

COLEÇÃO – PROFISSIONALIZANTE – EQUIPE PRAXIS

Coordenação Geral

Priscila Isaías da Silva

Coordenação Pedagógica

Maiara Caroline Braz Sobrinho

Projeto Gráfico, Editoração

Matheus Carrara de Oliveira Gerth

Revisão

Equipe Praxis

Montagem da Capa

Leonardo Moreira Campos

2023

III Caros alunos e alunas,

Sejam bem-vindos à nossa apostila. Nosso objetivo é fornecer um guia expositivo e norteador para o assunto em questão. Este material foi elaborado com o objetivo de fornecer informações claras e precisas sobre o assunto, de forma a permitir que o estudante comprehenda os conceitos e teorias básicas.

No entanto, é importante destacar que alguns dos conteúdos apresentados nesta apostila não incluem a parte prática, pois o foco principal, como já dito anteriormente, é fornecer uma base sólida e teórica sobre o assunto. Assim, é recomendável que o estudante complemente seus estudos com outros materiais sugeridos pelos professores.

Esperamos que esta apostila seja de grande ajuda para o seu aprendizado e compreensão sobre o assunto, e não hesite em entrar em contato conosco caso tenha alguma dúvida ou sugestão.

Atenciosamente,

Coordenação Pedagógica Qualifica DF

Equipe Praxis

Sumário Desenvolvedor de Aplicativo para Android

Dispositivos Móveis.....	5
Plataformas Mobile.....	6
Introdução ao Android	7
Facilidades Android.....	8
Principal ferramenta de desenvolvimento Android: Android Studio.....	8
Vamos conhecer um pouco mais da estrutura de nosso projeto.....	19
Melhorando o nosso primeiro projeto.....	20
Estudando a Activity.....	23
Ciclo de Vida Activity.....	24
Componentes gráficos.....	25
Conhecendo alguns Tipos de Layouts.....	27
Desenvolvendo uma Calculadora.....	28
Aprimorando o nossa calculadora.....	38
Intent.....	45
Intent Filter.....	47
Projeto com Intent.....	48
Listas.....	52
Menus.....	54
Menu de opção e barra de opção:.....	55
Menu de contexto:.....	55
Menu Popup.....	55
Novo Projeto Utilizando Menu.....	56
Popup Menu.....	67
Fragment.....	70
Ciclo de vida de Fragment.....	71
Criando Fragments:.....	73
Estilo e Tema.....	78
Herança de Estilo.....	79
AlertDialog.....	80
Classe MediaPlayer	85
Persistência de Dados em Android.....	88
SharedPreferences.....	89
Internal Storage.....	91
External Storage.....	92
SQLite Databases.....	94
Revisão geral	97
Google Play Services.....	164
Criando um Arquivo de distribuição.....	165

Criando um arquivo APK:	165
Publicando seu aplicativo na Google Play Store	167
Exercícios Extras	169
Projeto CursosPraxis	178
Projeto Posto Fácil	191
Projeto LembraTexto	198
Projeto Memes	201
Projeto CompartilhaText	204
Referencial Bibliográfico	208



Dispositivos Móveis.

A computação móvel, consiste em:

"Ter capacidade de acesso a informação a qualquer lugar e a qualquer hora, onde existe total mobilidade do usuário".

Com o avanço das tecnologias de TI, nasceram uma gama de novidades relativas à infraestrutura da computação móvel como hardwares, softwares, redes de computadores e outros. Entre eles, podemos destacar os dispositivos móveis.

Um dispositivo móvel(handheld) é um computador de bolso normalmente equipado com uma pequena tela(output) e um teclado em miniatura (input), ou como na maioria dos dispositivos hoje encontrados, com telas TouchScreen(tela sensível ao toque) que são responsáveis pela entrada e saída de informações.

O conceito base que impulsionou o desenvolvimento dos dispositivos móveis foi a mobilidade, que pode ser definida como:

"Capacidade de poder ser movido fisicamente e/ou ser utilizado enquanto está sendo movido".

Para isto, possui determinadas características como:

Pequeno em tamanho;

Leves em peso;

Capacidade de memória e processamento limitados;

Baixo consumo de energia;

Podem possuir conectividade ou não, ou ainda, conectividade limitada;

Curto tempo de inicialização;

Normalmente mais resistentes a quedas;

Monitoramento de nível de energia para prevenção de perda de dados;

Armazenamento de dados local e/ou remoto;

Sincronização de dados com outros sistemas;



Plataformas Mobile.

Com o crescimento no uso de dispositivos móveis ao redor do globo, surgiram vários sistemas operacionais e também uma crescente demanda por aplicativos.

Mas que tipo de aplicativo construir?

Específico para um determinado sistema operacional? Voltado para Web? Genérico?

Existem muitos tipos de sistemas operacionais para dispositivos móveis, para se iniciar um projeto é necessário definir a forma em que será desenvolvida a aplicação móvel. Existem várias formas, cito alguns exemplos a seguir:

Nativa: São desenvolvidas especificamente para uma determinada plataforma. Faz uso da linguagem de programação suportada pela plataforma móvel e seu respectivo SDK(Software Development Kit). Normalmente são instaladas através de uma loja de aplicativos, como por exemplo App Store e Google Play.

WebMobile: Diferente das aplicações nativas, o aplicativo consiste em um site com um layout otimizado para dispositivos. Faz uso de linguagens web(Html, Css, Javascript) e design responsivo. É desenvolvido para multiplataforma.

Híbrida: Tem se destacado muito nestes últimos tempos. Este consiste na combinação dos tipos nativos e WebMobile. Em geral, possui um navegador de internet customizado para o site do aplicativo. É desenvolvido para uma plataforma específica, por isto, deve ser baixado em uma loja de aplicativos específica.

Multiplataforma: Faz uso de ferramenta proprietária(Framework) para geração de aplicações móveis para várias plataformas móveis.

Principais plataformas de desenvolvimento

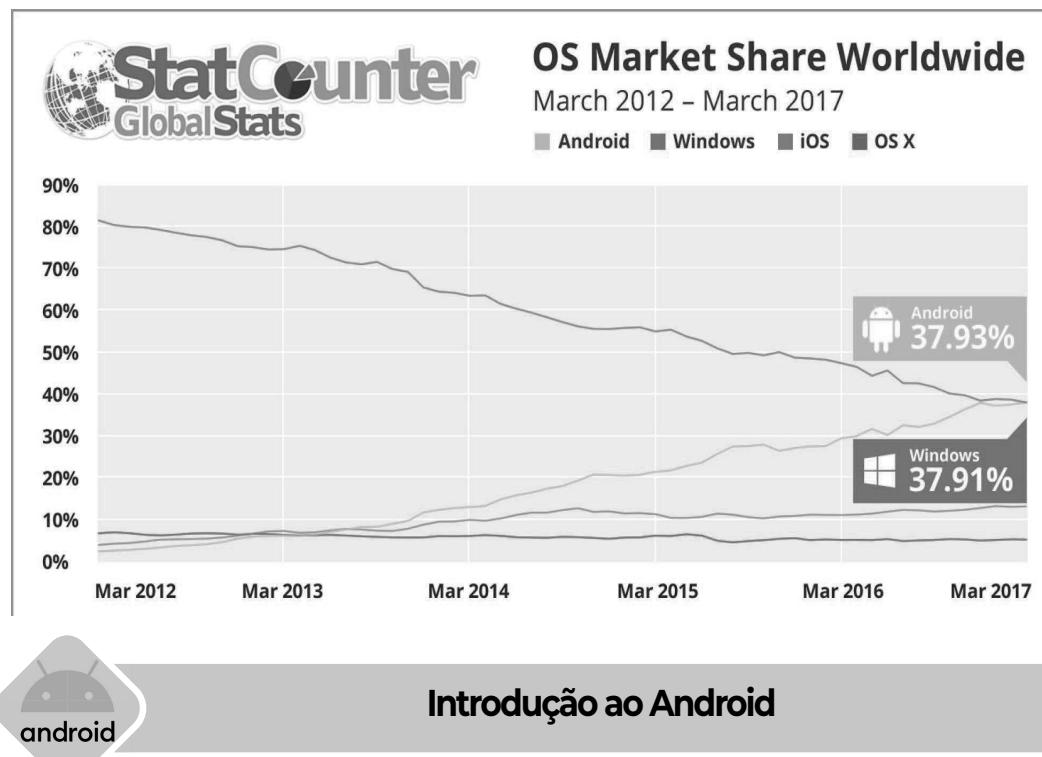
Existem atualmente vários sistemas operacionais para celulares e dispositivos móveis disponíveis no mercado. Abaixo estão relacionados alguns:

iOS

Android

Windows Phone

É público e notório que o Android, objeto de nosso estudo, se tornou o sistema operacional mais usado no mundo ultrapassando em números até o Windows, conforme podemos observar na imagem da próxima página:



Android é um software open-source, baseado no sistema operacional Linux para dispositivos móveis, como smartphones, tablets e computadores. Este foi desenvolvido por uma empresa chamada Android Inc.

Situada na Califórnia-EHA, e fundada por Andy Rubin, Rick Miner, Nick Sears e Chris White, era inicialmente uma empresa destinada a desenvolvimento de aplicativos para celulares. Foi adquirida pela Google em 2005, que manteve Andy Rubin como colaborador no desenvolvimento da plataforma Android.

Sua proposta consiste em que as aplicações devem ser capazes de rodar em diferentes dispositivos alimentados pelo sistema operacional do Android.

Diferente do Java, este não trabalha com uma Java Virtual Machine(JVM). Possui Dalvik Virtual Machine(DVM) ou Android Runtime(ART) que, além de possuir uma arquitetura diferente, é otimizada para cada dispositivo móvel.



Facilidades Android

Cada plataforma corresponde a versão do Sistema Operacional Android. Cada versão possui um número inteiro denominado API Level que corresponde a versão da plataforma Android. Por causa do conjunto de interfaces de programação do aplicativo (API) que vêm no android, você pode desenvolver facilmente aplicativos cheios de recursos em um intervalo de tempo relativamente curto. Depois de se registrar como desenvolvedor na Google Play Store, simplesmente faça upload de seus aplicativos e os publique.



Principal ferramenta de desenvolvimento Android: Android Studio.

O Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android e é baseado no IntelliJ IDEA. Além do editor de código e das ferramentas de desenvolvedor avançados do IntelliJ, o Android Studio oferece ainda mais recursos para aumentar sua produtividade na criação de aplicativos Android, como:

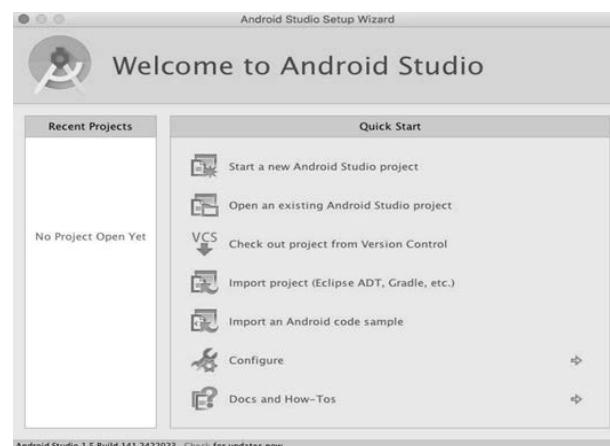
Recursos como edição de código de nível global, depuração, ferramentas de desempenho, sistema flexível de compilação e criação/implantação instantâneas permitem que você se concentre na criação de aplicativos exclusivos de alta qualidade. Um emulador rápido com inúmeros recursos.

Para fazer a instalação do Android Studio faça o download da versão mais recente no link:

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

Logo em seguida clique em download Android Studio.

A após aceitar os termos de compromisso e instalar o arquivo baixado o Android Studio estará instalado conforme tela abaixo:

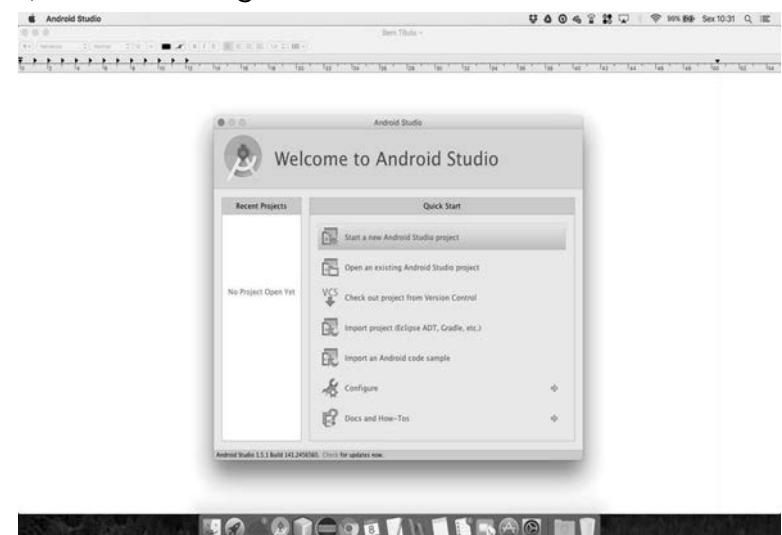


Vamos implementar um pequeno exemplo demonstrando uma pequena parte do desenvolvimento de uma aplicação com o Android Studio.

Execute o Android Studio. Será exibida a seguinte tela:

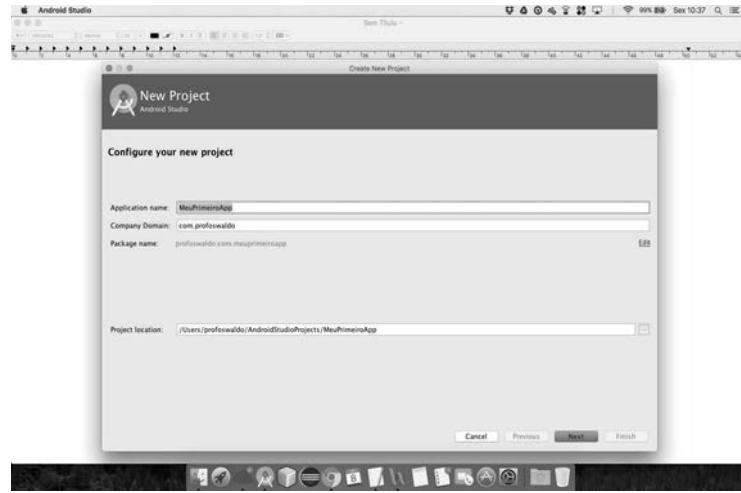


Depois de um certo tempo, e isso pode variar bastante de acordo com o computador utilizado, outra tela irá surgir:

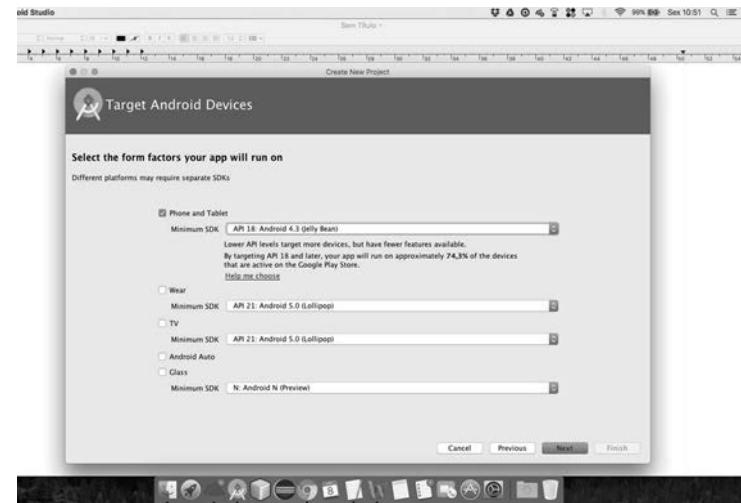


Como podemos observar, dentre outras opções, podemos criar um projeto novo, abrir um projeto e existente ou até mesmo importar um projeto. Para o nosso primeiro exemplo, criaremos um projeto novo (Start a new Android Studio project).

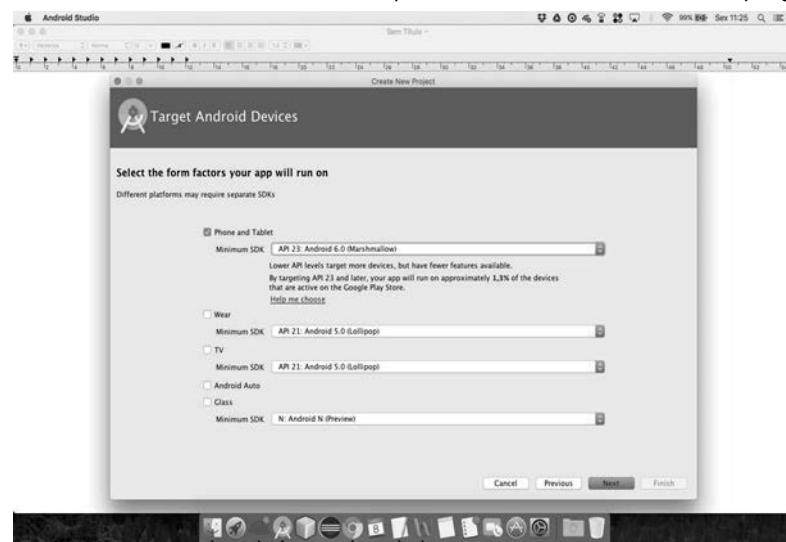
Logo após, preencha os campos conforme demonstrado na figura:



Após clicar no botão NEXT, será exibida uma tela, onde selecionamos o tipo de dispositivo para o qual desenvolverá seu aplicativo.

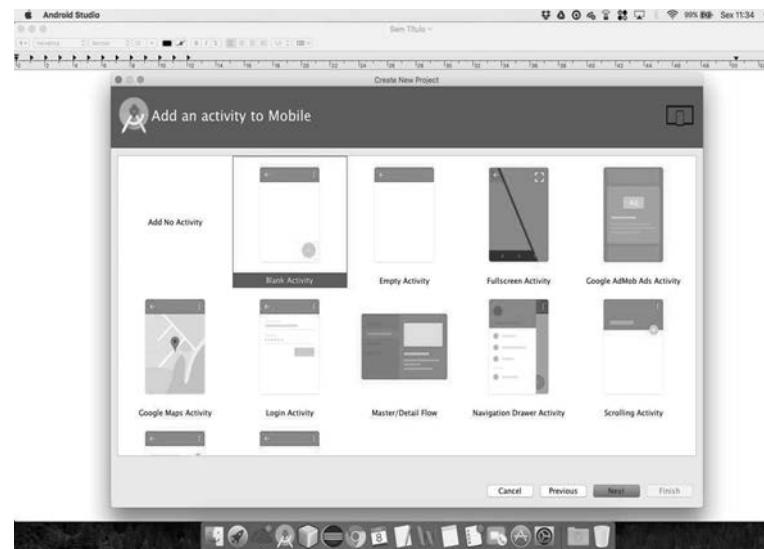


Nessa tela vamos escolher a API base que irá ser utilizada no nosso projeto.

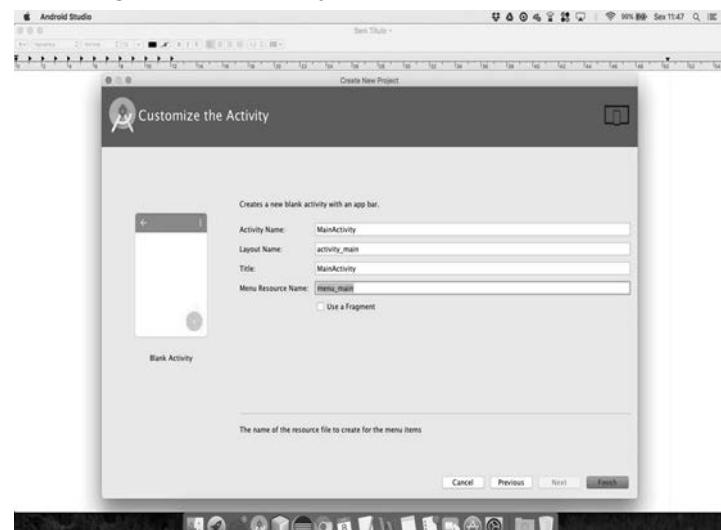


Ao avançamos a tela (Next), teremos que selecionar o tipo de Activity que desejamos implementar.

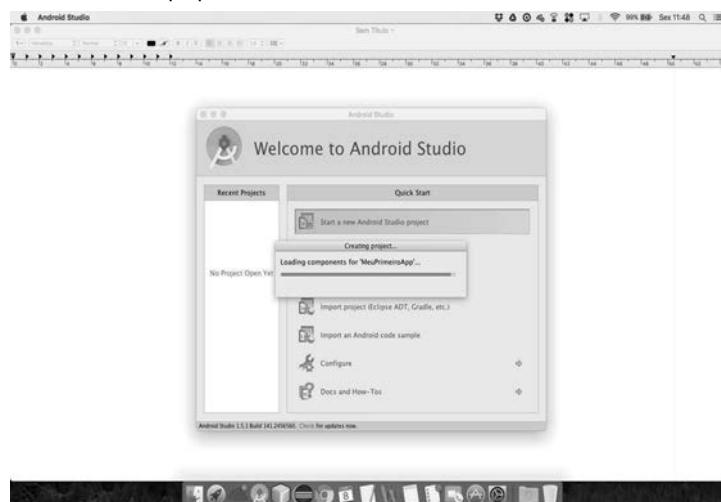
Em nosso exemplo vou selecionar a Black Activity. Este tipo corresponde a uma tela em branco com a mensagem Hello World. Depois avançaremos a tela novamente (Next).



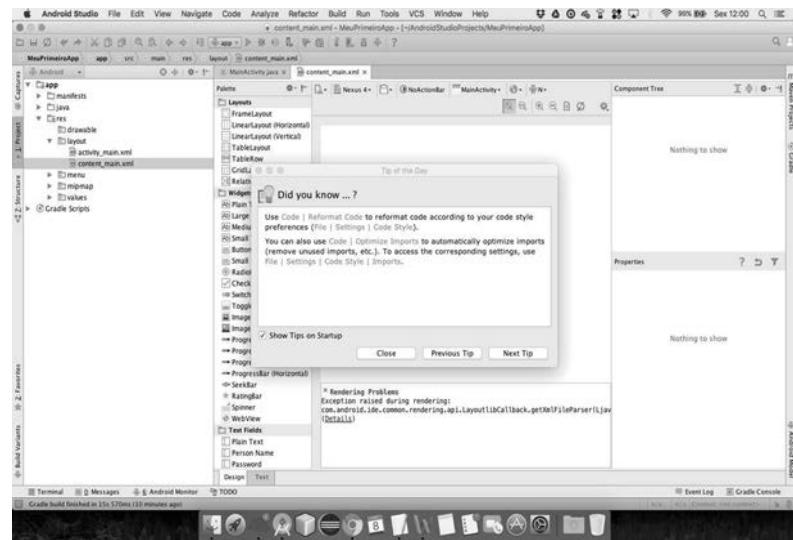
Agora vamos configurar este Activity criado conforme definido na tela abaixo:



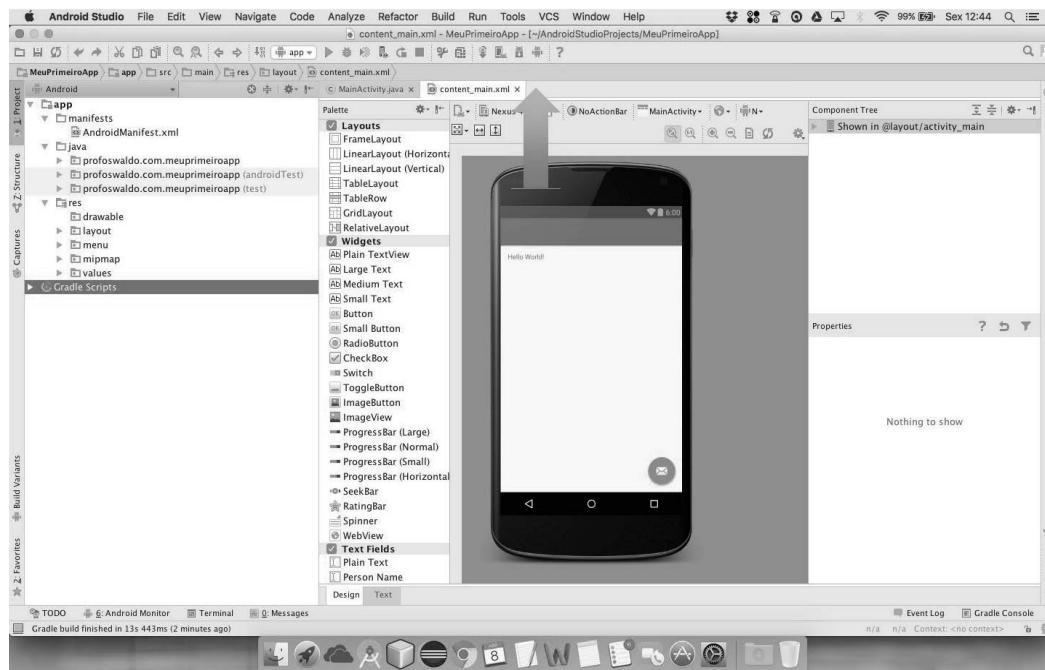
Quando selecionar o Finish, o Android Studio estará montando toda a estrutura necessária para que sua aplicação possa ser executada. Isto pode demorar algum tempo dependendo de seu equipamento.



