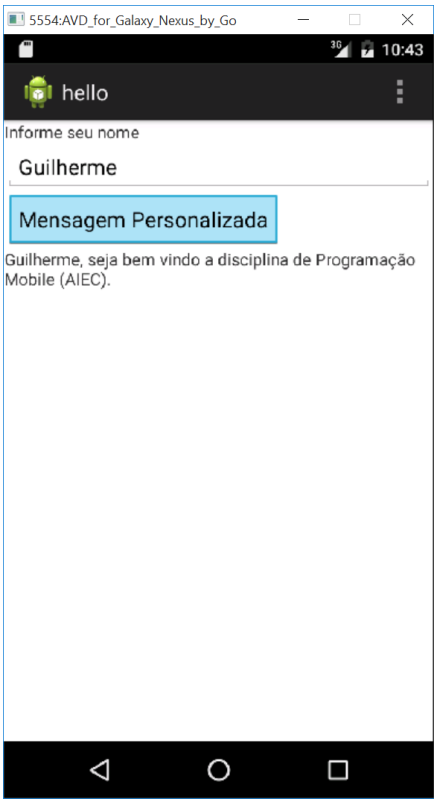
Informamos que o método a ser acionado após o clique do botão e aquele cujo nome é "mensagemPersonalizada", através da propriedade onClick. Este método **deve** necessariamente ser público e receber como parâmetro um objeto do tipo View que é uma referência do botão que foi pressionado. Nesse método, precisamos modificar o conteúdo do widget "boasVindasTextView" para que ele mostre o conteúdo informado no "EditText" acrescido do conteúdo personalizado pela variável "boas_vindas" presente no arquivo strings.xml. Abaixo, o código da classe MainActivity:

```
package com.example.hello;

import android.app.Activity;
import android.os.Bundle;
import android.text.Editable;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {

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

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is
present.
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		// Handle action bar item clicks here. The action bar will
		// automatically handle clicks on the Home/Up button, so long
		// as you specify a parent activity in AndroidManifest.xml.
		int id = item.getItemId();
		if (id == R.id.action_settings) {
			return true;
		}
		return super.onOptionsItemSelected(item);
	}
```

```
      public void mensagemPersonalizada(View view){
            EditText nomeEditText = (EditText)
findViewById(R.id.nomeEditText);
            TextView saudacaoTextView = (TextView)
findViewById(R.id.boasVindasTextView);
            Editable nome = nomeEditText.getText();
            String boasVindas =
getResources().getString(R.string.boas_vindas);
            String mensagem = nome.toString() + ", " + boasVindas;
            saudacaoTextView.setText(mensagem);
      }
}
```

09

Ao executar o aplicativo, o resultado deve ser algo semelhante ao mostrado na figura a seguir.



Neste módulo, melhoramos o aplicativo de exemplo do módulo anterior, buscamos entender seus detalhes e organização, além de fazer modificações no código e layout para termos nossa segunda experiência em programação de aplicativos móveis para o sistema Android.

10

2 - RESUMO

O objetivo deste módulo foi apresentar dois componentes visuais: EditText e Button. O EditText é um componente textual que permite ao usuário informar dados para o sistema enquanto que o Button é um componente de evento que permite ao usuário solicitar que o sistema execute alguma coisa. Além disso, esse módulo apresentou o modo de funcionamento de um dos possíveis layouts disponíveis (Linearlayout) na construção de GUI no sistema Android.

UNIDADE 4 – CONSTRUINDO APLICATIVOS ANDROID

MÓDULO 3 – INTENTS E INTENT FILTERS

01

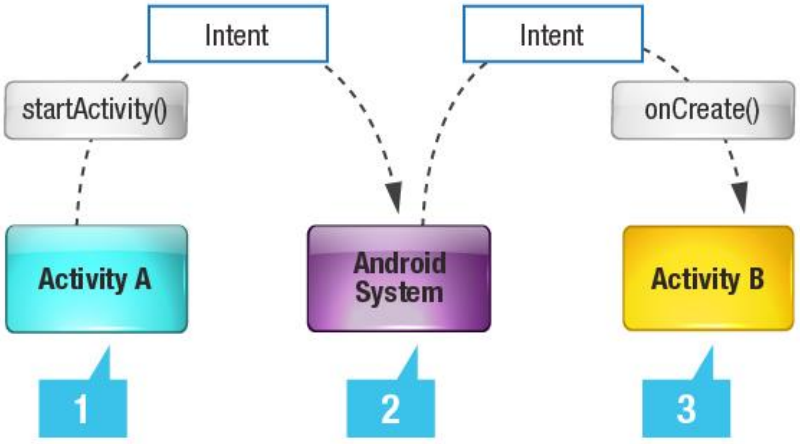
1 - INTENTS

As *Intents* geralmente são criadas a partir de ações do usuário e representam a intenção de se realizar algo, como, por exemplo, iniciar o aplicativo de correio eletrônico do Android, iniciar a reprodução de uma música, iniciar um *download*, acessar um conteúdo web, dentre outros.

Formalmente, as Intents podem ser definidas como mensagens enviadas por um componente da sua aplicação (uma activity, por exemplo) para o Android, informando a intenção de inicializar outro componente da mesma aplicação ou de outra aplicação qualquer..

Portanto, as Intents são consideradas um **mecanismo de IPC** (Inter Process Comunication), ou seja, um mecanismo que permite aos processos transferirem dados entre si. Além disso, todo o modelo de desenvolvimento Android é centrado na utilização de *Intents* para inicialização de componentes pertencentes a outros aplicativos procurando garantir assim, reusabilidade e baixo acoplamento entre as diversas funcionalidades existentes.

