

Esta apostila tem por objetivo mostrar de modo fácil como programar aplicativos para dispositivos móveis que utilizam o sistema operacional Android. Utilizando Eclipse, e através do desenvolvimento de vários programas e exemplos práticos, o leitor conhecerá os principais recursos que envolve o desenvolvimento de aplicações móveis para Android.

# Apostila de programação para Android

Prof. Msc. Regilan Meira Silva  
<http://www.regilan.com.br>  
regilan@hotmail.com.

---

## Sobre este material

Esta apostila tem como objetivo auxiliar os estudantes de escolas técnicas e nível superior, na aprendizagem de programação para aplicativos móveis baseados na Plataforma ANDROID.. Esta apostila não substitui os livros, sua finalidade é criar um roteiro resumido do ensino-aprendizagem realizado em sala de aula.

Este material foi construído a partir de slides elaborados para as minhas aulas no IFBA – Campus Ilhéus e também a partir apostilas, tutoriais, dicas e demais fontes de dados, obtidos a partir de pesquisas em sites de buscas na Internet. Além disto, este material tem como referência a seguinte bibliografia:

Por fim, este material é distribuído de forma gratuita, sendo vedado qualquer tipo de comercialização.

# Sumário

<b>1. INTRODUÇÃO</b>	<b>4</b>
1.1. Conceitos gerais sobre programação para dispositivos móveis	4
1.2. Visão geral da plataforma Android	6
1.3. Instalação e configuração do ambiente de desenvolvimento	10
1.3.1. Configuração do ambiente de desenvolvimento	10
1.3.2. Executando uma aplicação Android	18
<b>2. TIPOS DE LAYOUT E WIDGETS</b>	<b>22</b>
2.1. Tipos de layouts	22
2.2. Widgets	28
2.2.1. TextView	29
2.2.2. ImageView	31
2.2.3. EditText	31
2.2.4. Button	32
2.2.5. Relacionando widgets no código JAVA	33
2.2.6. Checkbox	41
2.2.7. Radiobutton e RadioGroup	44
2.2.8. ToogleButton e Switch	48
2.2.9. Seekbar	50
2.2.10. RatingBar	53
2.2.11. NumberPicker	54
2.2.12. DateTime e TimePicker	56
2.2.13. Spinner	57
2.2.14. ListView	60
<b>3. STRING.XML (INTERNACIONALIZAÇÃO) E LAYOUTS EM DIFERENTES ORIENTAÇÕES</b>	<b>63</b>
3.1. String.xml	63
3.2. Layout com orientação na vertical e horizontal	66
<b>4. TEMAS, ESTILO E GALERIA DE IMAGENS</b>	<b>70</b>
4.1. Temas e estilo	70
4.2. Temas e estilo	73
<b>5. ACTIVITIES, MENUS E ACTION BAR</b>	<b>77</b>
<b>6. PERSISTÊNCIA DE DADOS: SHARED PREFERENCES E INTERNAL STORAGES</b>	<b>85</b>
<b>7. PERSISTÊNCIA DE DADOS: BANCO DE DADOS LOCAL E REMOTO</b>	<b>89</b>
<b>8. RECURSOS DO DISPOSITIVO: CÂMERA, SMS, CHAMADAS E LOCALIZAÇÃO</b>	<b>110</b>

<b>8.1.</b>	<b>Chamadas telefônicas.....</b>	<b>110</b>
<b>8.2.</b>	<b>Envio de SMS .....</b>	<b>110</b>
<b>8.3.</b>	<b>Acesso a câmera .....</b>	<b>112</b>
<b>8.4.</b>	<b>Recursos de localização .....</b>	<b>116</b>

## 1. INTRODUÇÃO

### 1.1. Conceitos gerais sobre programação para dispositivos móveis

Um dispositivo móvel (DM) pode ser definido como um dispositivo eletrônico de pequeno porte com poder de processamento que atualmente, têm incorporado o teclado à tela e utilizado o recurso TouchScreen (sensível ao toque).

Há diferentes tipos de dispositivos móveis existente no mercado: tocadores de mídia digitais, telefones celulares, tablets, PDAs, consoles portáteis de videogame. Em geral as características deste dispositivo são:

- Pequenos em tamanhos
- Memória limitada
- Poder de processamento limitado
- Baixo consumo de energia
- Conectividade limitada
- Tempo curto de inicialização



Figura 1. Dispositivos móveis mais comuns

O desenvolvimento de aplicações e sistemas para dispositivos móveis é toda atividade e processo acerca do desenvolvimento de software para dispositivos móveis (handheld) como computadores de bolso, PDAs, smartphone, telefone celular, console portátil e Ultra Mobile PC combinado com tecnologias como GPS, TV portátil, touch, consoles, navegador de Internet, WAP, leitores de áudio, vídeo e texto, entre outros. Estes aplicativos podem ser instalados durante a fabricação do aparelho, através dos sistemas operacionais de cada dispositivo ou

distribuído através de arquivos de instalação pela web ou não. O desenvolvimento de aplicações para móveis possui particularidades do desenvolvimento tradicional devido as limitações tanto do processamento, tamanho de tela e área de trabalho, e a plataforma de desenvolvimento, já que existem vários sistemas operacionais disponíveis. Um dispositivo móvel está baseado em uma plataforma de desenvolvimento, que podemos hoje em dia classificar em:

- JAVA:
  - SUN Java ME
  - Android
  - RIM Blackberry
- Não JAVA:
  - Windows 8 (C#)
  - Iphone (Objective C)
  - Symbian (C/C++, Python)



*Figura 2. Plataformas de desenvolvimento para aplicativos móveis*

Aplicativos móveis são softwares utilizados para funções específicas em dispositivos móveis como smartphones e tablets. Em geral suas funções se baseiam na transferência de dados cliente-servidor e tem como objetivo tornar móveis as arquiteturas de softwares já existentes. Em vários casos são implantados por razões de negócio, como melhorar a produtividade, aumento de precisão e outras métricas. Além disto, em muitos casos, os aplicativos móveis precisam ser integrados às aplicativos existentes.

Entre os principais motivos de desenvolver aplicações móveis, merece destacar:

- Clientes potenciais em constante movimento;
- Existência vários fabricantes, modelos e funcionalidades;
- Crescimento explosivo de dispositivos móveis: smartphones, tablets, ultrabooks, híbridos;
- Diferentes recursos de multimídia: tela, captura de imagem, armazenamento, processamento, comunicação;
- Redes móveis: maior cobertura e largura de banda;
- Conteúdo: crescimento da mídia digital e mudanças na forma como o conteúdo é produzido;
- Tipos de aplicação: comunicação por voz, navegação pela web, acesso e arquivos de mídia, GPS, jogos , etc.

Alguns mitos da programação para dispositivos móveis:

**Desenvolver aplicações móveis é fácil?** Dificuldades: ergonomia, conectividade, telas de tamanho reduzido, etc.

**Desenvolver aplicações móveis é rápido?** Depende da complexidade, como ocorre no desenvolvimento de qualquer outra aplicação

**Desenvolver aplicações móveis é barato?** Nem o desenvolvimento das aplicações móveis, nem os dispositivos para teste são barato

## 1.2. Visão geral da plataforma Android

O Android é uma plataforma do Google voltada para dispositivos móveis. Em 5 de novembro de 2007, a empresa tornou pública a primeira plataforma Open Source de desenvolvimento para dispositivos moveis baseada na plataforma Java com sistema operacional Linux, na qual foi chamada de Android. Em novembro de 2007, foi formada a Open Handset Alliance, inicialmente formado por 34 empresas, para desenvolver Android, impulsionando a inovação na tecnologia móvel, melhorando a experiência do usuário e reduzindo os custos. Android é usado em Smartphones, dispositivos e-reader, tablets e outros dispositivos móveis.

A plataforma Android tem como principais características:

- Código-fonte aberto e gratuito
- Os aplicativos Android são desenvolvidos com Java
- Programação da interface gráfica é baseada em eventos: toques na tela e pressionamento de tecla
- Os dispositivos Android vêm com vários aplicativos incorporados: telefone, contatos, correio, navegadores, etc.
- Cada nova versão Android recebe um nome de sobremesa, em inglês.

Ninguém sabe ao certo, alguns dizem que as primeiras versões do sistema tinha nomes baseados em robôs de programas de TV, como “Astro”, baseado em Astro Boy e Bender do Futurama. Outros já dizem que seria “Apple Pie” e “Banana Bread”, por seguirem uma ordem alfabética, veja:

”A”Apple Pie

”B”ananá Bread’

”C”upcake

”D”onut

”E”clair

”F”loyo

”G”ingerbread

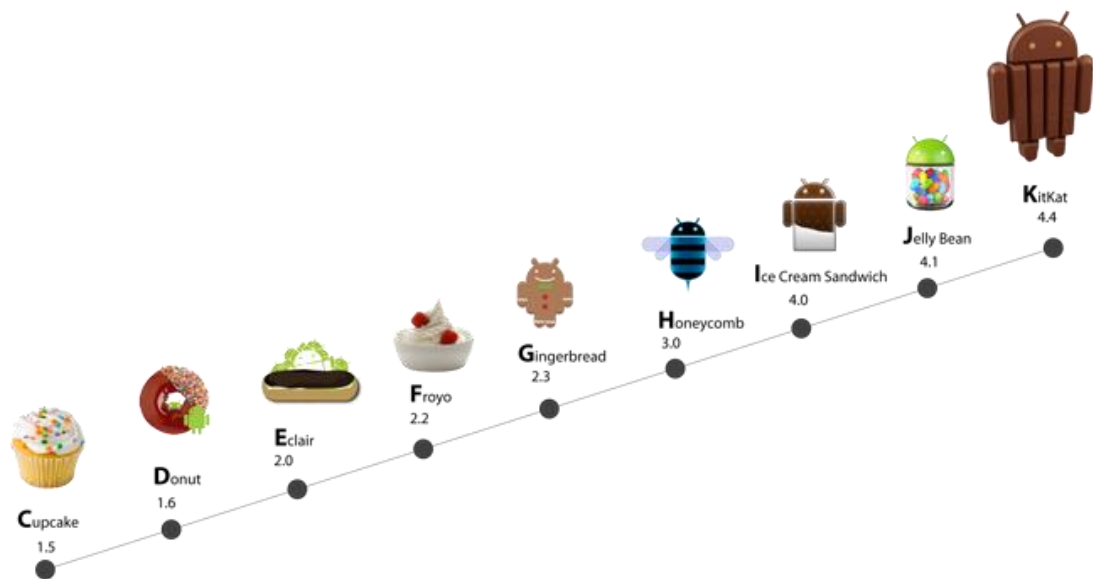
”H”oneycomb

”I”ce Cream Sandwich

”J”elly Bean

”K”it Kat





Até a versão 4.0, as versões do Android se dividiam em 2.x para Smartphones e 3.x: apenas para Tablets. Com o lançamento da versão Ice Cream Sandwich(4.0) passou a ter uma versão única para tablets e smartphones a partir da versão. As versões mais atuais são:

- 4.1 - 4.2 - 4.3: Jelly Bean (Junho de 2012)
- 4.4: KitKat (Versão atual lançada em Outubro de 2013)

De acordo com o Google, em maio de 2014, a distribuição de dispositivos x versões do Android está distribuído da seguinte forma:

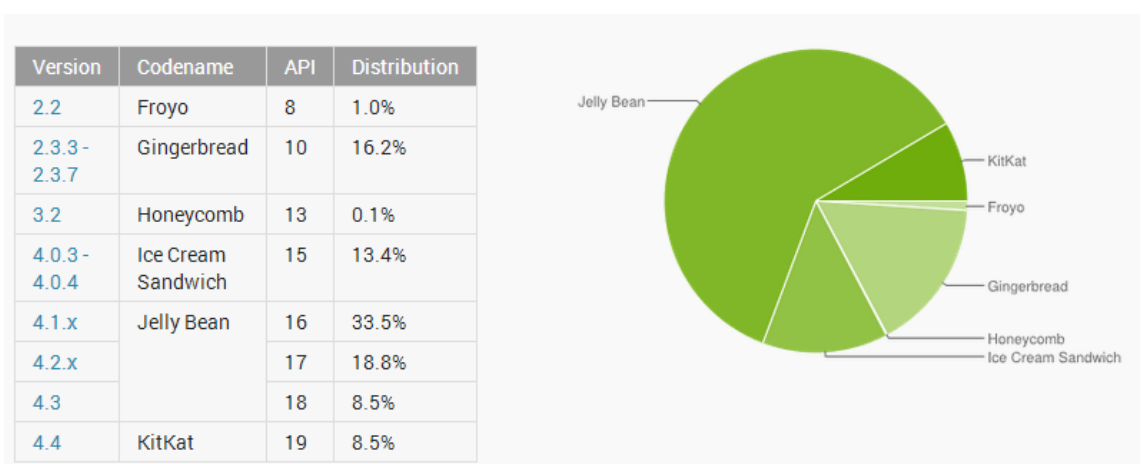


Figura 3. Dispositivos x versões do Android em maio de 2014

A plataforma de desenvolvimento para Android incluir algumas ferramentas, entre as quais o SDK – Software Development Kit que fornece as ferramentas necessárias para construir aplicativos Android. O SDK incluir a IDE Eclipse, um plugin ADT para aplicações Android, o Android SDK, a última versão da plataforma Android e um emulador para testar as aplicações. Existe também uma versão Preview de uma nova ferramenta: Android Studio 0.5.2 for Windows.

O SDK está disponível gratuitamente no site do Google para Android Developers, no seguinte endereço: <http://developer.android.com/sdk/index.html>

A IDE Eclipse, disponível no SDK para desenvolvimento Android é um ambiente de desenvolvimento integrado recomendado para desenvolvimento Android, que possui editor de código, depurador, preenchimento automático, sintaxe colorida e outros recursos.

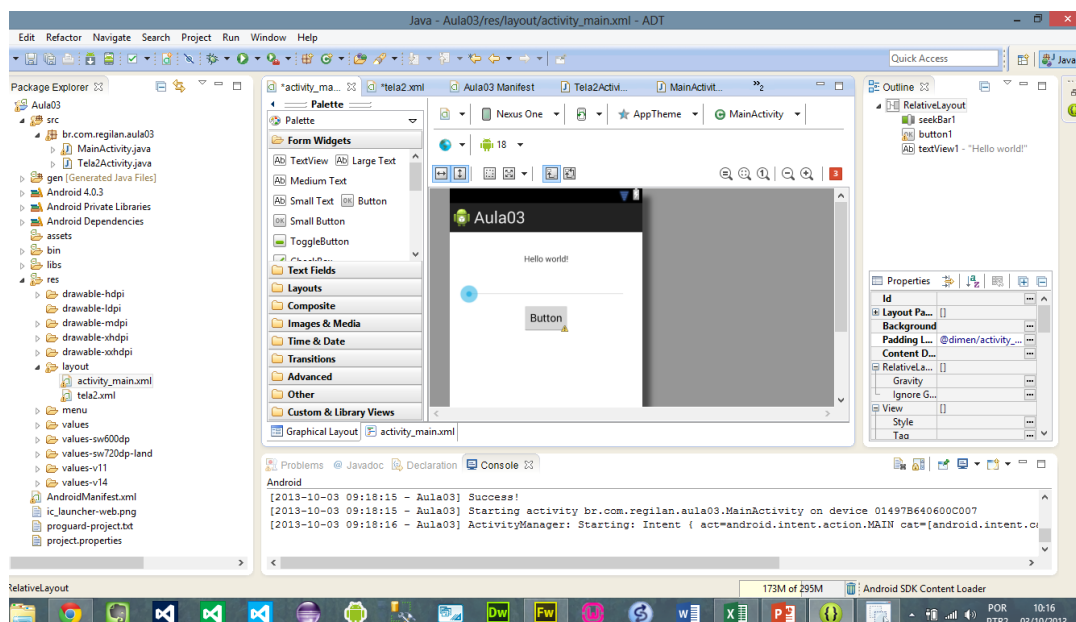
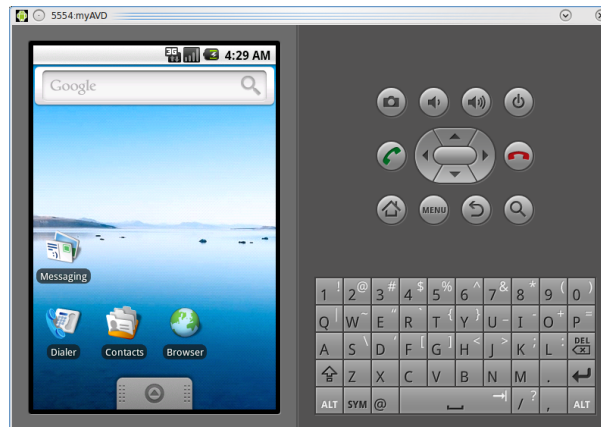


Figura 4. IDE Eclipse para desenvolvimento Android

O Plugin ADT (Android Development Tools) para Eclipse é uma extensão para o IDE Eclipse que permite criar, executar e depurar aplicativos Android, exportá-los para distribuição. O ADT contém uma ferramenta de projeto visual de interface gráfica do usuário, que podem ser arrastados e soltos no lugar para formar interfaces sem nenhuma codificação.

O emulador do Android, incluído no SDK do Android, permite executar aplicativos Android em um ambiente simulado dentro do Windows, Mac OS X ou Linux. O emulador exibe uma janela de interface de usuário, porém antes de executar o aplicativo no emulador, você

precisa criar um AVD (Android Virtual Device), o qual define as características do dispositivo, incluindo hardware, tamanho da tela, armazenamento, etc.



*Figura 5. Emulador Android*

A distribuição dos aplicativos desenvolvidos para Android é centralizado na loja virtual Google Play que é a loja online mantida pela Google para distribuição de aplicações, jogos, filmes, música e livros. No início a loja chamava-se Android Market. As aplicações do Google Play estão disponíveis de graça ou a um custo, e podem ser baixados diretamente para um dispositivo Android. Para distribuir seus produtos por meio do Google Play há uma taxa de registro de US\$ 25 cobrados por uma conta de Console do desenvolvedor do Google Play. Em caso de aplicativos pagos, o Google fica com 30% do preço de venda e repassa 70% ao desenvolvedor.

### **1.3. Instalação e configuração do ambiente de desenvolvimento**

#### **1.3.1. Configuração do ambiente de desenvolvimento**

Para iniciar a configuração do ambiente de desenvolvimento, devemos fazer o download da versão mais atual do SDK no endereço: <http://developer.android.com/sdk/index.html>

Com um único download obtemos:

- Eclipse + ADT plugin
- Android SDK Tools

- Android Platform-tools
- Última versão da plataforma Android
- Emulador para testar as aplicações

O processo de instalação é bem simples, bastando apenas descompactar o arquivo baixado e termos duas ferramentas importantes:

- Eclipse: IDE de desenvolvimento
- SDK Manager: Gerenciador do kit de desenvolvimento

Após a conclusão do download e descompactação do arquivo recomendamos executar o SDK Manager para baixar uma SDK para começarmos a programar (caso não deseje programar para a última versão do Android). Ao executar pela primeira vez, o SDK Manager irá verificar os repositórios do Android em busca das últimas versões do SDK.

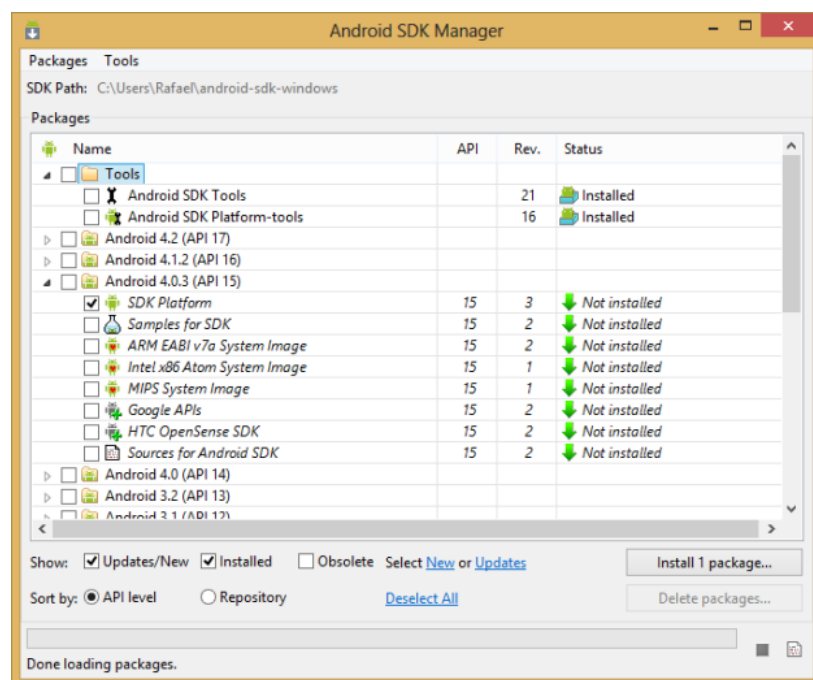


Figura 6. Android SDK Manager

Nos exemplos desta apostila usaremos versões superiores ou igual a 4.0.3 (API 15). Expanda a pasta Android 4.0.3 (API 15) e marque todas as opções, marque Accept All e então clique em Install. O download e configuração será feito automaticamente.

Após instalação, iremos configurar um AVD – Android Virtual Device. Para isto clique no menu Tools -> Manage AVD.

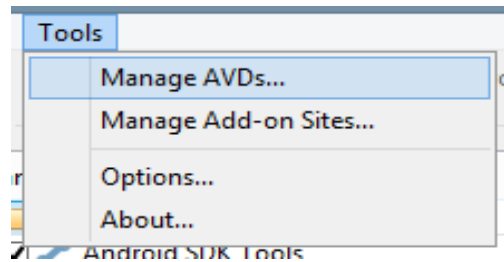


Figura 7. Gerenciador de ambiente virtual – AVD

Na janela que foi aberta, clique no botão NEW para configurar um novo AVD>

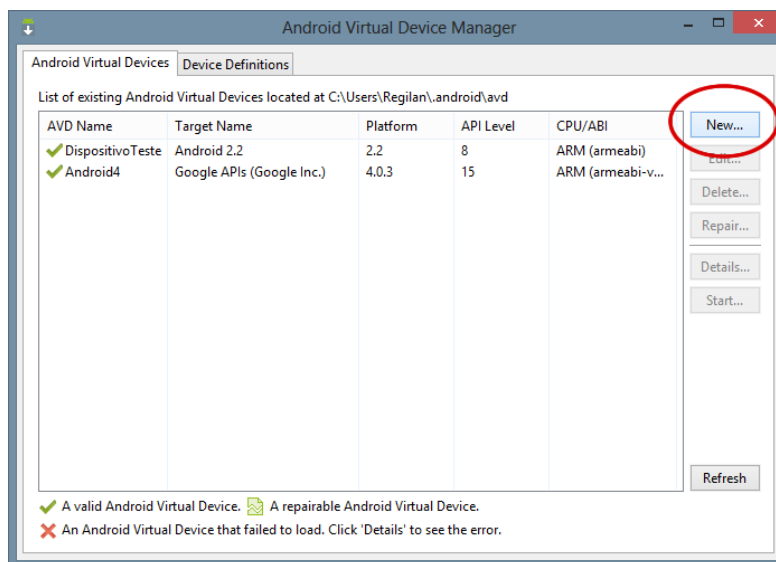


Figura 8. Adicionando um novo AVD

Em seguida, configure um nome para seu ADV (AVD Name). Durante a configuração defina um dispositivo (Device) e uma versão do Android (Target). Neste curso usaremos versões iguais ou superior a 4.0.3(API 15).

Note a opção de criar um SDCard. Este recurso serve para você salvar informações no emulador. Como Configurações, arquivos, aplicativos instalado e etc. Você deve informar um valor em MB por exemplo. EX: 50 MB



**OBS: API level é um valor inteiro que identifica uma versão do Android. Recomendamos escolher sempre a versão Google API, pois estas incluem recursos disponíveis da plataforma Google.**

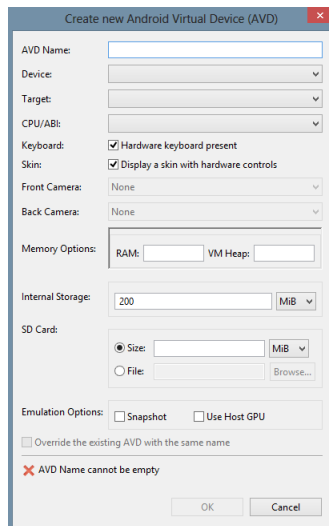


Figura 9. Configuração do AVD

Devido ao tempo consumido para iniciar um AVD, sempre que o mesmo for usado para testes recomendamos inicia-lo e somente fecha-lo após o termo da programação do aplicativo. Caso seja encerrado, uma nova inicialização será realizada.