Logo após, será exibida a tela abaixo. Aparece uma tela dando algumas dicas. Você pode fechar-a apenas clicando em close.

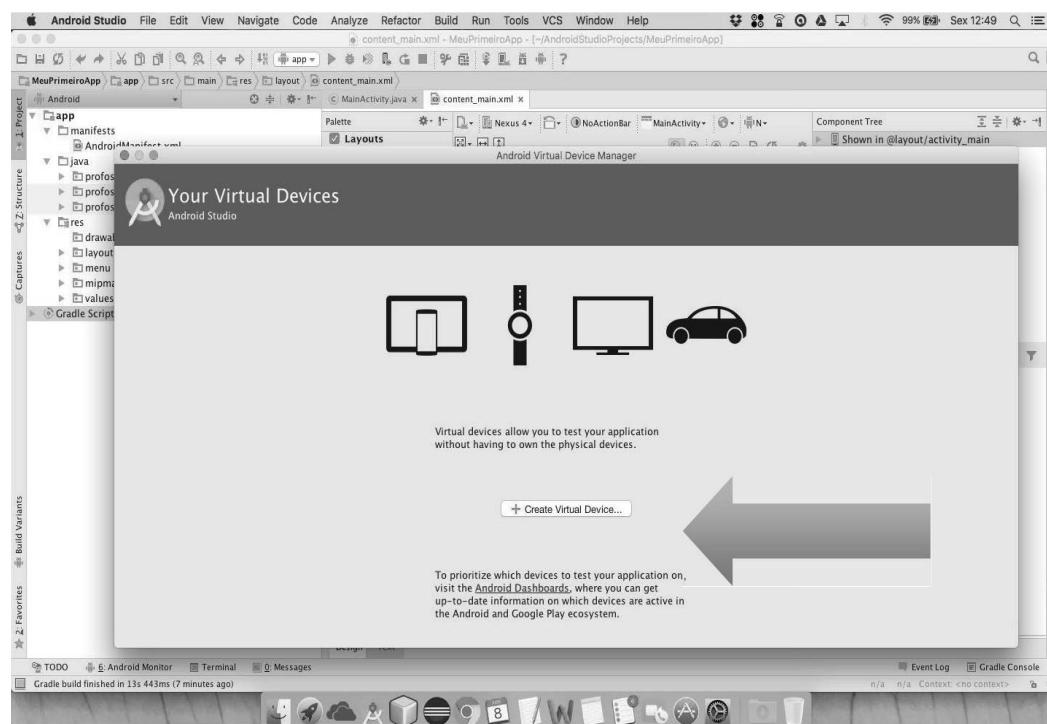


Pronto. Você já tem uma aplicação exemplo. Nessa Aplicação será exibida uma tela com a mensagem Hello World.

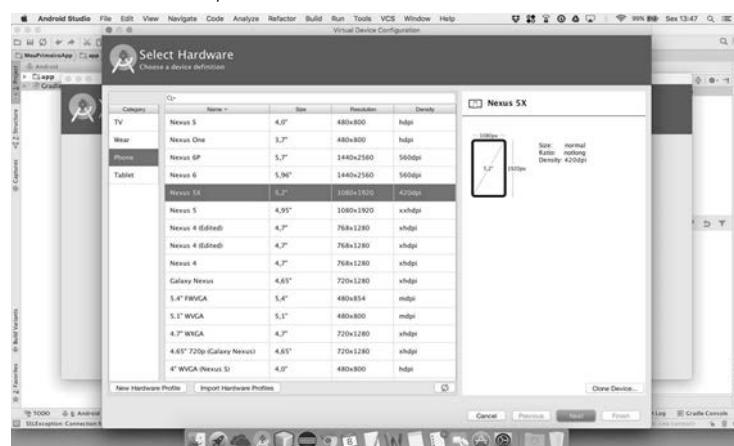


Para executar seu exemplo, é necessário que seu smartphone android esteja conectado ao computador com o modo de depuração usb ativado, ou se você pode utilizar um emulador referente ao seu dispositivo. Para tanto, na tela acima selecione o ícone indicado pela seta azul(AVD Manager).

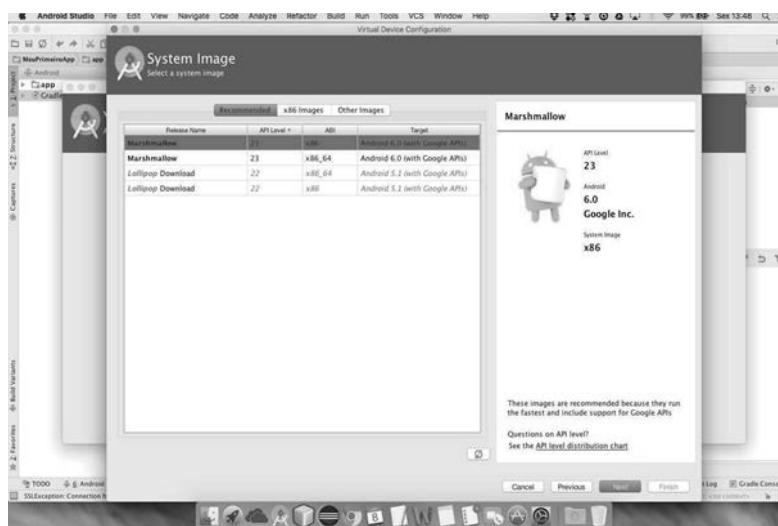
Será exibida a tela abaixo. Neste instante, como é a primeira vez que você está rodando um aplicativo, não haverá nenhum emulador definido. Precisamos criar então este dispositivo virtual. Clique no botão indicado pela seta azul.



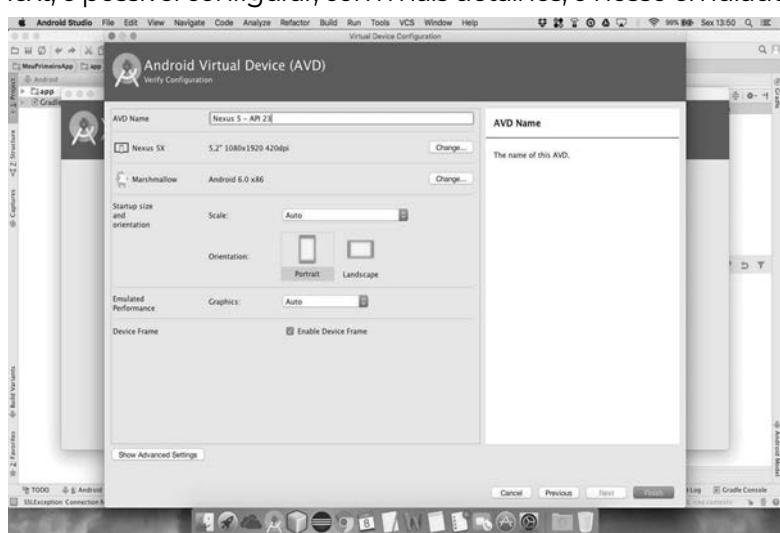
Será exibida uma tela que apresenta uma série de dispositivos de tamanhos e resoluções diferentes. Fica a seu encargo escolher o que desejar. Em nosso exemplo, vou selecionar o modelo Nexus 5x, conforme demonstrado na tela.



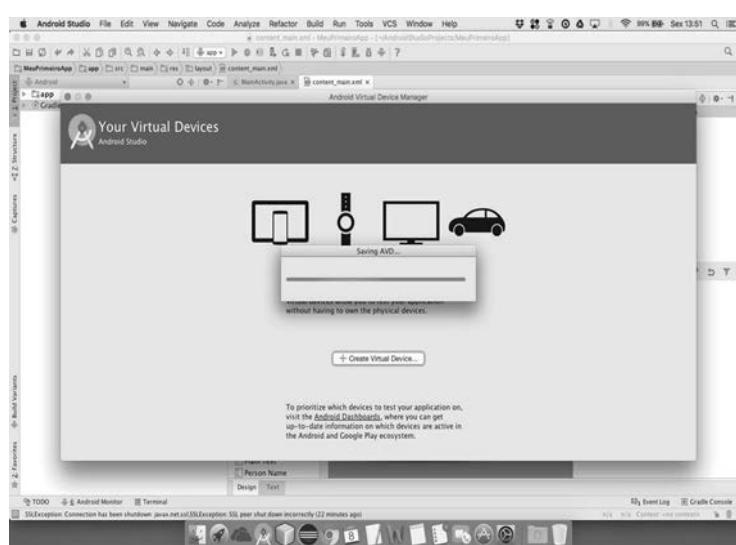
Na opção recommended serão exibidas as imagens em função dos SDKs que instalou. Você poderá verificar os no SDK Manager. No meu exemplo, estou selecionando a primeira opção, que é compatível com o meu SDK.



Após o next, é possível configurar, com mais detalhes, o nosso emulador.

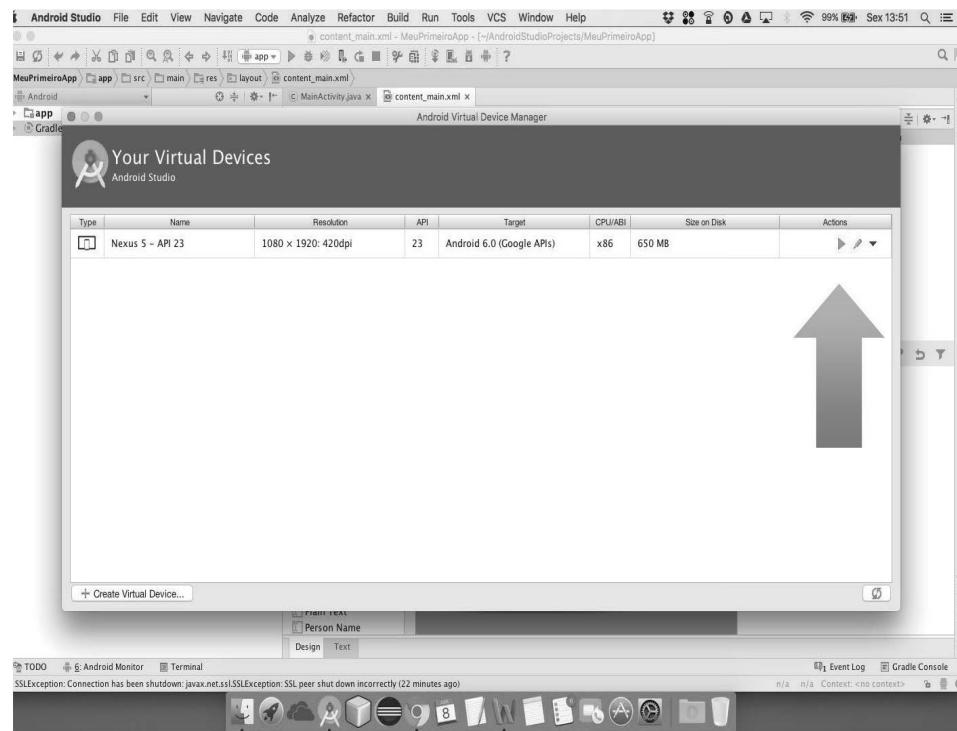


Selecione Finish e será construído o nosso emulador. Isto pode demorar um pouco.

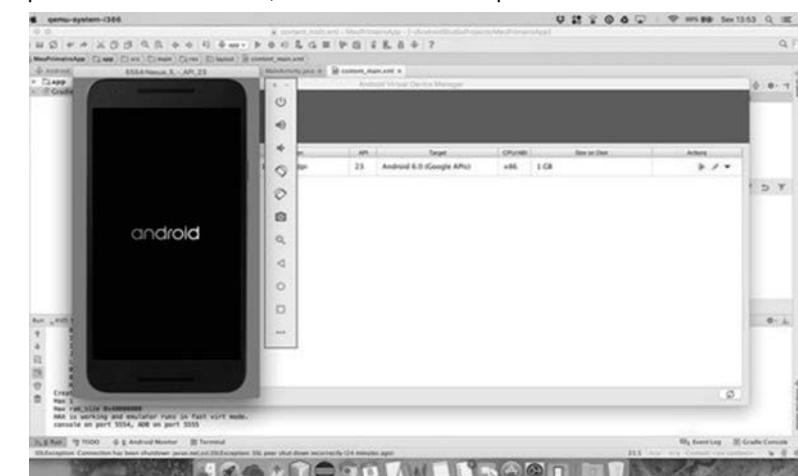


Finalmente, será exibida a tela abaixo. Precisamos executar o emulador, para que a nossa aplicação possa rodar neste.

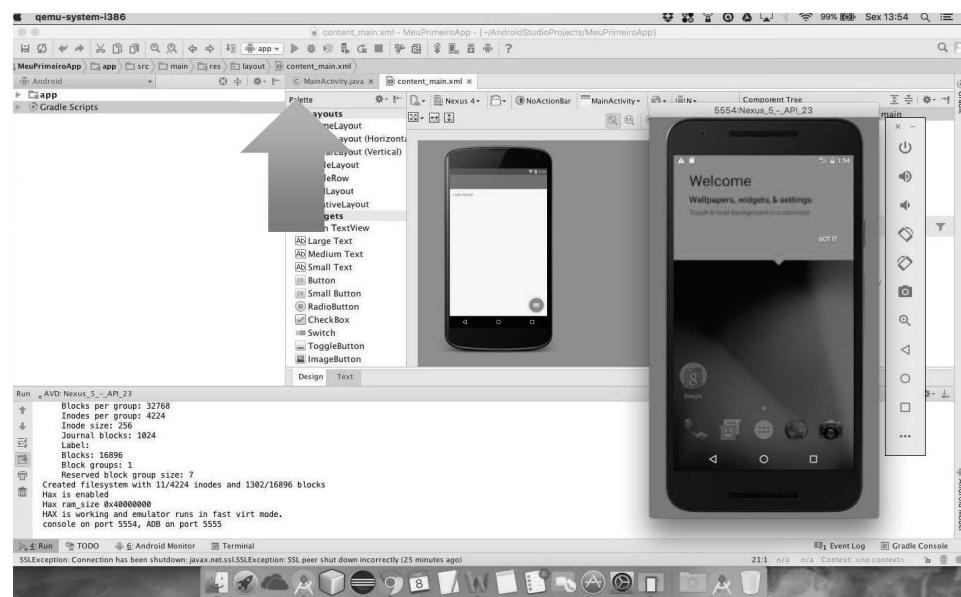
Click no ícone indicado pela seta azul.



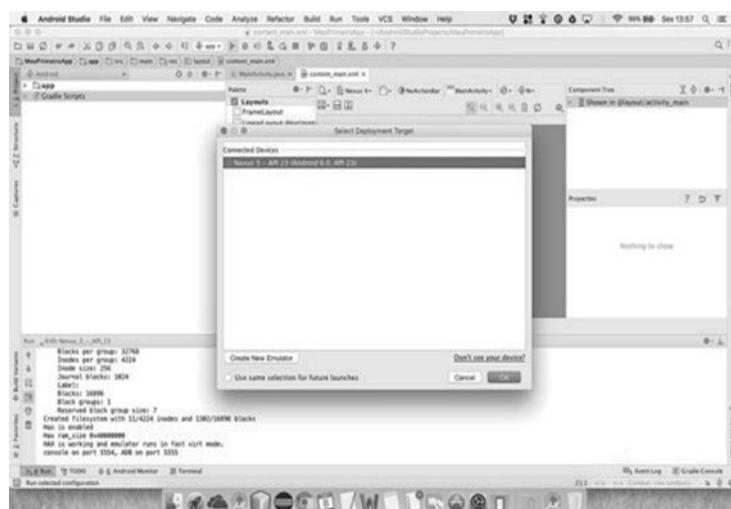
Em um primeiro momento, nossa tela ficará aproximadamente assim.



Aguarde mais um pouco até que nosso emulador fique a aparência semelhante à da tela abaixo. Para facilitar, apenas desloquei o emulador para o lado direito da tela.



Agora vamos rodar nosso aplicativo. Click no ícone indicado pela seta azul, na tela acima.

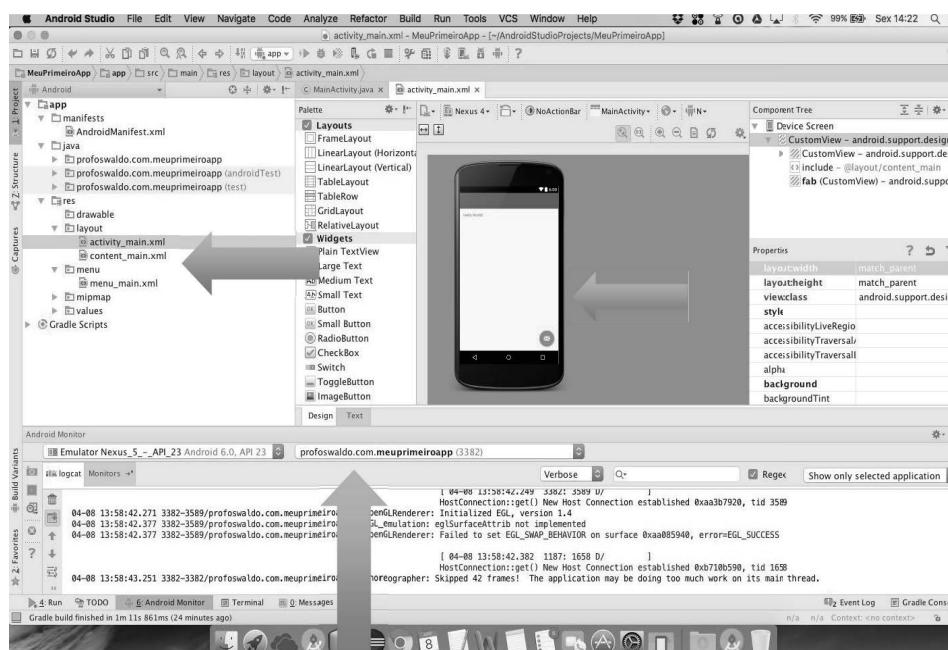


Será exibida a tela acima. Basta clicar em OK e pronto. Este processo pode demorar um pouco.

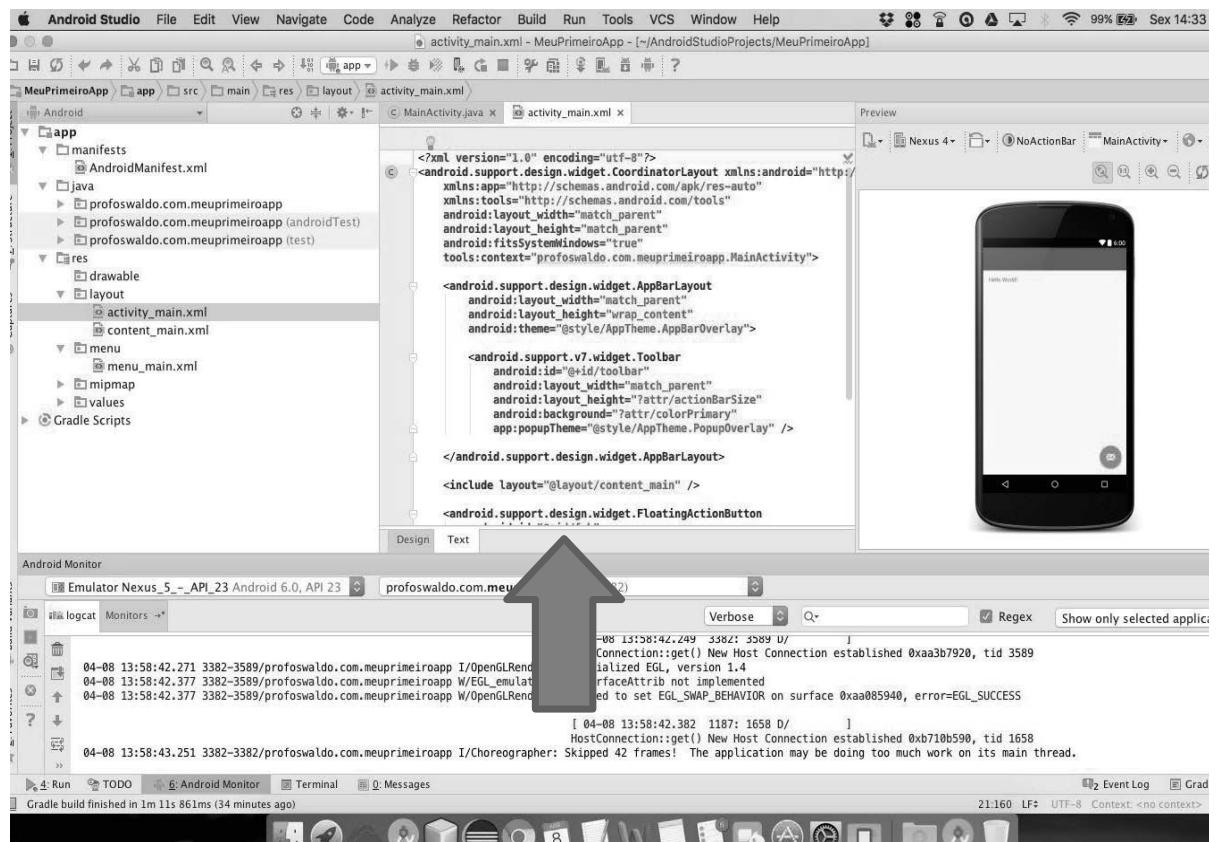
Isto vai depender também de seu hardware. Logo após seu aplicativo já estará rodando em seu emulador.



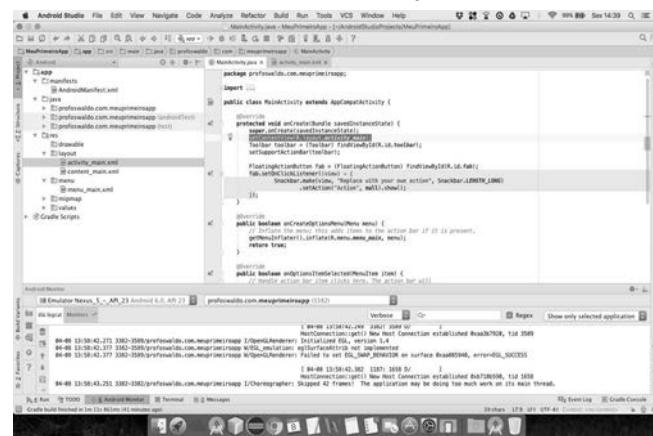
Pronto. Você já tem uma aplicação exemplo. Conforme informei anteriormente, só será exibida uma tela com Hello World.



Primeiramente, indicados pelas setas azuis horizontais na tela acima, temos os layouts que são arquivos XML que contém as definições das telas no Android. Você pode observar o código XML referente a esta tela na aba Text, indicada pela seta azul vertical.

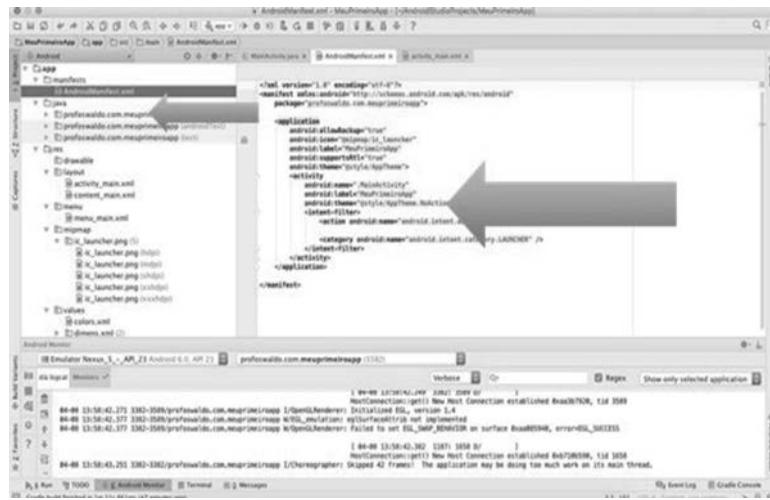


Vinculado à esta tela, temos uma Activity, no meu caso MainActivity. É um arquivo Java. Nele, dentre várias coisas, são definidos os tratamentos dos eventos gerados em nosso dispositivo. Selecione a aba MainActivity.