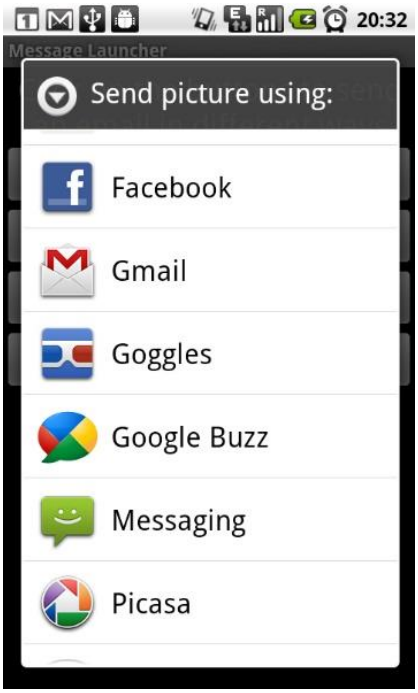
Abaixo, uma figura que demonstra um Intent implícita:



Em linhas gerais, a figura acima demonstra que a *Activity* A cria um *Intent* implícito e por meio do método *startActivity* transmite a intenção para o Android. De posse da *Intent*, o Android procura na base de aplicativos instalados no dispositivo, qual ou quais aplicativos podem atender à intenção do usuário. Caso exista apenas uma única opção que seja capaz de processar a intenção, o Android irá inicializar a nova Activity, neste caso B, por meio do método onCreate, ocasião em que o Intent transmitido pela Activity A é repassado a Activity B.

02

No entanto, caso existam várias opções que sejam capazes de processar a intenção de A, a imagem abaixo será exibida de modo que o usuário possa escolher qual o aplicativo irá tratar a intenção. Neste caso, as opções apresentadas pelo Android correspondem as aplicações que são capazes de compartilhar uma imagem, pois esse foi o *Intent* criado pela *Activity* A.



Fique Atento!

As *Intents* são um recurso-chave no Android pois é através delas que podemos fazer com que as aplicações colaborem entre si, disponibilizando, por exemplo, funcionalidades que podem ser reutilizadas, sem a necessidade de reescrever ou importar códigos, uma vez que tais práticas, consequentemente, geram dependências desnecessárias em sua aplicação.

Através de *Intents* é possível iniciar novas *activities*, como fazer uma busca e selecionar um contato do telefone, abrir a aplicação de mapas com as coordenadas de localização do GPS, abrir uma página da web, tirar fotos utilizando a câmera, capturar algum SMS específico, gravar o áudio de uma ligação telefônica, dentre inúmeras outros, apenas reaproveitando funcionalidades já existentes, disponibilizadas pelos aplicativos instalados no aparelho. Saiba+

Saiba+
Observe que o uso ou reuso de tais funcionalidades está condicionada a quais aplicativos estão devidamente instalados e configurados no aparelho em questão em um determinado instante do tempo. Ou seja, a forma como cada dispositivo responde aos Intents em tempo de execução depende da configuração específica do aparelho no qual o aplicativo está sendo executado.

03

Aplicativos de terceiros, assim como os nossos, podem disponibilizar novas funcionalidades acessíveis via *Intents*. Existem, por exemplo, aplicativos de leitura de códigos de barra que podem ser chamados pela sua aplicação para lê-los utilizando a câmera do aparelho e devolver o resultado para ser processado por um método da sua aplicação.

Podemos criar e utilizar as *Intents* de diversas maneiras e a seguir veremos alguns exemplos. O trecho de código abaixo mostra como abrir uma página utilizando o navegador que acompanha o Android:

```
Uri uri = Uri.parse("http://www.android.com");
Intent intent = new Intent (Intent.ACTION_VIEW, uri);
startActivity (intent);
```

Um objeto do tipo **Uri** foi criado a partir de uma string representando o endereço eletrônico que desejamos acessar via protocolo HTTP. Em seguida, instanciamos uma nova *Intent* informando a ação que gostaríamos de executar (*Intent.ACTION_VIEW*), juntamente com a Uri criada. Observe que o construtor da classe Intent, neste caso, recebeu dois parâmetros. Contudo, existem outros construtores da classe Intent que possuem comportamentos diferentes do mostrado neste exemplo. Saiba+ sobre os construtores.

Continuando a explicação do código fonte de exemplo, após a criação do objeto do tipo Intent, chamamos o método ***startActivity*** da classe *Activity* passando a *intent* como parâmetro. Repare que não indicamos exatamente a *activity* que deve ser iniciada para abrir o site desejado. Neste caso, a nossa *Intent* **é classificada como implícita**. Com base na ação *Intent.ACTION_VIEW* e o no conteúdo da Uri da *Intent*, o Android decide qual a activity, já registrada de maneira prévia no sistema, é mais adequada para resolver a URI informada. Neste caso o escolhido é o navegador web.

A seguir, temos um outro exemplo de como iniciar uma nova activity existente na nossa aplicação, passando no construtor da *Intent* a classe correspondente a activity que deve ser iniciada.

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

Saiba+
Para conhecer mais sobre os construtores acesse: http://developer.android.com/intl/pt-br/reference/android/content/Intent.html#pubctors.

04

Diferentemente do exemplo anterior, agora nos informamos exatamente qual *activity* deve ser iniciada, ou seja, agora nossa *Intent* é **explícita**. Geralmente as *Intents* explícitas são utilizadas apenas para interação entre componentes de uma mesma aplicação, já que é necessário conhecer o componente que deverá ser ativado pelo seu nome.

Apesar de ser possível saber o nome de componentes de outras aplicações, é terminantemente proibido utilizar a forma explícita para se comunicar com componentes de outra aplicação uma vez que tal prática vai de encontro ao modelo de desenvolvimento sugerido para sistemas Android. Ou seja, tal prática cria um alto acoplamento entre os componentes, o que dentro do modelo Android de desenvolvimento é algo considerado prejudicial.