Para executar um AVD devemos escolher um dispositivo configurado e clicar no botão Start.

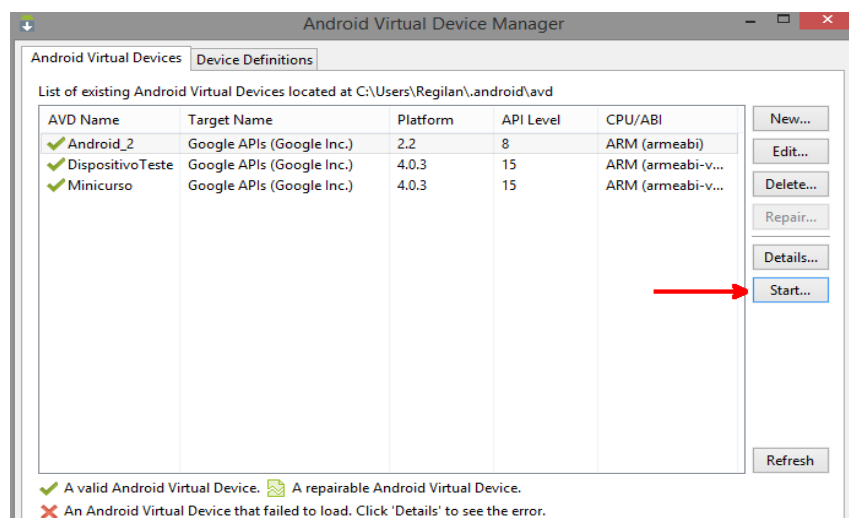
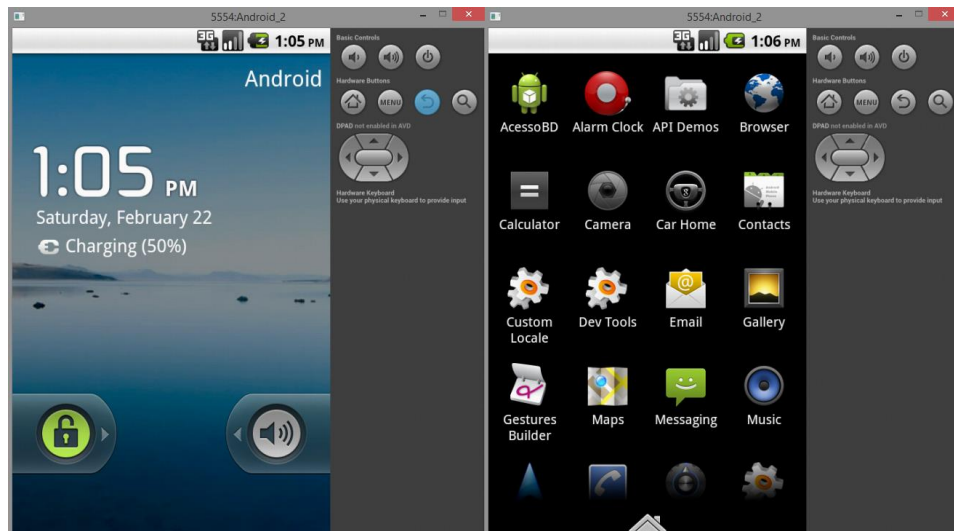


Figura 10. Iniciando um AVD

Após este processo será carregado uma janela com um dispositivo ANDROID com as opções da plataforma escolhida. Lembre-se, como se trata de um dispositivo virtual alguns periféricos estão indisponíveis como GPS, câmera, etc.



*Figura 11. AVD em execução*

Após a conclusão dos downloads e configuração do AVD, iniciaremos o Eclipse.



*Figura 12. Janela de inicialização do Eclipse – ADT*

Ao ser consultado sobre qual workspace (workspace é o local onde seus projetos serão salvos) utilizar, basta definir um local e utilizá-lo como padrão.

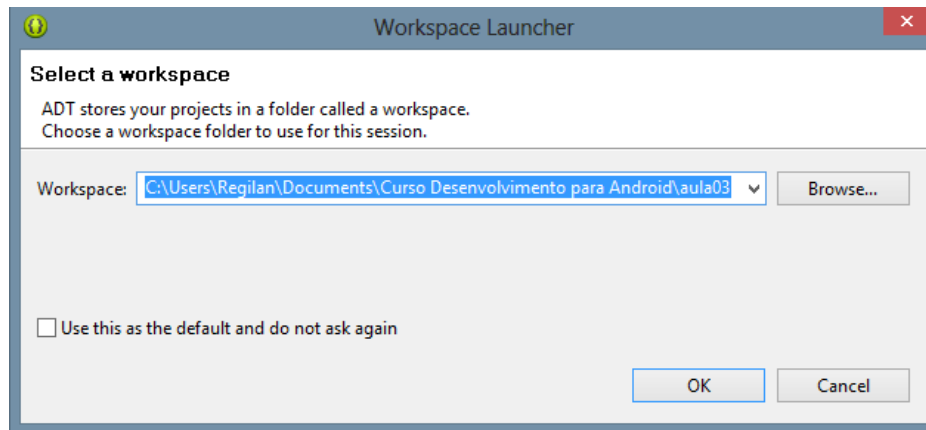


Figura 13. Janela de seleção de Workspace

Para criar um novo projeto para Android, clique no menu: FILE -> NEW -> ANDROID APPLICATION PROJECT

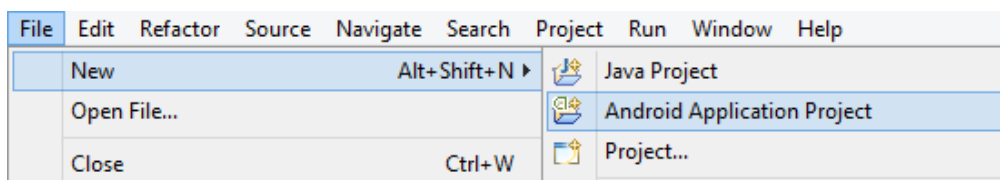


Figura 14. Novo projeto de aplicação para Android

Na janela que será carregada após clicar em Android Application Project deverá ser informado:

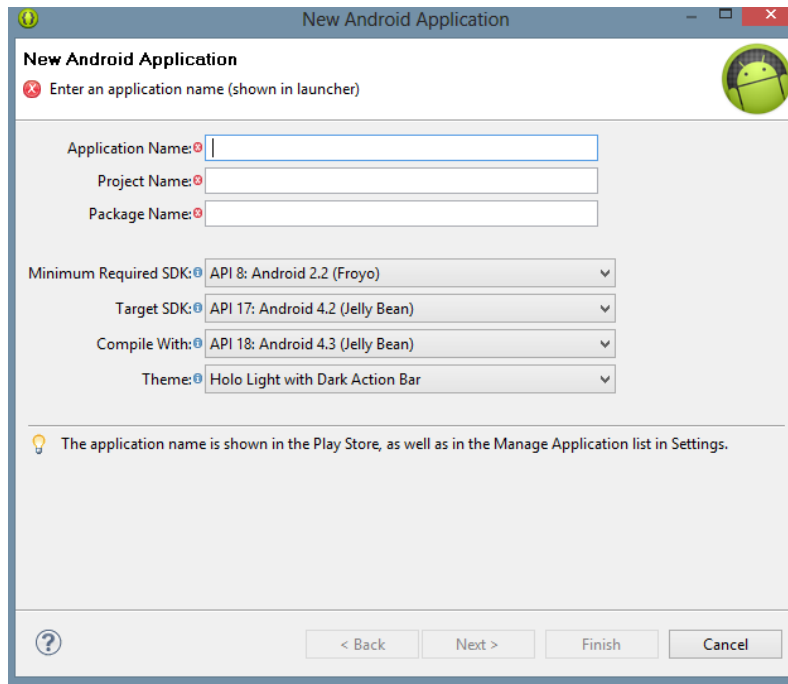
- Nome da aplicação (Iniciando em maiúsculo)
- Nome do projeto
- Nome do pacote (em geral no formato br.com.seudominio.nomeDaAplicacao)
- Em seguida configure as seguintes definições:
- SDK Mínimo
- Versão do Android (Target SDK)
- Versão da compilação (Compile With)

Clique em FINISH para iniciar as definições de configuração da aplicação.



**OBS: Durante esta etapa será necessário criar uma Activity (TELA). Configure de acordo com o especificado.**





*Figura 15. Configuração da nova aplicação para Android*

Quando criamos um novo projeto Android, temos uma estrutura de pastas para aproveitar o uso de recursos facilmente para a aplicação. Dentro de Eclipse na perspectiva Java, você deve ver seu projeto apresentado no painel Package Explorer no lado esquerdo da tela. Uma série de pastas são criadas automaticamente para dar funcionalidade a aplicação. Cada diretório tem um significado específico.

Recomendamos não APAGAR as pastas e arquivos criados, em especial a pasta (gen), que é atualizado automaticamente a cada novo recurso utilizado no aplicativo.

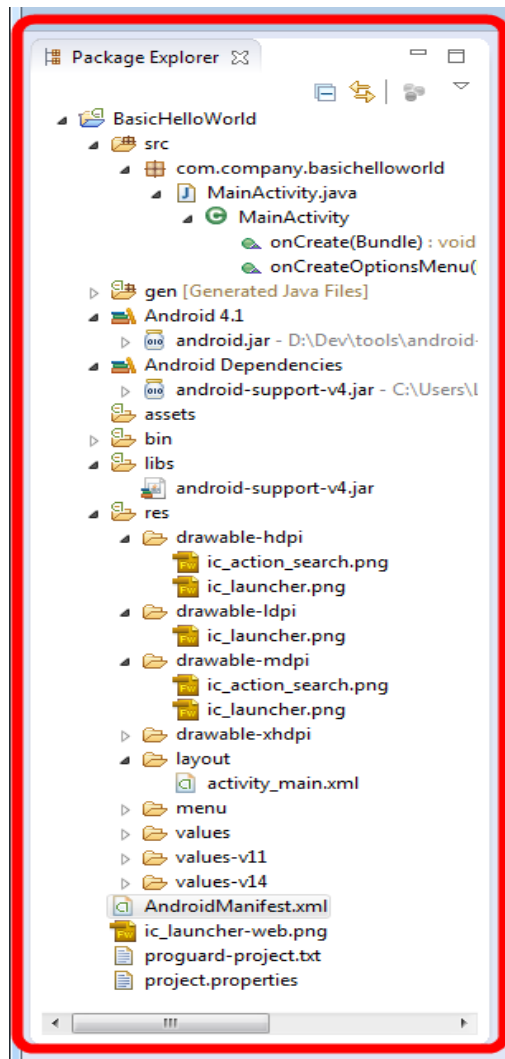


Figura 16. Package Explorer

O diretório **/src** contém os arquivos de origem Java associados ao seu projeto. Por exemplo, a classe Activity chama a MainActivity.java que é armazenado neste diretório com o nome do pacote especificado no assistente de projeto Android. Esta classe MainActivity fornece todo o código da aplicação associados com o aplicativo criado.

O diretório **/gen** contém os arquivos de origem Java e outros arquivos de código gerado pelo Eclipse, que estão associados com o seu projeto. Não edite estes arquivos diretamente. Por exemplo, o arquivo R.java é um arquivo gerado para vincular seus arquivos de recurso (como definido na estrutura do diretório **/res**) para uso em seus arquivos **/src** Java. Estes arquivos são recriados sempre que você adicionar recursos ao seu projeto ou recompilar seu projeto.

O diretório **/bin** contém os arquivos de aplicativos resultantes de pacotes associados com o seu projeto uma vez que foi construído. Arquivos do pacote, ou apks, são o produto que você realmente instala em um dispositivo Android.

O diretório **/res** contém os arquivos de recursos associados ao seu projeto. Todos os gráficos, valores, layouts e outros arquivos de recursos são armazenados na hierarquia de arquivo de recurso no diretório **/res**.

Diferentes tipos de recursos são armazenados em diretórios diferentes. Por exemplo, os gráficos são armazenados sob a tag diretório **/drawable**, enquanto valores e outras primitivas são armazenados sob a tag diretório **/values**. Recursos de interface do usuário são armazenados no diretório **/layout**. Tags especiais muitas vezes incluem alguma nomenclatura para organizar os recursos ainda por tipo de tela, versão do Android, e outros detalhes do dispositivo

O arquivo **AndroidManifest.xml** é um arquivo de configuração muito importante pois contém as especificações sobre a sua aplicação, desde o nome do aplicativo, ícone para acesso a aplicação e as permissões que seu aplicativo precisa para funcionar, entre muitos outros detalhes.

O arquivo **proguard-PROJECT.TXT** é gerado pelo assistente de projeto Android. Este arquivo é usado para configurar as definições Proguard associados ao seu projeto. ProGuard é uma ferramenta que pode ser usada para ajudar a proteger o código de pirataria de software usando ofuscação e otimização de código.

O arquivo **project.properties** é gerado pelo assistente de projeto Android. Este arquivo é usado para configurar as definições do projeto Eclipse. Desenvolvedores raramente, ou nunca, precisam editar este arquivo diretamente. Em vez disso, clique direito sobre o projeto no Eclipse, escolha propriedades, e faça as alterações necessárias usando a interface do usuário do Eclipse.

### **1.3.2. Executando uma aplicação Android**

Um projeto Android pode ser executado sob 2 formas:

- Utilizando um dispositivo físico
- Utilizando um AVD

Em caso de dispositivo físico, o mesmo deve estar conectado em uma porta USB e ter seus drives configurados. Ao executar o aplicativo, o Eclipse reconhecerá o dispositivo e executará.

Em caso de AVD, recomendamos antes de rodar a aplicação que o AVD seja iniciado através do AVD Manager, pois este processo é lento. OBS: UMA VEZ INICIALIZADO, NÃO FECHAR O AVD.

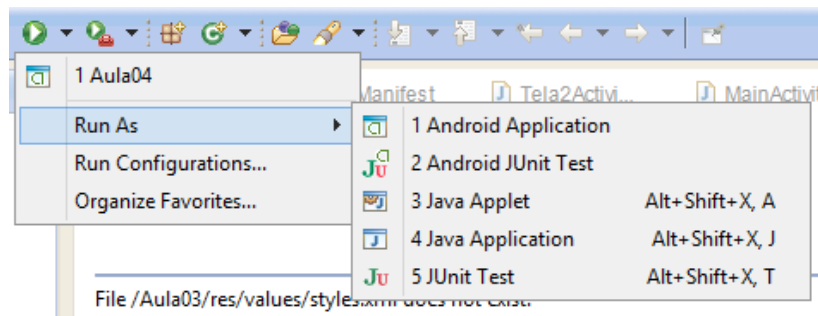


Figura 17. Executando uma aplicação Android

Recomendamos que em caso de execução via AVD o dispositivo seja iniciado (SDK Manager -> Tools -> Manager AVDs -> Start) antes de rodar aplicação devido ao tempo de inicialização. Uma vez iniciado é recomendável fechar o dispositivo somente quando os testes forem encerrados.

O Eclipse automaticamente instalará e executará o APP no dispositivo virtual.



**OBS: Através da opção RUN As -> RUN Configuration, podemos definir como o Eclipse executará o aplicativo, onde podemos definir 3 opções:**

- Sempre mostrar uma janela para escolher o dispositivo que receberá o APP
- Definir se o APP será executado por um dispositivo físico ou virtual
- Definir um dispositivo virtual padrão

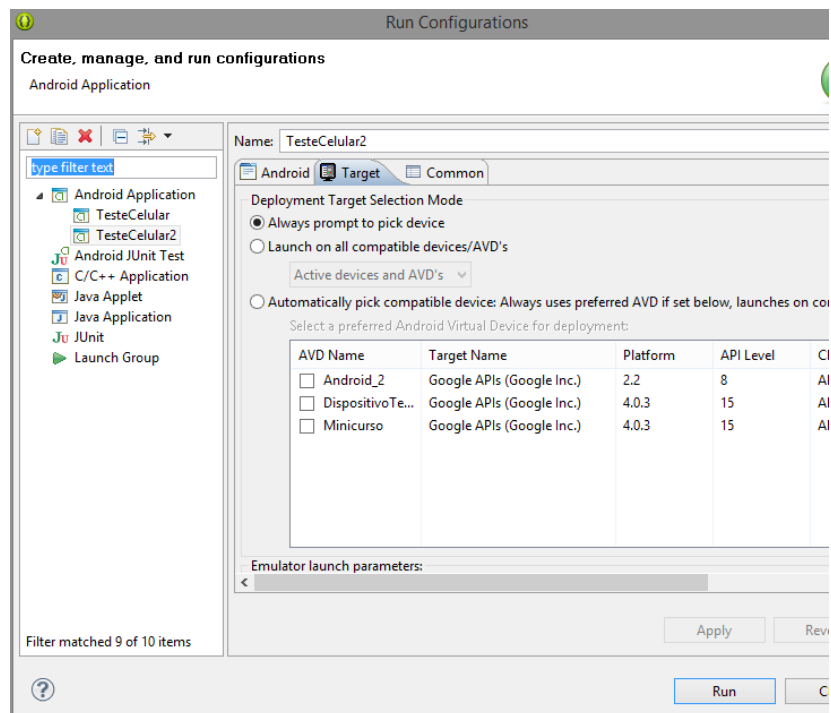


Figura 18. Configuração para execução de aplicação Android

Para a execução de aplicações Android em um dispositivo físico devemos seguir alguns passos:

**1º Passo:** Verificar na Configuração do aparelho se o modo Depuração USB esta ativo, caso não esteja ative-o.

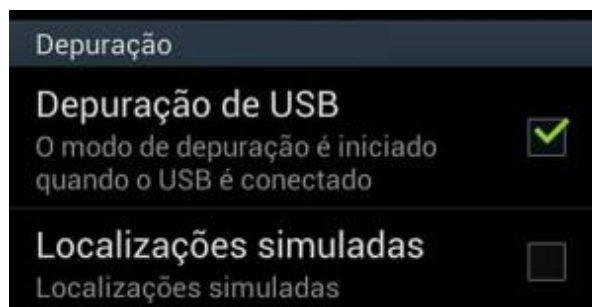


Figura 19. Depuração USB

**2º Passo:** Conectar o seu celular/Tablet no computador via USB.

**3º Passo:** Rodar sua aplicação (Run) e verificar se o seu dispositivo foi detectado. Caso não tenha sido detectado, geralmente é necessário instalar os drivers do aparelho.

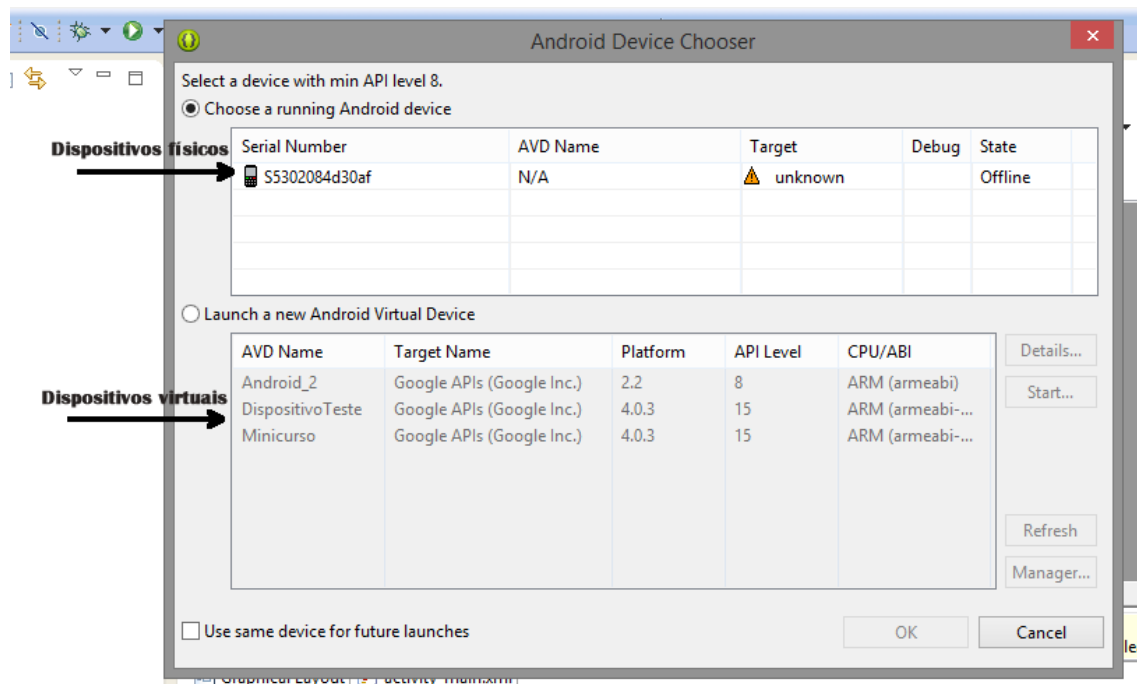


Figura 20. Escolha do dispositivo que executará uma aplicação Android

## 2. TIPOS DE LAYOUT E WIDGETS

### 2.1. Tipos de layouts

Na programação para Android existem diversos tipos de componentes gráficos também chamados de Widgets:

- **TextView**: Mostra um texto na tela. É com certeza o componente gráfico mais usado em Android.
- **EditTex**: Os componentes acima apenas mostram informações na tela. Já o EditText obtém um texto digitado pelo usuário, que poderá ser usado para interagir com a aplicação Android.
- **Button**: Este componente é um dos mais comuns em qualquer sistema de layout. Neste componente, uma ação é executada após um clique ser dado nele.
- **Outros**: Checkbox, RadioGroup, ListView, GridView, Spinner, SeekBart, etc.

Para que estes estejam organizados e apareçam na tela do dispositivo eles precisam estarem inseridos em um Layout. O diretório layout armazena todas os layouts da aplicação Android, que normalmente são arquivos “.xml”. Para quem conhece a combinação HTML + CSS + JavaScript, o Android é similar, é a combinação de XML + Java, logo todos os nosso componentes vão ser adicionados usando tags XML. Por padrão, o arquivo de layout é o activity\_main.xml. Veja exemplo no código abaixo:

```
<LinearLayout ← tags XML
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:orientation="vertical" >

    <TextView ← tags XML
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bem Vindo" />
    } definição das propriedades dos elementos

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Clique aqui"
        android:id="@+id/mostrar"
    />

</LinearLayout> ← tags XML
```

Figura 21. Exemplo de arquivo de layout XML

Um layout define a estrutura visual e organiza os componentes gráficos para compor a interface do usuário. Uma Layout funciona com uma espécie de container para os elementos que irão compor a interface gráfica do aplicativo. Toda aplicação deve iniciar um elemento de Layout. Existem diversos tipos de layout em Android. Em geral 3 tipos de layouts são os mais utilizados:

- LinearLayout
- RelativeLayout
- TableLayout

O **LinearLayout** é utilizado para dispor seus componentes em uma única direção (horizontal ou vertical). Para configurar um layout ou qualquer componente gráfico devemos especificar os valores das diferentes propriedades de cada elemento, que pode ser configurada de forma visual ou via código. Para configurar as propriedades de forma visual, clicamos no elemento visual a ser configurado (layout ou componente gráfico) e realizamos as alterações através da janela PROPERTIES.

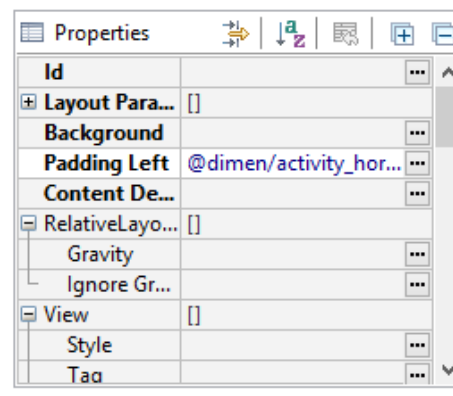


Figura 22. Janela de Propriedades

Para manipular as propriedades via código XML, clicamos na opção com o nome da activity.xml. Para visualizar de forma gráfica clicamos em Graphical Layout.