Vamos conhecer um pouco mais da estrutura de nosso projeto.



A menor seta indica os arquivos que fazem parte do nosso projeto android e a maior seta indica qual arquivo selecionado está aberto na tela para serem manipulados. Existem diversas pastas e sub-pastas todas com seus arquivos e funções em um projeto Android, vamos nos aprofundar um pouco mais nesses arquivos agora

A divisão simplificada com a qual vamos trabalhar é feita basicamente entre três pastas:

- ◆ manifestos: contém o arquivo `AndroidManifest.xml`.
- ◆ java: contém os arquivos de código-fonte do Java, incluindo o código de teste do JUnit.
- ◆ recursos: contém todos os recursos que não são código, como layouts XML, strings de IU e imagens em bitmap.

O arquivo `AndroidManifest.xml`:

Possui informações da aplicação, como por exemplo, as telas que possui, permissões, o que pode fazer entre outras coisas.

A pasta Java:

Nesta encontraremos o código Java propriamente dito. Em nosso exemplo temos somente a `MainActivity`, que foi criada pela IDE.

A pastas res:

Referente a recursos, como, por exemplo, imagens(bitmap), menus(menu), layouts(layout) e valores em geral(value). Dentro da pasta res são encontradas as sub-pastas seguintes:

Assets: diretório para o armazenamento de arquivos diversos utilizados por sua aplicação. Diferentemente dos recursos armazenados na pasta res, estes são acessíveis apenas programaticamente.

Libs: pasta para armazenar bibliotecas de terceiros que serão utilizadas pela aplicação e que não estão disponíveis por meio de um repositório compatível.

Drawable: pasta que contém as imagens da aplicação. Mipmap: pasta com o ícone da aplicação;

Layout: pasta que contém os arquivos XML de layouts para construir as telas da aplicação;

Menu: pasta que contém os arquivos XML que criam os menus da aplicação.

Values: pasta que contém os arquivos XML utilizados para a internacionalização, configuração de temas e outras configurações.

Para mais informações visite

<https://guiadodesenvolvedor.wordpress.com/2012/11/27/estrutura-basica-de-um-projeto-android/>.

<https://www.youtube.com/watch?v=KLaY0kT5Qt0>.



Melhorando o nosso primeiro projeto

Para melhorar a nossa aplicação, vamos incluir mais algumas coisas e aproveitar para entender alguns pontos fundamentais do desenvolvimento Android.

Em nossa versão melhorada do Hello World, o usuário informará seu nome em uma caixa de texto, pressionará um botão e a aplicação apresentará uma saudação personalizada.

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

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

Além disso, também teremos que incluir um botão, por meio do widget

Button:

```
<Button  
    android:id="@+id/saudacaoButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
    android:text="@string/surpreenda_me" />
```

Com estes novos elementos, podemos montar a tela completa do novo Hello World, que ficará como o código 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="wrap_content"  
    android:text="Digite Seu Nome" />  
  
<EditText  
    android:id="@+id/homeEditText"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:inputType="textPersonName" />  
  
<Button  
    android:id="@+id/saudacaoButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
    android:text="Surpreenda_me" />  
  
<TextView  
    android:id="@+id/saudacaoTextView"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />  
</LinearLayout>
```

Nos itens recém-adicionados, repare que colocamos um atributo android:id, que é importante, pois posteriormente precisaremos referenciar e manipular esses componentes visuais.

Ao executarmos a aplicação novamente, já perceberemos as mudanças realizadas. Como ainda não programamos nenhuma ação para o botão disponível na tela, ao pressioná-lo, nada de diferente acontece. Para obter o resultado esperado, criaremos um método na nossa MainActivity que responderá a esta ação apresentando ao usuário uma saudação personalizada. Logo, vamos configurar nosso botão para que, quando ele for pressionado, um método seja invocado. Para isso, utilizaremos o onClick:

```
<Button  
    android:id="@+id/saudacaoButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
    android:text="Surpreenda-me"  
    android:onClick="surpreenderUsuario"/>
```

Informamos que o método a ser acionado após o clique do botão é o surpreenderUsuario, por meio da propriedade onClick. Este necessariamente deve ser público e receber como parâmetro um objeto do tipo View, que é uma referência ao botão que foi pressionado:

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

Nesse método, precisamos modificar o conteúdo do widget saudacaoTextView, para que ele mostre o conteúdo informado no EditText. Para isso, a classe MainActivity deve possuir referência para esses elementos. Vamos começar declarando atributos para o EditText e o TextView:

```
public class MainActivity extends Activity {  
    private EditText nomeEditText;  
    private TextView saudacaoTextView;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
    public void surpreenderUsuario(View v) {}  
}
```

No método `onCreate`, temos que conseguir as referências para os componentes. Podemos fazer isso por meio do método `findViewById`:

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);  
    this.nomeEditText = (EditText) findViewById(R.id.nomeEditText);  
    this.saudacaoTextView = (TextView) findViewById(R.id.saudacaoTextView);  
}
```

Agora que já temos as referências para os objetos, podemos obter o valor digitado pelo usuário, que está armazenado no `EditText`, e atribuí-lo como conteúdo do `TextView`. Para realizar essa operação, basta implementar o método `surpreenderUsuario` dessa forma:

```
public void surpreenderUsuario(View v) {  
    String texto = this.nomeEditText.getText().toString();  
    this.saudacaoTextView.setText("Saudações " + texto);  
}
```

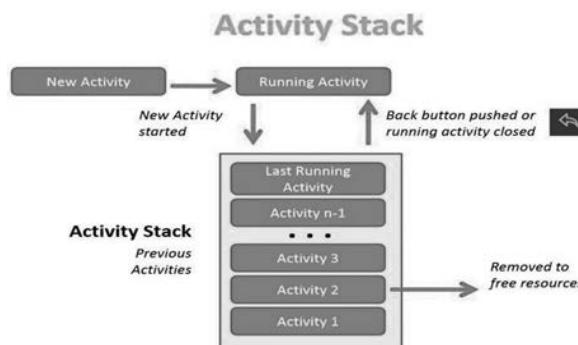


Estudando a Activity

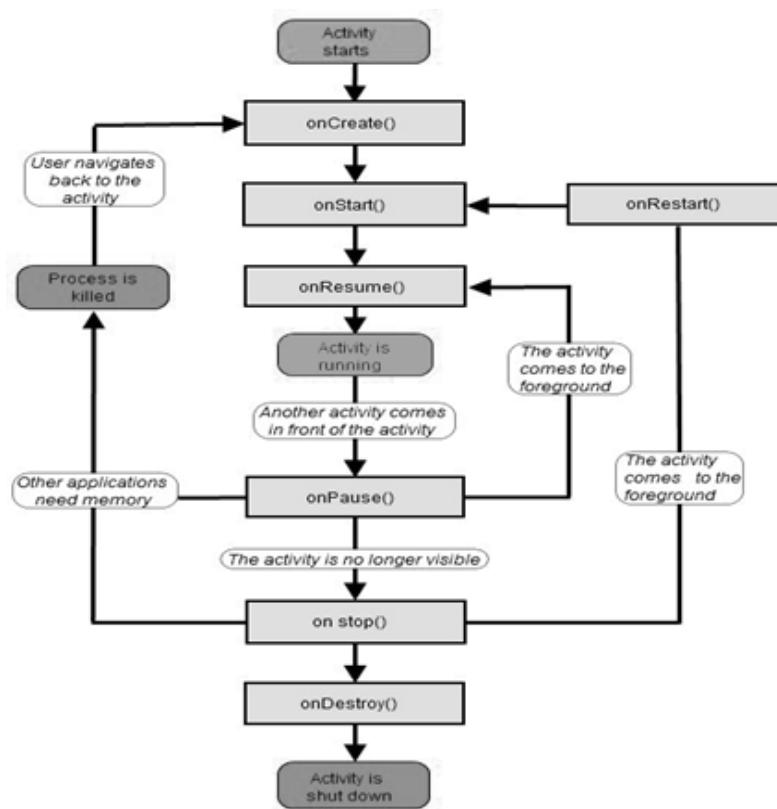
Agora que rodamos o nosso primeiro projeto, temos que entender uma das principais classes do android, a `Activity`. Ela é responsável por construir uma tela em Android, bem como tratar os eventos nela gerados. Toda aplicação Android deve implementar ao menos uma `Activity`, podendo chamar outras `activities`.

O Android é responsável por gerenciar o ciclo de vida dos `Activities`. Para tanto, faz uso do conceito de pilha, chamada de “activity stacks”(pilha de atividades). Toda `Activity` ao ser executada é inserida no topo desta pilha. A `Activity` anterior é parada e move-se para baixo da pilha.

O Android pode até mesmo encerra `Activities` se precisar de recursos. Nesse caso, ele verifica a pilha de atividade para determinar a prioridade das atividades e quais podem ser fechadas.



Ciclo de Vida Activity:



Principais métodos:

onCreate(): É a primeira função executada quando a activity é criada. Tem por responsabilidade, carregar os layouts XML, inicializar os objetos, variáveis e outras operações de inicialização. É importante lembrar que é executada somente uma vez.

onStart(): É executado antes da activity ficar visível na tela do dispositivo, podendo ser chamado após os métodos onCreate() ou onRestart().

onResume(): Representa o estado de que a activity está executando. É chamada logo após o evento onStart.

onRestart(): É chamado, imediatamente após ao método onStart(), quando uma Activity que estava parada volta ao foco.

onPause(): É chamado sempre que a tela do activity fechar. Isto ocorre quando uma outra activity (da sua aplicação ou não) ganhar o foco.

onStop(): É chamada após o método onPause(), quando a Activity não está mais visível e está sendo encerrada;

onDestroy(): É chamada antes da Activity ser destruída e logo após será liberada a memória.

Para mais informações acesse:

<https://developer.android.com/guide/components/activities.html?hl=pt-br>

Assita também: <https://www.youtube.com/watch?v=85MppyLJHz0>.

Atenção! conteúdo em inglês ative as legendas do vídeo para melhor entendimento.



Componentes gráficos.

O Android tem evoluído constantemente no que tange a interface gráfica. Ainda mais nos dias de hoje, onde existem dispositivos móveis muitas vezes com uma capacidade de tela superior até mesmo às televisões de última geração.

Para tanto, o Android oferece suporte nativo para o desenvolvimento de interfaces gráficas sofisticadas. Podemos destacar:

View

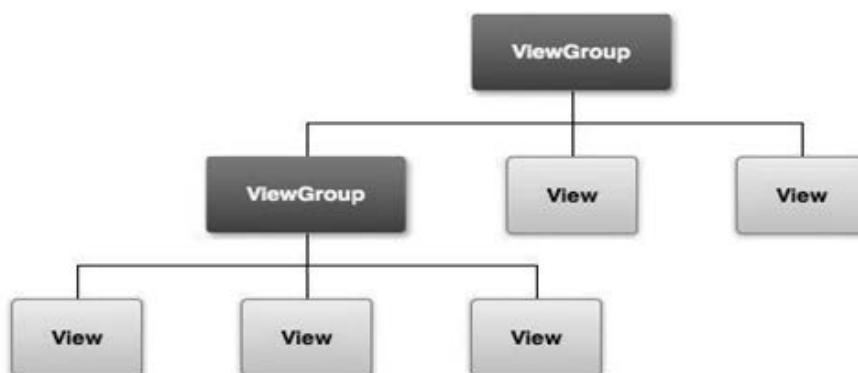
É a classe responsável pela criação de qualquer componente gráfico. Esta ocupa uma área retangular na tela e fica responsável por desenhar e controlar os eventos.

Widgets

Representa os componentes gráficos. Esta é base para a criação de componentes da interface do usuário interativos, como por exemplo, botões, caixas de texto, entre outros.

View Group

É a classe base para o desenvolvimento dos layouts.



Repare que o projeto implementado anteriormente utilizou alguns widgets para facilitar a interação entre o usuário e o nosso app. Existem vários outros que são fornecidos pela plataforma Android, e nos projetos futuros vamos implementar aplicações que demonstrem o uso deles.

Veja alguns exemplos:

Texview: Componente que funciona como um espaço onde podemos mostrar alguma informação, frase, texto e etc.

Button: Componente que representa um botão onde o usuário poderá clicar nele e assim o botão irá ativar uma ação que deverá ser executada no app.

CheckBox: Esse componente funciona como uma opção, onde esta poderá ser marcada ou desmarcada.

RadioButton: Esse componente funciona como uma opção, normalmente utilizado quando o usuário se encontra em uma situação onde ele poderá escolher uma opção dentre várias.

RadioGroup: Tem por objetivo agrupar estes radiobuttons, permitindo a seleção de apenas um RadioButton dentro deste Radiogroup.

Spinner: Esse componente nada mais é do que uma caixa de combinação. Nesse componente podemos adicionar vários itens que poderão ser selecionados pelo usuário.

ProgressBar: Esse componente exibe uma barra de progresso na tela e está disponível em 3 versões: normal, giratória e barra.

RatingBar: Esse componente é bastante utilizado para fazer sistemas de votações e classificações.

ImageView: Esse componente simplesmente serve para exibir imagens que são colocadas nele. Os formatos de imagens suportados por esse componente são: PNG, JPEG, BMP, GIF.

ImageButton: Esse componente é derivado do componente Button, só que ao invés de exibir um texto dentro dele, exibe uma imagem. Os formatos de imagem suportados são os mesmos do ImageView.

Para mais detalhes visite:

<https://developer.android.com/guide/topics/appwidgets/index.html>



Conhecendo alguns Tipos de Layouts

Atualmente existe um crescimento exponencial no uso de dispositivos móveis, em especial, smartphones e wearables. Com isto, nossos usuários já não são mais os mesmos.

Conceitos referentes velocidade, design, usabilidade, acessibilidade, tamanho de tela e outros estão em pauta para qualquer tipo de desenvolvimento de aplicação móveis. Então o desenvolvimento de interfaces inteligentes e amigáveis ao usuário é essencial para o sucesso do app. Mas nem sempre essa tarefa é tão fácil, no que tange a implementação. Nossas aplicações precisam ser executadas em vários tipos de dispositivos diferentes, mas nem todos possuem as mesmas características, como por exemplo, tamanho da tela. Já imaginou o trabalho que seria para organizar os componentes em cada tipo de tela? Neste ponto, o Android nos ajuda muito através de um conjunto de classes, pré-definidas, denominadas Gerenciadores de layout, que tem por finalidade organizar os componentes de nossas telas.

Como estudamos anteriormente, a classe View é a base para todo e qualquer componente visual no Android, que devem implementar o método `onDraw()`, responsável por desenhar o componente na tela. Os Widgets, que nada mais são do que componentes simples que herdam diretamente desta classe. As classes que herdam a classe View Group são conhecidas por gerenciadores de layout.

Então para dispor os widgets na tela do usuário, os gerenciadores de layout organizam os elementos da tela de acordo com a necessidade do desenvolvedor.

Veja alguns exemplos:

LinearLayout: Essa estrutura organiza os componentes de forma que os mesmos sejam distribuídos de forma horizontal ou vertical.

RelativeLayout: Essa estrutura organiza os componentes de forma que sejam distribuídos de forma livre na tela, mas um elemento é escolhido para ser o “pai” e os outros vão ser posicionados de acordo com a posição do elemento “pai”.

TableLayout: Essa estrutura organiza os componentes dentro dela de forma como se estivessem em uma tabela.

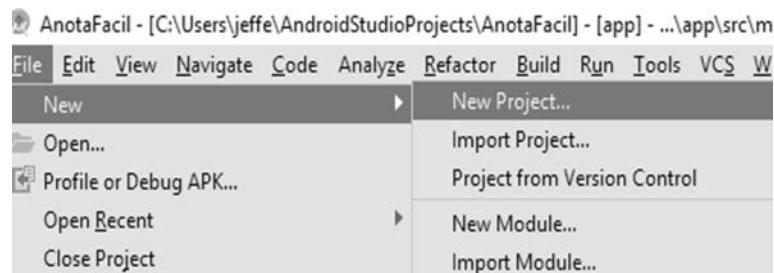
GridLayout: Essa estrutura organiza os componentes de forma como se estivessem em uma grade, dando mais possibilidades de posicionamentos que o tablelayout.

Para mais detalhes visite: <https://developer.android.com/studio/write/layout-editor.html?hl=pt-br>

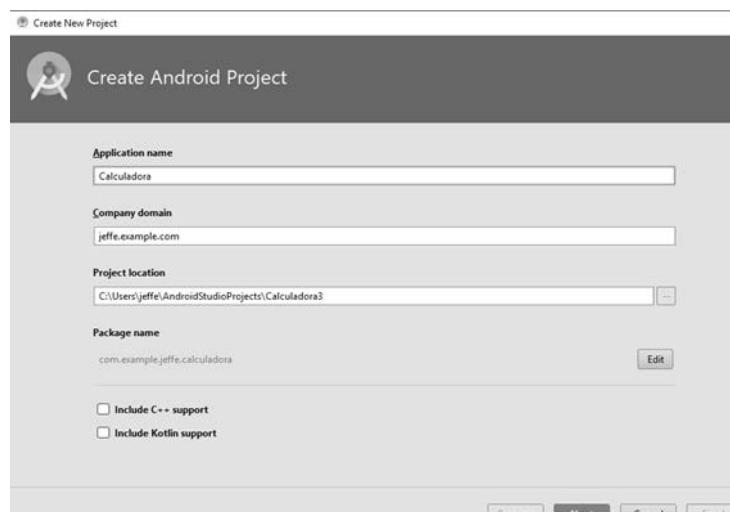


Desenvolvendo uma Calculadora

Vamos por em prática alguns conceitos estudados nos capítulos anteriores, e para começar iremos construir uma calculadora básica de soma. Para criar um novo projeto no Android Studio clique no Menu File/New/New Project. Confira na imagem a seguir.



Logo após a tela aonde aparecem os campos para escolhermos o nome da aplicação irá se apresentar.



Vamos preencher os campos com esses dados:

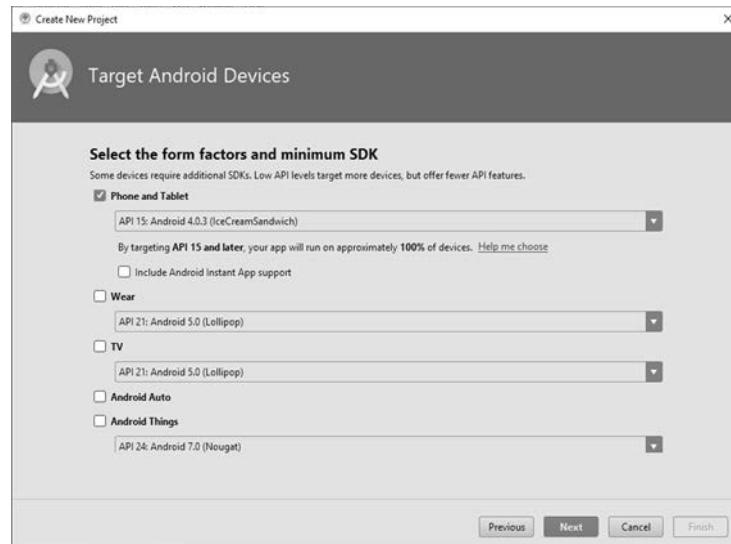
Application name: Calculadora.

Company Domain: Fica a sua escolha. A finalidade desse campo é para diferenciar os app na loja do Google, fique tranquilo você não precisa de um site para criar seu projeto.

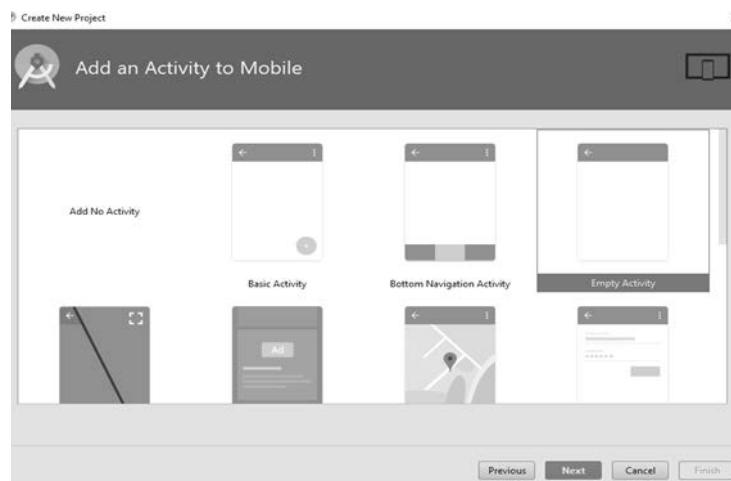
Project location: O próprio Android Studio já salva em uma pasta padrão, mas você pode modificar esse caminho.

Clique em Next.

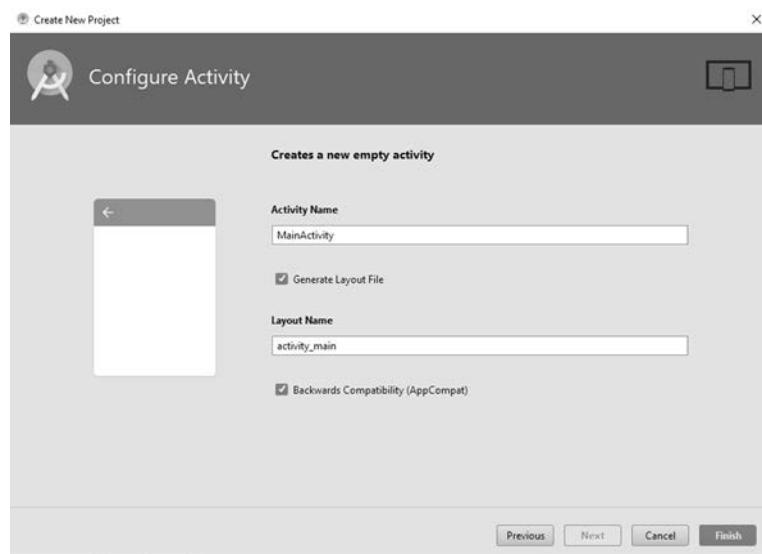
Logo após selecione a API 15 como o SDK mínimo clique em Next. Segue a figura abaixo:



Na próxima janela selecione a opção empty Activity, em outros projetos utilizaremos algumas variações de Activity.

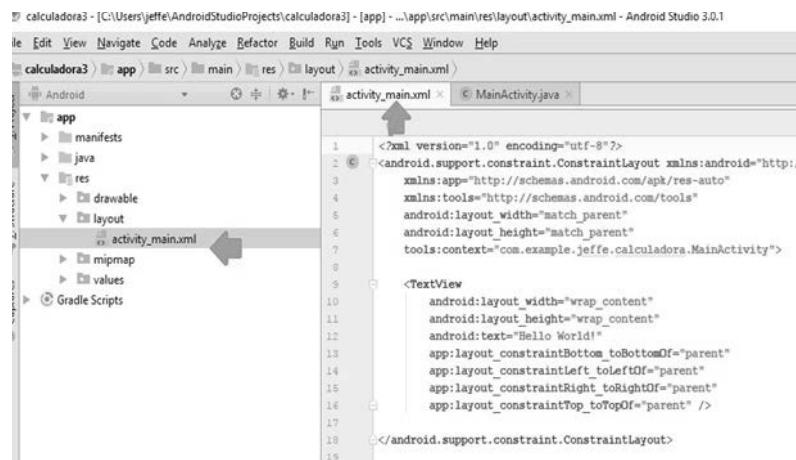


Clique em Next, logo em seguida aparecerá a seguinte tela:



Nessa tela será possível mudar o nome da Activity e do arquivo de layout responsável por ela. Apenas clique em Finish.

Na primeira parte do código iremos tratar do layout da nossa aplicação, portanto selecione o arquivo `activity_main.xml` conforme a próxima imagem.



A primeira coisa a se fazer é mudar o nosso tipo de layout para `LinearLayout` conforme o código abaixo:

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

Dentro do LienarLayout é importante usar o atributo orientation, pois ele irá informar ao Android se os elementos vão ficar na horizontal ou vertical. Logo após a modificação insira dentro do LinearLayout um TextView para que o app interaja com o usuário informando-o da ação que ele tem que executar. Esse TextView ficará assim:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.calculadora.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite o Primeiro Número" />

</LinearLayout>
```

Repare que o TextView foi inserido dentro do LinearLayout, pois o layout é responsável por posicionar os widgets na tela, não se esqueça. Logo em seguida vamos inserir um EditText para que o usuário seja capaz de digitar um número que deseja. Abaixo do TextView digite o seguinte código.