Portanto, caso queira que seu aplicativo se comunique com componentes de outro aplicativo**, a forma recomendada de uso de Intents é a forma implícita.** Tal forma é usada para ativar componentes de outra aplicação, fornecendo informações adicionais, como a ação e Uri, para que o Android localize o componente adequado.

Outra característica importante é que podemos colocar informações extras na *Intent,* as quais serão utilizadas posteriormente pelo componente iniciado por ela.

05

Para exemplificar, considere que a nossa aplicação deve tirar uma foto e armazená-la em uma pasta específica. O Android já possui um aplicativo que realiza esta tarefa de tirar foto, e o que queremos é chamá-lo a partir da nossa aplicação para capturar a imagem e salvá-la em um local determinado. O código abaixo inicia a Activity de câmera do aparelho, informando o local e nome desejado para o armazenamento da imagem capturada:

```
/*
* O exemplo considera que existe a pasta MinhasImagens
* e que o aplicativo tem permissão de escrita na mesma.
*/
```

```
Uri uri = Uri.fromFile (new File("/sdcard/MinhasImagens/hello_camera.jpg"));
Intent intent = new Intent (MediaStore.ACTION_IMAGE_CAPTURE);
```

```
intent.putExtra (MediaStore.EXTRA_OUTPUT, uri);

startActivity (intent);
```

Esse exemplo também utiliza *Intents* implícitas, pois em nenhum momento foi indicado qual a classe de *Intent* deveria ser utilizada.

Em resumo, uma *Intent* **é o conjunto de informações necessárias para ativar um componente de uma aplicação**.

Basicamente, podemos descrever seis informações básicas que devem ser consideradas quando do uso de *Intents*:

- nome,
- ação,
- dados,
- extras,
- categoria e
- sinalizadores.

Veremos o tratamento de cada uma dessas informações a seguir.

06

1.1- Nome (name)

O **nome do componente** é definido pelo nome completo da classe e o nome do pacote definido no AndroidManifest.xml que representam o componente que deve ser o encarregado de tratar a *Intent*.

Quando criamos uma *Intent* explícita com o construtor Intent (this, OutraActivity.class), o nome do componente é criado automaticamente. No entanto, também é possível defini-lo de forma programática, utilizando os métodos *setComponent(), setClass()* ou *setClassName()* da classe *Intent*.

1.2- Ação (Action)

A **ação** é uma string que define o que deve ser realizado. Existem diversas ações genéricas no Android, disponibilizadas como constantes na classe *Intent*.

Alguns exemplos de constantes são:

• ACTION_CALL - indica que uma chamada telefônica deve ser realizada.
• ACTION_VIEW - indica que algum dado deve ser exibido para o usuário.
• ACTION_EDIT - indica que se deseja editar alguma informação.
• ACTION_SENDTO - indica que se deseja enviar alguma informação.
• ACTION_DIAL - indica que se deseja exibir um número de telefone.

Enquanto a *Intent* declara **o que** deve ser feito, o componente que a recebe é o responsável por definir **como** a ação será executada. Ou seja, para uma mesma ação, podemos ter comportamentos distintos quando ela for executada por diferentes componentes. Exemplo

Exemplo
Um exemplo disso e a ACTION_VIEW, que pode ser utilizada tanto para indicar que desejamos abrir uma página da Internet quanto para abrir informações de um contato armazenado no telefone.

07

1.3- Dados (Data)

Os **dados** de uma *Intent* são representados através de uma Uri e a partir dela, a aplicação decide o que deve ser feito.

No primeiro exemplo de uso de *Intents* criamos uma Uri para a página que gostaríamos de visitar. Outro exemplo seria criar uma *Intent* informando uma Uri com valor “content://contacts/people/”, que abriria os contatos do telefone, conforme as instruções abaixo:

```
Uri uri = Uri.parse ("content://contacts/people/");
Intent intent = new Intent (Intent.ACTION_VIEW, uri);

startActivity (intent);
```

1.4- Extra (extra)

As **informações extras** são quaisquer outros dados necessários para que o componente execute a ação apropriadamente.

Eles podem ser informados através dos extras da *Intent*. No exemplo anterior, passamos uma Uri como extra para informar o local onde a foto deveria ser armazenada. Além disso, podemos também informar outros tipos de dados como strings, tipos primitivos, arrays e objetos serializáveis. Para incluir um dado como extra, utilizamos o método *putExtra* da classe *Intent*, fornecendo **o primeiro parâmetro** como

uma string que servirá como **identificador** do dado e **o segundo parâmetro** como o **extra** o qual possui um determinado **valor**. Relembre com o código abaixo, no qual o MediaStore.EXTRA_OUTPUT é o identificador da informação e a uri é o extra:

```
intent.putExtra (MediaStore.EXTRA_OUTPUT, uri);
```

08

1.5- Categoria (Category)

A **categoria**, representada apenas por uma string, serve como informação adicional para auxiliar o Android na escolha de qual componente e o mais adequado para receber a *Intent*.

Podemos adicionar várias categorias a uma *Intent* através do método *addCategory*. Assim como as ações, existem várias categorias predefinidas, como a *Intent.CATEGORY_APP_MUSIC*, que, quando colocada em uma
Intent, informara ao Android que uma *Activity* capaz de reproduzir músicas deve ser acionada.

1.6- Sinalizadores (flags)

Os sinalizadores definidos na classe *Intent* funcionam como metadados para a intenção.

Ou seja, os metadados são dados que descrevem outros dados, o que significa, por exemplo, que os sinalizadores podem instruir o sistema Android sobre como inicializar uma *activity* bem como tratá-las após sua inicialização. É importante frisar que a maioria dos flags disponíveis dependem do tipo de componente que será executado pelo Android baseado na Intent em questão.