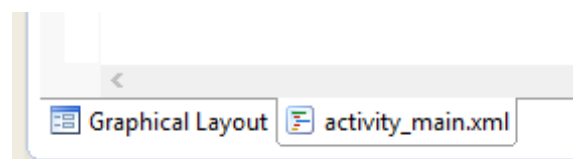


Figura 23. Modo gráfico e modo XML





**OBS: Não são todas as propriedades que podem ser configuradas visualmente. Algumas só podem ser definidas via código.**

Para definirmos um `LinearLayout` usamos a tag abaixo e em seguida devemos configurar cada propriedade deste elemento, que fica especificada antes de fechar a Tag de abertura do elemento.

```
<LinearLayout> </LinarLayout>
```

Na construção de layouts e dos elementos gráficos que irá compor a interface com usuários, sempre especificamos as propriedades de um elemento usando o prefixo “android:” em seguida um pós-fixado que define qual propriedade será manipulada.

**Exemplo:** `android:layout_width="match_parent"`

Este parâmetro define o tamanho da largura da tela que estamos criando. Este tamanho pode ser um tamanho fixo (em pixels, density pixels, ou outras unidades de formatação) ou em tamanhos expansíveis.

Existem 3 tamanhos expansíveis em Android:

- **fill\_parent:** Com esta opção, o tamanho do parâmetro será máximo (ou seja, o tamanho da tela corrente)
- **wrap\_content:** Com esta opção, o tamanho do parâmetro será mínimo, tendo como base os componentes-filhos do layout atual.
- **match\_parent:** Mantém o tamanho herdado pelo componente-pai. Caso não haja um componente-pai, o tamanho será máximo (ou seja, o tamanho da tela).

Outras propriedades relacionadas ao elemento Layout:

- `android:layout_height="match_parent"`: Este parâmetro define a altura do elemento que estamos criando.
- `android:layout_width="match_parent"`: Este parâmetro define a largura do elemento que estamos criando.

- `android:gravity="center_horizontal"`: Este parâmetro define o alinhamento que os componentes deste layout terão. Neste caso, o alinhamento será central e horizontal.
- `android:text="Exemplo"`: Este parâmetro define o texto do componente atual.
- `android:id="@+id/linearDados"`: Este parâmetro define um identificador ao elemento gráfico. Caso seja necessário manipular este elemento via código Java, usamos este identificador para acessá-lo
- `android:background="#A5B6C7"`: Este parâmetro define um background ao Layout que pode ser uma cor em formato hexadecimal ou uma imagem
- `android:orientation="vertical"`: Este parâmetro define a orientação dos elementos na tela, que pode ser na vertical ou horizontal.
- `android:padding="10dp"`: Define o espaçamento dos componentes gráficos (TextView, EditText, etc) em relação ao layout
- `android:margin="10dp"`: Define a margem do elemento gráfico em relação ao elemento que o contém.



**OBS:** as propriedades `padding` e `margin` podem ser configuradas separadamente usando as direções: `Top`, `Bottom`, `Left`, `Right`. Quando estas propriedades não estão configuradas separadamente, o valor definido é aplicado em todas as direções.

Um exemplo de um `LinearLayout` é apresentado através do código XML a seguir:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:orientation="vertical" >
</LinearLayout>
```

**Relative Layout** é um layout do tipo relativo, ou seja, ao contrário do `Linear Layout`, que especifica sempre uma direção horizontal ou vertical, no `RelativeLayout` posicionamos os

elementos por referência à outros elementos. Por exemplo, dizemos se o botão estará abaixo de um campo de texto, do **RelativeLayout** usamos a tag abaixo e em seguida devemos configurar cada propriedade deste elemento, que fica especificada antes de fechar a Tag de abertura do elemento.

```
<RelativeLayout></RelativeLayout>
```

Para definir o posicionamento dos elementos na tela em um RelativeLayout devemos configurar as propriedades:

- `android:layout_below="@id/label"`: define que o elemento gráfico está abaixo de um outro elemento definido pelo parâmetro `@id`
- `android:layout_above="@id/label"`: define que o elemento gráfico está acima de um outro elemento definido pelo parâmetro `@id`
- `android:layout_toRightOf="@id/ok"`: define que o elemento gráfico está a direita de um outro elemento definido pelo parâmetro `@id`
- `android:layout_toLeftOf="@id/ok"`: define que o elemento gráfico está a esquerda de um outro elemento definido pelo parâmetro `@id`

Um exemplo de um RelativeLayout é apresentado no código a seguir:

```
<RelativeLayout
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"
android:id="@+id/linearDados"
android:background="#A5B6C7"
tools:context=".MainActivity" >
</RelativeLayout>
```



**OBS:** As propriedades `android:layout_height` e `android:layout_width` definem respectivamente a altura e largura de uma elemento. Estas propriedades precisam estar definidas para que o elemento fique visível. Todos os elementos gráficos possuem estas propriedades.

**TableLayout** comumente é usado quando precisamos listar vários componentes em uma mesma linha, ao longo de uma mesma tela, no formato de uma tabela. Por exemplo, criar um layout com 18 TextView's divididas 3 a 3, ao longo de 6 linhas. Para criar as linhas em um **TableLayout** usamos o componente **TableRow**.

O número de colunas no componente TableLayout é definido pelo objeto TableRow que contém a maioria dos componentes. A altura de cada linha é determinada pelo componente mais alto dessa linha. Da mesma forma, a largura de uma coluna é definida pelo elemento mais largo nessa coluna – a não ser que configuramos para que as colunas da tabela se alonguem para preencher a largura da tela.

Por padrão, os componentes são adicionados da esquerda para direita em uma linha, mas podemos especificar o local exato de um componente.



**OBS: por padrão as linhas e colunas são numeradas a partir de 0. Uma propriedade importante é android:stretchColumns="0,1" que indica que as colunas devem ser alongadas horizontalmente,**

Para definirmos um TableLayout usamos a tag apresentada abaixo.

```
<TableLayout> </TableLayout>
```

Um exemplo de um TableLayout é apresentado no código a seguir:

```
<TableLayout
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"
android:id="@+id/linearDados"
android:background="#A5B6C7"
tools:context=".MainActivity" >

    <TableRow>
    </TableRow>

    <TableRow>
    </TableRow>

</TableLayout>
```

Duas propriedades importantes que podem ser configuradas aos elementos inseridos em um `TableLayout` são:

- `android:layout_column="1"`: Por padrão, ao se adicionar elementos em um componente `TableRow`, o primeiro deles é colocado na primeira coluna, o segundo é colocado na segunda coluna e assim por diante. Para começar em uma coluna diferente, precisamos especificar o número da coluna. Neste exemplo, estamos especificando que o elemento está na coluna nº 1 (ou seja segunda coluna)
- `android:layout_span="2"`. Por padrão cada elemento gráfico é inserido em uma coluna. Caso seja necessário que este elemento ocupe mais de uma coluna, devemos especificar quantas colunas este elemento ocupa (limitado ao número de colunas existente na tabela). Neste exemplo, o elemento ocuparia 2 colunas.

## **2.2. Widgets**

Widgets podem ser definidos como os componentes que compõem uma aplicação Android, como o `TextView`, um `Button`, um `EditText`, um `RadioButton` e etc. Um Widget pode ser adicionado arrastando o componente a janela referente a interface gráfica da aplicação ou através das TAGs XML de cada elemento. Os Widgets disponíveis para uma aplicação Android ficam localizadas na Paleta.

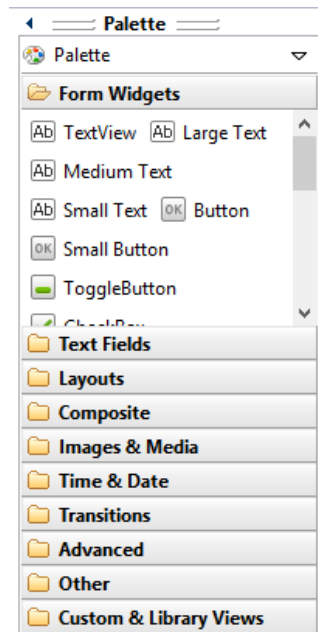


Figura 24. Paleta de widgets

### 2.2.1. TextView

O Widget TextView é utilizado para apresentar um texto não editável na tela. Qualquer componente gráfico pode ser adicionado arrastando da paleta até a tela gráfica ou criando o código XML deste elemento. O código XML que representa um TextView pode ser representado:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Seu texto aqui"
/>
```

Algumas propriedades importantes de uma TextView são

- `android:text="Texto 1"` : Este parâmetro define o texto que é exibido na TextView
- `android:id="@+id/tvNome "` : Este parâmetro define um identificador textView. Caso seja necessário manipular este elemento via código Java, usamos este identificador para acessá-lo

- `android:layout_height="match_parent"`: Este parâmetro define a altura do elemento que estamos criando.
- `android:layout_width="match_parent"`: Este parâmetro define a largura do elemento que estamos criando.
- `android:gravity="center_horizontal"`: Este parâmetro define o alinhamento que os componentes deste layout terão. Neste caso, o alinhamento será central e horizontal.
- `android:textColor="#A5B6C7"`: Este parâmetro define uma cor ao texto exibido. A cor definida deve estar em formato hexadecimal.
- `android:textSize="20dp"`: Este parâmetro define o tamanho do texto.
- `android:textStyle="bold"`: Define o estilo do texto (negrito, itálico ou normal)
- `android:textAllCaps="true"`: Define se o texto exibido aparecerá em caixa alta (true) ou em caixa baixa (false)
- `android:layout_gravity="center_horizontal"`: Define o alinhamento do texto
- `android:typeface="serif"`: Define os padrões de fonte, ou famílias, que no caso do Android são 3 famílias: Droid Sans, Droid Sans Mono e Droid Serif



**OBS:** De acordo com a documentação oficial de programação para Android é considerado uma boa prática “externalizar” strings, arrays de string, imagens, cores, e outros recursos que você ou outra pessoa possa gerenciá-los separadamente do código de seu aplicativo.

Para isto, adicione uma nova String em: **res -> values -> strings**

```
<string name="nome">Nome</string>
```

Em seguida configure o parâmetro `android:text` conforme abaixo

```
android:text="@string/nome"
```

Adicionando sombras:

- `android:shadowColor`: cor da sombra
- `android:shadowRadius`: o raio da sombra
- `android:shadowDx`: o distanciamento horizontal da sombra em relação ao texto
- `android:shadowDy`: o distanciamento vertical da sombra em relação ao texto

### 2.2.2. ImageView

O Widget `ImageView` é usado para adicionar uma imagem em uma activity (tela). Os parâmetros `id`, `gravity`, e outras propriedades comuns a todos os widgets são configurados da mesma forma aos já apresentados.

Para definir a imagem a ser exibida usamos a propriedade `src` conforme mostrado a seguir:

```
android:src="@drawable/ic_launcher"
```



**OBS:** Antes de utilizar uma imagem, é necessário colocá-la na pasta de imagem (`@drawable`). Para isto copie e cole a imagem na pasta específica: `res -> drawable`.

Um código que representa uma `ImageView` é apresentado a seguir:

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="86dp"  
    android:src="@drawable/ferrari" />
```

### 2.2.3. EditText

Um `EditText` é um componente gráfico que permite ao usuário interagir com o aplicativo através da inserção de textos. Quando o usuário do aplicativo tocar em um `EditText`, automaticamente será exibido o teclado virtual para que uma informação seja passada.



Na paleta de Widgets é possível incluir EditText com entradas pré-configuradas para permitir apenas números, campos no formato senha (password), etc.

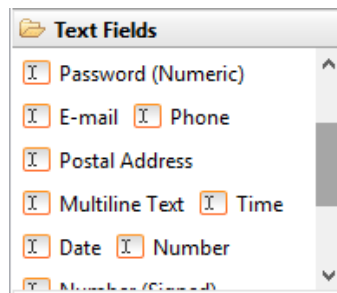


Figura 25. Tipos de EditText

Uma EditText também poderá ser adicionada via código XML, conforme abaixo:

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</EditText>
```

Os parâmetros id, layout\_width, layout\_height, gravity, e outras propriedades comuns a todos os widgets são configurados da mesma forma aos já apresentados.

Propriedades importantes:

- `android:hint=""`: este atributo exibe uma dica dentro de um componente EditText, a qual ajuda o usuário a entender o objetivo do componente. Quando o usuário iniciar a digitação neste componente, a dica de ajuda desaparece.
- `android:inputType="textPassword"`: configura o padrão de entrada de valores em uma EditText

#### 2.2.4. Button

Um Button é um componente gráfico que permite ao usuário interagir com o aplicativo através de cliques (toques) no botão.

Em geral os botões acompanham código JAVA que é acionado para realizar uma determinada função assim que o usuário do aplicativo toca-lo. Apesar de não ser o

recomendado podemos usar para isto a propriedade **onClick** para chamar uma função no código JAVA a qual o formulário está relacionado. Veremos mais a frente que todo evento de um componente deve ser realizado através da definição de **Listeners**.

As propriedades id, text, background, margin e outras propriedades comuns a todos os widgets são configuradas da mesma forma que os controles já apresentados

O código XML de um botão pode ser definido conforme a codificação a seguir:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="OK"
    android:onClick="cadastrar"
/>
```

Podemos ainda usar a tag **ImageButton** para definir um botão com uma imagem. Para isto usamos a propriedade **android:src**. A codificação de uma **ImageButton** é apresentada a seguir:

```
<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/botao_ok"
/>
```

### 2.2.5. Relacionando widgets no código JAVA

A programação de aplicações para Android é baseada na linguagem Java, por para manipular os componentes da interface gráfica que adicionamos via XML é necessário um código JAVA que esteja relacionado com a interface gráfica do aplicativo.

Na programação em Android, temos em geral um arquivo .XML que contém o layout e interface gráfica do aplicativo e uma classe JAVA associada a uma Activity.

Ao criar um novo projeto para Android, percebe-se que foi criado automaticamente pela ferramenta Eclipse um Layout .XML ( res -> layout -> nome do layout.xml) e uma **classe Java** ( **src -> nome do pacote -> nome da classe.java**

Ao acessar a classe Java criada automaticamente (esta classe tem o nome MainActivity.Java) observe que existe um método chamado onCreate:

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

O método **onCreate(Bundle)** é onde você inicia sua atividade e define a interface gráfica a qual está relacionada através do método **setContentView**.

Assim podemos obter os widgets (elementos de tela) através de um método chamado **findViewById()**.

Para que um arquivo .JAVA consiga se comunicar com o layout XML e outros recursos, existe uma classe intermediária, criada e atualizada automaticamente pela ferramenta, responsável por realizar esta ligação. Esta classe é chamada de R.Java.

A classe R.Java é o “coração” do sistema Android. Ela representa, em forma de atributos Java, todos os recursos da sua aplicação que estão dentro dos diretórios explicados de um aplicativo Android. Esta classe é gerada e atualizada automaticamente e não deve ser editada manualmente; o Eclipse fará isto automaticamente.

Por exemplo, temos dentro do diretório “res/drawable” a imagem “icon.png”; podemos acessá-la de dentro da nossa aplicação Android com a seguinte expressão: ‘R.drawable.icon’, onde “R” é a classe, “drawable” é o diretório e “icon” é o nome do recurso. Isto serve para quaisquer recursos presentes dentro dos diretórios de recursos.

Observe que o método setContentView relaciona a interface gráfica usando a classe R:

```
setContentView(R.layout.activity_main);
```

O método onCreate() - É a primeira função a ser executada em uma Activity (Tela). Geralmente é a responsável por carregar os layouts XML e outras operações de inicialização. É executada apenas uma vez.

O método onCreate() é importante pois é nele que iniciaremos nossos objetivos visuais e relacionamos os eventos (onClick, onTouch, etc) Neste método, programamos todas as funções que gostaríamos que fossem inicializadas quando o aplicativo for executado.



OBS: Este método só é executado uma única vez, no início da aplicação.

Os widgets adicionados à interface gráfica podem ser usados no nosso código Java. Se no código XML tivermos um widget do tipo EditText, para acessar esse componente pelo Java, é preciso fazer uso da classe EditText.

Cada widget no XML possui o seu respectivo em classe Java, logo, se possui um widget TextView, para acessá-lo devemos fazer uso da classe TextView e assim vai.

Podemos relacionar um widget a qualquer momento (onCreate, no clique de um botão – onTouch, onClick, etc.)

No exemplo abaixo, associamos os widgets no método onCreate da activity:

```
TextView tvBoasVindas;  
tvBoasVindas = (TextView) findViewById(R.id.tvBoasVindas);  
  
EditText txtNome;  
txtNome = (EditText) findViewById(R.id.txtNome);
```



OBS: Para associar um Widget inserido no Layout XML com um objeto Java, faz-se necessário especificar o atributo **android:id="@+id/"** de cada elemento a ser manipulado no código Java

No código apresentado anteriormente foi criado um objeto do tipo TextView e EditText na classe MainActivity.Java

Para relacionar o objeto criado com o seu respectivo correspondente na interface gráfica usamos o método findViewById e especificamos qual elemento está relacionado através da classe R.id.

Como JAVA é uma linguagem fortemente tipada, faz-se obrigatório converter o elemento para o tipo correto de dado, e isto é feito fazendo o casting (EditText), (TextView), etc.

```
txtNome = (EditText) findViewById(R.id.txtNome);
```

Este processo é feito semelhantemente para quaisquer recursos presentes dentro da interface gráfica XML.

A programação para Android, semelhantemente a outros ambientes, linguagens e ferramentas gráficas, é orientada a eventos, neste caso, aos cliques e toques na tela. Cada vez que um usuário toca em um botão, seleciona um item em uma lista, ou pressiona uma tecla, o sistema operacional gera um evento

Se uma aplicação está interessada em um evento específico (por exemplo, clique em um botão), deve solicitar ao sistema para “**escutar**” o **evento**. Se a aplicação não está interessada, seu processamento continua de forma normal.

É importante observar que a aplicação não espera pela ocorrência de eventos isso é controlado pelo sistema.

Para que um componente ou container possa “escutar” eventos, é preciso especificar um **LISTENER**. Listeners são classes criadas especificamente para o tratamento de eventos.

Um evento **LISTENER** é uma interface da classe **View** que contém um método simples de chamada. Esse método pode ser chamado pela framework **Android** quando a **View** a qual o listener está registrado é chamado por uma interação de usuário com um item da interface, ou seja, quando a ação correspondente ocorre no objeto. Por exemplo, quando um botão é clicado, o método **onClick()** é chamado no objeto.

Os principais eventos presentes na programação para **Android** são:

- **onClick():**  
Vem de **View.OnClickListener**. É chamado quando o usuário toca o item (quando estiver em modo de toque) ou foca o item através de teclas de navegação ou trackball e pressiona o botão de enter correspondente ou pressiona o trackbak (que também serve como enter).
- **onLongClick():**  
Vem de **View.OnLongClickListener**. É chamado quando o usuário toca um item e o segura (quando estiver em modo de toque) ou foca o item através de teclas de navegação ou trackball e pressiona o botão de enter correspondente e o segura ou pressiona o trackbak por pelo menos um segundo.
- **onFocusChange():**  
Vem de **View.OnFocusChangeListener**. É chamado quando o usuário navega para dentro ou para fora de um item, usando as teclas de navegação ou trackball.
- **onKey():**  
Vem de **View.OnKeyListener**. É chamado quando o usuário foca em um item e pressiona ou solta uma tecla no dispositivo. Exemplo: quando se está escrevendo uma mensagem, você pode tocar no botão virtual (ou físico) da letra **A** e, ao soltá-lo, a letra **A** é mostrada na tela.

- **onTouch():**

Vem de View.OnTouchListener. É chamado quando o usuário performa uma ação qualificada como um evento de toque, incluindo pressionar, soltar ou qualquer movimento de gesto na tela (dentro dos limites do item).

- **onCreateContextMenu():**

Vem de View.OnCreateContextMenuListener. É chamado quando um menu de contexto está sendo criado (como resultado de um long click).



Para exemplificar os listeners relacionados aos eventos, construiremos um layout semelhante ao apresentado ao abaixo:

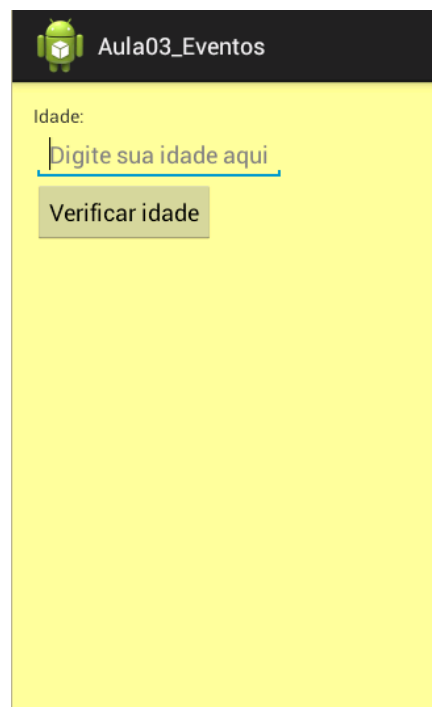


Figura 26. Exemplo de manipulação de eventos

Este layout contém um TextView, um EditText e um Button. Neste aplicativo chamaremos 2 listeners:

- onClick para o botão verificar idade
- onLongClick para o LinearLayout que envolve os widgets inseridos

O código XML do Layout acima é apresentado a seguir

```

<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    android:background="#FFFF99"
    android:id="@+id/layoutFormulario"
    tools:context=".MainActivity" >

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

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/txtIdade"
        android:hint="Digite sua idade aqui"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btVerificarIdade"
        android:text="Verificar idade"
        />

</LinearLayout>

```

Na classe JAVA associada a Activity (MainActivity) no método onCreate, iremos referenciar os objetos:

- EditText(txtIdade),
- Button(btVerificarIdade)
- LinearLayout(layoutFormulario)

```

//referenciando o LinearLayout
LinearLayout layoutFormulario;
layoutFormulario =
    (LinearLayout) findViewById(R.id.layoutFormulario);
//referenciando a EditText da idade
EditText txtIdade;
txtIdade = (EditText) findViewById(R.id.txtIdade);
//referenciando o botão de verificar idade
Button btVerificarIdade;
btVerificarIdade = (Button) findViewById(R.id.btVerificarIdade);

```

Para programar o listener referente ao toque longo (onLongClick) na tela (Linear Layout) usaremos o seguinte código:

```
View.OnLongClickListener toqueNaTela = new  
View.OnLongClickListener() {
```

Ao criar o Listener *toqueNaTela* o Eclipse cuidará automaticamente de inserir o código relacionado ao evento LongClick. Você deverá programar o que será realizado dentro da função:

```
@Override  
public boolean onLongClick(View v) {  
    // TODO Auto-generated method stub  
    return false;  
}
```

Após a programação do LongClick, deverá ser atribuído ao Layout, no método onCreate, o listener que acabou de ser criado:

```
layoutFormulario.setOnLongClickListener(toqueNaTela);
```

Código completo do método *toqueNaTela*:

```
View.OnLongClickListener toqueNaTela = new  
View.OnLongClickListener() {  
    @Override  
    public boolean onLongClick(View v) {  
        //referenciando a EditText referente a idade  
        EditText txtIdade;  
        txtIdade = (EditText)findViewById(R.id.txtIdade);  
        //cria uma variável inteira e coloca o valor da txtIdade nesta  
        variável  
        int idade;  
        idade = Integer.parseInt(txtIdade.getText().toString());  
        //A classe Toast é usada para mostrar uma mensagem na tela  
        Toast.makeText(getApplicationContext(), "A idade informada foi " +  
        idade, Toast.LENGTH_SHORT).show();  
        return false;  
    }  
};
```



No código apresentado acima recebemos o valor digitado na EditText (txtIdade). Este valor foi colocado em uma variável do tipo inteiro (int). Observe atentamente que antes da variável idade receber o valor que foi digitado, foi realizada uma conversão, utilizando o método Integer.parseInt()

O método getText() da EditText retorna o valor informado pelo usuário na caixa de texto.

A classe Toast é usada para exibir uma mensagem na tela. O método makeText é composto de 3 parâmetros: getBaseContext, "o texto a ser exibido" e a duração da mensagem.

Como segundo exemplo a partir do layout que trabalhamos, codificaremos um novo listener referente ao botão VERIFICAR IDADE. O listener com o evento deverá ser programado fora do método onCreate.



OBS: Digite apenas a criação da classe, que o Eclipse cuidará de criar automaticamente o método onClick: View.OnClickListener verificarIdade = new View.OnClickListener()

Código gerado pelo Eclipse:

```
View.OnClickListener verificarIdade = new {  
    View.OnClickListener()  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
    }  
};
```

Dentro do método public void onClick(View v) escreveremos o código associado ao clique. Neste caso, faremos um código para verificar a idade e classificar o valor informado e faixa etária (criança, adolescente, adulto)

Usaremos a estrutura if...else para verificar uma faixa de idade (criança, adolescente e adulto)

Ao final, exibiremos a faixa etária utilizando a classe Toast.