```
<EditText
    android:id="@+id/edt_num1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10" />
```

Perceba que no EditText foi inserido um atributo chamado id, ele serve para identificar os widgets que serão manipulados pelo usuário, portanto, ele deve ser único dentro da activity, ou seja, nenhum outro elemento deve ter o mesmo id. Com esse atributo os programadores podem manipula-los com códigos em java de acordo com a necessidade da situação.

Logo a seguir vamos replicar esses dois widgets para que o usuário possa digitar dois números e vamos adicionar um botão ao final para que o app faça a operação soma.

Segue o código completo do layout:

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

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.calculadora.MainActivity"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite o Primeiro Número" />

    <EditText android:id="@+id/edt_num1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:ems="10" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite o Segundo Número" />

    <EditText android:id="@+id/edt_num2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10" />

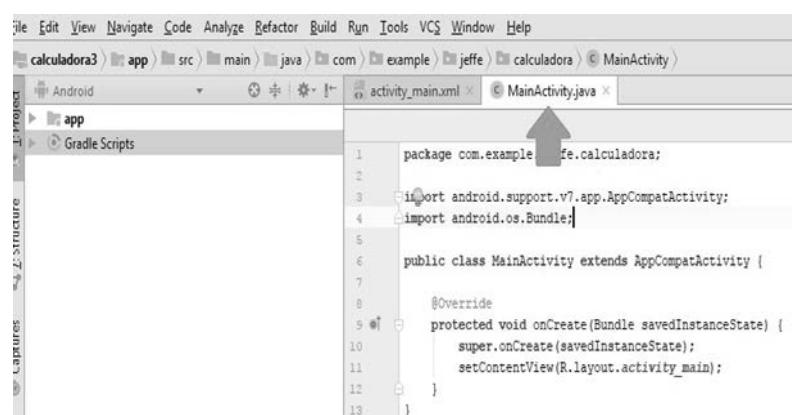
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Somar"
        android:onClick="soma"/>

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

</LinearLayout>
```

Repare que no final inserimos um botão com a propriedade onClick que irá chamar o método responsável por executar a ação do botão. Adicionamos também outro textView para que a resposta ao usuário seja mostrada na tela.

Mande executar seu projeto através do botão Run para que você veja a parte de layout pronta. Repare que ao clicar no botão soma nada acontece. Isso se deve ao fato que ainda não codificamos nenhuma ação para o botão. Para codificarmos a ação do botão selecione o arquivo MainActivity.java.



A screenshot of the Android Studio interface. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help. Below the menu is a breadcrumb navigation bar: calculadora3 > app > src > main > java > com > example > jeffe > calculadora > MainActivity.java. The left sidebar shows the Project structure with 'app' and 'Gradle Scripts'. The main editor area displays the MainActivity.java code:

```
1 package com.example.jeffe.calculadora;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class MainActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

Agora vamos analisar o código java por partes para que o entendimento seja facilitado.

package com.example.jeffe.calculadora;

Essa parte do código, nos definimos lá no começo nas telas iniciais de configuração do nosso app, o pacote é responsável por conter todos os arquivos do nosso projeto e também serve como identificação na Google Play para o aplicativo. Não é recomendável que você altere essa linha de código.

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

Essa parte do código é responsável pelos imports do nosso projeto, lembre-se a linguagem java é feita para rodar em qualquer dispositivo, por tanto, por padrão ela vem apenas com suas funcionalidades básicas, então, para que ela consiga fazer todas as tarefas, deve-se importar as funcionalidades extras que são necessárias para o projeto. Na maioria das vezes o Android Studio irá fazer os imports de forma automática, não se preocupe com isso.

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

No código acima, é criada a principal classe do nosso projeto, assim que nossa aplicação começar a rodar no android, essa será a primeira classe a ser lida. Note que dentro dela existe um método chamado OnCreate, que é responsável pela criação da nossa Activity (Lembre-se do ciclo de vida da activity). Não se preocupe o Android Studio cria tudo isso de forma automática.

Agora o próximo passo é começar a editar o código criado automaticamente pelo Android Studio:

Primeiramente devemos criar alguns objetos que serão responsáveis por controlar os widgets que irão interagir com o usuário.

```
public class MainActivity extends AppCompatActivity {  
  
    EditText num1, num2; Button botao;  
    TextView mostraSoma;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

Repare que os objetos foram criados dentro da classe, mas fora do método onCreate, essa é uma boa prática para que o método não fique sobrecarregado e não gere futuros problemas. Para criar um objeto, primeiro devemos falar de que tipo ele é, ou seja, se ele é um EditText, TextView, Button etc, depois simplesmente coloque um nome nele. Se eles forem do mesmo tipo você pode fazer a declaração em uma linha somente.

Logo após é necessário fazer referência aos ids do nosso layout para que efetivamente as nossos objetos possam controlar os nosso widgets, essas referencias são feitas do seguinte modo:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    num1 = findViewById(R.id.edt_num1);  
    num2 = findViewById(R.id.edt_num2);  
    botao = findViewById(R.id.somar);  
    mostraSoma = findViewById(R.id.mostraSoma);  
}
```

Observe que agora estamos dentro do método onCreate, aqui chamamos o objeto que já foi declarado e em seguida usamos um método findViewById que irá ligar o nosso objeto ao id que foi estabelecido ao widget no nosso layout. Para ter acesso ao id temos que chamar a classe R, pois ela é responsável por interligar os recursos do android.

Agora temos que criar nosso método que o botão somar chama ao ser clicado, lembre-se que atribuímos a ele uma propriedade chamada onClick, responsável por chamar um método no código java.

```
public void soma(View view) {  
  
    Double peganum1 = Double.parseDouble(num1.getText().toString());  
    Double peganum2 = Double.parseDouble(num2.getText().toString());  
    Double soma = peganum1+peganum2;  
    mostraSoma.setText(String.valueOf(soma));  
  
}
```

O nosso método irá nos retornar uma View, todo clique na tela do celular retorna um objeto view então quando o usuário clicar no botão soma em específico ele será capaz de pegar a view gerada e chamar o método soma. Dentro do método declaramos três variáveis, duas responsáveis por pegar o primeiro e o segundo número digitado: peganum1, peganum2. A variável soma é responsável por somar peganum1+ peganum2.

Vamos analisar mais cuidadosamente esse trecho:

```
Double peganum1 = Double.parseDouble(num1.getText().toString());
```

A variável acima é do tipo Double, então para que ela capture o que o usuário digitou no app ela precisa primeiro fazer uma conversão dos dados, pois tudo o que é digitado no widget EditText, é por padrão um Editable, portanto é necessário a conversão para que a nossa variável entenda o que o usuário digitou. Primeiro é convertido de Editable para String por isso o método `toString()`, logo após é convertido para Double por isso que utilizamos a classe e o método `Double.parseDouble()`.

Por fim vamos analisar esse trecho de código:

```
mostraSoma.setText(String.valueOf(soma));
```

Nesse trecho vamos utilizar o nosso objeto `mostraSoma` para mostrar o resultado da nossa operação para o usuário. Repare que atribuímos a ele o método `setText`. Ele é responsável por inserir um texto no nosso objeto, no caso foi o resultado da nossa soma. Por isso foi utilizado a classe e o método `String.valueOf()`, pois o resultado deve ser convertido de Double para String. Lembre-se os widgets não se dão muito bem com tipos numéricos.

Segue o código completo do nosso app:

```
package com.example.jeffe.calculadora;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    EditText num1, num2;
    Button botao;
    TextView mostraSoma;
```

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

    num1 = findViewById(R.id.edt_num1);
    num2 = findViewById(R.id.edt_num2);
    botao = findViewById(R.id.somar);
    mostraSoma = findViewById(R.id.mostraSoma);
}

public void soma(View view) {

    Double peganum1 = Double.parseDouble(num1.getText().toString());
    Double peganum2 = Double.parseDouble(String.valueOf(num2.getText()));
    Double soma = peganum1+peganum2; mostraSoma.setText(String.valueOf(soma));

}
}
```



Aprimorando o nossa calculadora

Nosso aplicativo está apenas realizando uma soma simples, vamos agora implementar a multiplicação, divisão e subtração. Repare que precisaremos agora de quatro botões distintos para ações diferentes. Nesse caso para uma maior performance do nosso app modificaremos um pouco os códigos e o layout obviamente. No arquivo responsável pelo layout(activity_main.xml) modifique o código acrescentando mais três botões, fique atento com o id de cada um e lembre-se, não repita o mesmo id. O código ficará conforme a seguir:

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite o Primeiro Número"
        android:layout_gravity="center"/>

    <EditText
        android:id="@+id/edt_num1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_gravity="center"/>

    <TextView
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Digite o Segundo Número"
        android:layout_gravity="center"/>

    <EditText
        android:id="@+id/edt_num2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"/>
```

```
        android:ems="10" android:layout_gravity="center"/>
```

```
<Button  
    android:id="@+id/somar"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Soma" />
```

```
<Button  
    android:id="@+id/subtrair"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Subtração" />
```

```
<Button  
    android:id="@+id/dividir"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Divisão" />
```

```
<Button  
    android:id="@+id/multiplicar"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Multiplicação" />
```

```
<TextView  
    android:id="@+id/mostraResultado"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Resultado"  
    android:layout_gravity="center"  
    android:textSize="35sp"  
    android:textStyle="bold"/>
```

```
</LinearLayout>
```

Repare que foi adicionado nos TextViews e EditTexts a propriedade android:layout_gravity="center" essa propriedade faz com que os objetos possam se centralizar na tela. Outra modificação feita é que agora não é mais necessário o atributo nos botões android:onClick pois agora quem irá disparar a ação do botão é o próprio código java, sem auxílio do layout. No último TextView foi modificado o seu nome, e adicionado dois atributos novos além do já citado android:layout_gravity. Esses dois atributos são: android:textSize que controla o tamanho do texto, android:textStyle que controla o estilo do texto.

Agora vamos modificar nosso código java. Vale lembrar que é muito importante que você não modifique a parte do código responsável pelos imports. Primeiramente é necessário implementar a classe View.OnClickListener ela vai ser responsável por "escutar" os nossos botões e disparar a ação correspondente para cada um. Esse novo código ficará da seguinte forma:

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {  
  
    EditText num1, num2;  
    Button botaosom,botaosub,botaodiv,botaomult;  
    TextView mostraResultado;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        num1 = findViewById(R.id.edt_num1);  
  
        num2 = findViewById(R.id.edt_num2);  
        botaosom = findViewById(R.id.sumar);  
        botaosub = findViewById(R.id.subtrair);  
        botaodiv = findViewById(R.id.dividir);  
        botaomult = findViewById(R.id.multiplicar);  
        mostrarResultado = findViewById(R.id.mostraResultado);  
    }  
  
    @Override  
    public void onClick(View view) {  
    }  
}
```

Repare que agora adicionamos uma palavra a mais na assinatura da nossa classe, o implements. Ele é responsável por chamar a classe View.OnClickListener para o nosso código. Foram adicionadas também mais três objetos do tipo botão em nosso código. Por enquanto o método onCreate permanecerá igual. Mais a baixo como chamamos uma nova classe devemos implementar o seu novo método que é o onClick. Ele é responsável por disparar a ação de cada botão do nosso layout.

Depois de implementada a nossa classe vamos focar no método onCreate. Além de fazer a integração do java com o layout agora nesse método devemos chamar o método setOnClickListener nos nossos botões. Esse método exige um contexto. Segue o código do método a seguir:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    num1 = findViewById(R.id.edt_num1);
    num2 = findViewById(R.id.edt_num2);
    botaosom = findViewById(R.id.somar);
    botaosub = findViewById(R.id.subtrair);
    botaodiv = findViewById(R.id.dividir);
    botaomult = findViewById(R.id.multiplicar);
    mostraResultado = findViewById(R.id.mostraResultado);

    botaosom.setOnClickListener(this);
    botaosub.setOnClickListener(this);
    botaodiv.setOnClickListener(this);
    botaomult.setOnClickListener(this);

}
```

Repare que o contexto é passado dentro do método com a expressão this.

O próximo passo é descartar o antigo método soma que era ativado pelo botão soma. Agora como estamos usando a classe View.OnClickListener devemos implementar o método onClick que é capaz de escutar os botões e executar as tarefas de cada um. Primeiramente vamos declarar as mesmas variáveis que utilizamos no método antigo. Segue o código:

```
@Override  
public void onClick(View view) {  
  
    Double peganum1 = Double.parseDouble(num1.getText().toString());  
    Double peganum2 = Double.parseDouble(String.valueOf(num2.get-  
Text()));  
  
}
```

Agora para diferenciar os botões devemos utilizar uma estrutura de dados chamada switch que é capaz de identificar cada botão e atribuir a sua ação específica. Segue o código:

```
switch (view.getId()){  
    case R.id.somar:  
        Double soma = peganum1+peganum2;  
        mostraResultado.setText(String.valueOf(soma));  
        break;  
    case R.id.subtrair:  
        Double subtrair = peganum1-peganum2;  
        mostraResultado.setText(String.valueOf(subtrair));  
        break;  
    case R.id.dividir:  
        Double dividir = peganum1/peganum2;  
        mostraResultado.setText(String.valueOf(dividir));  
        break;  
    case R.id.multiplicar:  
        Double multiplicar = peganum1*peganum2;  
        mostraResultado.setText(String.valueOf(multiplicar));  
        break;  
}
```

Esse código está dentro do método onClick não se esqueça!

O switch primeiro pega o id do botão pressionado pelo usuário com esse código:

```
switch (view.getId()){
```

Logo em seguida pergunta para cada case quem é responsável por esse id. Não se esqueça de colocar os dois pontos ao final.

```
case R.id.somar:
```

Então em seguida são executados os códigos dentro do case e é chamada a expressão break, que encerra o case e o switch. Não se esqueça de adicionar o break ao final de cada case pois se não todos os cases serão ativados e seu código apresentará erros inesperados.

Veja agora o código completo.

```
package com.example.jeffe.calculadora;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    EditText num1, num2;
    Button botaosom,botaosub,botaodiv,botaomult;
    TextView mostraResultado;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        num1 = findViewById(R.id.edt_num1);
```

```

num2 = findViewById(R.id.edt_num2);
botaosom = findViewById(R.id.somar);
botaosub = findViewById(R.id.subtrair);
botaodiv = findViewById(R.id.dividir);
botaomult = findViewById(R.id.multiplicar);
mostraResultado = findViewById(R.id.mostraResultado);
botaosom.setOnClickListener(this);
botaosub.setOnClickListener(this);
botaodiv.setOnClickListener(this);
botaomult.setOnClickListener(this);
}
@Override
public void onClick(View view) {
    Double peganum1 = Double.parseDouble(num1.getText().toString());
    Double peganum2 = Double.parseDouble(String.valueOf(num2.getText()));
    switch (view.getId()){
        case R.id.somar:
            Double soma = peganum1+peganum2;
            mostraResultado.setText(String.valueOf(soma));
            break;
        case R.id.subtrair:
            Double subtrair = peganum1-peganum2;
            mostraResultado.setText(String.valueOf(subtrair));
            break;
        case R.id.dividir:
            Double dividir = peganum1/peganum2;
            mostraResultado.setText(String.valueOf(dividir));
            break;
        case R.id.multiplicar:
            Double multiplicar = peganum1*peganum2;
            mostraResultado.setText(String.valueOf(multiplicar));
            break;
    }
}
}

```



Como estudamos, cada Activity corresponde a uma tela de nossa aplicação no Android, muito embora seja muito mais do que simplesmente uma tela, não é mesmo? Mas se precisássemos trabalhar com mais de uma tela? Devido a limitação de tamanho de nossos dispositivos, é muito comum distribuirmos componentes por várias telas visando facilitar o uso da aplicação.

O que fazer então? Dentre vários componentes fundamentais na programação de aplicativos Android, como Activities, Services e BroadCast Receivers, a Intent possibilita realizar a ligação, em tempo de execução, de componentes separados como, por exemplo, chamar Activities diferentes. Em outras palavras, podemos dizer que uma Intent nada mais é do que a intenção da aplicação em realizar uma determinada tarefa. Fique atento pois a intente é somente uma intenção, ou seja, não necessariamente vai ser executada, pois depende se o dispositivo permitirá.

Mas como é possível isto? Para tanto, a Intent envia ao sistema operacional o equivalente a uma mensagem(broadcast).

Este receberá a chamada e, dependendo do conteúdo, tomará as providências necessárias. Como exemplo, poderíamos:

- ◆ Iniciar uma nova Activity;
- ◆ Iniciar Bluetooth do aparelho;
- ◆ Ligar o GPS(Global Positioning System);
- ◆ Efetuar uma ligação telefônica;
- ◆ Abrir o programa de envio de SMS(Short Message Service);
- ◆ Chamar o navegador Web;
- ◆ Enviar mensagens para o Sistema Operacional;
- ◆ E muitas outras ações.

Uma Intent é basicamente um conjunto de dados que possuem informações de interesse para os componentes que a recebem e também para o Android. Este deve conter:

Nome do componente:

Representa o nome do componente que tratará a Intent. Deve ser o nome completo da classe alvo do componente. Este nome deve ser uma combinação do pacote do componente e o pacote definido no manifesto.

Ação:

É o que propriamente dito você deseja executar. Consiste de uma String com o nome da ação a ser executada. N caso de um broadcast, a ação que aconteceu e que está sendo reportada. Podemos possuir ações específicas de nossa aplicação, biblioteca ou ações predefinidas conforme tabela abaixo:

Constante	Componente alvo	Ação
ACTION_CALL	activity	Inicia uma chamada de telefone
ACTION_EDIT	activity	Mostra dados para o usuário editar
ACTION_MAIN	activity	Inicia a atividade que está marcada como inicial em modo va-zio e com nenhum retorno
ACTION_SYNC	activity	Sincroniza dados no servidor a partir do dispositivo móvel
ACTION_BATTERY_LOW	broadcast receiver	Mostra um aviso que a bateria está baixa
ACTION_HEADSET_PLUG	broadcast receiver	Mostra que um fone de ouvido foi plugado no dispositivo ou desplugado
ACTION_SCREEN_ON	broadcast receiver	A tela foi ligada
ACTION_TIMEZONE_CHANGED	broadcast receiver	As configurações da zona de tempo foi mudada

Dados:

URI e MIME type dos dados que vão adicionados à Intent. Diferentes ações são paradas com diferentes tipos de dados. Exemplo: Enviar uma foto para que um outro aplicativo trate, precisamos definir a URI e o MIME type.

Categoria:

String com informações adicionais sobre o tipo do componente que deve tratar a Intent. Qualquer número de categorias pode ser adicionada em uma Intent. Segue abaixo, as principais constantes predefinidas:

Constante	Significado
CATEGORY_BROWSABLE	A atividade alvo pode ser chamada de maneira segura pelo navegador para mostrar dados referenciados por um link (imagem ou email, etc)
CATEGORY_GADGET	A atividade pode ser embarcada dentro de outra atividade que hospede gadgets.
CATEGORY_HOME	A atividade mostra a tela home, a primeira tela que o usuário vê quando o dispositivo é ligado ou quando a tela HOME é pressionada.
CATEGORY_LAUNCHER	A atividade pode ser a atividade inicial ou uma tarefa e é listada no tipo da aplicação launcher.
CATEGORY_PREFERENCE	A atividade alvo é o painel de preferências

Extras:

Pares chave-valor para informações adicionais que devem ser entregues para o componente que tratará a Intent.

Flags:

Funcionam como instruções para o sistema Android lançar uma Activity ou como tratá-la depois de lançada. Todas as falas são definidas na classe Intent.



Intent Filter

Como saber se uma Intent vai ser entregue para o componente correto? Primeiramente precisamos entender melhor os tipos básicos de Intents.

Tipos de Intents:

As Intents podem ser classificadas como implícitas e explícitas.

Explícitas:

Como determina o componente pelo nome, é usada para passagem de mensagens na própria app.

Implícitas:

É enviada ao sistema operacional e pode ser tratada por componentes em outras app's. Neste caso, a definição de quem tratará a Intent é feita o conteúdo do objeto Intent é comparado com alguns filtros chamados Intent Filters.

Basicamente, um Intent Filter informa ao sistema quais Intents um certo componente pode tratar, um componente pode ter uma ou mais Intent Filters.

Com relação ao tipo implícito, a Intent será entregue se um dos filtros atender aos critérios da Intent. Já no caso da explícita, será entregue diretamente ao componente designado, não importando o filtro, pois nem chega a consultá-lo.

Para saber mais consulte:

<https://developer.android.com/training/basics/firstapp/starting-activity.html?hl=pt-br>

Veja também:

<https://www.youtube.com/watch?v=06bcsMx-7QQ>. Atenção conteúdo em inglês, ative as legendas do vídeo para melhor entendimento.

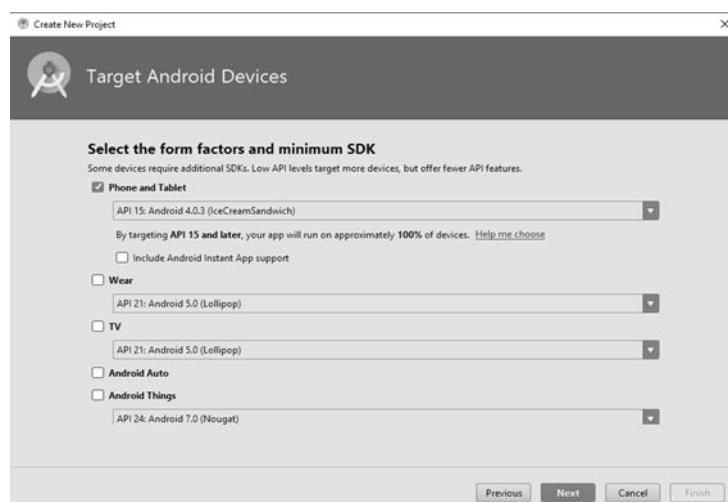


Nesse projeto iremos usar uma intent simples para que o app seja capaz de trocar de tela, o app contará apenas com um botão centralizado na tela que levará o usuário para uma segunda tela, então vamos começar.

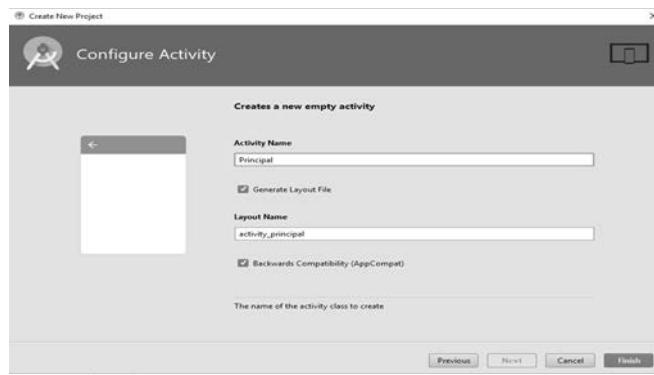
Em nosso exemplo, vou colocar o nome de nossa aplicação de trocaTela. Fique à vontade para colocar o domínio que você deseja.

Manterei o nome do pacote gerado a partir do nome da aplicação e do domínio da empresa. Você pode até alterar, mas não se esqueça que deve ser único.

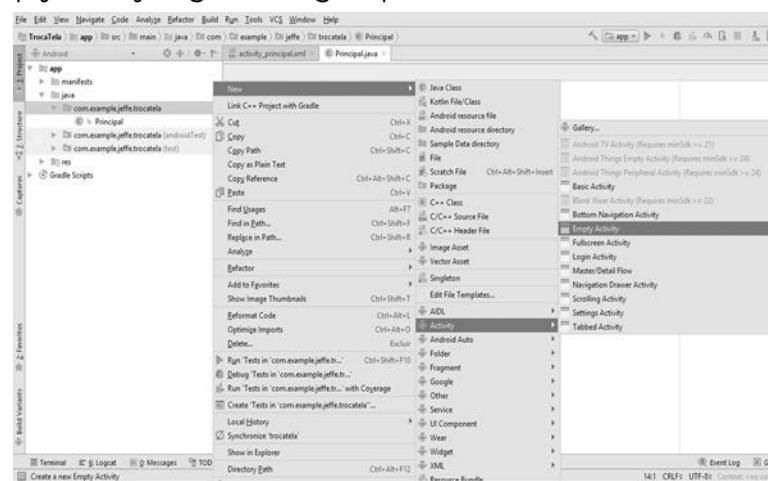
Para este exemplo, selecionarei o meu dispositivo para phone e tablet e o sdk mínimo para Android API 15 (Marshmallow).



Para simplificar, em nosso exemplo estarei selecionando o modelo Empty Activity. O nome da Activity será Principal. O nome do arquivo de layout será demônimo activity_principal. Lembrando que esse arquivo é responsável por informações do layout de nossa tela.



Após selecionar o botão finish, o projeto começará a ser construído. Não se preocupe, pois pode demorar um pouco, dependendo de seu equipamento. Depois do projeto pronto, precisaremos de duas telas para o nosso projeto, então vamos criar a segunda tela, para isso clique na pasta java e depois clique na pasta com o nome de domínio do seu projeto, atenção nessa parte pois provavelmente existirão três pastas com o mesmo nome mas duas delas são para testes, repare que na frente delas estará escrito test. Clique com botão direito na pasta que não é para testes, clique na opção new/activity/empty activity. Segue a imagem para te auxiliar.



Coloque o nome da nova activity de Segunda. Após criar a segunda tela, volte para a principal e vamos começar com os seguintes passos:

Primeiro vá ao arquivo de layout e insira um botão. Segue o código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.trocataela.Principal"
    android:orientation="vertical">
    <Button
        android:id="@+id/botao"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Troque a tela"
        android:layout_gravity="center"/>
</LinearLayout>
```

Agora vamos aos códigos java, selecione o arquivo Principal dentro da pasta java. Iremos declarar nosso widget botão, fazer a sua referência com o layout e depois construir o método que irá “escutar” o nosso botão. Segue o código:

```
public class Principal extends AppCompatActivity {
    Button botao;

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

        botao = findViewById(R.id.botao);

        botao.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
            }
        });
    }
}
```

Observe que como é um simples botão o método setOnClickListener foi instanciado dentro do método Oncreate sem problemas. Agora dentro da classe que é gerada pelo método setOnClickListener é criado um método onClick. Vamos colocar a ação do nosso botão dentro desse método, segue o código:

```
botao.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        Intent intent = new Intent(getApplicationContext(), Segunda.class);  
        startActivity(intent);  
  
    }  
});
```

Dentro do método onClick, repare que instanciamos um objeto do tipo Intent que pede os seguintes parâmetros:

- ◆ O contexto da Activity atual;
- ◆ A classe da Activity a ser iniciada.