09

2- INTENTS FILTERS

As informações contidas nas *Intents* são utilizadas pelo Android para localizar o componente adequado, geralmente uma *activity*, para executar a ação desejada. Quando o nome do componente é informado, o Android inicializa exatamente aquele componente, sem necessidade de avaliar a ação ou categoria. Por outro lado, quando o nome não é informado, é necessário consultar quais são os componentes existentes com a habilidade de executar a ação desejada e que pertencem as categorias existentes na *Intent*.

Adicionalmente, o Android também pode procurar por componentes capazes de resolver a Uri repassada e também de lidar com o formato dos dados, o MIME type, informado.

Um ponto importante no uso de *Intents* é saber determinar como que o Android sabe ou encontra a *Activity* que deve ser iniciada. Esse ponto é fundamental, pois isso é feito baseando-se apenas nessas informações básicas da *Intent*. Portanto, em algum lugar deve estar especificado que determinadas ações podem ser resolvidas por um dado componente.

A definição de quais ações um componente está apto a responder, bem como a quais categorias ele pertence e também quais dados ele sabe tratar, é realizada através de ***intent filters*** que são configurados no arquivo AndroidManifest.xml.

10

No nosso primeiro exemplo, o Hello World 1.0, já existe a declaração de um *intent filter* no AndroidManifest.xml para a *Activity* principal da nossa aplicação:

```
...
        <activity
       android:name=".MainActivity"
       android:label="@string/app_name" >
       <intent-filter>
          <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
    </activity>
...
```

Este *intent filter* indica que a activity *MainActivity* é aquela que deve ser iniciada ao abrir a aplicação e que também deve ser listada como uma aplicação do Android que pode ser utilizada por um usuário. Os *intent filters* podem ainda declarar, além da ação e da categoria, os tipos de dados com os quais o componente é capaz de lidar, como uma imagem por exemplo:

```
<data android:mimeType="image/*" />
```

Com base nestas três informações (action, category e data), o Android e capaz de selecionar qual é o componente mais adequado para responder a uma *Intent* implícita, comparando o que foi passado na *Intent* com aquilo que está declarado nos *intent filters* dos aplicativos. Nossas aplicações podem definir *intent filters* com ações e categorias próprias ou fazer uso das já existentes para expor funcionalidades para as demais aplicações.

Para saber mais sobre *Intents* e *Intents Filters*, acesse:
http://developer.android.com/intl/pt-br/guide/components/intents-filters.html

11

3- RESUMO

O objetivo deste módulo foi apresentar dois mecanismos, Intent e Intent Filters, importantes de IPC (Inter Process Comunication) utilizados frequentemente no sistema operacional Android. Os Intents podem ser implícitos ou explícitos sendo que o primeiro é frequentemente utilizado para comunicação entre componentes de processos diferentes enquanto que o segundo é frequentemente utilizado para comunicação entre componentes do mesmo processo. Já o Intent Filter é responsável por configurar quantos e quais são os componentes/processos responsáveis por executarem Intents implícitas.

UNIDADE 4 – CONSTRUINDO APLICATIVOS ANDROID

MÓDULO 4 – CICLO DE VIDA DE UMA ACTIVITY

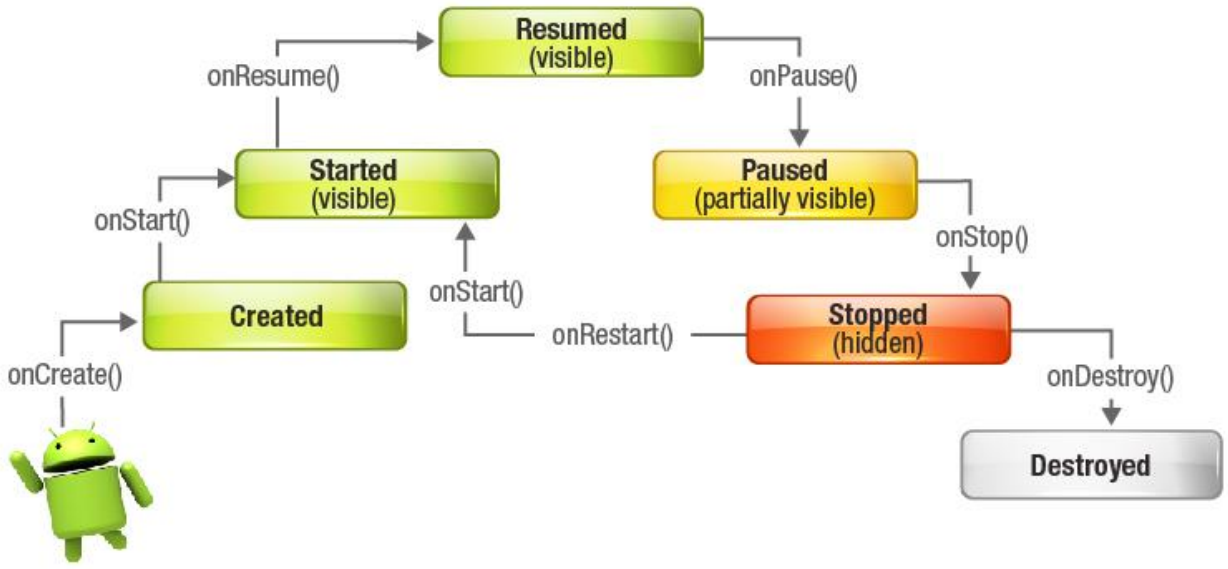
01

1 - CICLO DE VIDA DA ACTIVITY

A *Activity* é um componente de aplicação com um ciclo de vida específico. Quando o usuário acessa a aplicação, navega pelas opções, sai ou retorna para a mesma, as *activities* que a compõem passam por uma série de estados/estágios do ciclo de vida.