```
if ((idade > 0) && (idade <= 12))  
{  
    faixaEtaria = "Criança";  
}  
else if ((idade > 12) && (idade <= 18))
```

```

{
    faixaEtaria = "Adolescente";
}
else
{
    faixaEtaria = "Adulto";
}
//A classe Toast é usada para mostrar uma mensagem na tela
Toast.makeText(getBaseContext(), "Você é um(a) " + faixaEtaria,
    Toast.LENGTH_SHORT).show();

```



**Exercício:** Desenvolver uma APP para verificação de índice de massa corporal. A sugestão da interface gráfica é apresentada a seguir. Você deve programar o botão Verificar IMC de forma que ao toca-lo, o aplicativo apresentará o diagnóstico referente ao IMC para os dados informados.

Figura 27. Sugestão de layout para exercício prático IMC

### 2.2.6. Checkbox

O Widget Checkbox permite criar uma caixa de marcação múltipla, por padrão acompanhada de uma etiqueta ao lado. Na paleta de Widgets é possível incluir um Checkbox ou podemos adicioná-lo via código XML

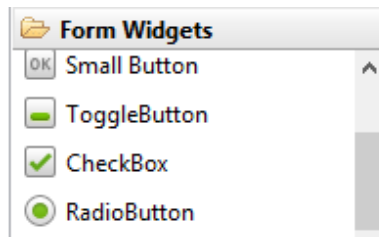



Figura 28. Widget CheckBox

O código XML que representa um CheckBox é mostrado a seguir:

```
<CheckBox
    android:id="@+id/ckEstudante"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Estudante"
/>
```



☐ Estudante



Para demonstrar a utilização de um CheckBox e como manipula-lo no código JAVA construiremos um layout semelhante ao apresentado ao abaixo:

**Aula04**

Nome:

☐ Estudante

☐ Trabalhador

Verificar

Figura 29. Exemplo com Checkbox

O código fonte do Layout XML par a interface gráfica acima é representado a seguir:

```
<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"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context=".MainActivity"
        android:orientation="vertical" >

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

        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Digite seu nome completo"
            android:id="@+id/txtNome"
            />

        <CheckBox
            android:id="@+id/ckEstudante"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Estudante"
            />

        <CheckBox
            android:id="@+id/ckTrabalhador"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Trabalhador"
            />

        <Button
            android:id="@+id/btVerificar"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Verificar"
            />
    </LinearLayout>

```

Para manipular um Checkbox na classe Java, devemos inicialmente fazer a associação deste elemento usando o método `findViewById`, assim como realizado com outros widgets.

Baseado no exemplo anterior, iremos programar o evento Click do botão VERIFICAR e exibir uma mensagem a partir da seleção da CheckBox, para isto criaremos um listener para o click do botão.

```
View.OnClickListener verifica = new View.OnClickListener()
```

Em seguida, programaremos o evento relacionado ao clique. OBS: Ao incluir o listener (onClickListener) o Eclipse irá incluir um código automaticamente relacionado ao onClick. Devemos programar nosso código entre as { } referente ao método onClick (View v)

O código completo do listener OnClickListener é apresetando a seguir:

```
View.OnClickListener verifica = new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub

        CheckBox ckEstudante;
        CheckBox ckTrabalhador;

        ckEstudante =
        (CheckBox) findViewById(R.id.ckEstudante);
        ckTrabalhador =
        (CheckBox) findViewById(R.id.ckTrabalhador);

        if (ckEstudante.isChecked() == true)
        {
            Toast.makeText(getApplicationContext(), "Estudante
selecionado!", Toast.LENGTH_SHORT).show();
        }
        if (ckTrabalhador.isChecked() == true)
        {
            Toast.makeText(getApplicationContext(), "Trabalhador
selecionado!", Toast.LENGTH_SHORT).show();
        }

    }

};
```

Após programar o clique é necessário alterar a propriedade onClickListener do botão.

```
btVerificar.setOnClickListener(verifica);
```

### 2.2.7. Radiobutton e RadioGroup

Estes controles trabalham juntos. O RadioGroup permite agrupar controles RadioButton. Um RadioButton, assim como o CheckBox é uma caixa de marcação única, por padrão acompanhada de uma etiqueta ao lado. O Android faz este processo de marcação/desmarcação de forma automática.

Na paleta de Widgets é possível incluir um Checkbox ou podemos adicioná-lo via código XML



Figura 30. RadioGroup e RadioButton



OBS: Para que um RadioButton tenha funcionalidade de marcação/desmarcação automática este deve estar inserido dentro de um RadioGroup.

O código XML que representa um RadioButton é mostrado a seguir:

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <RadioButton
        android:id="@+id/rbMasculino"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Masculino"/>
    <RadioButton
        android:id="@+id/rbFeminino"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Feminino"
    />
</RadioGroup>
```

A white rectangular box containing two radio button options. The first option is 'Masculino' with an empty radio button to its left. The second option is 'Feminino' with an empty radio button to its left.

Para demonstrar a utilização de um RadioButton e como manipulá-lo no código JAVA construiremos um layout semelhante ao apresentado ao abaixo:

Figura 31. Exemplo com RadioButton

O código fonte do Layout XML par a interface gráfica acima é representado a seguir:

```
<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"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity"
android:orientation="vertical" >

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

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Digite seu nome completo"
    android:id="@+id/txtNome"
/>

<CheckBox
    android:id="@+id/ckEstudante"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
```

```

        android:text="Estudante"
    />
    <CheckBox
        android:id="@+id/ckTrabalhador"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Trabalhador"
    />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sexo:"

    />

    <RadioGroup
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    >
        <RadioButton
            android:id="@+id/rbMasculino"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Masculino"
        />

        <RadioButton
            android:id="@+id/rbFeminino"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Feminino"
        />
    </RadioGroup>

    <Button
        android:id="@+id/btVerificar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Verificar"
    />

</LinearLayout>

```

Para manipular um Radiobutton na classe Java, devemos inicialmente fazer a associação deste elemento usando o método `findViewById`, assim como realizado com outros widgets.

Baseado no exemplo anterior, iremos programar o evento Click do botão VERIFICAR e exibir uma mensagem a partir da seleção do Radiobutton, para isto criaremos um listener para o click do botão.



```
View.OnClickListener verifica = new View.OnClickListener()
```

Em seguida, programaremos o evento relacionado ao clique.



OBS: Ao incluir o listener (onClickListener) o Eclipse irá incluir um código automaticamente relacionando ao onClick. Devemos programar nosso código entre as chaves { }

```
public void onClick(View v) {}
```

Após programar o clique é necessário alterar a propriedade onClickListener do botão **(btVerificar.setOnClickListener(verifica);)**

Assim como no Checkbox, o método isChecked() é utilizado para verificar se a checkbox está ou não marcada. Veja no código a seguir a programação do botão Verificar, que é utilizado para verificação de qual RadioButton está marcada.

```
RadioButton rbMasculino;  
RadioButton rbFeminino;  
rbMasculino = (RadioButton) findViewById(R.id.rbMasculino);  
rbFeminino = (RadioButton) findViewById(R.id.rbFeminino);  
if (rbMasculino.isChecked() == true)  
{  
    Toast.makeText(getApplicationContext(), "Masculino selecionado!",  
        Toast.LENGTH_SHORT).show();  
}  
else if (rbFeminino.isChecked() == true)  
{  
    Toast.makeText(getApplicationContext(), "Feminino selecionado!",  
        Toast.LENGTH_SHORT).show();  
}
```

### 2.2.8. ToogleButton e Switch

Um ToogleButton é um botão de alternância que permite os usuários alterar a configuração entre dois estado(Sim/Não, Verdadeiro/Falso)

Por padrão o texto deste botão é On/Off. Podemos alterar usando as propriedades:

```
android:textOn="Curtir"  
android:textOff="Não curtir"
```

Na paleta de Widgets é possível incluir um Togglebutton ou podemos adiciona-lo via código XML

```
<ToggleButton
    android:id="@+id/toggleButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Curtir"
    android:textOff="Não curtir"
/>
```

A versão Android 4.0 (API nível 14 em diante) introduz um outro tipo de botão de alternância chamado um Switch que fornece um controle deslizante, que você pode adicionar via XML.

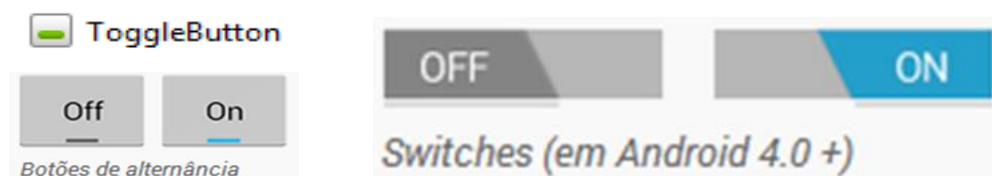



Figura 32. ToogleButton e Switch

Na paleta de Widgets é possível incluir um Tooglebutton ou podemos adiciona-lo via código XML:

```
<ToggleButton
    android:id="@+id/toggleButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Curtir"
    android:textOff="Não curtir"
/>
```

O Widget Switch não aparece na paleta de componentes, mas poderá ser adicionado através do código XML a seguir:

```
<Switch
    android:id="@+id/toggleButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Curtir"
    android:textOff="Não curtir"
/>
```



Manipulamos `ToggleButton` e `Switch` de forma semelhante a um `Checkbox` ou `RadioButton`. Neste caso, criaremos um listener específico para o click do `ToggleButton` ou `Switch`. Para verificar se valor marcado está `On` ou `Off` usamos o método `isChecked()`. Veja exemplo a seguir:

```
ToggleButton tgCurtir;
tgCurtir = (ToggleButton)findViewById(R.id.tgCurtir);
if (tgCurtir.isChecked() == true)
{
    Toast.makeText(getBaseContext(), "Você curtiu!!",
    Toast.LENGTH_SHORT).show();
}
else
{
    Toast.makeText(getBaseContext(), "Que pena!!!",
    Toast.LENGTH_SHORT).show();
}
```

### 2.2.9. Seekbar

Uma `SeekBar` é uma barra de progresso no qual o usuário pode tocar e arrastar para a esquerda ou para a direita para definir o nível de progresso. Na paleta de Widgets é possível incluir um `SeekBar` ou podemos adicioná-lo via código XML



Figura 33. SeekBar

Por padrão o valor máximo de uma `SeekBar` é 100 e o valor mínimo é 0. O intervalo é definido de 1 em 1. Apenas o valor máximo pode ser definido via XML. Os demais atributos só podem ser modificados via código Java.

O código XML para incluir o Widget Seekbar em uma tela é semelhante aos demais componentes. OBS: a propriedade **android:max** define o valor máximo da `SeekBar`

```
<SeekBar
    android:id="@+id/skIdade"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="120"
/>
```

Manipulamos um Seekbar de forma semelhante aos demais componentes. Para adicionar funcionalidade quando o usuário arrastar a barra, precisamos criar um listener. Neste caso definimos o `OnSeekBarChangeListener`.

```
SeekBar.OnSeekBarChangeListener avanca = new  
SeekBar.OnSeekBarChangeListener()
```

O Eclipse irá gerar automaticamente 3 eventos:

```
public void onStopTrackingTouch(SearchBar searchBar)  
  
public void onStartTrackingTouch(SearchBar searchBar)  
  
public void onProgressChanged(SearchBar searchBar, int progress,  
boolean fromUser)
```

Usamos o evento **`public void onProgressChanged(SearchBar searchBar, int progress, boolean fromUser)`** para capturar as mudanças iniciadas pelo usuário. Para obter o valor atual definido pelo usuário, usamos o parâmetro **`progress`**.

Após programar o evento `onProgressChanger` é necessário “setar” o listener relacionado ao componente inserido na tela, para que a operação seja realizada.

Exemplo:

```
SeekBar skIdade;  
skIdade = (SeekBar) findViewById(R.id.skIdade);  
skIdade.setOnSeekBarChangeListener(avanca);
```



Para demonstrar a utilização de um `RadioButton` e como manipula-lo no código JAVA construiremos um layout semelhante ao apresentado ao abaixo e programaremos o listener **`OnSeekBarChangeListener`**:

Figura 34. Exemplo com SeekBar

Programação JAVA do listener **onSeekBarChangeListener**:

```
SeekBar.OnSeekBarChangeListener avanca = new
SeekBar.OnSeekBarChangeListener() {

    @Override
    public void onStopTrackingTouch(SearchBar seekBar) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onStartTrackingTouch(SearchBar seekBar) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProgressChanged(SearchBar seekBar, int progress,
        boolean fromUser) {
        // TODO Auto-generated method stub

        TextView tvIdade;
        tvIdade = (TextView) findViewById(R.id.tvIdade);
        tvIdade.setText("Idade = " + progress);
    }
};
```

### 2.2.10. RatingBar

Um RatingBar é uma extensão da SeekBar e ProgressBar que mostra uma classificação em estrelas.



Na paleta de Widgets é possível incluir um SeekBar ou podemos adicioná-lo via código XML:

```
<RatingBar
    android:id="@+id/rtNota"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
/>
```

Os principais atributos de um Ratingbar são:

- `android:numStars`: define o número de itens (estrelas)
- `android:rating`: define o valor padrão da classificação (OBS: deve ser um número com casa decimal, ex: 1,2)
- `android:stepSize`: define o passo (variação) da classificação (OBS: deve ser um número com casa decimal, ex: 1,2)

O listener utilizado para programar uma ação quando o usuário alterar a quantidade de estrelas marcada é o : `OnRatingBarChangeListener` ().

O método: `getRating ()` retorna a classificação atual (número de estrelas cheias).



Para demonstrar a utilização de um RatingBar e como manipulá-lo no código JAVA construiremos um layout semelhante ao apresentado ao abaixo e programaremos o listener **OnClickListener** do botão Verificar:



Figura 35. Exemplo com RatingBar

O código JAVA para verifica a quantidade de estrelas marcadas é apresentado a seguir:

```
View.OnClickListener verifica = new View.OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
  
        RatingBar rtNota;  
        rtNota = (RatingBar) findViewById(R.id.rtNota);  
  
        double nota;  
  
        nota = rtNota.getRating();  
  
        Toast.makeText(getApplicationContext(), "Nota = " + nota,  
        Toast.LENGTH_LONG).show();  
  
    }  
};
```

### 2.2.11. NumberPicker

O NumberPicker é um widget que permite que o usuário selecionar um número a partir de um conjunto pré-definido, ou seja, permite criar um índice de numeração que pode aumentar ou diminuir, com valores fixos determinados pelo programador

Este componente fica na aba Advanced da paleta e pode ser adicionado arrastando a tela ou via código XML.

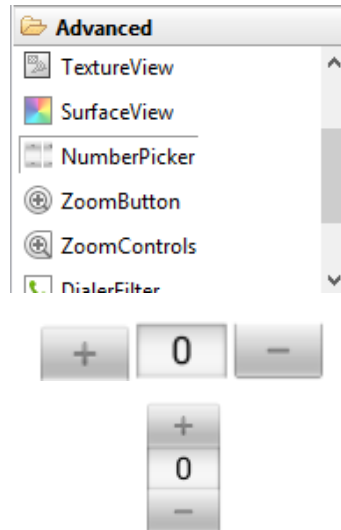


Figura 37. Objeto NumberPicker

Código XML para criação de um objeto do tipo NumberPicker

```
<NumberPicker
    android:id="@+id/npIngressos"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
/>
```

Após incluir o elemento na parte visual, faz-se necessário criar uma associação ao controle no código Java para determinar o valor máximo, mínimo e o incremento.

No código Java, no método onCreate da activity, devemos definir o valor máximo, mínimo e atual do NumberPicker.

```
NumberPicker npIngressos;
npIngressos = (NumberPicker) findViewById(R.id.npIngressos);
npIngressos.setMinValue(0);
npIngressos.setMaxValue(10);
npIngressos.setValue(0);
```

O listener OnValueChangeListener é usado para programar a mudança do valor atual. O método getValue() retorna o valor do seletor.



### 2.2.12. DateTime e TimePicker

O Android possui alguns widgets específicos para trabalhar com valores de data e hora. Na paleta de Widgets eles estão organizados na guia Time & Data.

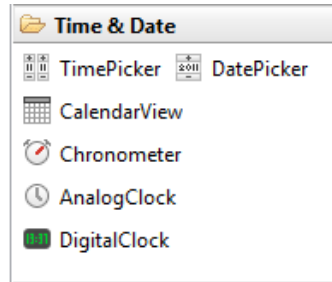


Figura 38. DateTime e TimePicker

Estes elementos podem ser adicionados arrastados ao formulário ou através do código XML.

```
<DatePicker
    android:id="@+id/datePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<TimePicker
    android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Os principais atributos de um DatePicker são:

- `android:calendarViewShown`: Se o ponto de vista da agenda será mostrada.
- `android:maxDate`: A data máxima mostrada no formato dd / mm / aaaa.
- `android:minDate`: A data mínima mostrado no formato dd / mm / aaaa.
- `android:spinnersShown`: Configura se os spinners são mostrados.
- `android: startYear`: O primeiro ano, por exemplo, "1940".

O listener **OnDateChangeListener()** é usado para realizar uma programação quando o usuário alterar a data.

Os métodos **GetDayOfMonth()**, **getMonth()**, **getYear()** entre outros são usados para obter os valores definidos pelo usuário.

O listener **OnTimeChangeListener()** é usado para realizar uma programação quando o usuário alterar o horário.

Os métodos **getCurrentHour()**, **getCurrentMinute()**, entre outros são usados para obter os valores definidos pelo usuário.



OBS: Para manipular um DatePicker ou TimePicker na classe Java, devemos inicialmente fazer a associação deste elemento usando o método **findViewById**, assim como realizado com outros widgets já estudados.

### 2.2.13. Spinner

Spinners é um widget que permite o usuário selecionar um valor a partir de um conjunto de dados. No estado padrão, um spinner mostra seu valor atualmente selecionado.

Um toque no spinner exibe um menu com todos os outros valores disponíveis, a partir do qual o usuário pode selecionar um novo.

Na paleta de Widgets é possível incluir um Spinner ou podemos adicioná-lo via código XML:

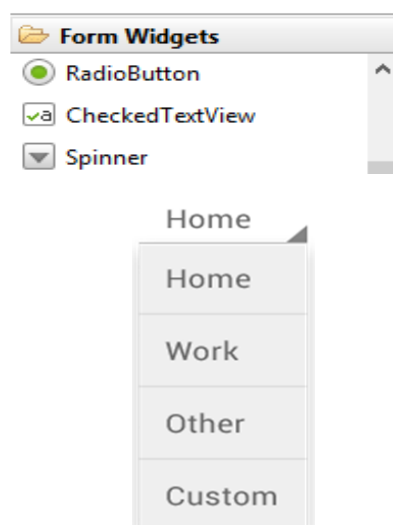


Figura 39. Spinner

O código XML para criação de um Spinner é representado a seguir:

```
<Spinner
    android:id="@+id/spTimes"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

Podemos preencher um spinner de diversas formas:

- Através de um ArrayList,
- Através de um Array de string definido em um arquivo resources,
- Através de uma consulta ao banco de dados (veremos isto quando estudarmos sobre persistência de dados).
- Outros

Inicialmente veremos o preenchimento através de um array de string definido em um arquivo resources. Para isto, vamos criar um Array de String no diretório: res -> values -> strings. Veja exemplo a seguir:

```
<string-array name="times">
    <item>Atlético MG</item>
    <item>Atlético PR</item>
    <item>Botafogo</item>
    <item>Corinthians</item>
    <item>Cruzeiro</item>
    <item>Flamengo</item>
    <item>Fluminense</item>
    <item>Grêmio</item>
    <item>Internacional</item>
    <item>Santos</item>
    <item>São Paulo</item>
    <item>Vasco</item>
    <item>Vitória</item>
</string-array>
```

Depois de definido um string-array em res -> values -> string, precisamos associar o objeto aos itens da Spinner. Para isto, no método onCreate da activity devemos relacionar a nossa fonte de dados (array de string) a um spinner desejado. Neste caso usaremos um objeto chamado ArrayAdapter.

```
Spinner spTimes = (Spinner) findViewById(R.id.spTimes);
// Cria um ArrayAdapter usando um array de string e um layout
padrão de spinner
```

```
ArrayAdapter adapter = ArrayAdapter.createFromResource(this,
R.array.times, android.R.layout.simple_spinner_item);
//alterar a fonte de dados(adapter) do Spinner
spTimes.setAdapter(adapter);
```

O resultado poderá ser visualizado na imagem abaixo:



Figura 40. Preenchimento de Spinner

Para capturar o texto selecionado usamos o método `getSelectedItem()`.

```
String texto;
texto = spTimes.getSelectedItem().toString();
```

Para capturar a posição (número do item selecionado – começando de 0) usamos o método `getSelectedItemPosition()`.

```
int a;
a = spTimes.getSelectedItemPosition();
```

Para manipular o evento de um Spinner quando o usuário alterar o item selecionado, devemos criar um listener específico para esta função. Neste caso usamos um listener para um `AdapterView`

```
AdapterView.OnItemSelectedListener escolha = new  
AdapterView.OnItemSelectedListener() {
```

Em seguida, programaremos o evento relacionado ao item selecionado.



OBS: Ao incluir o listener (AdapterView.OnItemSelectedListener) o Eclipse irá incluir um código automaticamente relacionando ao item selecionado.

Devemos programar nosso código no método:

```
public void onItemSelected
```

Após programar o clique é necessário alterar a propriedade setOnItemSelectedListener do Spinner:

```
spTimes.setOnItemSelectedListener(escolha);
```

Um exemplo completo é apresentado no código Java a seguir:

#### 2.2.14. ListView

Um ListView é um Widget que mostra itens em uma lista de rolagem vertical. Na paleta de Widgets (Composite) é possível incluir um ListView ou podemos adicioná-lo via código XML.

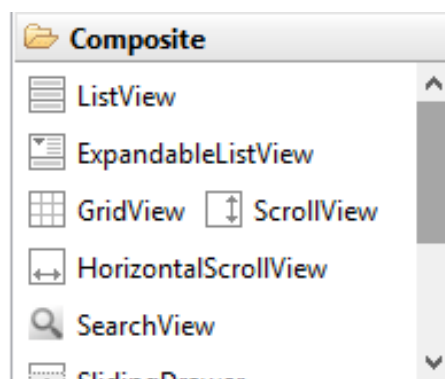


Figura 41. ListView

```
<ListView
    android:id="@+id/lvTimes"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```



OBS: Uma propriedade importante da ListView é a choiceMode. Usamos este atributo quando queremos transformar uma ListView em uma lista de seleção.

```
android:choiceMode="singleChoice"
```

Da mesma forma que uma Spinner, podemos preencher um ListView de diversas formas:

- Através de um ArrayList,
- Através de um Array de string definido em um arquivo resources,
- Através de uma consulta ao banco de dados (veremos isto quando estudarmos sobre persistência de dados).
- Outros

Estudaremos o preenchimento através de um array de string definido em um arquivo resources. No código Java, para preencher a ListView com uma lista de opções, precisamos especificar um ArrayAdapter e depois associa-lo a ListView desejada.



OBS: Neste Exemplo, usaremos a string de arrays já criada para popular o Spinner relacionado aos times de futebol.

O processo de preenchimento é semelhante ao da Spinner. Veja código Java abaixo:

```
// Cria um ArrayAdapter usando um array de string e um layout
padrão de spinner
// Neste caso, o layout padrão adotado possui a opção do usuário
CHECAR um item
ArrayAdapter timesSerieA = ArrayAdapter.createFromResource(this,
R.array.times, android.R.layout.simple_list_item_checked);
timesSerieA.setDropDownViewResource(android.R.layout.simple_list
_item_checked);
//Cria um ListView e relaciona com o Widget criado no XML
```

```

ListView lvTimes;
lvTimes = (ListView) findViewById(R.id.lvTimes);
//Associa a fonte de dados timesSerieA - Array de string
lvTimes.setAdapter(timesSerieA);

```

O resultado do código Java mostrado acima é visualizado como:

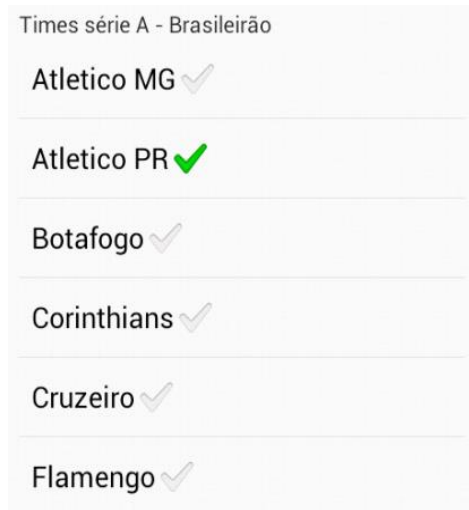


Figura 42. Spinner preenchido com string-array

Para capturar a posição de um elemento selecionado em uma ListView com a opção ChoiceMode configurada, usamos o método `getCheckedItemPosition()`

```

int a;
a = lvTimes.getCheckedItemPosition();

```

Para capturar o texto de um item selecionado usamos o método `getItemAtPosition()` de uma ListView.