Em nosso exemplo os parâmetros correspondem respectivamente ao método getApplicationContext() e a classe Segunda.class. Segue o código completo:

```
public class Principal extends AppCompatActivity {  
    Button botao;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_principal);  
        botao = findViewById(R.id.botao);  
  
        botao.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View view) {  
                Intent intent = new Intent(getApplicationContext(), Segunda.class);  
                startActivity(intent);  
            }  
        });  
    }  
}
```



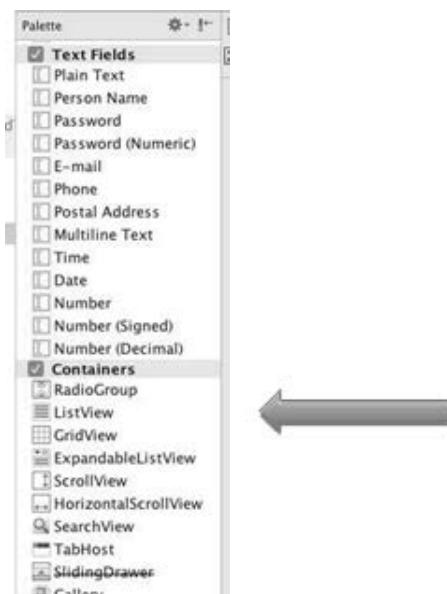
Listas

Sem sombra de dúvida, um dos componentes mais implementados no Android é o ListView. Normalmente é utilizado quando precisamos exibir uma grande quantidade de dados na forma de lista que pode possuir rolagem(scroll).

Podemos otimizá-lo com textos e imagens, tratar eventos de click e de seleção. Criando nossa primeira Lista.

Primeiramente devemos criar um projeto novo. No caso do menu exemplo, o nome é AulaListView. Assim como os exemplos anteriores, manterei o tipo do layout, nome da activity e do layout.

Logo após este primeiro passo, precisamos criar a nossa ListView. Poderíamos criar graficamente arrastando a ListView contida na Pallete para dentro do layout, mas para verificarmos com um nível de detalhamento maior, vamos trabalhar no arquivo xml do layout.



Para criarmos o ListView via códigos xml, precisamos alterar o arquivo activity_main.xml conforme definido no código abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.aulalistview.Principal">

    <ListView android:id="@+id/listview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>.
```

Feito isto, precisamos desenvolver a Activity Principal que possuirá os dados que irão compor a nossa lista. Para inserir os dados neste ListView, usaremos a classe adapter ArrayAdapter.

Não é a única forma de se implementar uma ListView, assim como os demais tipos de Adapter, ela representa a ligação entre a View e alguma fonte de dados. Altere sua MainActivity conforme o código abaixo:

```
public class Principal extends AppCompatActivity {
    ListView minhaListView;
    ArrayAdapter<String> arrayAdapter;
    String[] disciplinas
    {"matemática","Portugues","Física","Química","Inglês","Geografia", "História"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_principal);
        minhaListView = findViewById(R.id.listview);
        arrayAdapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,disciplinas);
        minhaListView.setAdapter(arrayAdapter);
    }
}
```

Primeiramente foi criado um atributo que referenciará a ListView que foi definida no arquivo activity_main.xml (@+id/listview). Depois foi preciso criar um vetor de String, onde foram inseridas várias disciplinas que irão popular a ListView. Através o método **findViewById**, minhaListView possui a referência a ListView criada.

Logo após, foi instanciada a classe **ArrayAdapter** que definirá não só o layout desta ListView, como definirá os dados que a comporão. Observe que o construtor da ArrayAdapter possui 4 parâmetros, sendo:

```
public ArrayAdapter (Context context, int resource, int  
textViewResourceId, T[] objects)  
context O contexto atual.
```

resource O id para o arquivo de layout que contém o layout a ser usado quando instanciarmos uma View.

textViewResourceId O id para a TextView do layout à ser populado.

objects Os objetos à serem representados na ListView.

Por último, apenas foi usado o método **setAdapter()** para inserir o arrayAdapter dentro da listView. Agora só precisamos executar para ver a tela abaixo:



Menus

Em aplicativos Android, um recurso extremamente utilizado é o de Menu. Basta você baixar alguns aplicativos da Google Play Store e os encontrará facilmente.

Embora sejam de fácil implementação, devemos reforçar os cuidados no que tange a usabilidade e aparência dos mesmos. É muito comum encontrarmos aplicativos sem nenhum menu, menus sem ícones ou até mesmo com títulos inapropriados. Lembre-se que um bom apelo visual e/ou textual, facilitara muito a interação do usuário com este aplicativo.

Podemos trabalhar com três tipos de menus em Android:



Menu de opção e barra de opção:

É o tipo de menu mais implementado em aplicativos Android. Normalmente encontramos neste menu as principais opções. Suporta mais de seis itens de menu, apresentando automaticamente na opção Mais(More) quando possuir mais de seis itens de menu.



Menu de contexto:

É exibido quando o usuário clica e segura, por mais de 2 segundos, um componente visual. Uma particularidade deste menu é que não suporta atalhos, ícones ou até mesmo, submenus.



Menu Popup.

Abrem quando tocamos no item de menu Options(Opções) ou em menu contextual;

- ◆ Não suporta ícones;
- ◆ Não suporta submenus aninhados;

Podemos desenvolver menus Android através de:

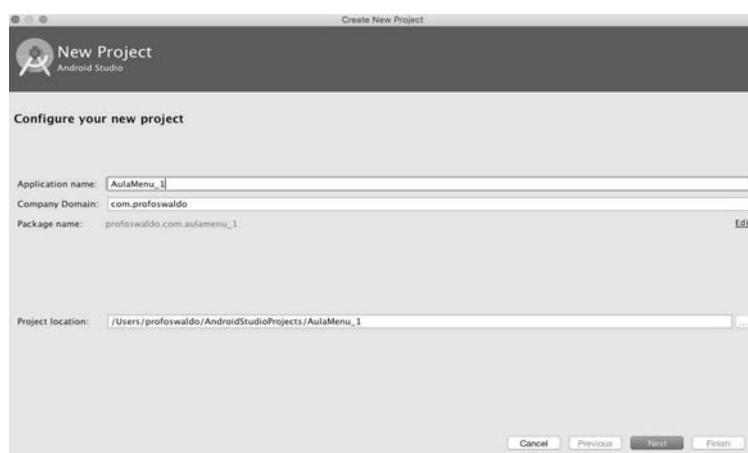
- ◆ codificação(Java): o Desenvolvido no código de sua atividade;
- ◆ arquivo XML: o Definido em um arquivo XML.

Uma boa prática é que você defina em um arquivo XML pois facilita a visualização da estrutura do menu, bem como, nos ajuda a implementar para diferentes versões do Android, tamanhos de tela, entre outros.

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



Crie um novo projeto Android. Para meu exemplo o chamarei de AulaMenu_1.
Isto sua tela deverá ficar similar a exibida abaixo:



Provavelmente não existe a pasta menu em sua árvore de recursos, então clique com o botão direito na pasta res e crie o diretório menu. Agora crie o arquivo xml chamado menu_main. Neste arquivo serão definidos os itens que comporão nosso menu. Veja a tela abaixo:



Segue o código do arquivo xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/item1"
        android:icon="@android:drawable/btn_star_big_on"
        android:title="Item1"
        app:showAsAction="always" />
    <item
        android:id="@+id/item2"
        android:icon="@android:drawable/btn_dialog"
        android:title="Item2" app:showAsAction="always" />
    <item
        android:id="@+id/item3"
        android:icon="@android:drawable/ic_delete"
        android:title="Item3" app:showAsAction="always" />
    <item
        android:id="@+id/item4"
        android:icon="@android:drawable/ic_input_add"
        android:title="Item4" app:showAsAction="always" />
</menu>
```

Se olhar mais atentamente o código, perceberá que cada tag item é referente a uma opção do menu, não é mesmo? No meu exemplo, os parâmetros são:

android:id: este id é exclusivo para cada item. É através deste que podemos identificar o item de menu;

android:title: o É o texto título de nosso item de menu;

android:icon: Embora não tenha usado neste exemplo, poderia definir um ícone para meu menu. O valor aqui é a referência a um drawable.

android:showAsAction: Define a forma de exibição do componente A constante que deve empregar são:

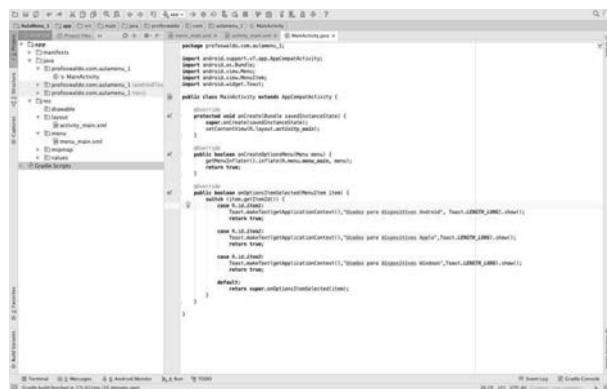
always: O componente sempre fica visível, recomendado para ações mais comuns do aplicativo; **ifRoom:** O componente é exibido na action bar se existir espaço é adequado para manter compatibilidade com diversos tipos de dispositivos e também com telas na vertical ou horizontal. **withText:** O componente exibe o seu título ao lado do ícone, caso tenha espaço disponível.

never: Não exibe o componente na action bar.

collapseActionView: Quando a view é grande deve ser contraída para exibir apenas um botão.

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

Agora segue o código java para o controle do menu(Lembre-se vá para o arquivo da classe Principal:



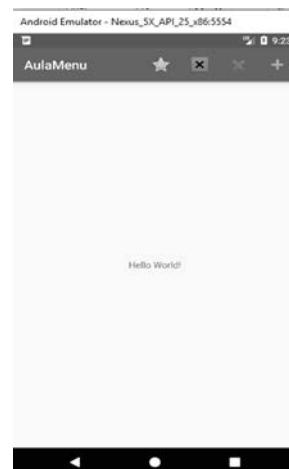
Segue o código da imagem:

```
public class Principal extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_principal);  
    }  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        getMenuInflater().inflate(R.menu.menu_main,menu);  
        return super.onCreateOptionsMenu(menu);  
    }  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) {  
        switch (item.getItemId()){  
            case R.id.item1:  
                Toast.makeText(getApplicationContext(),"Esse é o item 1",Toast.LENGTH_SHORT).show();  
                break;  
            case R.id.item2:  
                Toast.makeText(getApplicationContext(),"Esse é o item 2",Toast.LENGTH_SHORT).show();  
                break;  
            case R.id.item3:  
                break;  
        }  
        return true;  
    }  
}
```

```
Toast.makeText(getApplicationContext(),"Esse é o item 3",Toast.  
LENGTH_SHORT).show();  
break;  
case R.id.item4:  
Toast.makeText(getApplicationContext(),"Esse é o item 4",Toast.  
LENGTH_SHORT).show();  
break;  
  
}  
return super.onOptionsItemSelected(item);  
}  
}
```

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

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



Selecione uma das opções de nosso menu. Observe que a mensagem foi exibida na parte inferior da tela. Podemos alterar isto. Basta definirmos o método setGravity()

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

constante: constante Gravity. Existem várias constantes que você pode empregar e até mesmo combinar.

valor_x: o deslocamento x da posição

valor_y: deslocamento y da posição

Exemplo:

`toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);`

Vamos subir um pouco mais o nível de nosso menu?

Vá até o arquivo `menu.xml` e atualize seu código de acordo com as linhas abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/item1"
        android:icon="@android:drawable/btn_star_big_on"
        android:title="Item1"
        app:showAsAction="always">
        <menu>
            <item android:id="@+id/sub_item1"
                  android:title="Sub Item1"/>
            <item android:id="@+id/sub_item2"
                  android:title="Sub Item2"/>
        </menu>
    </item>
    <item
        android:id="@+id/item2"
        android:icon="@android:drawable/btn_dialog"
        android:title="Item2" app:showAsAction="always">
        <menu>
            <item android:id="@+id/sub_item3"
                  android:title="Sub Item1"/>
            <item android:id="@+id/sub_item4"
                  android:title="Sub Item2"/>
        </menu>
    </item>
</menu>
```

Agora aparecem mais tags entre elas estão:

menu: Representa o elemento raiz do XML. Este possui os itens de menus a serem definidos.

item: Representa itens de menu ou até mesmo de submenu; Em alguns menus pode aparecer também a seguinte tag:

group: Agrupa itens de menu. Mesmo sendo opcional e invisível, é muito comum o seu uso, como por exemplo para definir propriedades em comum entre itens.

Agora vamos ao layout e assim adicionaremos um botão para chamar um menu de contexto. Segue o código abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.aulamenu.Principal">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Mantenha pressionado" />
</LinearLayout>
```

Agora vamos para o nosso arquivo Principal onde iremos fazer algumas modificações no código. Segue a baixo as instruções.

Primeiramente temos que criar um objeto do tipo Button e logo em seguida fazer a referencia com o id do layout, depois vamos pedir para que o botão chame o método `setOnCreateContextMenuListener` para que seja criado o menu de contexto.

```
public class Principal extends AppCompatActivity {  
    Button botao;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_principal);  
  
        botao = findViewById(R.id.button);  
        botao.setOnCreateContextMenuListener(new View.  
    OnCreateContextMenuListener() {  
  
        @Override  
        public void onCreateContextMenu(ContextMenu contextMenu,  
View view, ContextMenu.ContextMenuItemInfo contextMenuItemInfo) {  
  
    };  
    }  
}
```

Dentro do método `OncreateContextMenu` vamos fazer a seguinte implementação:

```
    @Override  
    public void onCreateContextMenu(ContextMenu context-  
Menu, View view, ContextMenu.ContextMenuItemInfo contextMenuItemInfo) {  
        contextMenu.add(Menu.NONE, 1,Menu.NONE,"Menu 1");  
        contextMenu.add(Menu.NONE, 2,Menu.NONE,"Menu 2");  
        contextMenu.add(Menu.NONE, 3,Menu.NONE,"Menu 3");  
    }
```

Note que enviamos 4 parâmetros:

1º parâmetro: Refere-se ao groupId.

2º parâmetro: Nesse parâmetro enviamos o itemId que é justamente o id do menu, ou seja, é por meio dele que identificaremos esse menu.

3º parâmetro: Nele informamos o order, ou seja, a posição que queremos ordenar o menu.

4º parâmetro: Indica o nome que será exibido para o menu.

Note que tanto no groupId quanto no order colocamos o valor Menu.NONE. Essa constante equivale a 0 e significa que não queremos atribuir esses parâmetros. Em outras palavras, no código acima, criamos um menu com id 1 e nome “deletar”.

Então Implementamos o método **onCreateContextMenu(ContextMenu menu, View view, ContextMenu.ContextMenuItemInfo menuInfo)**. Este é bem similar ao método **onCreateOptionsMenu(Menu menu)**, só que é referente ao menu de contexto. O mesmo ocorre com o método **onContextItemSelected(MenuItem item)** que é da mesma forma equivalente ao método **onOptionsItemSelected(MenuItem item)**.

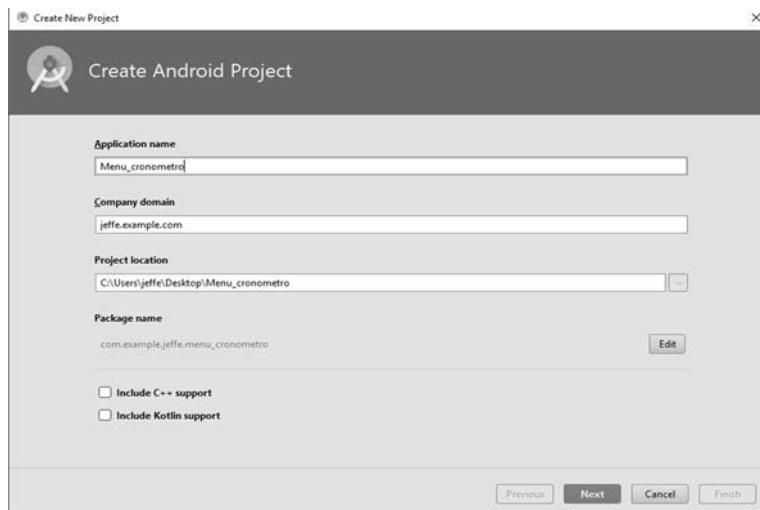
Agora é só executar para ter uma tela similar a que está registrada abaixo:



Não se esqueça de testar os sub-menus criados.

Agora que sabemos em linhas gerais como implementar menus em Android, vamos implementar outros projetos.

Crie um projeto Android, conforme tela abaixo, para implementar o Context-Menu.



Vamos definir os itens que comporão nosso menu. Se seu projeto não possuir a pasta menu, crie-a e aproveite para implementar o seguinte código no arquivo do menu:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/start_timer"
        android:title="Start" />
    <item
        android:id="@+id/stop_time"
        android:title="Stop" />
    <item
        android:id="@+id/reset_timer"
        android:title="Reset" />
</menu>
```

Chegou a hora de desenvolvemos o layout de nosso aplicativo. Para tanto, estaremos acrescendo o componente Chronometer no arquivo responsável pelo layout do nosso projeto. Segue o código do layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.menu_cronometro.Principal" />
```

```
    android:orientation="vertical">

    <Chronometer
        android:id="@+id/chronometer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" android:textSize="40dp"/>
</LinearLayout>
```

Este corresponde ao cronômetro que contará o tempo. Agora vamos aos códigos na nossa classe Principal.

```
public class Principal extends AppCompatActivity {

    Chronometer c;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_principal);
        c = findViewById(R.id.chronometer); registerForContextMenu(c);

    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenu.ContextMenuInfo menuInfo) {
        if(v.getId() == R.id.chronometer){ getMenuInflater().inflate(R.
            menu.menu,menu);
        menu.setHeaderIcon(android.R.drawable.ic_media_play);
        menu.setHeaderTitle("Controle do Timer");
    }

    super.onCreateContextMenu(menu, v, menuInfo);
}
```

```
}

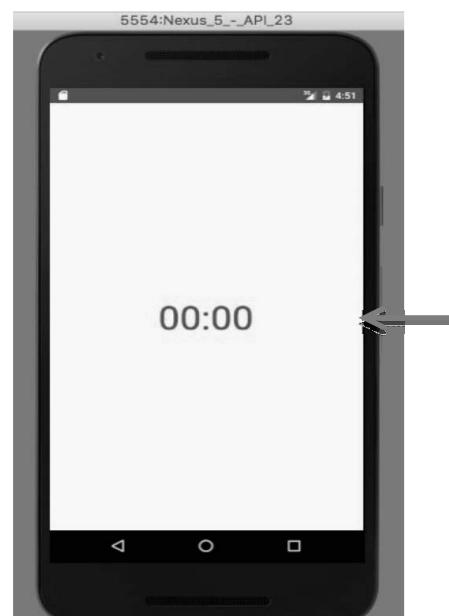
@Override
public boolean onContextItemSelected(MenuItem item) {

    if(item.getTitle().toString().equals("Start")){
        c.start();
    }
    if(item.getTitle().toString().equals("Stop")){
        c.stop();
    }
    if(item.getTitle().toString().equals("Reset")){
        c.setBase(SystemClock.elapsedRealtime());
        c.start();
    }

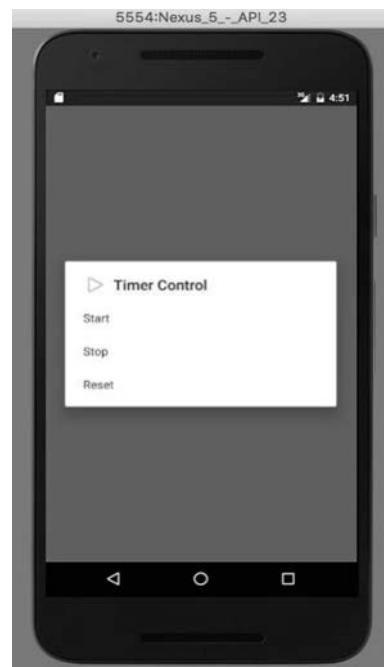
    return super.onContextItemSelected(item);
}
}
```

Perceber que já havíamos discutido estes métodos em exemplos anteriores?

Para iniciar, basta clicar continuamente por alguns segundos no cronômetro indicado abaixo.



Logo após, será exibido o menu abaixo. Click em **Start** para iniciar, **Stop** para parar e **Reset** para Zerar nosso contador de tempo.



Experimente como exercício, atribuir o Start, Stop e Reset a botões. Você consegue!

Vamos lá!



PopupMenu

Finalizando, vamos implementar um exemplo. Crie um projeto com o nome de PopupMenu e nomeie a activity de Principal.

Nosso menu será composto de 3 itens. Para isto crie a pasta responsável pelo menu (já fizemos isso anteriormente) e dentro da pasta crie um arquivo `popup_menu.xml`. Se quiser, pode incluir novos itens para verificar o efeito em nosso aplicativo. Segue o código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/aluno"
        android:title="Aluno"/>

    <item
        android:id="@+id/coordenador"
        android:title="Coordenador"/>
```

```
<item  
    android:id="@+id/professor"  
    android:title="Professor"/>  
  
<item  
    android:id="@+id/outros"  
    android:title="Outros"/>  
/>/menu>
```

Nosso layout definido no arquivo activity_principal.xml é bastante simples porque estamos estudando, no momento, menus. Possui apenas um botão com propriedades bem conhecidas. Segue o código completo:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context="com.example.jeffe.popupmenu.Principal">  
  
<Button  
    android:id="@+id/button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="Abrir Popup"/>  
</LinearLayout>
```

Somente a MainActivity.java apresenta novidades mesmo.

Segue o código:

```
public class Principal extends AppCompatActivity {  
  
    Button botao;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_principal);  
        botao = findViewById(R.id.button);
```

```
botao.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        PopupMenu popupMenu = new PopupMenu(Principal.this,botao);
        popupMenu.getMenuInflater().inflate(R.menu.menu_popup,popupMenu.
        getMenu());
        popupMenu.setOnMenuItemClickListener(new PopupMenu.
        OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem menuItem) {
                Toast.makeText(getApplicationContext(),"Você selecionou o item " +
                menuItem.getTitle(),Toast.LENGTH_SHORT).show();
                return true;
            }
        });
        popupMenu.show();
    }
});
```

Primeiramente precisamos registrar o evento para o botão. Em nossos exemplos, estou implementando com o conceito de classe anônima Java.