Entender como o ciclo vida funciona é importante para preparar a aplicação para lidar com situações que podem interferir na sua execução, tais como o recebimento de uma ligação, o desligamento da tela do aparelho ou ainda a abertura de outra aplicação feita pelo usuário.

A imagem abaixo ilustra o ciclo de vida da *Activity*.



Sempre que a Activity transita de estado, o Android aciona um método (*callback*) correspondente. Assim que o usuário inicia uma aplicação, o Android cria a activity principal que está declarada no AndroidManisfest.xml e invoca o seu método *onCreate*. Não se esqueça que é neste método que atribuímos qual *layout* será utilizado pela nossa activity além de poder também inicializar um conjunto de variáveis/objetos e demais recursos necessários.

Em seguida, o Android invoca os métodos *onStart* e *onResume* em sequência, ou seja, primeiro *onStart* e depois *onResume*. Entre o fim do onStart e o início do onResume, a Activity torna-se visível para o usuário no estado denominado de *Started*. Após a execução do método *onResume*, a *activity* se encontra no estado *Resumed*. Uma vez neste estado, permanecerá nele até que os métodos onPause (visível parcialmente) ou onDestroy serem chamados.

02

Quando a Activity está no estado ***Resumed*** dizemos que ela está no primeiro plano (foreground) e pode realizar interação com o usuário. Neste plano foreground o usuário pode interagir com o aplicativo fornecendo ou recebendo dados do mesmo.

A activity transita para o estado ***Paused*** quando for, por exemplo, parcialmente encoberta por outra activity inicializada pelo usuário, que pode não ocupar toda a tela ou ser transparente. Se o usuário sair da aplicação ou iniciar outra activity que encubra totalmente a que está sendo executada, então o método *onStop* é invocado e a activity vai para o segundo plano (background). Mesmo não sendo mais visível pelo usuário em background, a activity continua instanciada na memória de trabalho e com seu estado interno representado, ou seja, da forma como estava quando em execução no estado *Resumed*.

Quando uma activity está nos estados de ***Paused*** ou ***Stopped***, o sistema operacional Android poderá removê-la da memória de trabalho, invocando o seu método *finish* ou encerrando arbitrariamente o seu processo. Isso ocorre em função da política de escalonamento de processos e threads do sistema operacional conforme apresentamos em ocasiões anteriores. Nestas condições, o método *onDestroy* é disparado no momento imediatamente anterior a eliminação do referido objeto que representa um activity qualquer. Após destruída, se a Activity for aberta novamente, ela será recriada e não reinicializada.

Conhecer esse funcionamento é importante para preservar o estado dos objetos envolvidos. Portanto, é dever do programador sobrescrever esses métodos do ciclo de vida para acrescentar ações que devem ser realizadas em determinado estágio dessas possíveis transições, caso ocorram.

Por exemplo, quando a Activity não estiver mais visível, podemos liberar recursos tais como uma conexão de rede, ou ainda, salvar os dados digitados pelo usuário no método *onPause* e encerrar as threads em execução no método *onDestroy*.

03

O código a seguir mostra os possíveis métodos que devemos sobrescrever, em função da relação hierárquica de herança com a classe android.app.Activity, bem como possibilita a execução e consequente visualização dos comportamentos narrados ao longo deste texto para o ciclo de vida de uma *activity*.

```
package com.example.hello;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class CicloVidaActivity extends Activity {
	String msg = "Android by AIEC: ";

	/** Chamado quando a activity é criada pela primeira vez */
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.ciclo_vida);
		Log.d(msg, "Método callback onCreate() executado");
	}

	/** Chamado quando a activity está prestes a tornar-se visível. */
	@Override
	protected void onStart() {
		super.onStart();
		Log.d(msg, "Método callback onStart() executado");
	}

	/** Chamado quando a activity tornou-se visível. */
	@Override
	protected void onResume() {
		super.onResume();
		Log.d(msg, "Método callback onResume() executado");
	}

	/** Chamado quando uma outra activity está tomando foco. */
	@Override
	protected void onPause() {
		super.onPause();
		Log.d(msg, "Método callback onPause() executado");
	}

	/** Chamado quando a activity não é mais visível . */
```

```
        @Override
        protected void onStop() {
                super.onStop();
                Log.d(msg, "Método callback onStop() executado");
        }

        /** Chamado apenas antes da activity ser destruída. */
        @Override
        public void onDestroy() {
                super.onDestroy();
                Log.d(msg, "Método callback onDestroy() executado");
        }
}
```

Lembre-se sempre de **invocar a implementação padrão do método que está sendo sobrescrito**. Por exemplo, se estiver sobrescrevendo o método *onStop*, então invoque antes o método *super.onStop().*

Caso queira recordar um pouco sobre os motivos pelos quais isso deve de fato ser implementado, retorne ao conteúdo sobre "Serialização e Parcelable", uma vez que tais motivos fazem, por exemplo, o uso da classe "android.os.Bundle" cuja finalidade é a de preservar o estado de objetos que compõem uma determinada activity.

04

Além do código fonte java para demonstrar o ciclo de vida de uma activity, é preciso do código de layout abaixo.