```

String texto;
texto = lvTimes.getItemAtPosition(a).toString();

```

### 3. STRING.XML (INTERNACIONALIZAÇÃO) E LAYOUTS EM DIFERENTES ORIENTAÇÕES

#### 3.1.String.xml

No desenvolvimento de aplicações para Android **NÃO** é considerado uma boa prática de programação colocar os textos do aplicativo diretamente na propriedade text conforme exemplificado a seguir.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Nome: "
    android:id="@+id/tvNome"
/>
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Passaporte"
    android:id="@+id/tvPassaporte"
/>
```

Como boas práticas de programação um aplicativo Android contém um arquivo chamado string.xml, neste arquivo são armazenadas os textos exibidos no aplicativo, como por exemplo, o nome do aplicativo, nome de botões, nome de layouts. Deixando todos os textos do aplicativo em um só arquivo, facilita a manutenção além de podermos utilizar este recurso para internacionalizar nosso APP.

Para suportar internacionalização, o ambiente Android possui arquivos de “recursos” (resource), em XML, contendo os textos a serem exibidos na aplicação.

No arquivo string.xml criamos cada objeto (<string> </string>) e em seguida referenciamos em nosso layout. Veja exemplo no código abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internacionalização</string>
    <string name="lb_nome">Nome: </string>
    <string name="lb_passaporte">Passaporte: </string>
    <string name="lb_endereco">Endereço: </string>
    <string name="lb_telefone">Telefone: </string>
    <string name="lb_enviar">Enviar</string>
    <string name="lb_limpar">Limpar</string>
</resources>
```



Após definido no arquivo string.xml, podemos referenciar os objetos criado através da propriedade text, conforme verificado abaixo:

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/btEnviar"
    android:text="@string/lb_enviar"
/>
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/btLimpar"
    android:text="@string/lb_limpar"
/>
```

Para realizar internacionalização de nossas aplicações, configuramos um arquivo string.xml para cada idioma previsto, mas isto deve ser feito copiando a pasta values e em seguida renomeando esta pasta com a extensão do idioma (en para inglês, es para espanhol, fr para francês, etc.)

Veja abaixo:

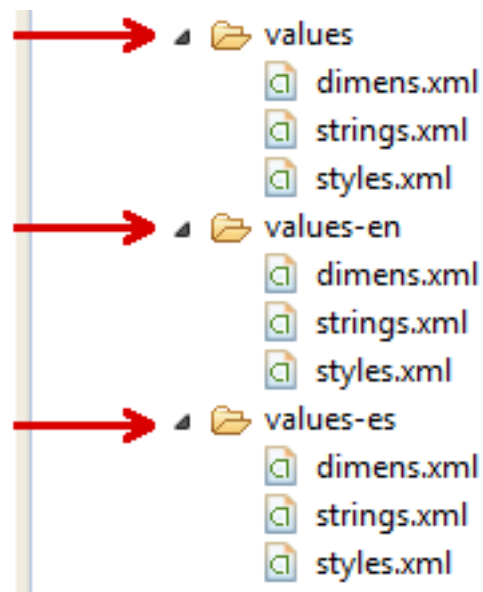


Figura 43. Internacionalização com strings.xml

Depois de criado as pastas para cada idioma, configuramos os objetos criados no arquivo string.xml com a respectiva tradução. Como exemplo, configuramos a tradução para o arquivo de string.xml apresentado no início deste capítulo. Veja exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internacionalization</string>
    <string name="lb_nome">Name: </string>
    <string name="lb_passaporte">Passport: </string>
    <string name="lb_endereco">Address: </string>
    <string name="lb_telefone">Phone number: </string>
    <string name="lb_enviar">Send</string>
    <string name="lb_limpar">Clear</string>
</resources>
```

De acordo com o idioma do dispositivo ele vai saber qual pasta irá buscar. Por exemplo, se o smartphone de um usuário estiver no idioma inglês ele vai buscar pela pasta values-en , se o estiver no idioma espanhol ele vai buscar na pasta values-es. Este processo é feito automaticamente pelo sistema operacional Android.

Caso não exista um idioma específico o Android irá utilizar o arquivo definido na pasta values. Para testar a aplicação, iremos acessar a Configuração do dispositivo e alterar o idioma para algum idioma definido no string.xml.

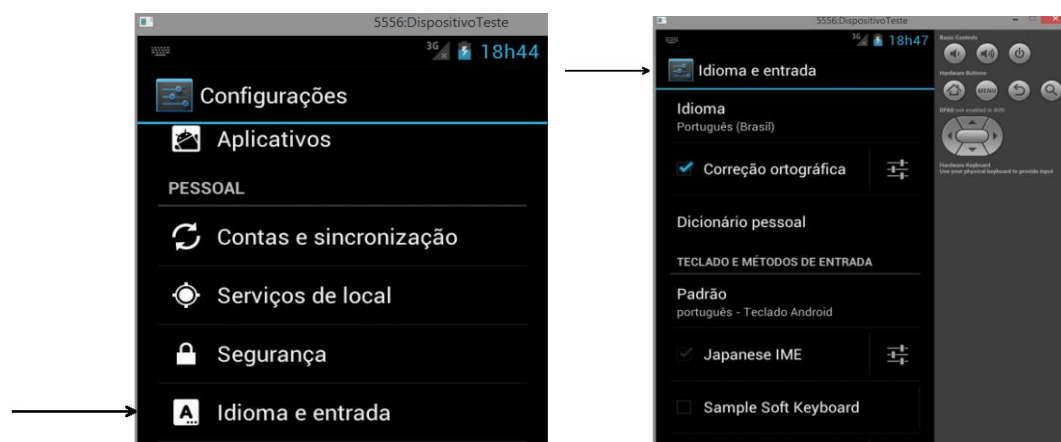


Figura 44. Alterando o idioma do dispositivo Android

O resultado poderá ser visualizado conforme a imagem a seguir:

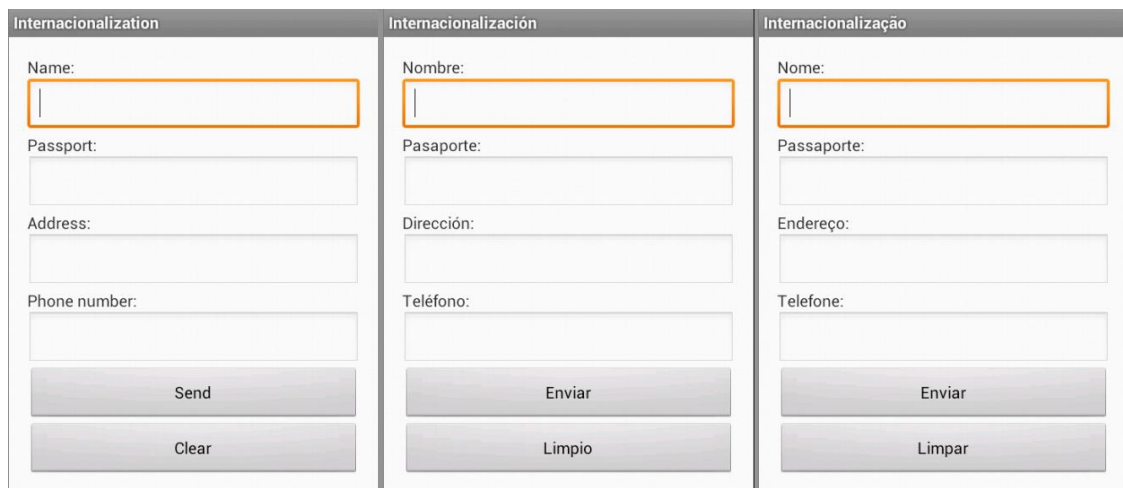


Figura 45. Execução de aplicativo com suporte a múltiplos idiomas

### 3.2. Layout com orientação na vertical e horizontal

Quando desenvolvemos aplicativos móveis precisamos criá-lo de modo que o layout mantenha-se perfeito nas diferentes orientações de tela. No Android este trabalho não é tão difícil quanto em outras ferramentas.

Depois de criado o projeto no eclipse clique com o botão direito na pasta “res” (onde ficam todos os recursos do aplicativo como, imagens, xml, layouts), em “New” e em “Android XML File”, como mostrado na imagem abaixo:

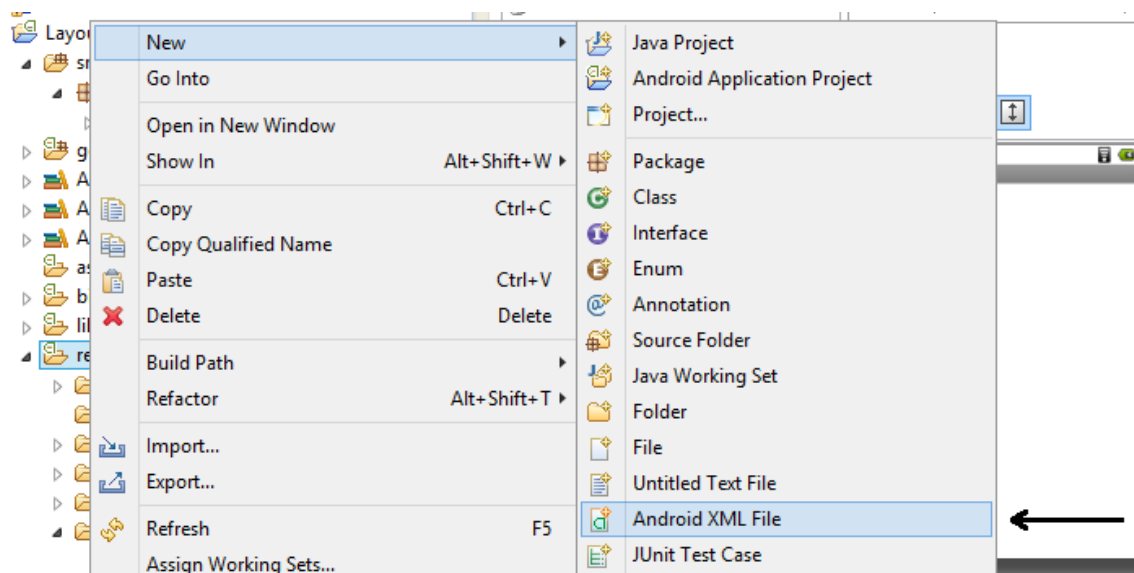


Figura 46. Adicionando novos arquivos de layout.xml

Na tela de criação do xml marque a opção “Layout”, em Available Qualifiers escolha ORIENTATION e clique no botão referente a seta para o lado direito “Chosen Qualifiers” . Na caixa de seleção “Screen Orientation” defina Portrait. Com isso você perceberá que uma nova pasta será criada com o nome de “layout-port”. Faça o processo novamente sendo que desta vez escolha no combo-box a opção Landscape e mais uma nova pasta será criada com o nome de “layout-land” como nas imagens abaixo:

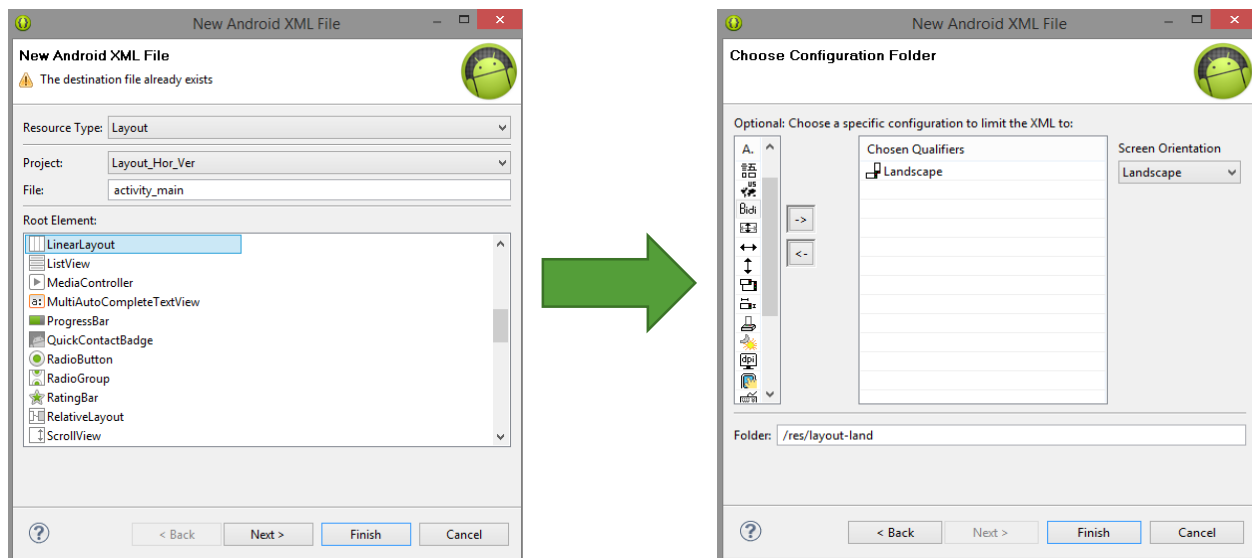


Figura 47. Configurando um novo arquivo de layout.xml

Com isso temos mais duas pastas no nosso projeto. Para que a troca de layouts funcione é preciso criar um layout xml com o mesmo nome em ambas as pastas e o android reconhecerá como o mesmo layout para orientações diferentes.

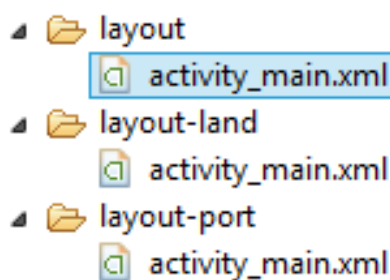


Figura 48. Estruturas de pastas para layout horizontal e vertical

No arquivo .xml referente aos layout (activity\_main.xml) definimos uma imagem diferente para o background da orientação horizontal e outra para a orientação vertical.

Layout horizontal:

```
<?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:background="@drawable/layout_horizontal"
    android:orientation="vertical" >
</LinearLayout>
```

Layout vertical:

```
<?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"
    android:background="@drawable/layout_vertical" >
</LinearLayout>
```

Resultado dos layouts nas orientações vertical e horizontal:



Figura 49. Execução de aplicativo com orientações de layouts na vertical e horizontal



OBS: OS LAYOUTS NA VERTICAL E HORIZONTAL DEVEM POSSUIR OS MESMOS ELEMENTOS. DEVERÁ SER ALTERADO APENAS AS CONFIGURAÇÕES RELACIONADAS DE CADA LAYOUT ESPECÍFICO.



**Exercício:** Em um hospital, na ala de internamento, as enfermeiras em cada turno verificam alguns sinais vitais dos pacientes: pressão arterial, batimentos cardíacos e, temperatura do corpo. Também é necessário anotar o nome do paciente e número do leito. Com os recursos já estudados, desenvolva uma APP que permita auxiliar as enfermeiras neste processo, de forma que possa substituir as anotações numa ficha de papel por um APP no Android. Após preenchimento a enfermeira deve clicar em um botão e enviar os dados.

Utilize TextView, EditText e Button para montar a interface gráfica

Utilize o recurso de string para disponibilizar sua APP em pelo menos 2 idiomas de livre escolha.

## 4. TEMAS, ESTILO E GALERIA DE IMAGENS

### 4.1. Temas e estilo

Um estilo é uma coleção de propriedades que especificam o visual e formato de uma View ou janela. Um estilo por especificar propriedades como altura, espaçamentos, cor de fonte, tamanho da fonte, cor de fundo e muito mais. Um estilo é definido em um recurso XML que é reparado do XML que especifica o layout.

Estilos em Android compartilham uma filosofia similar ao que é o CSS para web design - eles permitem a você separar o design do conteúdo.

Por exemplo, usando um estilo, o código a seguir:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello"
/>
```

Poderá ser reduzido para:

```
<TextView
    style="@style/Brasil"
    android:text="@string/Pais" />
```

Todos os atributos relacionados a estilo foram removidos do layout XML e colocados em um estilo definido pelo programador, que é então aplicado com o atributo style.

Um tema é um estilo aplicado em toda uma atividade ou aplicação, ao invés de apenas em uma View individual (como no exemplo acima). Quando um estilo é aplicado como um tema, cada View na atividade vai ter o estilo aplicado onde for suportado. Por exemplo, você pode aplicar o mesmo estilo como um tema para uma Activity (Janela) e então todo o texto dentro da atividade vai ter a configuração definida no arquivo de estilo.

Para criar um conjunto de estilos, abra o arquivo style.xml na pasta res/values dentro do seu projeto. Ao abrir este arquivo você irá verificar o nó raiz do arquivo XML de estilo deverá ser <resources>.

Para cada estilo que você queira criar adicione um elemento **<style>** ao arquivo com um **name** que identifique unicamente o estilo. Então adicione elementos **<item>** para cada propriedade do estilo, com o **name** que declare a propriedade de estilo e o valor a ser usado.

O valor para o **<item>** pode ser uma string, um código hexadecimal para cor, uma referência para outro recurso ou outro valor dependendo das propriedades do estilo.

Vejamos um exemplo de um estilo:

```
<style name="Textos" parent="AppTheme">
    <item name="android:textColor">#0000FF</item>
    <item name="android:textSize">20px</item>
    <item name="android:typeface">serif</item>
</style>
<style name="Botao" parent="AppTheme">
    <item name="android:textColor">#00FF00</item>
    <item name="android:textSize">40px</item>
    <item name="android:textAllCaps">true</item>
    <item name="android:typeface">serif</item>
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
</style>
```

O atributo **parent** dentro do elemento **<style>** é opcional e especifica o ID de recurso de um outro estilo a partir do qual esse estilo deverá herdar propriedades.

Cada filho do elemento **<resources>** é convertido em um objeto de recurso da aplicação em tempo de compilação, que pode ser referenciada pelo valor do atributo **name** dentro do elemento **<style>**. Esse estilo de exemplo pode ser referenciado a partir de um XML de layout usando **@style/nomeDoEstilo**.

Para utilizar um estilo criado em um arquivo de layout, usamos o atributo **style** conforme a seguir:

```
<TextView
    android:text="Endereço"
    style="@style/Textos" />
<Button
    android:text="Verificar"
    style="@style/Botao" />
```

O resultado do estilo aplicado a **TextView** e o **Button** poderá ser visualizado como:





Figura 50. Visualizando a aplicação de estilos

Um **theme** é um estilo aplicado a uma **Activity** ou aplicação inteira, ao invés de elemento por elemento. Quando um estilo é aplicado como um theme, todos os widgets na Activity ou aplicação irão usar todas as propriedades de estilo por ele definidas.

Para aplicar uma definição de estilo como um tema, você deve aplicá-lo à atividade ou aplicação dentro do manifesto (AndroidManifest.xml) do Android.



Figura 51. Definindo um tema para a aplicação

Quando você faz dessa maneira, cada elemento dentro da atividade ou aplicação vai aplicar as propriedades que são suportadas. Quaisquer elementos que não suportem alguma das propriedades não a terão aplicada. Se um elemento suporta apenas uma das propriedades, apenas essa propriedade será aplicada.

## 4.2. Temas e estilo

O Componente Gallery permite criar uma galeria de imagens em uma aplicação Android. Uma das formas mais comuns de apresentação é visualizar um conjunto de imagens presentes nas pastas res/drawable de nossa aplicação.

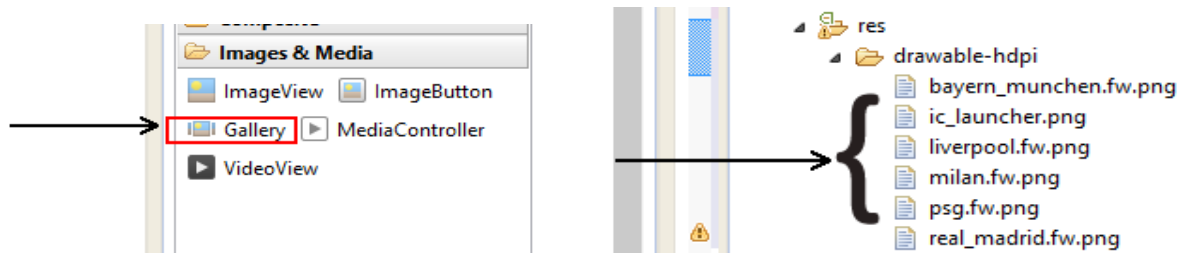
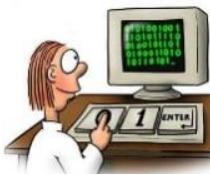


Figura 52. O componente Gallery



Para demonstrar a utilização de uma Galeria de construiremos um layout semelhante ao apresentado ao abaixo. Observação: as imagens utilizadas(escudos de times europeus) é apenas um exemplo, você poderá utilizar imagens de livre escolha.

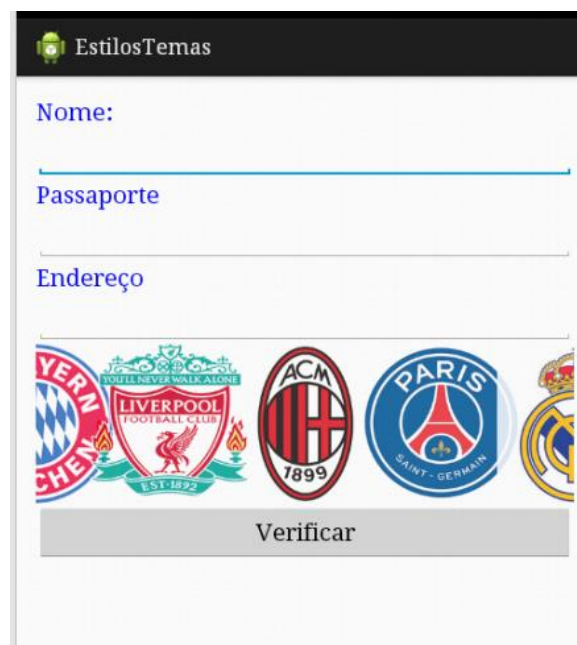


Figura 53. Exemplo de galeria de imagens

O componente Gallery pode ser adicionado no layout.xml conforme a seguir:

```
<Gallery
    android:id="@+id/galeriaImagem"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:spacing="15px"/>
```

Para popular o Gallery com um conjunto de imagens, precisamos programar uma classe Java baseada na classe BaseAdapter. Esta classe funciona com um adaptador de dados que fará a ligação entre as imagens presentes na aplicação e componente Gallery.

Felizmente esta classe já está criada por programadores ao redor do mundo, não sendo necessário programá-la. Neste exemplo iremos apenas configurar as imagens que serão utilizadas.

Crie uma nova classe Java e adicione o código da classe ImageAdapter apresentado a seguir:

```
package com.example.estilostemas;

import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter{

    Context Con;
    //array to hold the values of image resources
    int [] Resources;
    List<ImageView> views;

    public ImageAdapter(Context con,int[] resources)
    {
        Con=con;
        Resources=resources;
        views=new ArrayList<ImageView>(resources.length);
    }

    @Override
    public int getCount() {
        // TODO Auto-generated method stub
        return Resources.length;
    }
}
```

```

@Override
public Object getItem(int position) {
    // TODO Auto-generated method stub
    return views.get(position);
    //return position;
}

@Override
public long getItemId(int position) {
    // TODO Auto-generated method stub
    //return views.get(position).getId();
    return position;
}

@Override
public View getView(int position, View convertView,
ViewGroup parent) {
    // TODO Auto-generated method stub
    ImageView img=new ImageView(Con);
    img.setImageResource(Resources[position]);
    views.add(img);
    return img;
}
}

```



OBS: VERIFIQUE SE O PACKAGE DA CLASSE IMAGEADAPTER ESTÁ RELACIONADO AO PACOTE DE SUA APLICAÇÃO.