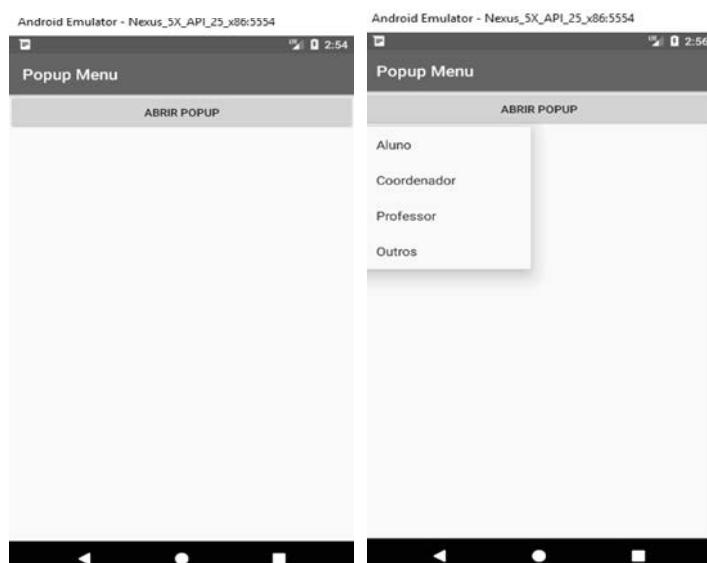
```
botao.setOnClickListener(new OnClickListener() {

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

Dentro do método onClick(View v), estamos criando uma instância da classe PopupMenu, que está recebendo o contexto e uma view como parâmetros.

O método setOnMenuItemClickListener será o responsável por responder aos eventos dos itens de menus.

Execute o projeto e click continuamente por alguns segundos o botão Seleciona o Usuário e verifique a tela abaixo:

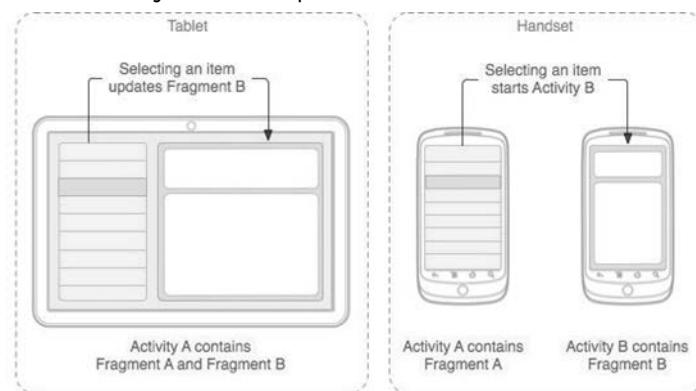


Fragment

Bastante similar a uma Activity, um Fragment consiste é uma pequena porção de Activity que permitem um projeto mais modular. Não seria errado afirmarmos que um Fragment é uma espécie de sub-activity.

Este conceito surgiu com Android 3.0 (Honeycomb) devido a necessidade de customizar o Android para as interfaces dos aplicativos, em função da pluralidade de tipos e tamanhos de dispositivos.

Embora um Fragment seja muito mais do que dividir a tela em duas ou mais, o exemplo da tela abaixo nos ajuda a compreender o conceito:



Podemos perceber que na segunda figura(SmartPhone) ao selecionarmos uma opção da lista é necessário apresentar outra tela, devido ao tamanho da existente no dispositivo. Já na primeira figura(Tablet), a naveabilidade se dá na mesma tela.

Algumas características importantes deste conceito são:

Possui seu próprio layout e seu próprio comportamento com os seus próprios retornos de chamada do ciclo de vida.

Permite adicionar ou remover *fragments* de uma *activity* em execução. Permite combinar vários *fragments* em uma única *activity*.

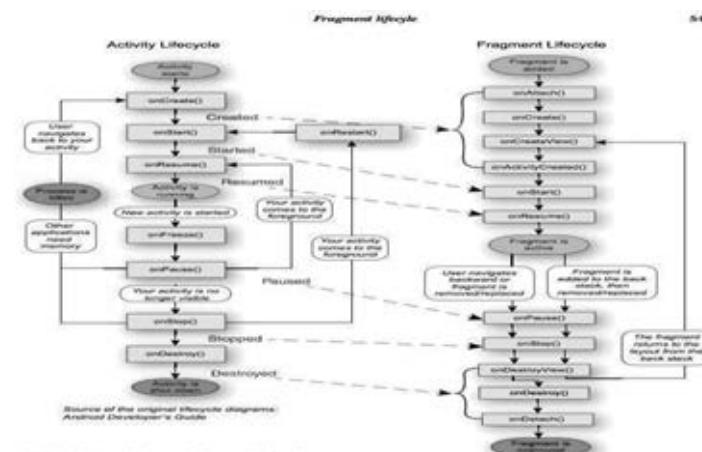
Pode ser utilizado em várias *activities*.

Ciclo de vida de um *fragment* está intimamente relacionado com o ciclo de vida da sua *activity* de acolhimento o que significa que quando a *activity* estiver em pausa, todos os *fragments* disponíveis na *activity* também serão interrompidos.

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



Embora o Fragment possua seu próprio ciclo de vida, que é bastante similar ao de uma activity, este não é uma entidade independente. Na verdade é parte de uma Activity hospedeira, que orienta o ciclo de vida do fragment. Isto pode ser verificado na figura abaixo:



A lista abaixo mostra os principais métodos do ciclo de vida:

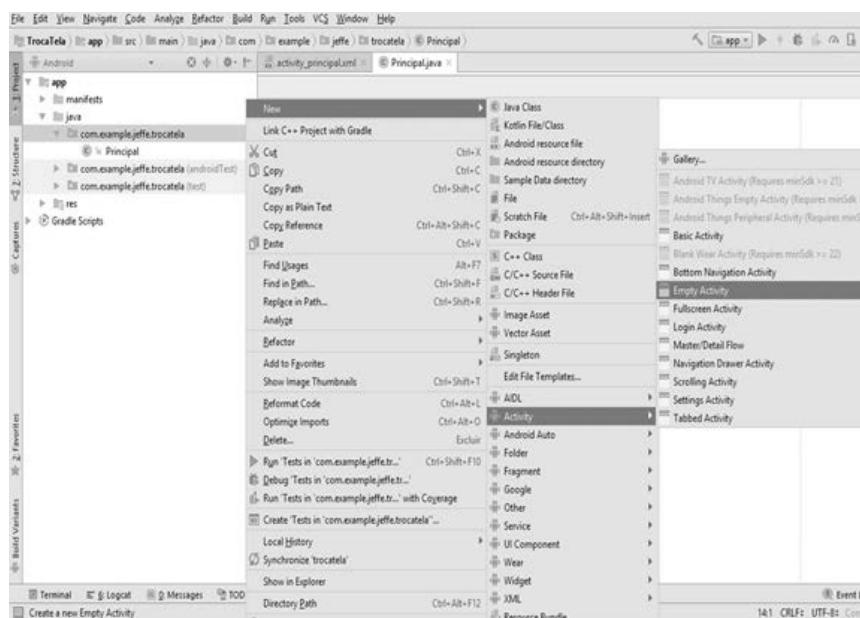
Método	Descrição
public void onAttach(Activity activity)	Chamado quando a Activity está no estado Created de- pois que o fragment é associado com sua Activity
public void onCreate(Bundle savedInstanceState)	Chamado quando a Activity está no estado Created para fazer a criação inicial do fragment.
public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



Criando Fragments:

Crie um projeto chamado AulaFragments.

Depois de criada a activity Principal crie dois fragments de nome Fragmento1 e Fragmento2. Para criar os fragments o processo é o mesmo que criar uma nova Activity, só que ao invés de clicar em new/ Activity, você vai clicar em new/Fragment. Segue a imagem da criação de um fragmento abaixo:



Depois de criado os Fragments é hora de implementarmos o layout da activity Principal. Acesse o arquivo activity_principal e faça as seguintes modificações:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.testandofragment.Principal"
    android:orientation="vertical">

    <Button
        android:id="@+id/btn1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Frag1" />
```

```

    android:onClick="mudarFragment"/>
<Button
    android:id="@+id(btn2"
    android:layout_width="match_parent" android:layout_height="wrap_content"
    android:text="Frag2"
    android:onClick="mudarFragment"/>
<fragment
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/fragment_place"
    android:name="com.example.jeffe.testandofragment.Fragmento1"

</fragment>
</LinearLayout>

```

Observe que foram adicionados dois botões, cada um irá ser responsável por ativar o seu respectivo fragmento. Logo abaixo dos botões vem o fragment, que será uma espécie de janela para as outras duas telas que criaremos. O atributo android:name é de suma importância, pois todo fragment deve ser nomeado ou na execução do app o android irá acusar erro no seu código.

Agora o próximo passo é implementar a codificação no nosso arquivo Principal que é responsável pelos códigos java.

Para um fragment funcionar temos que instanciar um objeto do tipo FragmentManager, depois outro objeto do tipo FragmentTransaction. O primeiro será o responsável por gerenciar os fragments e o segundo será responsável por fazer a troca entre os fragments. O código fica da seguinte forma:

```

Public class Principal extends AppCompatActivity {

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

    public void mudarFragment(View view){
        Fragment fragment;

```

```
if(view == findViewById(R.id.btn1)){
    fragment = new Fragmento1();
    FragmentManager fm = getSupportFragmentManager();
    FragmentTransaction ft = fm.beginTransaction();
    ft.replace(R.id.fragment_place,fragment);
    ft.commit();

}

if(view == findViewById(R.id.btn2)){
    fragment = new Fragment2();

    FragmentManager fm = getSupportFragmentManager();
    FragmentTransaction ft = fm.beginTransaction();
    ft.replace(R.id.fragment_place,fragment);
    ft.commit();
}

}
```

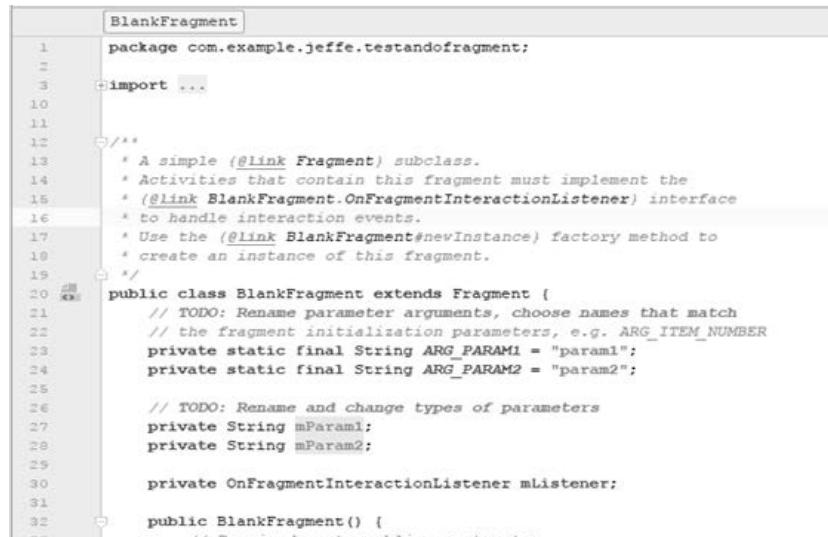
Vamos entender o código:

A primeira coisa que feita por nós, foi o método mudarFragment, ele é o responsável por fazer a troca de fragments. Logo depois usamos um if para saber qual foi o botão pressionado pelo usuário e assim fazer com que o fragment correto apareça na tela.

```
if(view == findViewById(R.id.btn1)){
    fragment = new Fragmento1();
    FragmentManager fm = getSupportFragmentManager();
    FragmentTransaction ft = fm.beginTransaction();
    ft.replace(R.id.fragment_place,fragment);
    ft.commit();
}
```

Logo depois disso é instanciado um novo objeto do tipo Fragmento1, outros dois são instanciados, são eles o FragmentManager e o FragmentTransaction. Depois de instanciados colocamos eles todos no fragment da nossa activity principal e mandamos ele rodar.

Agora depois de feito esses processo devemos arrumar os nossos fragments, pois quando eles são criados pelo Android Studio, eles vem cheios de códigos que não são necessários.



```
1 package com.example.jeffe.testandofragment;
2
3 import ...
4
5 /**
6  * A simple (@link Fragment) subclass.
7  * Activities that contain this fragment must implement the
8  * (@link BlankFragment.OnFragmentInteractionListener) interface
9  * to handle interaction events.
10 * Use the (@link BlankFragment#newInstance) factory method to
11 * create an instance of this fragment.
12 */
13 public class BlankFragment extends Fragment {
14     // TODO: Rename parameter arguments, choose names that match
15     // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
16     private static final String ARG_PARAM1 = "param1";
17     private static final String ARG_PARAM2 = "param2";
18
19     // TODO: Rename and change types of parameters
20     private String mParam1;
21     private String mParam2;
22
23     private OnFragmentInteractionListener mListener;
24
25     public BlankFragment() {
26
27
28
29
30
31
32 }
```

Apague todos os códigos e deixe somente esses:

```
Public class Fragmento1 extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_fragmento1, container, false);
    }

}
```

Após fazer tais modificações, abra os arquivos de layout dos fragments e digite o seguinte código para diferencia-los.

O código abaixo é referente ao arquivo fragment_fragmento1.xml

```
<FrameLayout    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.testandofragment.Fragmento1">
```

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:text="Esse é o fragmento 1"  
    android:background="@android:color/holo_blue_dark"/>
```

```
</FrameLayout>
```

O Código abaixo é referente ao arquivo fragmente_fragmento2.xml

```
<FrameLayout    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context="com.example.jeffe.testandofragment.Fragment2">
```

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:text="Esse é o fragmento 2"  
    android:background="@android:color/holo_green_dark"/>
```

```
</FrameLayout
```

Pronto, agora mande executar o seu projeto. O resultado irá ser algo parecido com o da imagem abaixo.



Estilo e Tema

Um estilo em Android corresponde a um conjunto de propriedades como tamanho, altura, largura, cor de fonte, cor do fundo, tamanho de fonte e muito mais propriedades que a plataforma android te dá liberdade para se modificar uma View.

O estilo é definido em um arquivo XML que é separado do arquivo de layout, fazendo assim com que uma única propriedade possa servir para vários objetos em uma mesma tela ou em telas diferentes, o que tem como efeito a redução de duplicação de código.

É bastante simples definir um conjunto de estilos. Se o Android Studio não criar automaticamente basta criar um arquivo XML no diretório res/values/styles.xml. Apenas não podemos esquecer que a tag <resources> deve ser a raiz. O código abaixo define a especificação de um estilo chamado appTema no arquivo styles.xml:

```
<?xml version="1.0" encoding="utf-8"?> <resources>
<style name="MeuEstilo">
<item name="android:textColor">#424242</item>
<item name="android:textSize">40sp</item>
<item name="android:textStyle">"bold"</item>
</style>
</resources>
```



Herança de Estilo

A plataforma Android permite aplicar o conceito de herança que vem da programação orientada a objetos para estilos. Com isto, podemos:

Fazer uso das propriedades definidas em um estilo pai; Modificar ou adicionar novas propriedades;

Herdar de estilos pré-definidos; Herdar estilos criados;

É bastante simples de implementar a herança de estilo. Para herdar estilos pré- definidos do Android basta seguir o exemplo abaixo:

```
<style name="TextoVerde"
parent="@android:style/TextAppearance">
<item name="android:textColor">#00FF00</item>
</style>
```

Já para herdar estilos definido na aplicação, não devemos usar o atributo parent.

Apenas usamos o nome do estilo herdado e acrescentamos o “.” + o novo nome.

Veja o exemplo abaixo:

```
<style name="TextoCinza">
<item name="android:textColor">#424242</item>
</style>

<style name="TextoCinzaeGrande">
<item name="android:textSize">60sp</item> </style>
```

Um tema é um estilo aplicado a uma Activity ou aplicação inteira, ao invés de uma View individual.

Quando um estilo é aplicado como um tema, todas as Views na Activity irão usar essas propriedades definidas.

Para definir um tema para aplicação ou uma atividade específica precisamos editar o arquivo **AndroidManifest.xml** conforme exemplo abaixo:

Aplicação inteira:

```
<application android:theme="@style/TemaApp">
```

Activity específica:

```
<activity android:theme="@style/TemaActivity">
```

A plataforma Android oferece uma grande coleção de estilos e temas que você pode usar em seus aplicativos. Você pode encontrar uma referência de todos os estilos disponíveis no link:

<https://developer.android.com/reference/android/R.style.html>.

Esta documentação não descreve minuciosamente os estilos e temas. Caso precise de mais detalhes, acesse os links:

<https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/values/styles.xml>

<https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/values/themes.xml>

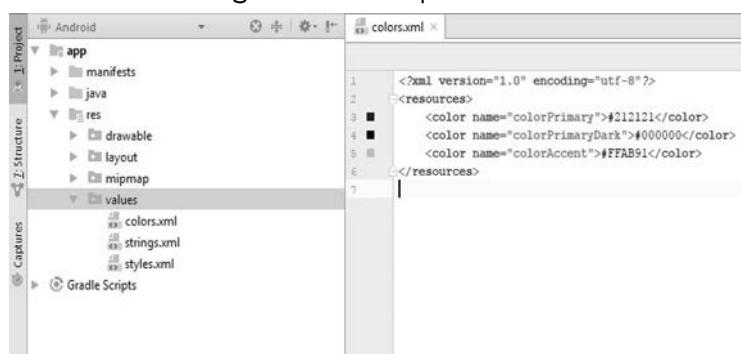


Caixa de diálogo que pode exibir um título, uma frase qualquer, até três botões cada um com uma ação diferenciada, uma lista de itens selecionáveis ou um layout totalmente personalizado.

Exemplo AlertDialog:

Vamos criar um exemplo prático de AlertDialog, primeiramente crie um novo projeto com o nome de Aula_Dialog, faça as configurações de sempre: API 15, Blank Activity e nome da Activity de principal.

Após esses procedimentos que devem ser feitos em todo projeto novo, vamos iniciar da seguinte forma, primeiramente vamos acessar o arquivo responsável por gerenciar as cores do nosso aplicativo e modifica-las. lembre-se acesse a pasta res/values/colors. Abaixo observe a imagem com o arquivo colors.xml aberto.



Segue o código do arquivo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="colorPrimary">#212121</color>
<color name="colorPrimaryDark">#000000</color>
<color name="colorAccent">#FFAB91</color>
</resources>
```

Repare que existem três cores, a colorPrimary que irá colorir a toolbar do projeto, a colorPrimaryDark, que irá colorir a área de notificações do nosso projeto, e por último temos a colorAccent, que irá colorir botões, cursores e etc.

Depois de definidas as cores iremos modificar o nosso tema padrão. Para modificá-lo acesse o arquivo styles.xml na mesma pasta do arquivo colors.xml. Logo após modifique o código da seguinte forma:

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item
            name="colorAccent">@color/colorAccent</item>
        <item name="android:textColor">#FFAB91</item>
        <item name="android:textSize">35sp</item>
        <item name="android:textStyle">bold</item>
    </style>

</resources>
```

Repare que para não precisar ficar adicionando propriedades de cor, tamanho e estilo de texto, em cada widget da nossa View, adicionamos logo aqui no arquivo responsável pelo tema e ele irá valer para toda a nossa aplicação, isso nos poupa tempo e evita linhas de códigos redundantes.

Chegou a vez de definirmos o layout de nossa activity Principal. Ele terá um simples botão para chamar o nosso alertDialog. Segue o código de exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.aula_dialog.Principal">
```

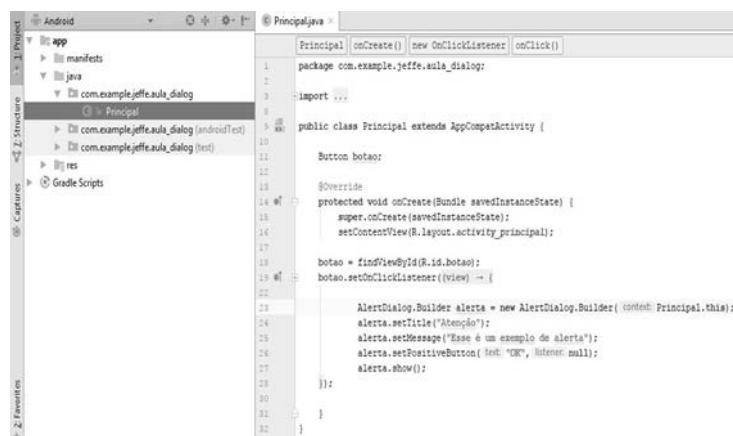
```

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/botao" android:text="Clique aqui"
    android:textColor="#fff"/>

</LinearLayout>

```

Por último, vamos programar nossa activity principal. Acesse o arquivo na pasta java conforme a imagem abaixo mostra:



Através da classe AlertDialog.Builder vamos configurar a caixa de diálogo(título, mensagem, botões), definiremos o evento onClick e exibiremos através da classe AlertDialog. Veja no código abaixo:

```

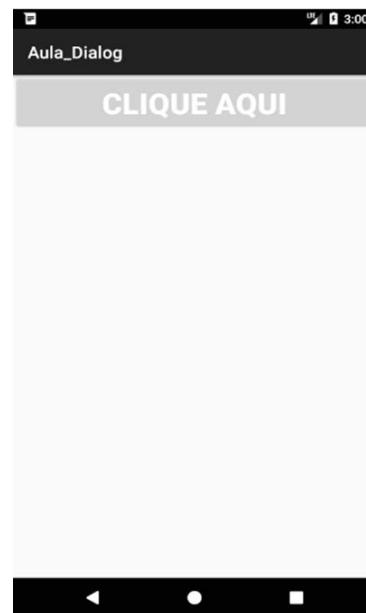
public class Principal extends AppCompatActivity { Button botao;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_principal);
botao = findViewById(R.id.botao);
botao.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
AlertDialog.Builder alerta = new AlertDialog.Builder(Principal.this);
alerta.setTitle("Atenção");
alerta.setMessage("Esse é um exemplo de alerta");
alerta.setPositiveButton("OK",null);
alerta.show();
}});}}

```

Vamos analisar o código. Primeiramente foi criado um objeto do tipo alertDialog, e logo depois inserimos nesse objeto um título com o método setTitle, uma mensagem com o método setMessage e um botão com o método setPositiveButton.

Lembre-se, ao final não se esqueça do método show(), pois ele é o responsável por mostrar a caixa de diálogo na aplicação.

Execute seu aplicativo e verá a tela abaixo:



Clique no botão cinza claro e será exibida a AlertDialog abaixo:



Agora vamos implementar o Clique no botão ok, só para você ter uma ideia de como implementar ações dentro de um botão do AlertDialog. Dentro do método OnClick implemente o seguinte código:

```
public void onClick(View view) {  
  
    AlertDialog.Builder alerta = new AlertDialog.Builder(Principal.this);  
    alerta.setTitle("Atenção");  
    alerta.setMessage("Esse é um exemplo de alerta");  
    alerta.setPositiveButton("OK", new DialogInterface.OnClickListener() {  
        @Override  
        public void onClick(DialogInterface dialogInterface, int i) {  
            Toast.makeText(getApplicationContext(),"Você clicou no botão  
Ok",Toast.LENGTH_SHORT).show();  
        }  
    });  
    alerta.show();  
}
```

Repare que no método setPositiveButton adicionamos uma nova classe que é responsável por pegar em qual botão o usuário clicou no alertDialog. Depois isso é só implementar a ação do botão que no nosso caso é um simples toast que avisará ao usuário em qual botão ele clicou.

Termine o código e vamos testar, o Toast irá aparecer quando você clicar no botão ok do AlertDialog.

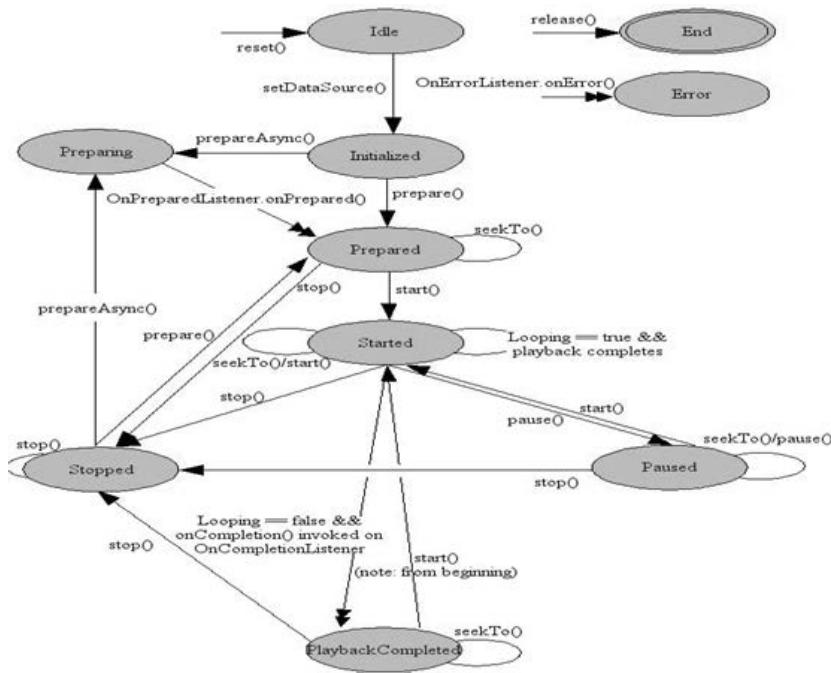




Classe MediaPlayer

A classe MediaPlayer(android.media.MediaPlayer) é responsável por controlar a reprodução de recursos multimídia, como áudios e vídeos.

É muito importante compreender o ciclo de vida desta classe. Para tanto, segue o seu Diagrama de Estados abaixo:



Principais métodos:

Nome do método	Estados válidos	Estados inválidos
attachAuxEffect	{Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted}	{Idle, Error}
getAudioSessionId	any	{}
getCurrentPosition	{Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted}	{Error}
getDuration	{Prepared, Started, Paused, Stopped, PlaybackCompleted}	{Idle, Initialized, Error}

getVideoHeight	{Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted }	{Error}
getVideoWidth	{Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted }	{Error}
isPlaying	{Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted }	{Error}
pause	{Started, Paused, PlaybackCompleted }	{Idle, Initialized, Prepared, Stopped, Error}
prepare	{Initialized, Stopped}	{Idle, Prepared, Started, Paused, PlaybackCompleted, Error}
prepareAsync	{Initialized, Stopped}	{Idle, Prepared, Started, Paused, PlaybackCompleted, Error}
release	any	{}
reset	{Idle, Initialized, Prepared, Started, Paused, Stopped,	{}
reset	PlaybackCompleted , Error}	
seekTo	{Prepared, Started, Paused, PlaybackCompleted }	{Idle, Initialized, Stopped, Error}
setAudioAttributes	{Idle, Initialized, Stopped, Prepared, Started, Paused, PlaybackCompleted }	{Error}

	setAudioSessionId	{Idle}	{Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted, Error}
	setAudioStreamType	{Idle, Initialized, Stopped, Prepared, Started, Paused, PlaybackCompleted }	{Error}
	setAuxEffectSendLevel	any	{}
	setDataSource	{Idle}	{Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted, Error}
	setDisplay	any	{}
	setSurface	any	{}
	setVideoScalingMode	{Initialized, Prepared, Started,	{Idle, Error}
	setVideoScalingMode	Paused, Stopped, PlaybackCompleted }	{Idle, Error}
	setLooping	{Idle, Initialized, Stopped, Prepared, Started, Paused, PlaybackCompleted }	{Error}
	isLooping	any	{}
	setOnBufferingUpdateListener	any	{}
	setOnCompletionListener	any	{}
	setOnErrorListener	any	{}
	setOnPreparedListener	any	{}

setOnSeekCompleteListener	any	{}
setPlaybackRate	any	{}
setPlaybackParams	any	{}
setScreenOnWhilePlaying	any	{}
setVolume	{Idle, Initialized, Stopped, Prepared, Started, Paused, PlaybackCompleted}	{Error}
setWakeMode	any	{}
start	{Prepared, Started, Paused, PlaybackCompleted}	{Idle, Initialized, Stopped, Error}
stop	{Prepared, Started, Stopped, Paused, PlaybackCompleted}	{Idle, Initialized, Error}
getTrackInfo	{Prepared, Started, Stopped, Paused, PlaybackCompleted}	{Idle, Initialized, Error}
addTimedTextSource	{Prepared, Started, Stopped, Paused, PlaybackCompleted}	{Idle, Initialized, Error}
selectTrack	{Prepared, Started, Stopped, Paused, PlaybackCompleted}	{Idle, Initialized, Error}
deselectTrack	{Prepared, Started, Stopped, Paused, PlaybackCompleted}	{Idle, Initialized, Error}

Para maiores informações, acesso o link: <https://developer.android.com/reference/android/media/MediaPlayer.htm>



Persistência de Dados em Android.

Em Android podemos persistir dados de nossa aplicação de várias formas diferentes.

Dentre as opções de armazenamento de dados, podemos escolher:

- ◆ Shared Preferences: o Armazenar dados particulares primitivos em pares chave valor;
- ◆ Internal Storage: o Armazena dados privados na memória do dispositivo.
- ◆ External Storage o Armazenar dados públicos sobre o armazenamento externo compartilhado.
- ◆ SQLite Databases o Armazena dados estruturados num banco de dados privado.
- ◆ Network Connection o Armazena dados na web no seu servidor de rede.



SharedPreferences

É muito comum a necessidade de nossas aplicações precisarem armazenar pequenas quantidade de informações.

A classe SharedPreferences se mostra como uma alternativa ao uso de banco de dados.

Ela permite salvar e recuperar pares de chave/valor de tipos de dados(boolean, floats, ints, longs, e strings), pois associa um “nome” a uma determinada informação para que depois se possa recuperá-la através deste nome.

Normalmente usamos esta opção para poucas informações. Como exemplo deste tipo de armazenamento temos:

Preferências que o usuário definiu na aplicação. Data do último acesso ao servidor.

Pontuação de um jogo.

Para criarmos um arquivo de preferências podemos fazer de duas formas: getSharedPreferences() o se precisar de múltiplos arquivos de preferências

identificados por um nome que é passado como parâmetro; Exemplo:

```
SharedPreferences sharedpreferences =getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
```

getPreferences() o se precisar de um único arquivo de preferências para a Activity.

Como se trata de um único arquivo, não é necessário fornecer um nome.

Exemplo:

```
SharedPreferences sharedpreferences=getPreferences(Context.MODE_PRIVATE);
```

Além de privado, existem outros modos disponíveis que estão listados abaixo:

Modo	Descrição
MODE_APPEND	se o arquivo já existe, então, escrever dados para o final do arquivo existente em vez de apagá-lo.
MODE_ENABLE_WRITE_AHEAD_LOGGING	aplicado à banco de dados
MODE_MULTI_PROCESS	verifica se há modificação de preferências, mesmo que a instância sharedpreference já foi carregada. (deprecated)
MODE_PRIVATE	É o modo padrão, onde o arquivo criado só pode ser acessado pelo aplicativo de chamada (ou todos os aplicativos que compartilham o mesmo ID de usuário).

Para salvar usamos classe SharedPreferences.Editor que nada mais é do que uma classe auxiliar que escreve no arquivo. Veja o exemplo abaixo:

```
Editor editor = sharedpreferences.edit(); editor.putString("key", "value"); editor.commit();
```

Não devemos nos esquecer do método commit que efetiva a escrita no arquivo. Além do método putString(), a classe Editor possui vários outros métodos.

Entre eles podemos destacar:

Modo	Descrição
apply()	confirmar as alterações para o objeto sharedPreference
clear()	remove todos os valores
remove(String key)	remove o valor cuja chave foi passada como um parâmetro
putLong(String key, long value)	salvar um valor longo
putInt(String key, int value)	salvar um valor inteiro
putFloat(String key, float value)	salvar um valor flutuante
putString(String key, String value)	salvar um valor String
putBoolean(String key, boolean value)	salvar um valor booleano
getLong(String key, long value)	retorna o valor Long referente a chave ou um valor predefinido caso a chave não seja encontrada
getInt(String key, int value)	retorna o valor Int referente a chave ou um valor predefinido caso a chave não seja encontrada
getFloat(String key, float value)	retorna o valor Float referente a chave ou um valor predefinido caso a chave não seja encontrada
getString(String key, String value)	retorna o valor String referente a chave ou um valor predefinido caso a chave não seja encontrada
getBoolean(String key, boolean value)	retorna o valor Boolean referente a chave ou um valor predefinido caso a chave não seja encontrada

commit()	salva as preferências no objeto SharedPreferences associado e em disco
----------	--

Para maiores informações, acesse o link:

<https://developer.android.com/reference/android/content/SharedPreferences.html>



Esta segunda opção de persistência de dados, possibilita que sejam salvos arquivos na memória interna do aparelho.

É bastante oportuno ressaltar que esta memória interna é a mesma onde são armazenados os arquivos da SharedPreferences e que quando o aplicativo é desinstalado, seus dados são apagados, pois são dados privados de seu aplicativo.

Dentre várias classes que nos permitem efetuar este tipo de persistência, podemos destacar:

Para gravação de arquivos:

FileWriter:

Exemplo:

```
File arquivo = new File(getFilesDir().getPath() + "/dados.txt");
FileWriter fw;
fw = new FileWriter(arquivo, true);
fw.append(nome); fw.append("\n");
fw.append(idade); fw.append("\n");
fw.flush();
fw.close();
```

FileOutputStream.

Exemplo:

```
FileOutputStream fOut = openFileOutput("arquivoTeste", MODE_WORLD_READABLE);
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Para leitura de arquivos:

FileReader:

Exemplo:

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

Exemplo:

```
FileInputStream fin = openFileInput("arquivoTeste");  
int c;  
String temp="";  
while( (c = fin.read()) != -1){  
temp = temp + Character.toString((char)c);  
}  
fin.close();
```

Para mais detalhes consulte o link: <https://developer.android.com/reference/java/io/FileOutputStream.html>

<https://developer.android.com/reference/java/io/FileInputStream.html>

<https://developer.android.com/reference/java/io/FileWriter.html>

<https://developer.android.com/reference/java/io/FileReader.html>



External Storage

Muitas aplicações precisam efetuar sua persistência em algum tipo de memória externa, como exemplo um SDCard, que é removível. Felizmente o Android oferece suporte nativo para este tipo de persistência, porém precisamos ter o cuidado de sempre efetuar a chamada do método Environment.getExternalStorageState().

Este é responsável por nos informar se a mídia esta disponível ou não.

Exemplo:

```
boolean mExternalStorageAvailable = false;  
boolean mExternalStorageWriteable = false;  
String state = Environment.getExternalStorageState();  
if (Environment.MEDIA_MOUNTED.equals(state)) {  
    mExternalStorageAvailable = mExternalStorageWriteable = true;  
}  
else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {  
    mExternalStorageAvailable = true; mExternalStorageWriteable = false;  
} else {  
    mExternalStorageAvailable = mExternalStorageWriteable = false;  
}
```



SQLite Databases

O SQLite é banco de dados que ao contrário da maioria de software de banco de dados, como Oracle, Sybase e SQL Server, SQLite não está sob o controle de um programa servidor, em separado do programa do cliente.

Em vez disso, o programa do usuário possui uma biblioteca SQLite compacto, que lida com todo o acesso ao banco de dados. Isso simplifica o software e elimina a necessidade de o usuário a instalar, configurar e manter informações de banco de dados complexo.

Este pequeno e notável banco de dados tem se tornado cada vez mais popular na comunidade da TI, principalmente por desenvolvedores Android e iOS, uma vez que ambas plataformas oferecem suporte nativo.

Método	Descrição
<code>SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)</code>	este construtor cria um objeto para criar, abrir e gerenciar o banco de dados.
<code>SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)</code>	este construtor cria um objeto para criar, abrir e gerenciar o banco de dados. Ele especifica o gerenciador de erro.
<code>public abstract void onCreate(SQLiteDatabase db)</code>	método chamado apenas uma vez quando a base de dados é criada pela primeira vez.
<code>public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)</code>	método chamado quando banco de dados precisa ser atualizado.
<code>public synchronized void close ()</code>	método que fecha o banco de dados.

Classe SQLiteOpenHelper

No Android, os bancos criados serão acessíveis pelo nome para qualquer classe da aplicação, mas não poderão ser acessados externamente, salvo se for utilizado um provedor de conteúdo.

O método recomendado para criar um banco de dados SQLite novo é criar uma subclasse de SQLiteOpenHelper e implementar os métodos `onCreate()` e `onUpgrade()`.

A tabela abaixo destaca alguns métodos da classe SQLiteOpenHelper:

Para mais detalhes consulte o link:

<https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>

Classe SQLiteDatabase

Esta classe contém métodos a serem executadas no banco de dados SQLite, como criar, atualizar, excluir, selecione etc.

Existem muitos métodos na classe SQLiteDatabase. Alguns deles são os seguintes:

Métodos	Descrição
void execSQL(String sql)	Executa script SQL que não seja SELECT. Exemplo: CREATE TABLE, INSERT, UPDATE, etc.
long insert(table, null, ContentValues values)	Insere um registro e retorna o id. INSERT INTO <table> (values) VALUES (val- ues)
int update(table, ContentValues values, whereClause, whereArgs)	Altera registro(s) e retorna quantidade de linhas modificadas. UPDATE <table> SET <values> WHERE <whereClause+whereArgs>
int delete(table, whereClause, whereArgs)	Deleta registro(s) e retorna quantidade de linhas modificadas. DELETE FROM <table> WHERE <whereClause+whereArgs>

<code>boolean isOpen()</code>	Verifica se está aberto
<code>Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)</code>	Mostra e executa um SQL de consulta na forma: SELECT <distinct> <columns> FROM <table> WHERE <selection+selectionArgs> GROUP BY <groupBy> HAVING <having> ORDER BY <orderBy> LIMIT <limit>
<code>Cursor query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)</code>	
<code>static SQLiteDatabase openDatabase(String path, CursorFactory factory, int flags)</code>	Abre banco de dados de acordo com os flags: OPEN_READWRITE, OPEN_READONLY, CREATE_IF_NECESSARY, NO_LOCALIZED_COLLATORS.
<code>static SQLiteDatabase openOrCreateDatabase(String path, CursorFactory factory)</code>	Abre ou cria banco de dados.
<code>static SQLiteDatabase openOrCreateDatabase(File file, CursorFactory factory)</code>	

Exemplo:

```
public class MeuBanco extends SQLiteOpenHelper {  
    public MeuBanco (Context context, String name, int version) {  
        super(context, name, null, version);  
    }  
    @Override  
    public void onCreate(SQLiteDatabase sqld) {  
        sqld.execSQL("CREATE TABLE usuarios_tbl ( \"id_usuarios"  
        " INTEGER PRIMARY KEY autoincrement, usuario varchar(45) NOT NULL, senha  
        varchar(45) NOT NULL, nome_completo varchar(45) NOT NULL," )  
    }  
    @Override  
    public void onUpgrade(SQLiteDatabase sqld, int i, int il) {  
    }  
}
```

Para mais detalhes consulte o link:

[https://developer.android.com/reference/android/database/sqlite/ SQLite- Database.html](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html)



Revisão geral

Nesse capítulo faremos uma revisão geral de todos os aspectos abordados na nossa apostila ao final iremos produzir uma aplicação de agenda que será capaz de guardar informações de contatos, utilizando banco de dados, menus, intent, avisos ao usuário. Vamos começar.

Crie seu novo projeto com o nome de agenda. Lembre-se de usar a api 15 que irá ajudar seu app a ser compatível com quase todos os androids disponíveis no mercado. Se estiver com dúvidas de como criar um novo projeto por favor retorne a leitura na página 6. Lembre-se de que quando for escolher o tipo de activity, escolha a opção empty activity. Nomeie sua activity como ListaAlunos Activity.

Lembrando que ao criar a activity o Android Studio cria dois arquivos:

A Activity, que corresponde a tela do Android e representa o comportamento da aplicação. A Activity está na ListaAlunosActivity.java.

O .xml, que contém os componentes da tela, isto é, o seu conteúdo. A classe .xml está dentro da «pasta» Layout e na aba activity_lista_alunos.xml.

Vamos começar a codificar o layout, para isso abra o arquivo activity_listalunos.xml, e começa a implementar o seguinte código:

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

Vamos acrescentar agora um componente diferente do Android, a ListView. Esse componente representa uma lista.

O código do layout ficará da seguinte forma:

```
<LinearLayout    xmlns:android="http://schemas.android.com/apk/res/an-  
droid"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical">  
    <ListView android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:id="@+id/lista_alunos" />  
  
</LinearLayout>
```

Vamos agora acessar a ListaAlunosActivity.java e nela vamos declarar apenas uma Array simples, uma String e dentro dela inserimos os alunos da nossa lista. Então, temos a seguinte Array: String[] = {}. Dentro das chaves inserimos os nomes dos alunos. Não podemos esquecer de acrescentar também o nome da variável, no nosso caso, “alunos”. Ficaremos com:

```
public class ListaAlunosActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_lista_alunos);  
  
        String [] alunos= {"Daniel","Ronaldo","Jeferson","Felipe"};  
        ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos);  
    }  
}
```

Repare que foi usado método findViewById para que seja feito a referência entre o objeto do código java e o widget lista do layout

Agora falta inserir na lista os nomes dos alunos. Para fazer isso precisamos converter os alunos da String para View. A classe ArrayAdapter faz exatamente isso! Para instancia-la, pulamos para a próxima linha e inserimos new ArrayAdapter <>:

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_lista_alunos);  
  
    String[] alunos = {"Daniel", "Ronaldo", "Jeferson", "Felipe"};  
    ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos);  
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, alunos);  
}
```

Entre os argumentos <>, note que preenchemos com uma String, pois é isso que desejamos importar.

Os parâmetros que a ArrayAdapter pede são o contexto e o layout. O contexto serve para identificação. Já o layout serve para instruir ao android como construir a lista. Utilizamos um layout padrão da lista. Ao final informamos os dados que serão convertidos no adapter, no nosso caso são os alunos.

Agora, temos dois objetos separados, uma Lista de alunos, que é a ListView e também, um Adapter que irá converter os alunos que são Strings em View para serem introduzidos na lista. Iremos agora juntar as duas coisas!

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_lista_alunos);  
  
    String[] alunos = {"Daniel", "Ronaldo", "Jeferson", "Felipe"};  
    ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos);  
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, alunos);  
    listaAlunos.setAdapter(adapter);
```

Agora sim colocamos o ArrayAdapter dentro do nosso objeto listaAlunos, pois somente o adapter é capaz de pegar os textos converter e inserir dentro da lista. Vamos rodar o emulador para ver o que acontece.

Temos uma lista completa, com todos os nomes dos alunos, espaçamento e até uma animação, mas agora para continuar inserindo novos alunos é preciso um novo formulário, ou seja, uma nova Activity: “File> New> Activity> Blank Activity”.

Seguindo esse caminho abrirá uma nova janela e nela vários dados serão pedidos como o nome da activity, o título, etc. Vamos nomear essa nova activity de «FormularioActivity»

Vamos primeiramente implementar nosso layout inserindo os campos de nome, Endereço, Site, Nota e Telefone. Para que o usuário consiga inserir as informações é necessário o EditText e dentro dele inserimos o atributo hint, que indica o que constará nesse campo, no caso, nomes dos alunos:

EditText android:hint="Nome" Ficará assim:

```
<ScrollView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    xmlns:android="http://schemas.android.com/apk/res/android" >  
  
<LinearLayout android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical">  
  
    <EditText android:hint="Nome"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:id="@+id/formulario_nome" />  
  
    <EditText android:hint="Endereço"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:id="@+id/formulario_endereco" />  
  
    <EditText android:hint="Telefone"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:id="@+id/formulario_telefone"/>  
  
    <EditText android:hint="Site"  
        android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:id="@+id/formulario_site"/>

    <RatingBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:numStars="5"
        android:layout_gravity="center"
        android:id="@+id/formulario_nota" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Salvar"
        android:id="@+id/formulario_salvar" />

</LinearLayout>

</ScrollView>

```

Passamos a ter duas telas. Mas, o Android só pode mostrar uma de cada vez, portanto, é preciso dizer ao Android qual das telas deve ser aberta!

Para isso, abra o arquivo `AndroidManifest.xml` e repare no código

```

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

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

```

```
    android:label="@string/title_activity_formulario" >
  </activity>
</application>
```

Perceba que nossa nova Activity não tem nenhum intent filter, ou seja, quando a aplicação for iniciada ela simplesmente será ignorada para concertar isso por enquanto faça as seguintes modificações:

```
<application
    android:allowBackup="true" android:icon="@mipmap/ic_launcher" android:label="Agenda"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".ListaAlunosActivity" android:label="@string/app_name" >
    </activity>

    <activity
        android:name=".FormularioActivity" android:label="@string/title_activity_formulario" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

Com isso transformamos o FormularioActivity em nossa tela principal

Pronto, agora mande a aplicação rodar e veja como ficou o layout da nossa nova tela.

Para que a lista de alunos reapareça vamos de novo modificar o arquivo AndroidManifest.xml da nossa aplicação ela ficará assim:

```
<application android:allowBackup="true" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:theme="@style/AppTheme" >
    <activity android:name=".ListaAlunosActivity" android:label="@string/app_name" >
        <intent-filter>
```

```
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity
    android:name=".FormularioActivity"
    android:label="@string/title_activity_formulario" >
</activity>
</application>
```

Pronto! Agora, ao rodar a aplicação o que encontramos é a lista de alunos.

O próximo passo é criar uma forma de sair da lista para chegar ao formulário. Para solucionar a questão, vamos inserir um botão “novo aluno” na activity_lista_alunos.xml.

Vamos estilizar nosso botão criando um fundo redondo para ele. Selecione a pasta “res” e clique com o botão direito do mouse em “drawable” e “New > Drawable resource file”. Seguindo esse caminho abrirá uma janela que pedirá o “File Name”, que denominaremos de “Fundo”. Abra o arquivo recém criado e digite o seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid
        android:color="#ff0000"/>

</shape>
```

O próximo passo é abrir o arquivo activity_lista_alunos.xml e inserir o seguinte código:

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

```

<Listview
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/lista_alunos"/>

    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/formulario_telefone"/>

<EditText android:hint="Site"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/formulario_site"/>

<RatingBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:layout_gravity="center"
    android:id="@+id/formulario_nota" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Salvar"
    android:id="@+id/formulario_salvar" />

</LinearLayout>

</ScrollView>

```

Passamos a ter duas telas. Mas, o Android só pode mostrar uma de cada vez, portanto, é preciso dizer ao Android qual das telas deve ser aberta!

Para isso, abra o arquivo `AndroidManifest.xml` e repare no código

```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher" android:label="Agenda"
    android:theme="@style/AppTheme" >

```

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

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".FormularioActivity" android:label="@string/title_activity_formulario" >
    </activity>
</application>
```

Perceba que nossa nova Activity não tem nenhum intent filter, ou seja, quando a aplicação for iniciada ela simplesmente será ignorada para concertar isso por enquanto faça as seguintes modificações:

```
<activity
    android:name=".FormularioActivity" android:label="@string/title_activity_formulario" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>
```

Com isso transformamos o FormularioActivity em nossa tela principal

Pronto, agora mande a aplicação rodar e veja como ficou o layout da nossa nova tela.

Para que a lista de alunos reapareça vamos de novo modificar o arquivo AndroidManifest.xml da nossa aplicação ela ficará assim:

```
<application android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".ListaAlunosActivity" a
        ndroid:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

        android:name=".FormularioActivity"
        android:label="@string/title_activity_formulario" >
    </activity>
</application>
```

Pronto! Agora, ao rodar a aplicação o que encontramos é a lista de alunos.

O próximo passo é criar uma forma de sair da lista para chegar ao formulário. Para solucionar a questão, vamos inserir um botão “novo aluno” na activity_lista_alunos.xml.

Vamos estilizar nosso botão criando um fundo redondo para ele. Selecione a pasta “res” e clique com o botão direito do mouse em “drawable” e “New > Drawable resource file”. Seguindo esse caminho abrirá uma janela que pedirá o “File Name”, que denominaremos de “Fundo”. Abra o arquivo recém criado e digite o seguinte código:

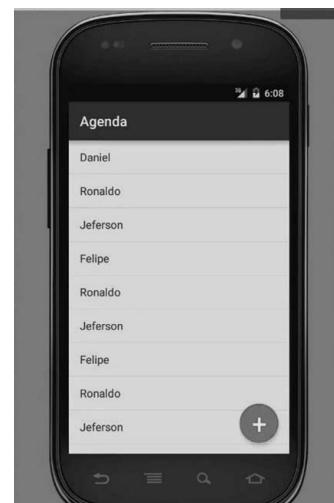
```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid
        android:color="#ff0000"/>

</shape>
```

O próximo passo é abrir o arquivo activity_lista_alunos.xml e inserir o seguinte código:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android">  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical">  
  
    <Listview android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:id="@+id/lista_alunos"/>  
  
    <Button android:layout_width="56dp" android:layout_height="56dp"  
        android:text "+"  
        android:textColor="#ffffffff"  
        android:textSize="40sp"  
        android:elevation="6dp"  
        android:layout_alignParentBottom="true"  
        android:layout_alignParentRight="true"  
        android:layout_marginBottom="16dp"  
        android:layout_marginRight="16dp"  
        android:background="@drawable/fundo"  
        android:stateListAnimator="@null" android:id="@+id/novo_aluno"/>  
  
    </RelativeLayout>
```

Para ver como ficou, vamos salvar e rodar:



Agora que temos um botão flutuante, queremos que ele tenha uma ação real. Queremos que ele nos leve até o formulário e que nós possamos salvar os nomes dos alunos!