Salve o arquivo de layout com o nome "ciclo_vida.xml" na pasta res/layout do seu projeto.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/cicloVidaTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="CICLO DE VIDA DE UMA ACTIVITY"/>
</LinearLayout>
```

05

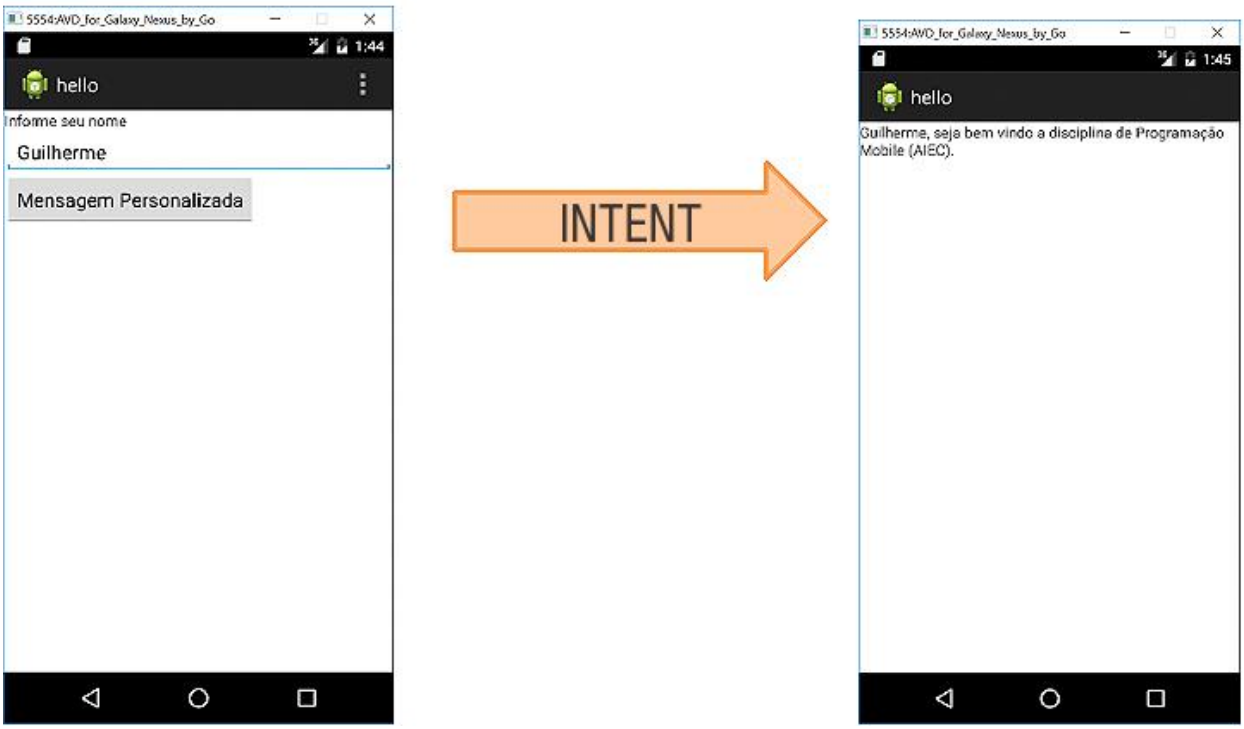
2- HELLO WORLD 3.0

Agora que já sabemos como funciona o ciclo de vida e também como utilizar as *Intents* e os *Intent Filters* apresentados no módulo anterior, vamos tirar proveito disto melhorando o nosso HelloWorld 2.0 para incluir uma *activity* que irá responder a uma *intent* implícita.

A ideia e que na nossa aplicação de exemplo existam duas activities:

- a ***MainActivity***, que continuará sendo utilizada para o usuário informar o seu nome, e
- a ***BoasVindasActivity***, que será responsável apenas por exibir uma mensagem de boas-vindas para o usuário a partir das informações contidas na *intent*.

Ela também possuirá uma categoria própria e responderá a uma ação especifica. A figura abaixo demonstra essa ideia.

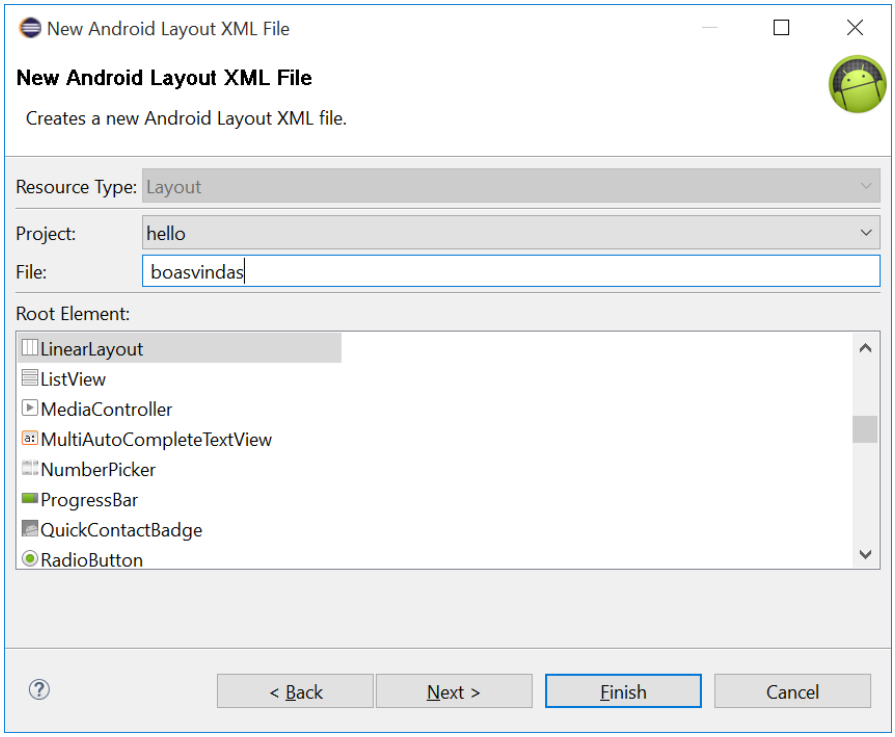


06

Primeiramente, devemos criar um novo XML de layout que será utilizado pela BoasVindasActivity para exibir a mensagem de boas-vindas para o usuário. Para isto, acesse o menu File > New > Android XML Layout File.

Informe o nome do arquivo como “boasvindas” e clique no botão Finish.

Não é necessário alterar nenhuma outra informação, como na figura abaixo:

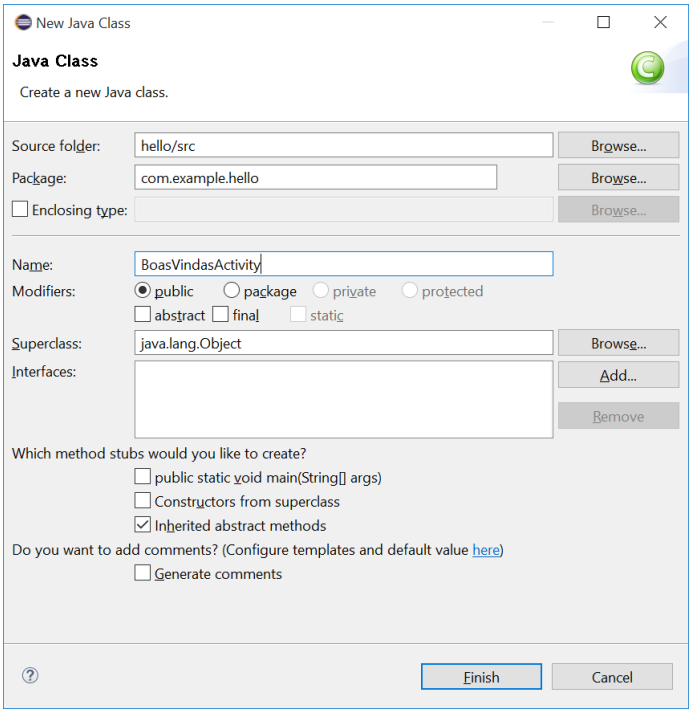


Neste novo layout incluiremos apenas um TextView para mostrar a mensagem de boas-vindas ao usuário. O código ficará assim:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <TextView
      android:id="@+id/boasVindasTextView"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />

</LinearLayout>
```