Após criação da classe ImageAdapter, programaremos o método onCreate da classe relacionada a Activity. Inicialmente iremos relacionar a galeria de imagem criada no layout.xml.

```

galateria = (Gallery)findViewById(R.id.galeriaImagem);

```

Em seguida será criado um vetor com as imagens presentes na pasta drawable (neste exemplo as imagens de escudos de clubes de futebol da europa).

```

//CRIAÇÃO DE UM VETOR DE INTEIRO COM AS IMAGENS ARMAZENADAS NA
PASTA DRAWABLE
int []imagens =new
int[] {R.drawable.bayern_munchen,R.drawable.liverpool,R.drawable.
milan,R.drawable.psg,R.drawable.real_madrid};

```

Por fim criamos um objeto do tipo ImageAdapter (classe que foi criada para relacionar uma fonte de dados com um objeto Gallery) e em seguida alteramos a fonte de dados da Gallery para este objeto.

```
//CRIA UM OBJETO DO TIPO IMAGE ADAPTER ATRIBUINDO O VETOR DE
IMAGENS
ImageAdapter imgAdapter=new ImageAdapter(this, imagens);
//ALTERA A FONTE DE DADOS DA GALERIA
galeria.setAdapter(imgAdapter);
```

O resultado será verificado quando a aplicação for executada. O usuário ao arrastar o dedo na tela poderá verificar a alteração das imagens.

Para verificar qual imagem foi selecionada podemos utilizar o evento **OnItemClickListener**.

Para isto devemos programar o código relacionado ao clique da imagem e em seguida atribuí-lo ao objeto Gallery. Veja como seria este processo:

```
OnItemClickListener selecionar = new OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int
arg2,
long arg3) {
// TODO Auto-generated method stub
Toast.makeText(getApplicationContext(), "Você selecionou a imagem
de nº 1" + (arg2 + 1), Toast.LENGTH_SHORT).show();
}
};
```

## 5. ACTIVITIES, MENUS E ACTION BAR

### 5.1. Activities

A classe Activity é quem gerencia a interface com o usuário. Esta classe é responsável por receber as requisições, trata-las e processa-las. Na programação de aplicativos para Android, uma Activity está relacionada a uma Tela, mas é preciso entender também o ciclo de vida de uma Activity.

Imagine que nossa aplicação seja interrompida de forma inesperada, seja por que o usuário abriu uma outra Activity e com isso houve algum fator que fez a Activity ser fechada, e até mesmo quando o Android finaliza a mesma quando vê a necessidade de liberar memória. Desta forma é preciso entender cada ciclo de vida de uma Activity para realizar a programação necessária a cada ciclo de uma Activity.

Uma Activity possui vários estados em seu ciclo de vida, conheceremos os diversos estados a seguir:

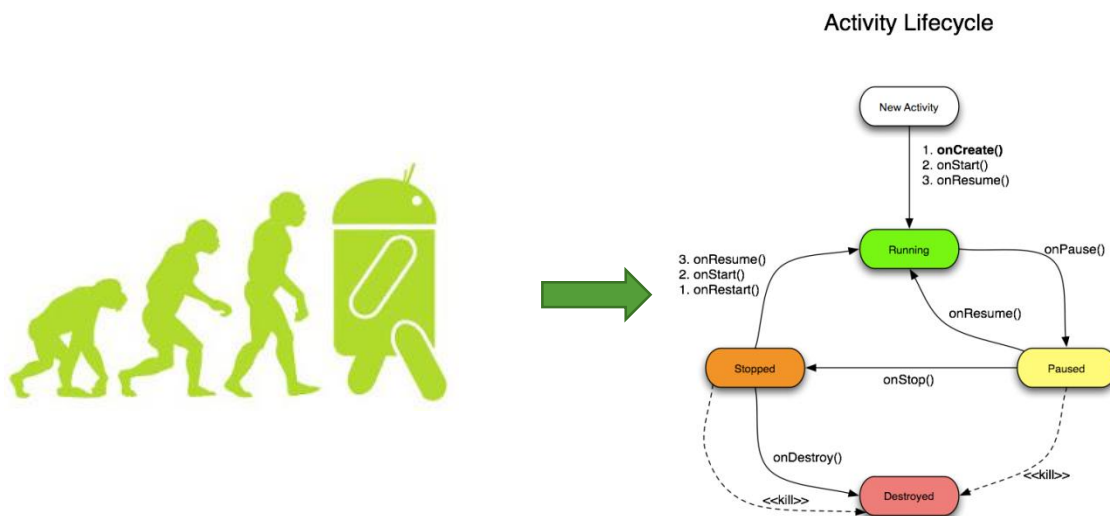


Figura 54. Ciclo de vida de uma aplicação Android

- **onCreate()** - É a primeira função a ser executada em uma Activity. Geralmente é a responsável por carregar os layouts XML e outras operações de inicialização. É executada apenas uma vez.
- **onStart()** - É chamada imediatamente após a onCreate() – e também quando uma Activity que estava em background volta a ter foco.

- **onPause()** - É a primeira função a ser invocada quando a Activity perde o foco (isso ocorre quando uma nova Activity é iniciada).
- **onRestart()** - Chamada imediatamente antes da onStart(), quando uma Activity volta a ter o foco depois de estar em background.
- **onResume()** - Assim como a onStart(), é chamada na inicialização da Activity e também quando uma Activity volta a ter foco. Qual a diferença entre as duas? A
- **onStart()** - É chamada quando a Activity não estava mais visível e volta a ter o foco, a onResume() é chamada nas “retomadas de foco”.
- **onDestroy()** - A última função a ser executada. Depois dela, a Activity é considerada “morta” – ou seja, não pode mais ser relançada. Se o usuário voltar a requisitar essa Activity, um novo objeto será construído.

Para criar novas telas em uma aplicação Android, é necessário trabalhar com o conceito de Activity. Uma aplicação Android pode ter várias Activities, porém o usuário só interage com uma de cada vez. As inicializações e configurações da interface com o usuário devem ser feitas sempre no método onCreate(), pois ele é o primeiro método a ser executado quando a Activity é iniciada.

Para criar uma Activity, inicialmente devemos adicionar um novo arquivo de layout configurando o arquivo de acordo com as especificações necessárias a construção da interface gráfica. Clique com botão direito do mouse no diretório Res -> Layout de sua aplicação:

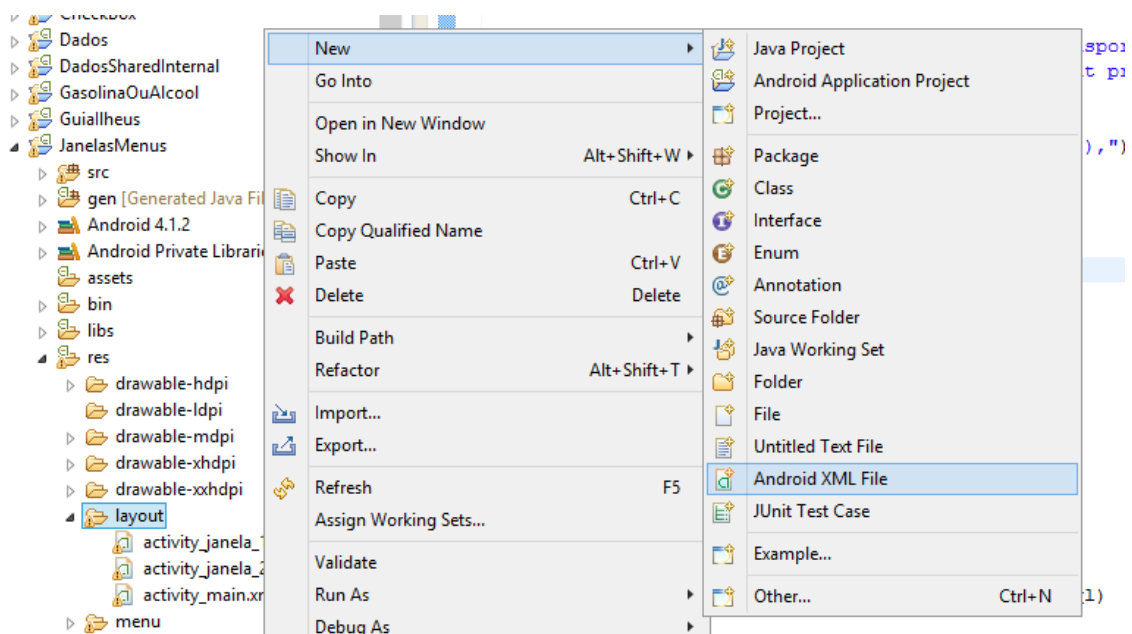


Figura 55. Adicionando um novo arquivo de layout

Em seguida devemos criar uma classe no Java que herdará a classe `android.app.Activity`. Para criar uma nova classe, devemos clicar com botão direito do mouse no diretório `src` -> pacote da aplicação e escolher a opção `New` -> `Class`.

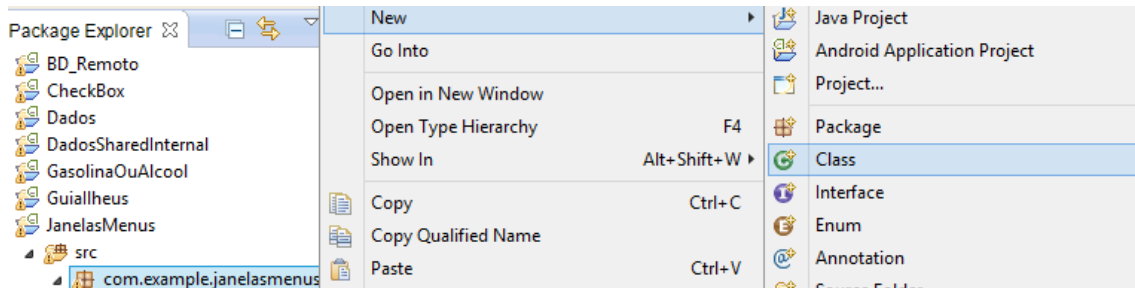


Figura 56. Criação de nova classe Java

Após estendermos a classe é obrigatório rescrever o método `onCreate` para selecionar um arquivo XML relacionado ao layout, para isto no método **`setContentView`** iremos informar qual o layout associado a classe Java.

```
setContentView(R.layout.janela2);
```

Este arquivo será carregado quando a classe for iniciada através da função `setContentView` no método `onCreate` da classe Java.

```
public class Janela2Activity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Especificamos abaixo qual o layout
        setContentView(R.layout.janela2);
    }
}
```

Após criarmos a classe é necessário registrar a activity. O Android precisa saber quais classes são atividades, não basta simplesmente criar uma classe e a estender, é necessário registra-la.

Para você registrar a activity, basta abrir o arquivo `AndroidManifest.XML`, e no mesmo deve ser feito o registro da activity, ou seja, incluir uma nova activity.

```
<activity
    android:name="com.example.aula06.Janela2Activity"
    android:label="Janela 2">
```



```
</activity>
```

Em android:name deverá ser informado o nome da classe.



OBS: As marcações `intent-filter` é utilizado para definir qual a Activity inicial de nossa aplicação.



Para demonstrar o processo de criação de vários layouts em uma aplicação iremos criar duas activities. A classe “MainActivity” é a activity (atividade) principal do projeto e a classe “Janela2Activity” que será chamada pela primeira tela.

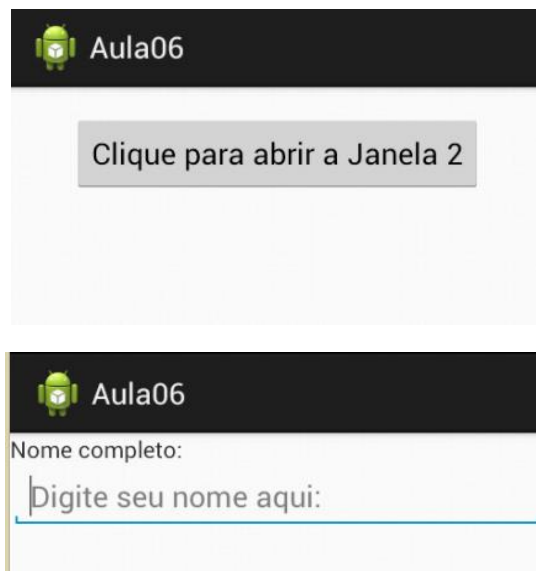


Figura 57. Exemplo de aplicação com múltiplas activities

A primeira Activity é simplesmente um botão, que quando clicado abre a segunda atividade, que por sua vez consiste em um simples campo de texto editável. A segunda activity não precisa de um botão "Sair", pois o usuário pode simplesmente clicar no botão "Voltar" do emulador ou dispositivo físico.

Para carregar uma nova Activity devemos criar uma Intent (Intenção). Intent são comandos que podemos enviar ao Sistema Operacional Android para realizar alguma ação.

Com Intents você podemos enviar ou recuperar dados. O método `startActivity(Intent intent)` envia dados e requisições de telas, o método `getIntent()` recupera uma intent enviada por meio do `startActivity()`, o método `putExtra("nome_de_identificação", valor)` insere na intent algum valor, entre outras operações.

Neste exemplo usaremos uma Intent para requisitar uma tela (activity). O código da Intent para abrir uma activity está programado no Click do botão inserido na MainActivity.

```
public void onClick(View v) {  
    //Cria uma nova intenção passando como parâmetro a classe atual  
    e a nova classe  
    Intent i = new Intent(MainActivity.this, Janela.class);  
    //inicia uma atividade passando como parâmetro a intenção criada  
    acima  
    MainActivity.this.startActivity(i);  
}
```

É possível enviar e receber dados entre as activities. Para enviar dados para uma activity que será carregada usamos o comando `putExtra` conforme o exemplo a seguir:

```
Intent dados = new  
Intent(MainActivity.this, DadosActivity.class);  
dados.putExtra("id", id);  
dados.putExtra("nome", nome);  
dados.putExtra("telefone", telefone);
```

Para receber os dados enviados por uma activity usamos o comando `getStringExtra` conforme exemplo a seguir.

```
//cria uma nova intenção para buscar os dados enviados pela  
activity anterior  
Intent valores = getIntent();  
//pega os valores enviados da activity anterior e preenche os  
campos  
String nome, telefone;  
  
nome = valores.getStringExtra("nome");  
telefone = valores.getStringExtra("telefone");
```

## 5.2. Menu e actionBar

Um menu é útil para disponibilizar acesso a diferentes partes de uma aplicação através da ActionBar. A ActionBar é a barra com o título que aparece no topo de uma aplicação

Android a partir da versão Honeycomb). Para adicionar um menu a uma aplicação Android devemos criar/utilizar um arquivo XML para os itens do menu, que fica localizado no diretório **res -> menu**.

Definem-se as opções individuais usando o elemento item. Especificam-se os atributos id, icon e title para cada opção.



OBS: Durante a criação do projeto, caso o programador tenha optado por um template com ActionBar, uma pasta menu juntamente com um arquivo .XML já estará criado no projeto.

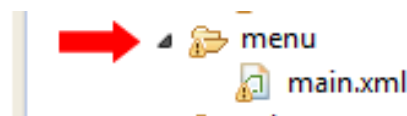


Figura 58. Diretório de menus

A criação de itens de menu é realizada editando o arquivo menu -> main.xml e adicionando itens conforme mostrado a seguir.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
>
  <item
    android:id="@+id/menuJanela2"
    android:title="Janela 2"/>
  <item
    android:id="@+id/menuJanela3"
    android:title="Janela 3"/>
  <item
    android:id="@+id/menuJanela4"
    android:title="Janela 4"/>
</menu>
```

O atributo showAsAction permite especificar quando e como é que cada opção aparece-se na ActionBar. Os valores disponíveis são: ifRoom, withText, never e always.

Depois é preciso atualizar a Activity onde o menu vai aparecer. É preciso criar o método onCreateOptionsMenu para construir o menu. Esta função já está incluída na Activity.

```
//CÓDIGO GERADO AUTOMATICAMENTE PARA CRIAR OS ITENS DO MENU
ADICIONADO EM RES -> MENU - MAIN.XML
@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;
}
```

O método `getMenuInflater().inflate` é responsável por “inflar” ou preencher um menu com os itens definidos no arquivo `.XML`. Veja como seria renderizado no Android o menu proposto.



Figura 59. Renderização de menus na ActionBar

Usamos o método `onOptionsItemSelected()` para programar a opção do menu selecionada pelo usuário. Veja código a seguir:

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    if (item.getItemId() == R.id.menuJanela2)
    {
        Toast.makeText(getBaseContext(), "Você clicou na Janela 2",
        Toast.LENGTH_SHORT).show();
        return true;
    }
    else if (item.getItemId() == R.id.menuJanela3)
    {
        Toast.makeText(getBaseContext(), "Você clicou na Janela 3",
        Toast.LENGTH_SHORT).show();
    }
    return false;
}
```

Podemos usar os itens do menu para chamar uma Activity, assim como mostrado no botão para chamar a Janela 2. Para isto usamos o método `onOptionsItemSelected()` conforme o código apresentado a seguir:

```
public boolean onOptionsItemSelected(MenuItem item)
{
    if (item.getItemId() == R.id.menuJanela2)
    {
        Intent i = new Intent(MainActivity.this, Janela.class);
        MainActivity.this.startActivity(i);
        return true;
    }
    else if (item.getItemId() == R.id.menuJanela3)
    {
        Intent i = new
            Intent(MainActivity.this, Janela2Activity.class);
        MainActivity.this.startActivity(i);
        return true;
    }
    return false;
}
```

## 6. PERSISTÊNCIA DE DADOS: SHARED PREFERENCES E INTERNAL STORAGES

Na maioria das aplicações precisamos ter algum tipo de persistência de dados. Para guardar informações de forma persistente no Android podemos usar os seguintes recursos:

- **Shared Preferences:** Armazena dados primitivos privados em pares chave-valor.
- **Internal Storage:** Armazena dados privados na memória do dispositivo.
- **External Storage:** Armazena dados públicos no cartão de memória.
- **SQLite Databases:** Armazena dados estruturados num banco de dados privado.
- **Network Connection:** Armazena dados na web no seu servidor de rede.

Neste capítulo abordaremos a persistência de dados utilizando SharedPreferences e InternalStorages.

### 6.1. Shared Preferences

A classe SharedPreferences é utilizada para armazenar informações, na qual é possível salvar entradas do tipo chave-valor, onde se associa um “nome” a uma determinada informação para que depois se possa recuperá-la através deste nome. Podemos usar SharedPreferences para salvar qualquer tipo primitivo: boolean, floats, ints, longs, e strings.

O Android salva tudo em um arquivoXML dentro da estrutura interna “de disco” em cada aplicação. O Android também oferece funções para a tarefa de salvar e depois buscar novamente o que foi salvo.

Essa opção é indicada para poucas informações e que sejam simples, como números, strings e valores booleanos. Por isso esse mecanismo é geralmente utilizado para armazenar as preferências que o usuário definiu na aplicação.

Para registrar um valor no SharedPreferences primeiro criamos (ou abrimos se já existir) o SharedPreferences informando o nome do arquivo e o modo de acesso que ele deve ter. Depois criamos um Editor, que nada mais é que uma classe auxiliar para escrever no arquivo, e salvamos o que foi passado, informando uma chave para cada informação. Por último realizamos o commit, para efetivamente escrever tudo no arquivo. Veja no código fonte a seguir o processo para registrar/escrever uma chave-valor no SharedPreferences.

```
// Cria ou abre.
SharedPreferences prefs =
getSharedPreferences("preferencia", Context.MODE_PRIVATE);
// Precisamos utilizar um editor para alterar Shared
Preferences.
Editor ed = prefs.edit();
// salvando informações de acordo com o tipo
ed.putString("USUARIO", "Regilan Silva");
ed.putString("IDADE", 30);

// Grava efetivamente as alterações.
ed.commit();
```

Mais simples que escrever no Shared Preferences é ler as informações dele. Basta acessá-lo da mesma forma e recuperar cada informação pela sua chave identificadora (o seu “nome”), informando após o que deve retornar caso nada seja encontrado para aquela chave.

```
// Acesso às Shared Preferences usando o nome definido.
SharedPreferences prefs = getSharedPreferences("preferencia",
Context.MODE_PRIVATE);

// Acesso às informações de acordo com o tipo.
String texto = prefs.getString("TEXTO", "não encontrado");
String idade = prefs.getString("IDADE", "não encontrado");
```

## 6.2. InternalStorages e ExternalStorage

A opção do Storage nada mais é que um espaço “em disco” que cada aplicação tem onde é possível salvar arquivos. Existe a opção do Internal Storage (espaço na estrutura de arquivos interna da aplicação, que é o mesmo onde fica(m) o(s) arquivo(s) de Shared Preferences) e do External Storage, que geralmente é um espaço no SDCard – podendo ser público (pastas de música ou fotos por exemplo) ou da aplicação – e nem sempre estará disponível (se o SDCard for removido, por exemplo).

Existem várias maneiras (classes) disponíveis de manipulação de arquivos, utilizaremos uma maneira mais simples de acessar dados de arquivo.

Primeiro criamos um arquivo no diretório do Internal Storage (retornado pelo método `getFilesDir()`) e depois criamos uma instância da classe auxiliar de escrita no arquivo.

Nesta classe auxiliar usamos o modo append (escreve a partir do fim do arquivo, sem apagar ou sobrescrever o que já existia antes). Escrevemos uma informação em cada linha

(escrevendo a “quebra de linha”) para facilitar a leitura posteriormente, e efetivamos a escrita das informações no arquivo.

O acesso a arquivos em disco pode gerar erros em vários pontos, por isso recomendamos o uso do tratamento de exceções (try...catch). O Eclipse automaticamente pedirá para incluir o código dentro de uma estrutura de tratamento de exceções.

A seguir apresentamos o código fonte para escrever dados em um arquivo localizado no InternalStorages:

```
// Cria o arquivo onde serão salvas as informações.
File arquivo = new File(getFilesDir().getPath()+ "/dados.txt");
// Cria a classe auxiliar para manipular arquivos.
FileWriter escrever;
escrever = new FileWriter(arquivo, true);
escrever.append(nome); // Registra um valor no arquivo.
escrever.append("\n"); // Quebra de linha.
escrever.append(idade);
escrever.append("\n"); // Quebra de linha.
// Escreve no arquivo.
escrever.flush();
// Fecha o arquivo para escrita de dados.
escrever.close();
```

Assim como ocorreu na escrita de dados, para ler dados gravados em um arquivo passamos o caminho onde criamos o arquivo e utilizamos uma classe auxiliar (dessa vez de leitura de arquivos), além de um buffer que vai nos permitir fazer a leitura linha a linha (por isso salvamos separando por linhas) sem nos preocuparmos com a quantidade que deve ser lida por vez.

Utilizamos então um “laço” para ler o arquivo até o seu fim (quando o método readLine() retornar null).



OBS: Cada aplicativo possui uma pasta com o seu nome no caminho /data/data/. Neste local encontramos todos os arquivos que foram criados para a aplicação.

O código fonte para leitura de dados em um arquivo é apresentado a seguir:

```
FileReader ler;
BufferedReader bufferDados;
```



```
String dados="";
ler = new FileReader(getFilesDir().getPath()+"/dados.txt");
bufferDados = new BufferedReader(ler);
String linha;
while ((linha = bufferDados.readLine()) != null) {
    dados += linha;
}
bufferDados.close();
```

Para utilizar o External Storage, precisaríamos apenas testar antes se ele está disponível e indicar o caminho correto na criação do arquivo. Para escrever nesse local ainda precisamos adicionar uma permissão no Manifest da aplicação.

Os arquivos salvos na memória externa podem ser lidos por qualquer outro dispositivo ou aplicação e podem ser modificados pelo usuário quando ele ativar a transferência USB ao conectar o dispositivo com um computador.

Arquivos armazenados na memória externa pode desaparecer se o usuário montar a mídia em um computador ou remover o cartão, e não existe nenhuma medida de segurança sobre os arquivos salvos nesse tipo de memória. Todas as aplicações podem ler e escrever arquivos localizados em memórias externas e o usuário pode remove-los. RECOMENDAMOS USAR SHARED PREFERENCES OU INTERNAL STORAGE para as situações apresentadas neste capítulo.



Exercício: Criar uma APP conforme ilustração a seguir e realizar a gravação de dados utilizando SharedPreferences e InternalStorages

Figura 60. Exercício sobre SharedPreferences e InternalStorages

## 7. PERSISTÊNCIA DE DADOS: BANCO DE DADOS LOCAL E REMOTO

Dando prosseguimento ao estudo sobre persistência de dados, neste capítulo abordaremos o armazenamento de dados em um banco de dados SQLite local e também em uma base de dados MySQL remota. Inicialmente abordaremos sobre a base de dados SQLite disponível no Android e como utiliza-la para armazenamento de dados.

### 7.1. Banco de dados local: SQLite

O Android fornece suporte completo ao banco de dados SQLite. Qualquer banco de dados será acessível pelo nome por qualquer classe da aplicação, mas não fora da aplicação.

O SQLite é uma biblioteca em linguagem C, open source, que implementa um banco de dados SQL embutido, ou seja, não tem processo servidor separado, lendo e escrevendo diretamente no arquivo de banco de dados no disco. O SQLite está disponível em todos os dispositivos Android, utilizá-lo não requer qualquer configuração ou administração de banco de dados, precisamos somente definir os comandos SQL para criar e atualizar o banco. Depois disso, o banco de dados é gerenciado automaticamente para você pela plataforma Android.

Quando criado, o banco de dados por padrão é armazenado no diretório /data/data/<nome-do-pacote-utilizado>/databases/nome-do-arquivo.sqlite.

Existem várias formas de acessar uma base de dados SQLite, neste caso, para efeitos didáticos e de simplicidade implementaremos nosso acesso a dados através da criação de métodos específicos na classe MainActivity.

Uma outra forma muito utilizada de acesso a dados é estender a classe **SQLiteOpenHelper** que contém uma série de métodos específicos para acesso a dados.



Para demonstrar o processo de armazenamento de dados no SQLite criaremos uma aplicação para armazenar contatos de uma pessoa, que incluirá o nome e telefone.

Nosso banco de dados chamará dados\_telefone e contará com uma tabela chamada contato, com os seguintes campos:

- `_id INTEGER PRIMARY KEY AUTOINCREMENT`
- `nome varchar(100)`
- `telefone varchar(10)`

O primeiro passo será criar um objeto do tipo SQLiteDatabase. Este objeto será de acesso global a toda a classe é através dele que criaremos e usaremos um banco de dados do SQLite.

```
public class MainActivity extends Activity {  
    SQLiteDatabase db;
```

O segundo passo será construir um método privado que será responsável pela criação do banco de dados e da tabela contato.

Para o desenvolvimento deste método usaremos o objeto db (SQLiteDatabase). Inicialmente usamos o comando openOrCreateDatabase para abrir um banco de dados existente ou criar um novo.

```
//Cria ou abre um banco de dados  
db =  
openOrCreateDatabase("dados_telefone.db", Context.MODE_PRIVATE,  
    null);
```

Em seguida usamos um objeto do tipo StringBuilder construir um comando SQL, neste caso o comando de criação das tabelas.

```
//Objeto usado para construir a STRING do comando SQL a ser  
executado  
//Neste caso a string SQL conterá o comando de criação de  
tabelas  
StringBuilder sql = new StringBuilder();  
//Obrigatoriamente tem que ter uma coluna id no banco SQL Lite  
sql.append("CREATE TABLE IF NOT EXISTS contato(");  
sql.append("_id INTEGER PRIMARY KEY AUTOINCREMENT,");  
sql.append("nome varchar(100),");  
sql.append("telefone varchar(10) )");
```



OBS: O método sql.append tem a finalidade de concatenar os comandos escritos.

Posteriormente acionaremos um comando para executar uma query armazenada em uma string (sql). O método **execSQL** executa o comando sql definido na query.

Usaremos a estrutura **try...catch** para maior consistência de nossa aplicação.

```
try
{
//executa um comando SQL, neste caso a StringBuilder SQL
db.execSQL(sql.toString());
}
catch(Exception ex)
{
Toast.makeText(getBaseContext(), "Error = " + ex.getMessage(),
Toast.LENGTH_LONG).show();
}
```

O processo acima criará no banco de dados dados\_telefone.db a tabela contato.

Antes de implementar o código relacionado a inserção de dados, construiremos uma interface gráfica composta por 2 campos (EditText) que será responsável por receber os dados que posteriormente serão armazenado no banco de dados. Além dos campos para entrada de dados, usaremos 2 botões para a realização das ações (INSERIR e MOSTRAR)



Figura 61. Interface gráfica para armazenamento de dados

Para criação do método INSERIR utilizaremos o mesmo princípio apresentando anteriormente. Criaremos um método private chamado INSERIR.

```
private void Inserir()
{
}
```

Inicialmente nosso método buscará os dados digitados pelo usuário nos campos do tipo EditText definidos no layout da interface gráfica.

```

EditText txtNome,txtTelefone;
String nome, telefone;
txtNome = (EditText)findViewById(R.id.txtNome);
txtTelefone = (EditText)findViewById(R.id.txtTelefone);
nome = txtNome.getText().toString();
telefone = txtTelefone.getText().toString();

```

Em seguida usamos um objeto do tipo StringBuilder construir um comando SQL, neste caso o comando INSERT para registrar os dados na tabela.

```

StringBuilder sql = new StringBuilder();
sql.append("INSERT INTO contato(nome,telefone) VALUES (");
sql.append("'" + nome + "'");
sql.append(",");
sql.append("'" + telefone+ "'");
sql.append(")");

```



OBS: No comando INSERT não é necessário incluir o campo `_id`, uma vez que este é um campo autoincrement, ou seja, gerado sequencialmente pelo próprio SQLite.

Posteriormente acionaremos um comando para executar uma query armazenada em uma string (sql). O método `execSQL` executa o comando sql definido na query. Usaremos a estrutura `try...catch` para maior consistência de nossa aplicação

```

try
{
    db.execSQL(sql.toString());
    Toast.makeText(getApplicationContext(), "OK - Inserido",
        Toast.LENGTH_SHORT).show();
}
catch(SQLException ex)
{
    Toast.makeText(getApplicationContext(), sql.toString() + "Erro = "
        + ex.getMessage(), Toast.LENGTH_LONG).show();
}

```

Por fim, para a chamada do método `INSERIR` criaremos um listener do tipo `OnClick` para o botão inserir, de forma que o usuário ao clicar(tocar) no botão inserir o código de inserção seja realizado.

```
View.OnClickListener inserir = new View.OnClickListener() {
@Override
public void onClick(View v) {
// TODO Auto-generated method stub
Inserir(); //CHAMADA DO MÉTODO INSERIR
}
};
```

Para codificação do botão mostrar, precisamos realizar uma busca na tabela TELEFONE e apresentar os dados armazenados na base de dados. Os dados resultantes da consulta a tabela TELEFONE serão apresentados como itens de uma ListView, sendo que um item de lista pode ser um texto simples (um TextView) ou , por exemplo, dois TextView, um contendo o nome de uma pessoa e outro o seu telefone.

Para apresentação de dados iremos criar uma ListView customizada, ou seja, criaremos um layout específico para apresentação dos dados. Este layout deverá ser criado dentro do diretório res -> layout e chamará dados.

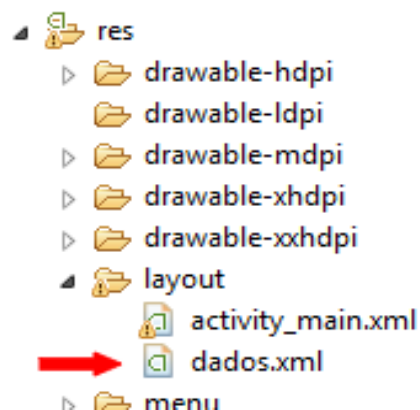


Figura 62. Local de criação de listview customizada

O layout será composto por duas TextView para apresentação dos dados(nome, telefone). Como este layout não é uma Activity, não se faz necessário registra-lo no arquivo AndroidManifest.xml.

O código de criação do layout dados.xml para apresentação dos resultados me uma ListView é mostrado a seguir:

```
<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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/tvId"
    android:visibility="invisible"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/tvNome"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/tvTelefone"/>
</LinearLayout>

```

Após a criação do Layout XML, partiremos para a programação no código JAVA do botão MOSTRAR. Da mesma forma que os códigos anteriores, definimos um método PRIVATE para obter os dados armazenados nas tabelas do banco de dados.

```

private void Buscar()
{
}

```

Em seguida usamos um objeto do tipo StringBuilder para construir um comando SQL, neste caso o comando SELECT para retornar dados na tabela. Usaremos um objeto CURSOR para armazenar os dados obtidos na consulta a tabela TELEFONE.

```

StringBuilder sql = new StringBuilder();
sql.append("SELECT * FROM contato");
Cursor dados = db.rawQuery(sql.toString(), null);

```

A conexão com os dados armazenados no Cursor e os campos criados no layout da ListView consistirá de:

- A TextView nome deverá receber os dados da coluna nome,
- A TextView telefone deverá receber os dados da coluna telefone.

Para isto usaremos um SimpleCursorAdapter. Antes de realizar a conexão dos dados através do Adapter, criaremos 2 vetores. O primeiro do tipo INT que armazenará os ID dos

campos definidos no layout da ListView; e o segundo com os nomes dos campos da consulta que serão exibidos.

```
//Array com os ID dos campos do layout dados
int[] to = {R.id.tvId,R.id.tvNome,R.id.tvTelefone};
//Array com o nome dos campos da tabela que serão mostrados
String[] from = {"_id","nome","telefone"};
```

O SimpleCursorAdapter fará a ligação entre os campos retornados da consulta e os campos (TextView) definidos no layout da ListView.

```
try
{
SimpleCursorAdapter ad = new
SimpleCursorAdapter(getApplicationContext(), R.layout.dados, dados,
from, to,0);
ListView lvDados;
lvDados = (ListView)findViewById(R.id.lvDados);
lvDados.setAdapter(ad);
}
catch(Exception ex)
{
Toast.makeText(getApplicationContext(), sql.toString() + " Erro = " +
ex.getMessage(), Toast.LENGTH_LONG).show();
}
```

Por fim, para a chamada do método Buscar criaremos um botão com o nome Mostrar e um listener do tipo OnClickListener para o botão mostrar, de forma que o usuário ao clicar-lo o código de inserção seja realizado.

```
View.OnClickListener mostrar = new View.OnClickListener() {
@Override
public void onClick(View v) {
// TODO Auto-generated method stub
Buscar(); // Chamada do método BUSCAR
}
};
```

A imagem a seguir ilustra a apresentação dos resultados após clicar no botão Mostrar.



Telefone

Nome:  
Digite seu nome aqui

Telefone:  
Digite seu telefone aqui

Inserir

Mostrar

CBA  
321

ghi  
789

def  
456

Figura 63. Visualização dos dados armazenados na tabela TELEFONE

Para criação do código referente aos comandos EDITAR e EXCLUIR, construiremos uma nova Activity, de forma que o usuário ao clicar (tocar) em um dos itens exibidos na ListView seja carregado uma nova janela com os dados selecionado e as opções de EDITAR e EXCLUIR.

Como se trata de uma nova Activity é necessário definir o Layout e em seguida criar uma classe para associar ao layout e também registrá-la no arquivo AndroidManifest.xml. A prosta de interface gráfica para as ações de EDITAR e EXCLUIR é mostrada a seguir:

Telefone

Nome:  
CBA

Telefone:  
321

Atualizar

Excluir

Figura 64. Activity para edição e exclusão

A Activity acima, que receberá os dados para edição e/ou exclusão será carregada a partir do Listener OnItemClickListener de uma ListView da Activity anterior, ou seja, ao tocar em um dos dados(Nome e Telefone) a aplicação reportará para a interface gráfica(Activity) relacionada a Edição e Exclusão do elemento selecionado. Desta forma, no listener relacionado a seleção de um item da ListView será buscado os dados selecionados e enviados para a nova Activity que exibirá os dados com as opções de EDITAR e EXCLUIR.

Para permitir a edição e exclusão de um item da lista de telefone, inicialmente desenvolveremos a programação do Listener relacionado ao clique do item selecionado na ListView. Veja no código a seguir o processo para verificar qual o item selecionado:

```
OnItemClickListener itemClicado = new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> arg0, View arg1, int
    arg2,
    long arg3) {
        // TODO Auto-generated method stub
        //A listview criada tem vários itens e é necessário saber
        //o item selecionado e buscar o id, nome e telefone relacionado
        ao item
        //O comando getChildAt pega os filhos da listview clicada.
        //Arg2 refere-se ao item selecionado
        View view = lvDados.getChildAt(arg2);
        String id,nome,telefone;
        TextView tvId = (TextView) view.findViewById(R.id.tvId);
        id = tvId.getText().toString();
        TextView tvNome = (TextView) view.findViewById(R.id.tvNome);
        nome = tvNome.getText().toString();
        TextView tvTelefone = (TextView)
        view.findViewById(R.id.tvTelefone);
        telefone = tvTelefone.getText().toString();

        //cria uma intenção para carregar uma nova activity
        Intent dados = new
        Intent(MainActivity.this,DadosActivity.class);
        //envia os dados para a activity que será carregada
        dados.putExtra("id", id);
        dados.putExtra("nome", nome);
```

```

dados.putExtra("telefone", telefone);
//inicia a activity
startActivity(dados);
}
};

```

A Activity criada para receber os dados e mostrar as opções de EDITAR e EXCLUIR, já mostrada anteriormente, deverá conter os widgets: TextView, EditText e Button. Sua apresentação visual é mostrada a seguir:



Figura 65. Activity para edição e exclusão

No método onCreate da Activity acima abrimos o banco de dados, associamos os Widgets do layout e pegamos os dados enviado pela Activity anterior.

```

//Cria ou abre um banco de dados
db =
openOrCreateDatabase("dados_telefone.db", Context.MODE_PRIVATE,
null);

//associa os campos criados na activity
tvIdEditar = (TextView) findViewById(R.id.tvIdEditar);
txtNomeEditar = (EditText) findViewById(R.id.txtNomeEditar);
txtTelefoneEditar =
(EditText) findViewById(R.id.txtTelefoneEditar);

//cria uma nova intenção para buscar os dados enviados pela
activity anterior
Intent valores = getIntent();

```

```
//pega os valores enviados da activity anterior e preenche os campos

tvIdEditar.setText(valores.getStringExtra("id"));
txtNomeEditar.setText(valores.getStringExtra("nome"));
txtTelefoneEditar.setText(valores.getStringExtra("telefone"));
```

Para criação do método EDITAR utilizaremos o mesmo princípio apresentando nos comandos INSERIR e MOSTRAR. Criaremos um método private chamado EDITAR.

```
private void Editar()
{
}
```

Inicialmente nosso método buscará os dados digitados pelo usuário nos campos do tipo EditText definidos no layout da interface gráfica.

```
//associa os campos criados na activity
tvIdEditar = (TextView) findViewById(R.id.tvIdEditar);
txtNomeEditar = (EditText) findViewById(R.id.txtNomeEditar);
txtTelefoneEditar =
(EditText) findViewById(R.id.txtTelefoneEditar);

//pega os valores de cada campo e coloca em variáveis
id = tvIdEditar.getText().toString();
nome = txtNomeEditar.getText().toString();
telefone = txtTelefoneEditar.getText().toString();
```

Em seguida usamos um objeto do tipo StringBuilder construir um comando SQL, neste caso o comando UPDATE para atualizar os dados na tabela.

```
//cria uma string SQL para atualizar o contato selecionado
StringBuilder sql = new StringBuilder();
sql.append("UPDATE contato SET nome = " + nome + "");
sql.append(",telefone = " + telefone + "");
sql.append(" WHERE _id = " + id);
Toast.makeText(getApplicationContext(), sql, Toast.LENGTH_LONG).show();
```



OBS: No comando UPDATE é necessário incluir o campo `_id`, uma vez que deve-se apenas atualizar o registro solicitado.

Posteriormente acionaremos um comando para executar uma query armazenada em uma string (sql). O método execSQL executa o comando sql definido na query. Usaremos a estrutura try...catch para maior consistência de nossa aplicação.

```
try
{
db.execSQL(sql.toString());
Toast.makeText(getApplicationContext(), "OK - Editado",
Toast.LENGTH_SHORT).show();
//volta para a janela anterior
Intent Main = new Intent(DadosActivity.this,MainActivity.class);
startActivity(Main);
}
catch(SQLException ex)
{
Toast.makeText(getApplicationContext(), sql.toString() + "Erro = " +
ex.getMessage(),
Toast.LENGTH_LONG).show();
}
```

Por fim, para a chamada do método EDITAR criaremos um listener do tipo OnClickListener para o botão EDITAR, de forma que o usuário ao clicar (tocar) no botão EDITAR o código de atualização dos dados seja realizado.

```
View.OnClickListener editar = new View.OnClickListener() {
@Override
public void onClick(View v) {
// TODO Auto-generated method stub
Editar();
}
};
```

Para criação do método EXCLUIR utilizaremos o mesmo principio apresentando nos comandos anteriores. Criaremos um método Private chamado EXCLUIR.

```
private void Excluir()
{
}
```

Inicialmente nosso método buscará o \_id relacionado ao item a ser excluído.

```
String id;
//associa os campos criados na activity
tvIdEditor = (TextView) findViewById(R.id.tvIdEditor);
```

```
//pega o Id e coloca na variável id  
id = tvIdEditar.getText().toString();
```

Em seguida usamos um objeto do tipo `StringBuilder` construir um comando SQL, neste caso o comando `DELETE` para excluir os dados na tabela.

```
//cria uma string SQL para excluir o contato selecionado  
StringBuilder sql = new StringBuilder();  
sql.append("DELETE FROM contato WHERE _id = " + id);
```



OBS: No comando `DELETE` é necessário incluir o campo `_id`, uma vez que deve-se apenas excluir o registro solicitado.

Posteriormente acionaremos um comando para executar uma query armazenada em uma string (sql). O método `execSQL` executa o comando sql definido na query. Usaremos a estrutura `try...catch` para maior consistência de nossa aplicação.

```
try  
{  
    db.execSQL(sql.toString());  
    Toast.makeText(getApplicationContext(), "OK - Excluído",  
        Toast.LENGTH_LONG).show();  
    //volta para a janela anterior  
    Intent Main = new Intent(DadosActivity.this, MainActivity.class);  
    startActivity(Main);  
}  
catch(SQLException ex)  
{  
    Toast.makeText(getApplicationContext(), sql.toString() + "Erro = " +  
        ex.getMessage(), Toast.LENGTH_LONG).show();  
}
```

Por fim, para a chamada do método `EXCLUIR` criaremos um listener do tipo `OnClick` para o botão `EXCLUIR`, de forma que o usuário ao clicar (tocar) no botão `EXCLUIR` o código de exclusão dos dados seja realizado.

```
View.OnClickListener excluir = new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        Excluir();  
    }  
};
```



Figura 66. Resultado da exclusão

## 7.2. Banco de dados remoto: MySQL, PHP e JSON

O objetivo deste tópico é desenvolver uma aplicação para dispositivos com sistema operacional Android a partir do qual será possível buscar e enviar dados em um banco de dados na Internet. O desenvolvimento da aplicação que conectará a uma base de dados na Internet será separada em duas partes:

- Criação de serviços web em PHP para acesso a base dados
- Criação de uma aplicação Android

Focaremos no desenvolvimento de uma aplicação Android. Os serviços PHP já estão criados e serão informados pelo professor durante a aula. Para realização de conexão com base de dados MySQL usaremos o PHP e objetos JSON.

O servidor web onde está armazenado a base de dados será composto de uma série de scripts PHP que se encarregam de ser intermediários entre o aplicativo no Android e banco de dados MYSQL no servidor WEB. Podemos também fazer com que o dispositivo fique encarregado em todo este processo, porém como em geral os dispositivos podem ter baixo poder de processamento o recomendável é disponibilizar rotinas em um servido web utilizando uma linguagem de programação como por exemplo o PHP que ficará encarrado de executar ou retornar dados a partir de um comando enviado de um dispositivo Android.

Através destes scripts será possível fazer consultas usando o método HTTP GET, que irá responder com objetos JSON, que conterá os dados retornados da consulta SQL.

Não iremos apresentar a programação em PHP pois a programação é PHP não é a proposta desta apostila, mas vamos considerar que foram programadas e estão disponibilizadas em um servidor web as seguintes páginas .php:

- **cliente\_inserir.php:** Irá registrar os dados recebidos em uma tabela. O script recebe os parâmetros através de uma solicitação HTTP GET, e execute um comando (INSERT,UPDATE ou DELETE) e retorna um true se a operação foi bem sucedida, caso contrário, uma mensagem de erro é enviada.
- **cliente\_selecionar\_por\_id:** receberá como parâmetro um número identificador e retorna uma matriz JSON que contém o resultado da consulta SQL.
- **cliente\_selecionar\_todos:** retorna uma matriz JSON que contém o resultado da consulta SQL.

O desenvolvimento da aplicação Android que acessa os scripts em PHP consistirá dos seguintes passos:

- 1 – Criação de um novo projeto no Eclipse – Android
- 2 – Criação do layout XML, como mostrado ao lado
- 3 – Adicionar permissão de Internet no arquivo AndroidManifest
- 4 – Adicionar a biblioteca JSON – 1.0.jar ao projeto (pasta libs)
- 5 – Criar uma classe para conexão ao servidor WEB que executará as rotinas de acesso a dados.
- 6 – Programar a classe MainActivity para chamar os métodos criados na classe de conexão ao servidor WEB

O 1º passo envolve a criação de um novo projeto no Eclipse Android. Em seguida precisamos acessar o layout.xml e definir a interface gráfica deste aplicativo que constará de:

**3 TextView:** usado para passar informações ao usuário na tela

**3 EditText:** receberá os valores a serem incluídos e pesquisado no banco de dados

**3 Button:** acionará os eventos: Enviar, Obter e Pesquisar

**1 ListView:** receberá os dados vindos do banco de dados que serão apresentados em formato de lista

A proposta de interface gráfica é apresentada a seguir:



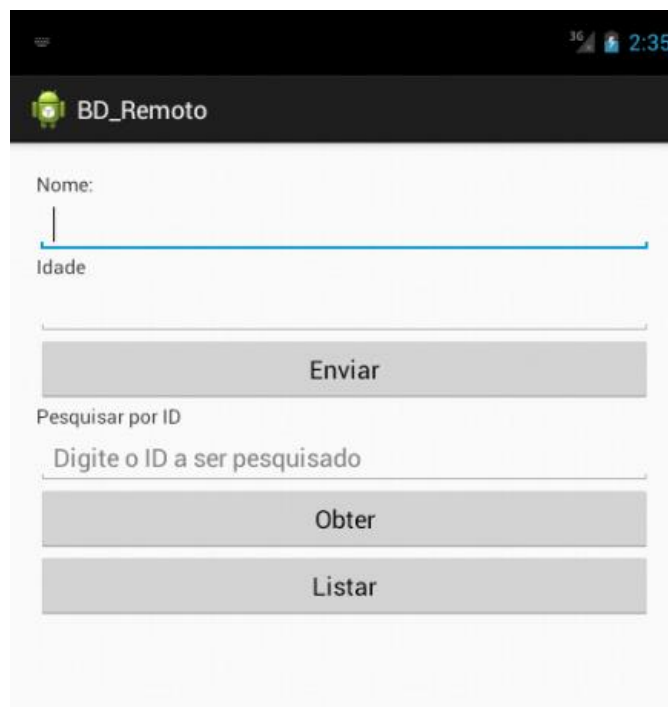


Figura 67. Interface gráfica para aplicação de acesso a base de dados na internet

Após codificação do layout xml, precisamos adicionar uma permissão de internet no arquivo AndroidManifest do projeto. Esta permissão faz-se necessária pois iremos invocar uma página web a partir de uma conexão HttpClient do Android.

Podemos adicionar a permissão acessando o arquivo AndroidManifest e adicionando a seguinte linha de código:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

O próximo passo é adicionar o arquivo JSON - 1.0.jar no diretório libs do projeto. Faremos isto copiando este arquivo e colando dentro do diretório libs.

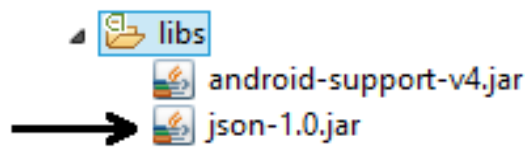


Figura 68. Adicionando bibliotecas externas

Nesta próxima etapa iremos criar uma classe responsável pela conexão via HttpClient de uma página PHP que fará uma consulta a uma base de dados MySQL. Para criar uma classe

clicamos com o botão direito do mouse no diretório src e na opção new escolhemos Class. Daremos o nome a esta classe de ConexaoHttpClient.