07

Com o layout pronto, vamos criar uma nova *Activity* para a aplicação, através do menu File > New > Class. Na caixa de diálogo apresentada, selecione o pacote com.example.hello e para o nome da classe informe BoasVindasActivity, conforme a figura abaixo e pressione Finish.



Podemos então criar a nossa BoasVindasActivity, que herdara de Activity conforme abaixo:

```
package com.example.hello;

import android.app.Activity;

public class BoasVindasActivity extends Activity{

        //teremos que implementar o método onCreate
}
```

08

Na implementação do método *onCreate*, teremos a chamada para *super.onCreate* e em seguida precisamos indicar qual o layout será utilizado, que no nosso caso será o layout “boasvindas” já criado anteriormente:

```
package com.example.hello;

import android.app.Activity;
import android.os.Bundle;

public class BoasVindasActivity extends Activity{

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

Podemos definir constantes para identificar o extra que a *intent* possui, pois vamos utilizá-la nos métodos *onCreate*:

```
package com.example.hello;

import android.app.Activity;
import android.os.Bundle;

public class BoasVindasActivity extends Activity{

	public static final String EXTRA_NOME_USUARIO = "helloworld3.EXTRA_NOME_USUARIO";

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

09

Com isso, podemos recuperar a *Intent*, que nos foi passada através do método *getIntent*, e checar se existe um extra com o identificador definido, ou seja, se a *Intent* possui o nome do usuário para a

exibição da mensagem de boas-vindas. Caso exista um extra, obtemos o seu valor utilizando o método *intent.getStringExtra(EXTRA_NOME_USUARIO)*. Se a *intent* fornecida não possui nenhum extra, então apresentamos um aviso para o usuário.

```
package com.example.hello;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class BoasVindasActivity extends Activity {

	public static final String EXTRA_NOME_USUARIO = "helloworld3.EXTRA_NOME_USUARIO";

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

		TextView boasVindasTextView = (TextView) findViewById(R.id.boasVindasTextView);
		Intent intent = getIntent();

		if (intent.hasExtra(EXTRA_NOME_USUARIO)) {
			String boasVindas = getResources().getString(R.string.boas_vindas);
			boasVindasTextView.setText(intent.getStringExtra(EXTRA_NOME_USUARIO)+ ",
" + boasVindas);
		} else {
			boasVindasTextView.setText("O nome do usuário não foi informado");
		}
	}
}
```

10

Em seguida, precisaremos alterar o método mensagemPersonalizada da *MainActivity* para deixar de exibir a mensagem e criar a *Intent* que acionara a nova *activity* (BoasVindasActivity). Com isso, vamos ter que definir também a ação e a categoria da *activity* que acabamos de criar. Podemos fazer isso definindo mais duas constantes na classe BoasVindasActivity conforme o código abaixo:

```
package com.example.hello;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
```

```
public class BoasVindasActivity extends Activity {

	public static final String EXTRA_NOME_USUARIO = "helloworld3.EXTRA_NOME_USUARIO";

	public static final String ACAO_EXIBIR_BOASVINDAS =
"helloworld3.ACAO_EXIBIR_BOASVINDAS";

	public static final String CATEGORIA_BOASVINDAS =
	"helloworld3.CATEGORIA_BOASVINDAS";

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

		TextView boasVindasTextView = (TextView) findViewById(R.id.boasVindasTextView);
		Intent intent = getIntent();

		if (intent.hasExtra(EXTRA_NOME_USUARIO)) {
			String boasVindas = getResources().getString(R.string.boas_vindas);
			boasVindasTextView.setText(intent.getStringExtra(EXTRA_NOME_USUARIO)+ ",
" + boasVindas);
		} else {
			boasVindasTextView.setText("O nome do usuário não foi informado");
		}
	}
}
```

11

No método mensagemPersonalizada da nossa classe *MainActivity*, criamos uma nova *Intent* com a ação desejada e nela adicionamos a categoria definida anteriormente. Em seguida, incluímos como informação extra o valor informado no EditText. E por fim, iniciamos uma nova *activity* passando a *Intent* criada.

```
	public void mensagemPersonalizada(View view) {

		Intent intent = new Intent(BoasVindasActivity.ACAO_EXIBIR_BOASVINDAS);
		intent.addCategory(BoasVindasActivity.CATEGORIA_BOASVINDAS);

		EditText nomeEditText = (EditText) findViewById(R.id.nomeEditText);
		String texto = nomeEditText.getText().toString();

		intent.putExtra(BoasVindasActivity.EXTRA_NOME_USUARIO, texto);

		startActivity(intent);
	}
```

Já que a exibição da mensagem de boas-vindas passou a ser responsabilidade de outra activity, devemos excluir do layout utilizado pela *MainActivity* o *TextView* que tinha esse papel. No arquivo de layout activity_main.xml, remova o último *TextView* declarado, com o id@boasVindasTextView, pois não precisaremos mais dele. As últimas alterações para que nossa BoasVindasActivity possa responder a uma *Intent* serão feitas no arquivo AndroidManifest.xml.

Vamos adicionar um novo bloco de *activity*, declarações nas quais estabelecemos que a *BoasVindasActivity* responde pela ACAO_EXIBIR_USUARIO e também atende a *intents* que pertencem a CATEGORIA_BOASVINDAS:

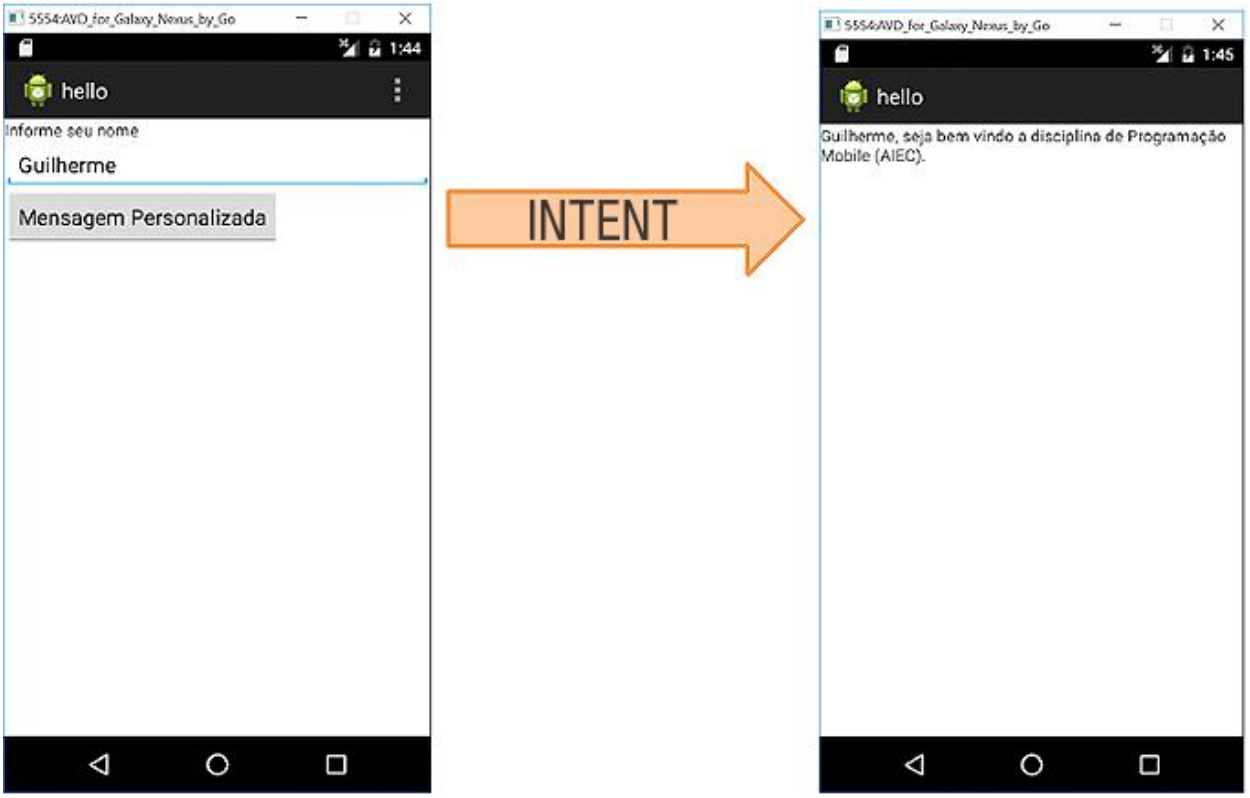
```
        <activity android:name="com.example.hello.BoasVindasActivity" >
      <intent-filter>
        <action android:name="helloworld3.ACAO_EXIBIR_BOASVINDAS" />

        <category android:name="helloworld3.CATEGORIA_BOASVINDAS" />

        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
```

Quando desejamos que uma *activity* receba *intents* implícitas, é obrigatório que no *intent filter* também seja incluída a categoria *android.intent.category.DEFAULT*.

12

Para cada *activity* é possível definir vários *intent filters*, com configurações diferentes de ação e categoria.

Já podemos executar a aplicação para testar! O resultado deve ser o mesmo apresentado na figura abaixo:



13

3 - RESUMO

O objetivo deste módulo foi apresentar o ciclo de vida de uma activity bem como seus métodos de callback: onCreate, onStart, onResume, onPause, onStop, onRestart e onDestroy. Além disso, a unidade apresentou por meio de um exemplo (Hello World 3.0) uma forma de uso de Intents implícitas.