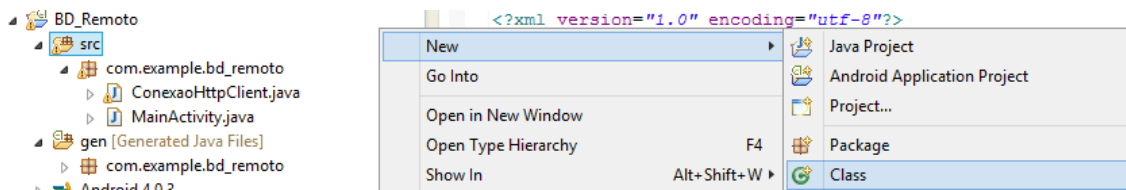


Figura 69. Adicionando uma nova classe ao projeto

Esta classe possuirá três métodos:

- `private String httpGetData(String mURL)`: Método privado responsável pela comunicação de dados com uma página web
- `public boolean Enviar(String url)`: Método para enviar os dados a partir de uma URL - Via GET
- `public JSONArray Obter(String url)`: Método para obter os dados a partir de uma URL

O método `httpGetData` é privado, ou seja, só é executado dentro da própria classe. Os métodos `Enviar` e `Obter` são públicos e se comunicam com o método `httpGetData` para enviar e retornar informações do banco de dados. A codificação para o método `httpGetData` é mostrada a seguir:

```
//Método privado responsável pela comunicação de dados com uma
página web
private String httpGetData(String mURL) {
    StrictMode.ThreadPolicy policy = new
    StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);
    String response="";
    mURL=mURL.replace(" ", "%20");
    Log.i("LocAndroid Response HTTP Threas","Ejecutando get 0:
    "+mURL);
    HttpClient httpclient = new DefaultHttpClient();
    Log.i("LocAndroid Response HTTP Thread","Ejecutando get 1");
    HttpGet httpget = new HttpGet(mURL);
    Log.i("LocAndroid Response HTTP Thread","Ejecutando get 2");
    try {
        Log.i("LocAndroid Response HTTP","Ejecutando get");
        // Execute HTTP Post Request
        ResponseHandler<String> responseHandler=new
        BasicResponseHandler();
        response = httpclient.execute(httpget,responseHandler);
        Log.i("LocAndroid Response HTTP",response);
    }
}
```

```

catch (ClientProtocolException e) {
Log.i("LocAndroid Response HTTP ERROR 1",e.getMessage());
// TODO Auto-generated catch block
} catch (IOException e) {
Log.i("LocAndroid Response HTTP ERROR 2",e.getMessage());
// TODO Auto-generated catch block
}
catch (Exception e) {
// TODO: handle exception
Log.i("LocAndroid Response HTTP ERROR 3",e.getMessage());
}
return response;
}

```

O método Enviar receberá uma URL com os dados a ser enviado para uma página via método GET, como exemplo: nomeDaPagina.php?dados1=valor1&dados2=valor2

```

/* Método para enviar os dados a partir de uma URL - Via GET
Este método recebe uma URL com os dados a ser enviados para
página via GET,
ou seja, junto com o endereço web: ex. pagina.php?dados=valor
O método retorna um booleano - se tudo der certo retorna true,
caso erro retorna false */
public boolean Enviar(String url)
{
try{
httpGetData(url);
return true;
}catch(Exception e){
return false;
}
}

```

O método Obter recebe uma URL que quando executada retornar um objeto JSON com os dados de um SELECT. Ao final a classe retorna este objeto JSON com os dados. Caso aconteça erro ou mesmo não exista os dados a partir de uma consulta o objeto retorna vazio.

```

/* Método para obter os dados a partir de uma URL
Este método recebe uma URL que quando executada retornar um
objeto JSON com os dados de um SELECT. Ao final a classe retorna
este objeto JSON com os dados. Caso aconteça erro ou mesmo não
exista os dados a partir de uma consulta o objeto retorna Vazio
*/
public JSONArray Obter(String url)
{
JSONArray ja=null;
try {
String data;

```

```

        data=httpGetData(url);
if(data.length()>1)
        ja=new JSONArray(data);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return ja;
}

```

Após a criação da classe ConexaoHttpClient e definição dos métodos, devemos programar os botões: Enviar, Obter e Buscar.

Para cada botão será criado um Listener do tipo OnClickListener

- OnClickListener enviar = new OnClickListener()
- OnClickListener obter = new OnClickListener()
- OnClickListener listar = new OnClickListener()



OBS: Antes de realizar os listeners, deve-se declarar os objetos e no método onCreate da classe MainActivity relacionar os objetos Java com os componentes usados no layout XML. Exemplo:

```

//Associando os objetos JAVA com os objetos do Layout XML
edt_nome = (EditText)findViewById(R.id.edt_nome);
btEnviar = (Button)findViewById(R.id.btEnviar);
//Associando o evento Click aos botões
btEnviar.setOnClickListener(enviar);

```

O código referente ao onClickListener do botão Enviar consistirá de:

```

try{
ConexaoHttpClient conexao = new ConexaoHttpClient(); //Criação
do objeto que fará a conexão com a página em PHP+MySQL
//Chamando o método ENVIAR com a URL e os dados via GET que são
os valores das 2 editext
boolean retorno =
conexao.Enviar("http://www.ALGUM_SITE.com.br/cliente_inserir.php
?nome="+edt_nome.getText().toString() +
"&idade="+edt_idade.getText().toString());
if (retorno == true) //Caso verdadeiro = Sucesso!
{
    Toast.makeText(getApplicationContext(), "Registro inserido!",
    Toast.LENGTH_LONG).show();
    //Limpa as editext para serem adicionados mais valores
    edt_idade.setText("");
    edt_nome.setText("");
}
}

```

```

} catch (Exception e) {
//Mensagem de ERRO!
Toast.makeText(getApplicationContext(), "Erro no envio dos
dados. " + e.getMessage(), Toast.LENGTH_LONG).show();
}

```

O código referente ao onClickListerner do botão Obter consistirá de:

```

// Criação do Array que receberá os dados da página PHP a ser
executada
JSONArray ja = new JSONArray();
// Criação do objeto que irá realizar a conexão e execução dos
dados em uma página PHP
ConexaoHttpClient conexao = new ConexaoHttpClient();
JSONObject ja_dados; //objeto que receberá uma linha de dados do
JSON Array
try
{
// Método para obter os dados de uma página PHP - É passado uma
URL com o id via método GET
ja =
conexao.Obter("http://www.ALGUM_SITE.com.br/cliente_selecionar_p
or_id.php?id="+edt_pesquisar_id.getText());
for (int i = 0; i < ja.length(); i++) {
ja_dados = ja.getJSONObject(i);
//Preenche as edittext com os dados retornados e armazenados no
JSON Array
edt_nome.setText(ja_dados.getString("nome"));
edt_idade.setText(ja_dados.getString("idade"));
}
}
catch (JSONException e) //Caso aconteça erro
{
edt_nome.setText("");
edt_idade.setText("");
Toast.makeText(getApplicationContext(), "Houve um erro durante a
conexão de dados. Error= " +
e.getMessage(), Toast.LENGTH_LONG).show();
}

```

Por fim temos o código referente ao onClickListerner do botão Listar que consistirá de:

```

// Criação do Array que receberá os dados da página PHP a ser
executada
JSONArray ja = new JSONArray();
// Criação do objeto que irá realizar a conexão e execução dos
dados em uma página PHP
ConexaoHttpClient conexao = new ConexaoHttpClient();

```

```

JSONObject ja_dados; //Como o JSON array retorna várias linhas
de dados
//usamos um JSONObject para pegar os valores de cada linha de
dados
try
{
// Método para obter os dados de uma página PHP - É passado uma
URL com o id via método GET
ja =
conexao.Obter("http://www.ALGUM_SITE.com.br/cliente_selecionar_t
odos.php");
ArrayList<String> listaDados = new ArrayList<String>(); //Lista
com os dados obtidos
//percorre todas as linhas do JSONArray que foi gerado a partir
da execução da página PHP
for (int i = 0; i < ja.length(); i++)
{
ja_dados = ja.getJSONObject(i); //Pega uma linha de dados
String texto = ja_dados.getString("nome"); //obtem um dado(neste
caso o nome)
listaDados.add(texto); //adiciona o dado a uma lista
}
ArrayAdapter<String> adapter; //adaptador de dados para fazer a
conexão entre os dados e o listView
adapter = new
ArrayAdapter<String>(getBaseContext(), android.R.layout.simple_li
st_item_1, listaDados);
lvDados.setAdapter(adapter); //atualiza a fonte de dados da
listview
}
catch (JSONException e) //Caso aconteça erro
{
Toast.makeText(getBaseContext(), "Houve um erro durante a
conexão de dados. Error = " +
e.getMessage(), Toast.LENGTH_LONG).show();
}
}

```

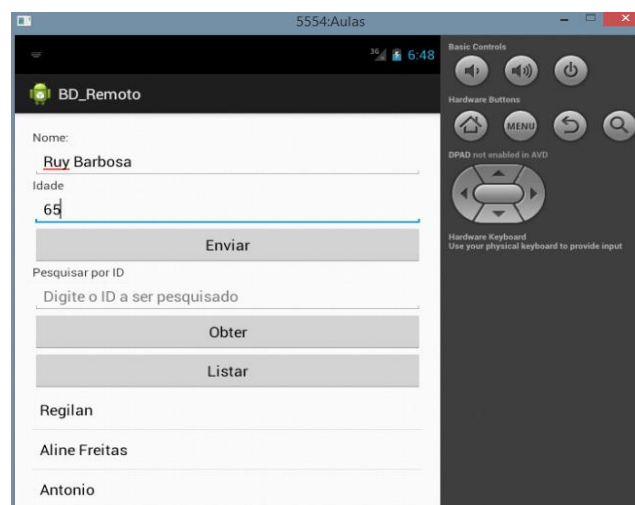


Figura 70. Exemplo do funcionamento da aplicação para acesso a base de dados na internet

## 8. RECURSOS DO DISPOSITIVO: CÂMERA, SMS, CHAMADAS E LOCALIZAÇÃO

Nesta capítulo iremos aprender como invocar recursos de um dispositivo Android para:

- Fazer chamadas telefônicas
- Enviar SMS
- Utilizar a câmera
- Obter localização

### 8.1. Chamadas telefônicas

Antes de criar uma intenção (INTENT) para fazer uma ligação precisamos da permissão do Sistema Operacional e para isso no arquivo AndroidManifest.xml devemos colocar a permissão android.permission.CALL\_PHONE, veja sua implementação completa abaixo:

```
<uses-permission android:name="android.permission.CALL_PHONE">
</uses-permission>
```

Em seguida criamos uma Intent passando no seu construtor o tipo de ligação, ACTION\_DIAL ou ACTION\_CALL.

ACTION\_DIAL abre a janela com a opção de alterar o telefone, já ACTION\_CALL, faz a ligação direta. O código a seguir representa uma intenção para realização de uma chamada telefônica.

```
//Tipos de chamadas
//ACTION_DIAL
//ACTION_CALL

Intent chamada = new Intent(Intent.ACTION_DIAL);

//altera o número do telefone
Uri uri = Uri.parse("tel:"+telefone);
chamada.setData(uri);
startActivity(chamada);
```

### 8.2. Envio de SMS

A plataforma Android fornece métodos de envio de SMS pela aplicação. Podemos enviar mensagem SMS de duas formas:

- 1ª forma: realizando uma chamada do aplicativo de mensagens com o texto que queremos passar e/ou o número;
- 2ª forma: Enviar o SMS direto da sua aplicação

Para usar esses métodos é necessário declarar uma permissão no AndroidManifest:

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Utilizando a 1ª forma, o que precisamos fazer é apenas chamar o aplicativo de mensagens com o texto que queremos passar e/ou o número:

```
Intent smsIntent = new Intent(Intent.ACTION_VIEW);  
smsIntent.setType("vnd.android-dir/mms-sms");  
smsIntent.putExtra("address", "NUMERO DO TELEFONE");  
smsIntent.putExtra("sms_body", "MENSAGEM A SER ENVIADA");  
startActivity(smsIntent);
```

Feito isso a mensagem e número é passado para outra aplicação e não nos preocupados com o envio propriamente dito.

A segunda forma de envio, seria encaminhar o SMS direto da aplicação. Precisamos para isto utilizar um objeto do tipo SmsManager.

```
SmsManager smsManager = SmsManager.getDefault();
```

Depois utiliza o seguinte comando se você tiver certeza que sua mensagem terá menos de 160 caracteres:

```
smsManager.sendTextMessage("07380808080", null, "Mensagem que  
estou enviando", null, null);
```

Se você não tem certeza se sua mensagem será menor que 160 caracteres use o seguinte método:



```
smsManager.sendMultipartTextMessage ("07187853344", null,  
smsManager.divideMessage("Mensagem muito grande que estou  
enviando"), null, null);
```

Grande parte dos dispositivos Android possui uma ou mais câmeras, que permitem tirar fotos e gravar vídeos. Podemos utilizar este recurso na programação de nossas APPs.

A integração de uma aplicação com a câmera de um dispositivo pode ser feita de duas formas. A primeira, mais simples, utiliza a aplicação de câmera nativa que já existe no Android. E a segunda permite que o desenvolvedor tenha controle total do hardware da câmera. As duas formas serão abordadas na sequência.

Utilizaremos a forma mais simples uma vez que a aplicação já existente no dispositivo já está pronta para utilização.

### **8.3. Acesso a câmera**

Grande parte dos dispositivos Android possui uma ou mais câmeras, que permitem tirar fotos e gravar vídeos. Podemos utilizar este recurso na programação de nossas APPs.

A integração de uma aplicação com a câmera de um dispositivo pode ser feita de duas formas. A primeira, mais simples, utiliza a aplicação de câmera nativa que já existe no Android. E a segunda permite que o desenvolvedor tenha controle total do hardware da câmera. As duas formas serão abordadas na sequência.

Utilizaremos a forma mais simples uma vez que a aplicação já existente no dispositivo já está pronta para utilização.

A forma mais simples de integrar uma aplicação com a câmera é chamar a aplicação nativa de câmera do dispositivo. A chamada de outras aplicações no Android é feita através do disparo de uma intent para a plataforma, que fica responsável por carregar a aplicação.

Para exemplificar a utilização da câmera iremos criar uma aplicação simples que fará uma chamada ao aplicativo da câmera e após a foto ser tirada a mesma será visualizada em uma ImageView.



OBS: Este exemplo só funcionará em um dispositivo físico com Android.

A aplicação consistirá de uma interface gráfica apresentada a seguir com os seguintes elementos:

- 1 Button para invocar a câmera física do dispositivo
- 1 Label com o texto “Resultado”
- 1 ImageView para receber a foto obtida pela câmera.



Figura 71. Interface gráfica para utilização de câmera fotográfica

O código XML da aplicação acima é representado por:

```
<Button
    android:id="@+id/btAbrirCamera"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/lbChamar" />
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/lbResultado"/>
<ImageView
    android:id="@+id/ivFoto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

A programação Java consistirá de 2 classes:

- MainActivity: classe relacionada ao layout XML

- **usarCamera**: classe criada para codificar os métodos de manipulação da Câmera

A classe **usarCamera** consistirá de 2 métodos:

- **public void carregarFoto (String nomeArquivoFoto, ImageView ivFoto)**: Esta classe carrega um arquivo de imagem em uma **ImageView**
- **public Intent capturarFoto(String nomeArquivoFoto)**: Esta classe captura uma imagem retirada na câmera do dispositivo e retorna a imagem retirada com o nome passado no parâmetro do método(**nomeArquivoFoto**)

O código fonte do método **carregarFoto**, utilizado para exibir um arquivo de imagem em uma **ImageView** é representado a seguir:

```
public void carregarFoto(String nomeArquivoFoto, ImageView ivFoto)
{
    //Local onde será buscado a foto a ser carregada na ImageView
    //Environment.DIRECTORY_PICTURES pega o diretório Pictures do Android
    String local = Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES) + "/" + nomeArquivoFoto;
    //Cria uma imagem a partir de um caminho onde se encontrar um arquivo
    Bitmap imagem = BitmapFactory.decodeFile(local);
    //Altera um imageView para uma nova imagem, neste caso a imagem com o caminho especificado acima
    ivFoto.setImageBitmap(imagem);
}
```

O código fonte para capturar uma foto de uma câmera fotográfica é representada a seguir:

```
public Intent capturarFoto(String nomeArquivoFoto)
{
    //Cria uma intenção de Capturar de Imagem ou seja, usar a camera
    Intent i = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Environment.DIRECTORY_PICTURES pega o diretório Pictures do Android
    //Usaremos este local para armazenar as imagens
    File picsDir = Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES);
    File imageFile = new File(picsDir, nomeArquivoFoto);
    i.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(imageFile));
    // Arquivo a ser armazenado
    // Inicia a captura da imagem
}
```

```
return i;
}
```

A classe MainActivity além das declarações de objetos, associações entre objetos Java e layout Xml (findViewById), teremos:

- 1 Listener para chamar a câmera ao tocar no botão CHAMAR
- 1 Método onActivityResult para para pegar o resultado gerado após ser tirado uma foto, ou seja, carregar a foto em uma ImageView.

O listener relacionado ao botão CHAMAR utilizará a classe usarCamera e o método capturarFoto() para abrir o aplicativo de câmera fotográfica.

```
View.OnClickListener abrirCamera = new View.OnClickListener() {
@Override
public void onClick(View v) {
//Cria a classe usarCamera
usarCamera c = new usarCamera();
//Chamada do método CapturarFoto com o nome do arquivo usado
quando for gravado a foto
//Este método retornar uma intenção para que depois seja
iniciada a captura da imagem
Intent i = c.capturarFoto("regilan.jpg");
//Inicia a intenção ou seja, captura a imagem
startActivity(i);
//Toast.makeText(getBaseContext(), "Mensagem = " + mensagem,
Toast.LENGTH_LONG).show();
}
};
```

Em seguida codificaremos o método onActivityResult() para carregar a fotografia retirada quando a foi chamado o método capturarFoto()

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
super.onActivityResult(requestCode, resultCode, data);
Toast.makeText(getBaseContext(), "requestCode = " + requestCode
+ " resultCode = " + resultCode, Toast.LENGTH_LONG).show();
if (requestCode == 100) { //requestCode = 100 para Imagem
if (resultCode == RESULT_OK) { //resultCode = OK se tudo deu
certo
//Cria a classe usarCamera
```

```

        usarCamera c = new usarCamera();
        //Chama o método para carregar uma foto no ImageView
        c.carregarFoto("regilan.jpg", ivFoto);
    } else if (resultCode == RESULT_CANCELED) {
        Toast.makeText(getBaseContext(), "Erro ao carregar a
foto!", Toast.LENGTH_LONG).show();
    } else {}
    }
}

```

No método onCreate da MainActivity ainda deve conter a chamada ao método OnClickListener. Também será feita uma chamada do método carregarFoto() para que ao ser iniciado o aplicativo, uma foto seja carregada na ImageView.

```

//Relacionamento dos objetos no Java com o Arquivo de Layout
ivFoto = (ImageView) findViewById(R.id.ivFoto);
btAbrirCamera = (Button) findViewById(R.id.btAbrirCamera);
//Associar o Click ao botão de Abrir Camera
btAbrirCamera.setOnClickListener(abrirCamera);
//Cria a classe usarCamera
usarCamera c = new usarCamera();
//Chama o método para carregar uma foto no ImageView
c.carregarFoto("regilan.jpg", ivFoto);

```

#### 8.4. Recursos de localização

Um dos recursos bastante interessantes dos smartphones mais modernos é a presença do GPS. GPS é o acrônimo para sistema de posicionamento global, é um sistema que permite ao dispositivo determinar a posição exata do aparelho no globo terrestre segundo coordenadas passadas por um dos satélites mundiais.

Para apresentar como obter informações de coordenadas geográfica do GPS, iremos desenvolver um exemplo com 1 botão, 2 textView e 2 EditText, que serão carregados com as informações de GPS. A visualização de nosso aplicativo deve ser semelhante a esta:



Figura 72. Recursos de localização

O código XML da aplicação acima é representado a seguir:

```
<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="${packageName}.${activityClass}"
android:orientation="vertical"
>

<TextView
    android:id="@+id/textView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Latitude" />

<EditText
    android:id="@+id/edtLatitude"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
</EditText>

<TextView
    android:id="@+id/textView2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Longitude" />

<EditText
    android:id="@+id/edtLongitude"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

<Button
    android:id="@+id/btLocalizar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Obter localização" />

</LinearLayout>
```

No código JAVA faremos a programação para quando o botão ObterLocalização for tocado, seja carregado nas EditText as coordenadas relacionadas a localização do dispositivo.

A lógica necessária para que o nosso aplicativo funcione consistirá basicamente em dois passos:

- Criação de um método chamado startGPS que usará os recursos de localização para obter as coordenadas de latitude e longitude
- Criação de um listener do tipo OnClick para ficar associado ao toque do botão

O código fonte JAVA para obter as coordenadas GPS é apresentado a seguir:

```
public class MainActivity extends Activity {
    EditText edtLatitude, edtLongitude;
    Button btLocalizar;

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

        //Associação dos objetos JAVA com os componentes do layout
        XML
        edtLatitude = (EditText) findViewById(R.id.edtLatitude);
        edtLongitude = (EditText) findViewById(R.id.edtLongitude);

        btLocalizar = (Button) findViewById(R.id.btLocalizar);
        btLocalizar.setOnClickListener(buscarCoordenadas);
    }

    OnClickListener buscarCoordenadas = new OnClickListener() {

        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            //Chama o método startGPS para obter uma localização
            startGPS();
        }
    };

    //Método que faz a leitura de fato dos valores recebidos do GPS
    public void startGPS() {
        LocationManager lManager =
        (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        LocationListener lListener = new LocationListener() {
            public void onLocationChanged(Location locat) {
                //Obtem as coordenadas de latitude e longitude
                Double latitude = locat.getLatitude();
                Double longitude = locat.getLongitude();

                edtLatitude.setText(latitude.toString());
                edtLongitude.setText(longitude.toString());
            }
            public void onStatusChanged(String provider, int status,
            Bundle extras) {}
            public void onProviderEnabled(String provider) {}
            public void onProviderDisabled(String provider) {}
        };
        lManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
        0, 0, lListener);
    }
}
```

Antes de executar a aplicação devemos adicionar as permissões de acesso ao GPS do celular no arquivo AndroidManifest.xml. Para isto, devemos inserir estas linhas de código:

```
<uses-permission  
android:name="android.permission.ACCESS_FINE_LOCATION"/>  
<uses-permission  
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

O aplicativo em execução é mostrado a seguir:

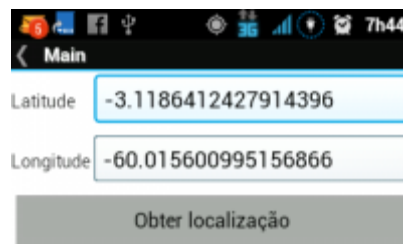


Figura 73. Visualização de localização através de GPS

Para finalizar iremos incrementar nossa aplicação adicionando um botão que mostrará no GoogleMaps a nossa localização. Para isto adicione ao layout um novo botão de forma que a interface gráfica fique semelhante a imagem a seguir:

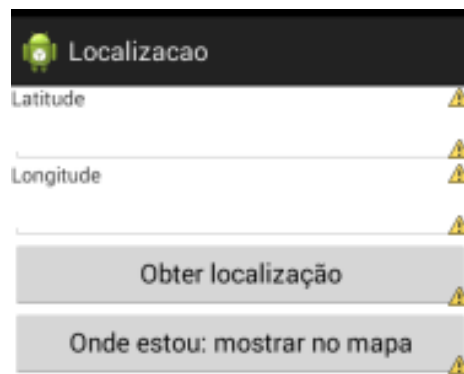


Figura 74. Interface gráfica para abrir o GoogleMaps com uma localização

Em seguida, no código JAVA, usaremos uma intent com o código necessário para chamar o GoogleMaps com a localização obtida através do botão localização e mostrada na EditText Latitude e Longitude. Veja a seguir como fica esta programação:

```
OnClickListener ondeEstou = new OnClickListener() {  
  
    @Override  
    public void onClick(View v) {
```



```
        // TODO Auto-generated method stub
        String label = "Minha Localização ";
        String uriBegin = "geo:" + latitude + "," +
longitude;
        String query = latitude + "," + longitude + "(" +
label + ")";
        String encodedQuery = Uri.encode(query);
        String uriString = uriBegin + "?q=" + encodedQuery +
"&z=16";
        Uri uri = Uri.parse(uriString);
        Intent intent = new
Intent(android.content.Intent.ACTION_VIEW, uri);
        startActivity(intent);
    }
};
```