Se estamos falando de um comportamento que queremos introduzir no botão, vamos alterar o `ListaAlunosActivity.java`.

Implementaremos o seguinte código nela:

```
AlunosActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_lista_alunos);  
        String [] alunos= {"Daniel", "Ronaldo", "Jeferson", "Felipe"};  
        ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos);  
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.  
        simple_list_item_1, alunos);  
        listaAlunos.setAdapter(adapter);  
  
        Button novoAluno = (Button) findViewById (R.id.novo_aluno);  
        novoAluno.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick (View v) {  
                Intent intentVaiProFormulario = new Intent (ListaAlunosActivity.this, Formu-  
                larioActivity.class)  
                startActivity(intentVaiProFormulario);  
  
            }  
        });  
    }  
}
```

Vamos salvar e dar play para ver como ficou no emulador. Bom, depois que apertamos o botão de “+” somos redirecionados para o formulário, mas ao chegar no formulário e selecionarmos o salvar permanecemos na tela do formulário.

Para resolver esse problema agora vamos voltar na o arquivo `FormulárioActivity.java` e usar a mesma ideia. Siga o código dentro do método `onClick`:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_formulario);  
  
    Button botaoSalvar = (Button) findViewById(R.id.formulario_salvar);  
    botaoSalvar.setOnClickListener(new View.OnClickListener() {  
        @Override public void onClick(View v) {  
            Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.LENGTH_SHORT).  
            show();  
            finish();  
        }  
    });  
}
```

Execute o veja o seu progresso.

Agora Vamos introduzir o botão de salvar junto a nossa barra de “Formulário” ficando assim como um menu. Desse modo, se o formulário for muito grande e a barra tiver que rolar, o botão ficará em cima dela.

Para fazer essas modificações, primeiramente temos que criar uma pasta de menu dentro da pasta res. Depois crie um arquivo de menu dentro dessa nova pasta e coloque o seguinte código:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
      xmlns:app="http://schemas.android.com/apk/res-auto"  
      xmlns:tools="http://  
                  schemas.android.com/tools"  
      tools:context="br.com.jefferson.agenda.FormularioActivity">  
  
<item android:id="@+id/menu_formulario_ok"  
      android:title="Ok"  
      android:icon="@drawable/ic_input_add" app:showAsaction="always"/>  
  
</menu>
```

Agora que criamos o .xml precisamos utilizá-lo em nosso código. Vamos ao arquivo FormularioActivity.java e codificaremos o seguinte método:

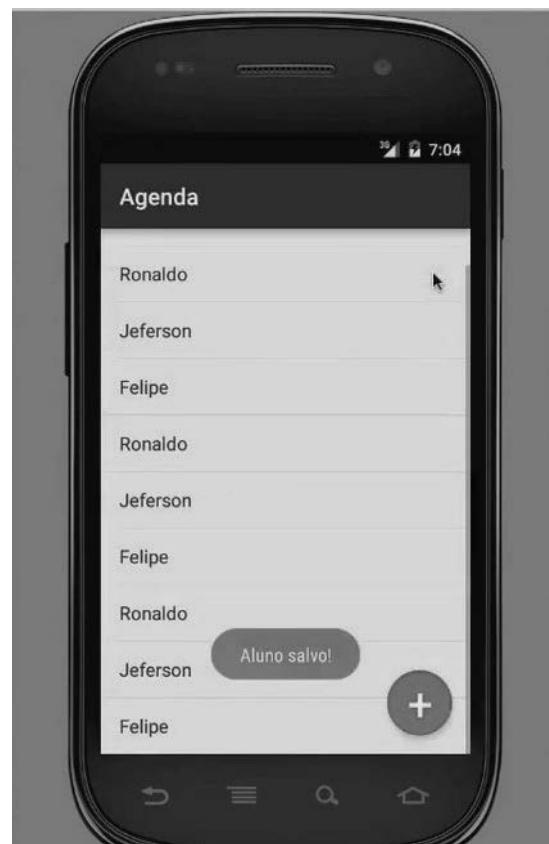
```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.menu_formulario, menu);  
  
    return super.onCreateOptionsMenu(menu);  
}  
  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_formulario_ok:  
            Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.LENGTH_SHORT).show();  
            finish(); break;  
  
    }  
    return super.onOptionsItemSelected(item);  
}
```

Vamos deletar também o botão de salvar, pois ele não existe mais na nossa tela.

Deletamos o seguinte:

```
Button botaoSalvar = (Button) findViewById(R.id.formulario_salvar);  
botaoSalvar.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
    }  
});
```

Vamos salvar e rodar o emulador?



Ao adicionar um novo aluno, preencheremos o formulário e dando um "Ok", ele nos avisa que o aluno foi salvo e retorna para a lista!

Agora de fato vamos implementar a funcionalidade para salvar o aluno. Para isso, realizaremos alterações no arquivo, FormularioActivity.

Lembrando

O Android avisa qual item do menu é clicado através do onOptionsItemSelected. No onOptionsItemSelected acrescentamos também um caso e o Toast para que uma mensagem apareça quando o botão for clicado.

Para salvar os alunos recém adicionados, precisaremos recuperar os dados novos que inserimos. Então, vamos adicionar algumas informações na Override da onOptionsItemSelected. Temos:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) { switch (item.getItemId()) {  
    case R.id.menu_formulario_ok;  
        Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.LENGTH_SHORT).show();  
}
```

```

EditText campoNome = (EditText) findViewById(R.id.formulario_
nome);
String nome = campoNome.getText().toString();

finish(); break;
}

return super.onOptionsItemSelected(item);
}

```

Damos um “Enter” e na linha abaixo de ‘Toast’ iremos acrescentar algumas informações. Como necessitamos armazenar os dados que introduzimos no formulário. A referência será buscada através do findViewById, portanto, utilizaremos entre parênteses a classe R., Id e o nome da tela, que é formulario_nome e indica quem está sendo procurado. Ao digitar a palavra «formulario», serão mostradas algumas opções. Ficaremos com, findViewById(R.Id.formulario_nome).

Como esse componente é do tipo EditText, vamos adicionar EditText e o campoNome, ficaremos com, EditText campoNome = findViewById(R.Id.formula-
rio_nome).

Em cima do campoNome, para importar a classe, utilizaremos o atalho “Alt+Enter”. O Android vai pedir para fazermos o Cast, então, damos um “Alt+Enter” de novo e pronto.

Ficaremos com EditText campoNome = (EditText) findViewById(R. Id.formula-
rio_nome). Primeiro, pegaremos o campo e agora pegaremos o valor dele acrescen-
tando campoNome. . e getText. Adicionaremos uma String na frente disso, seguida de «nome» e um «=». Teremos, String nome = campoNome.getText.

Mas, se fizermos apenas isso ele devolverá um editable e não queremos isso. Então, adicionaremos depois do getText mais um . e toString. Faremos isso para con-
verter o campo em algo que possa ser colocado de fato na String. Teremos, String
nome = campoNome.getText().toString(). Ficaremos com :

```

@Override
public boolean onOptionsItemSelected(MenuItem item) { switch (item.get
itemId()) {
    case R.id.menu_formulario_ok;
        Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.LENGTH_
SHORT).show();
}

```

```

        EditText campoNome = (EditText) findViewById(R.id.formulario_nome);
        String nome = campoNome.getText().toString();

        finish(); break;
    }

    return super.onOptionsItemSelected(item);
}

```

Fizemos isso com o campo “nome” e faremos o mesmo com o campo endereço. Para isso, daremos um “Enter” depois do que acabamos de acrescentar e na próxima linha teremos que pegar novamente a referência do campo, no caso, o Id de endereço. Digitaremos, findViewById(R.id.formulario_endereco) e como também é um EditText, adicionaremos isso e campo endereço. Na sequência, importaremos e faremos o Cast. Por fim, teremos EditText campoEndereco = campoEndereco findViewById(R.id.formulario_endereco). Na próxima linha adicionaremos o String, endereco, =, campoEndereco, o getText, que serve para pegar o texto e o to String. Teremos:

```

EditText campoEndereco = (EditText) findViewById(R.id.formulario_Endereco);
String Endereco = campoEndereco.getText().toString();

```

Faremos o mesmo nos campos de telefone e site, basta selecionar o que acabamos de escrever, do EditText do nome até a String do Endereço, dar um «Comand+C» ou

«Ctrl+C» e «Comand+V» ou «Ctrl+V». Colaremos isso abaixo do Endereço. Basta alterar os nomes pelos campos que desejamos, no caso «Site» e «Telefone». Teremos:

```

@Override
public boolean onOptionsItemSelected(MenuItem item) { switch (item.getItemId()) {
    case R.id.menu_formulario_ok;
        Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.LENGTH_SHORT).show();
        EditText campoNome = (EditText) findViewById(R.id.formulario_nome);
        String nome = campoNome.getText().toString();
        Endereco);
        Telefone);
    }
    EditText campoEndereco = (EditText) findViewById(R.id.formulario_

```

```

String Endereco = campoEndereco.getText().toString();

EditText campoTelefone = (EditText) findViewById(R.id.formulario_
String Telefone = campoTelefone.getText().toString();
EditText campoSite = (EditText) findViewById(R.id.formulario_Site); String
Site = campoSite.getText().toString();

finish(); break;

return super.onOptionsItemSelected(item);
}

```

Imagine se tivéssemos mantido o antigo botão de salvar, aquele que ficava na parte de baixo do formulário. Teríamos que repetir todo o código que acabamos de escrever para ele ter a mesma função. Nessas situações, quando é preciso repetir muitas vezes o mesmo código, podemos criar uma classe para que ela faça isso por nós. Portanto, vamos deslocar os códigos em uma classe auxiliar do formulário.

Para criar uma nova classe vamos no pacote , clicamos com o botão direito e “New > Java Class”. Vai aparecer uma janela com um campo Name para ser preenchido, nomearemos nossa classe de «FormularioHelper». Abrirá uma tela com a nova classe:

```

public class FormularioHelper {
}

```

Vamos fazer todos os findViewById que estão na aba FormularioActivity.java, na nova classe, justamente, para facilitar. Assim, não precisaremos ficar recriando eles sempre que necessário. Vamos introduzir os findViewById no início, no construtor do «FormularioHelper».

Para fazer o construtor basta dar um enter depois do public class FormularioHelper e ele já aparece na tela, basta escrever public FormularioHelper. Agora, dentro, iremos adicionar os findViewById.

Na próxima linha adicionaremos o findViewById e digitaremos o campo que queremos puxar, no caso, o formulario_nome. Ficaremos com, findViewById(R. Id.formulario_nome). Repare que o findViewById ficará em vermelho, pois não é possível localizar o método.

O `findViewById` é um método oriundo da `AppCompatActivity`, e quando estamos dentro da `activity` podemos chamar o método normalmente. Mas, estamos no `<FormularioHelper.java>`. Para utilizar o `findViewById` no

`<FormularioHelper.java>` necessitamos de uma referência da nossa `activity`. Vamos adicionar essa referência no construtor da nossa classe, adicionaremos entre os parênteses que seguem o `public formularioHelper` a referência de que estamos trazendo isso da `FormularioActivity` e nomearemos ela de «`activity`». Ficaremos com `FormularioHelper(FormularioActivity activity)`.

Como essa referência vem de fora do `FormularioHelper.java` temos que referenciar também o `findViewById`. Digitaremos na frente de tudo `activity` e um ponto, teremos, `activity.findViewById(R.Id.formulario_nome)`.

Falta acrescentar que é um `EditText` do campoNome que será = a `activity.findViewById(R.Id.formulario_nome)`. Teremos, `EditText campoNome = activity.findViewById(R.Id.formulario_nome)`. Damos um “Alt+Enter” para importar e fazemos também o Cast.

Vamos ter o seguinte:

```
public class FormularioHelper {  
    public FormularioHelper(FormularioActivity activity) {  
        EditText campoNome = (EditText) activity.findViewById(R.Id.formulario_  
nome);  
    }  
}
```

Agora, temos o campo do nome pronto!

Basta reproduzir o que já escrevemos. Selecionei com o mouse, ‘`EditText campoNome = (EditText) activity.findViewById(R.Id.formulario_nome)`’ e damos um “Command+C” e “Command+V”. Agora, trocaremos apenas os nomes do `EditText` e alteraremos os `Id`.

Acrescentaremos um último campo, que é o da nota. Atenção aqui, ele não é um `EditView`, ele é um `RatingBar`, então, temos que alterar os tipos das variáveis. Ficaremos com `RatingBar campoNota = (RatingBar) activity.findViewById(R.Id.formula-
rio_nota)`. Não esquecendo de importar a classe com o “Alt+Enter”.

E ficaremos com:

```
public class FormularioHelper {  
  
    public FormularioHelper(FormularioActivity activity) {
```

```

EditText campoNome = (EditText) activity.findViewById(R.id.formulario_
nome);
EditText campoEndereco = (EditText) activity.findViewById(R.id.formulario_
endereco);
EditText campoTelefone = (EditText) activity.findViewById(R.id.formulario_
telefone);
EditText campoSite = (EditText) activity.findViewById(R.id.formulario_site);
RatingBar campoNota = (RatingBar) activity.findViewById(R.id.formulario_
nota);
}
}

```

Agora, na aba FormularioActivity.java podemos apagar os EditText que tínhamos e apagaremos o seguinte:

```

EditText campoNome = (EditText) findViewById(R.id.formulario_nome); String
nome = campoNome.getText().toString();
EditText campoEndereco = (EditText) findViewById(R.id.formulario_Endereco);

String endereco = campoEndereco.getText().toString();

```

```

EditText campoTelefone = (EditText) findViewById(R.id.formulario_Telefone);
String telefone = campoTelefone.getText().toString();

```

```

EditText campoSite = (EditText) findViewById(R.id.formulario_Site); String site =
campoSite.getText().toString();

```

E ficaremos apenas com:

```

@Override
public boolean onOptionsItemSelected(MenuItem item) { switch (item.get-
itemId()) {
    case R.id.menu_formulario_ok;
        Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.
LENGTH_SHORT).show();

        finish(); break;
}

return super.onOptionsItemSelected(item);
}

```

Agora, precisamos substituir. Quando formos buscar o item dentro da classe ItemSelected, iremos introduzir a classe do FormularioHelper. Vamos instanciar essa classe em algum lugar, por exemplo, no onCreate. Então, na onCreate adicionamos, helper = new FormularioHelper. Ficaremos com helper = new FormularioHelper. Só que ele ainda pede uma referência para a activity, então, digitaremos o this entre os parênteses, como parâmetro. Ficaremos com FormularioHelper helper = new FormularioHelper (this).

Mas, adicionando no Toast o helper, ele não é encontrado, por isso, apagamos o FormularioHelper que tínhamos escrito e transformamos isso em atributo. Repare que

o helper fica em vermelho, damos um «Alt+Enter» e escolhemos «Create final helper». O Android irá criar, acima da Override um private FormularioHelper helper. Teremos:

```
private FormularioHelper helper;
```

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_formulario);  
  
    helper = new FormularioHelper (this);  
}
```

E quando quisermos utilizar o helper, vamos embaixo, na classe onOptionsItemSelected e depois do Toast, na próxima linha, acrescentamos helper e damos um .. Se não tivéssemos criado o atributo na onCreate não iríamos conseguir localizar o helper e ele continuaria em vermelho.

O que faremos a partir do Helper é pegar os dados dos aluno. Para isso, vamos adicionar uma String, teremos String nome = helper.peganome e faremos isso com todos os outros campos. O que não é muito prático.

Através do helper desejamos sinalizar todos os campos que aparecem no formulário. Os campos da tela do celular, o nome, o endereço, a nota, o site e o telefone representam um «aluno» no sistema. O ideal é que tivéssemos um único método no helper, que seria String nome = helper.pegaAluno, responsável por trazer todos os dados que queremos. Mas, no lugar de String devemos ter um objeto de tipo Aluno. Ficaremos com Aluno aluno = helper.pegaAluno.

Teremos:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_formulario_ok;  
            Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.  
LENGTH_SHORT).show();  
            Aluno aluno = helper.pegaAluno();  
            finish();  
            break;  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```

Vamos fazer um “Alt+Enter” no objeto Aluno. O Android Studio sugere que criemos a classe alunos, a «Create class Aluno». Portanto, vamos selecionar essa opção e criar a classe.

Abrirá uma janela, que tem um campo Destination package. Vamos criar um pacote que já conhecemos e chamá-lo de modelo. Damos um “Ok”.

Ele criará uma pasta chamada “modelo” e ela estará junto com as outras, no lado direito da tela. Nela teremos a classe “Aluno” e dentro do ‘Aluno.java’ colocaremos os dados do aluno.

Vamos acrescentar abaixo do public class, na próxima linha, o private String nome e faremos o mesmo com os demais campos que queremos trazer para o Aluno.java. A nota, entretanto, será Double e não String. Logo menos, começaremos a trabalhar com o banco de dados, vamos acrescentar também um Id para esse aluno e já deixar isso para o futuro. Logo na linha de baixo de public class digitamos private Long id. Teremos:

```
public class Aluno { private Long id;  
    private String nome;  
    private String endereco  
    private String telefone;  
    private String site;  
    private Double nota;  
}
```

Como colocamos que todos os nossos objetos estão privados, teremos que criar um Getter and Setter. Damos uns dois “Enter” depois da última linha e no Mac utilizamos o atalho «Command+M» e no Windows «Alt+Insert» e selecionamos o «Getter and Setter». Vai abrir um janela e nela selecionaremos todos os campos que gostaria que ele gerasse um «Getter» e um «Setter» e damos um «Ok».

Automaticamente nossa classe terá vários “Getter” e “Setter” definidos.

Agora que a classe “Aluno” está certa, vamos voltar no FormularioActivity. No Helper acrescentamos o pegaAluno, mas ele ainda está em vermelho, falta acrescentar um método para ele. Para isso, damos um “Alt+Enter” em cima do pegaAluno e selecionamos

«Create method <pegaAluno>». Pronto, agora temos um public Aluno pegaAluno. Ficaremos com:

```
public class FormularioHelper {  
  
    public FormularioHelper(FormularioActivity activity) {  
        EditText campoNome = (EditText) activity.findViewById(R.id.formulario_  
nome);  
        EditText campoEndereco = (EditText) activity.findViewById(R.id.formulario_  
endereco);  
        EditText campoTelefone = (EditText) activity.findViewById(R.id.formulario_  
telefone);  
        EditText campoSite = (EditText) activity.findViewById(R.id.formulario_site);  
        RatingBar campoNota = (RatingBar) activity.findViewById(R.id.formulario_  
nota);  
    }  
  
    public Aluno pegaAluno() { return null;  
    }  
}
```

Podemos introduzir abaixo da classe pegaAluno um aluno, assim, acrescentamos o Aluno aluno = new Aluno. E na linha de baixo digitamos aluno.setNome. O nome do aluno vem do formulário, então, acrescentamos campoNome..getText. Ficaremos com aluno.setNome(campoNome.getText).

Mas, o campo nome ficará em vermelho, isso ocorre porque antes declaramos que ele era um EditText. Para resolver esse problema basta apagar o EditText de todos os campos da classe FormularioHelper e deixar apenas o uso deles:

```
public class FormularioHelper {  
  
    public class FormularioHelper(FormularioActivity activity) {  
        campoNome = (EditText) activity.findViewById(R.id.formulario_nome);  
        campoEndereco = (EditText) activity.findViewById(R.id.formulario_endereco);  
        campoTelefone = (EditText) activity.findViewById(R.id.formulario_telefone);  
        campoSite = (EditText) activity.findViewById(R.id.formulario_site);  
        campoNota = (RatingBar) activity.findViewById(R.id.formulario_nota);  
    }  
    public Aluno pegaAluno() { Aluno aluno = new Aluno ();  
        aluno.setNome(campoNome.getText());  
  
        return null;  
    }  
}
```

Agora, nenhum deles está definido. Então, em todos os campos faremos “Alt+Enter” e criaremos “Fields”. Todos os campos estarão com atributos:

```
public class FormularioHelper {  
  
    private EditText campoNome; private EditText campoEndereco; private  
    EditText campoTelefone; private EditText campoSite; private Ratingbar campoNo-  
    ta;  
    public class FormularioHelper(FormularioActivity activity) { campoNome =  
        (EditText) activity.findViewById(R.id.formulario_  
            nome);  
        campoEndereco = (EditText) activity.findViewById(R.id.formulario_endere-  
            co);  
        campoTelefone = (EditText) activity.findViewById(R.id.formulario_  
            telefone);  
        campoSite = (EditText) activity.findViewById(R.id.formulario_site);
```

```
campoNota = (RatingBar) activity.findViewById(R.id.formulario_nota);
}

public Aluno pegaAluno() { Aluno aluno = new Aluno ();
aluno.setNome(campoNome.getText());

return null;
}
}
```

Agora conseguimos acessar a classe 'pegaAluno'!

Já digitamos aluno.setNome(campoNome.getText) e como ele nós devolve um editable, devemos converter isso. Usaremos para tanto um toString. Não esqueça de fechar com um ; Vamos repetir o aluno.setNome(campoNome.setText().toString()) por quatro vezes para inserir também os outros campos. Alteramos os nomes do campo inserindo o que ainda falta: endereco, telefone, site e nota.

A nota é um pouco diferente, para descrevê-la acrescentamos um getProgress que devolverá um Double e nele escreveremos valueOf. Para a nota teremos: aluno.setNota(Double.valueOf(campoNota.getProgress()))). Por fim, devolvemos o aluno que acabamos de construir com return aluno. Teremos:

```
public Aluno pegaAluno() {
Aluno aluno = new Aluno ();
aluno.setNome(campoNome.getText().toString());
aluno.setEndereco(campoEndereco.getText().toString());
aluno.setTelefone(campoTelefone.getText().toString());
aluno.setSite(campoSite.getText().toString());
aluno.setNota(Double.valueOf(campoNota.getProgress()));
return aluno;
}
```

Vamos voltar ao FormularioActivity.java. Observe onde estávamos:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_formulario_ok;  
            Toast.makeText(FormularioActivity.this, "Aluno salvo!", Toast.LENGTH_  
SHORT).show();  
            Aluno aluno = helper.pegaAluno();  
  
            finish(); break;  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```

Para confirmar se o código está correto vamos deslocar o `Toast.makeText(FormularioActivity.this, "Aluno salvo!"` para a linha de baixo do `Aluno aluno = helper.pegaAluno();`, através do “Command+V” e “Command+C”. No `Toast.concatenate()` depois de “Aluno”, o nome. Digitaremos:

```
Toast.makeText(FormularioActivity.this, "Aluno" + aluno.getNome() + "salvo!",  
Toast.  
LENGTH_SHORT).show();
```

E no todo teremos:

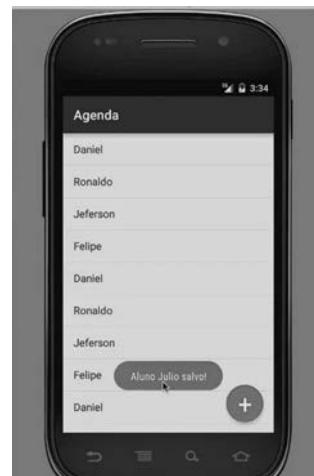
```
@Override  
public boolean onOptionsItemSelected(MenuItem item) { switch (item.ge-  
tItemId()) {  
    case R.id.menu_formulario_ok;  
  
        Aluno aluno = helper.pegaAluno(); Toast.makeText(FormularioActivity.this,  
"Aluno" + aluno.getNome() +  
"salvo!", Toast.LENGTH_SHORT).show();  
  
        finish();  
        break;  
    }  
}
```

```
return super.onOptionsItemSelected(item);  
}
```

Vamos rodar o emulador! No campo formulário acrescentamos um novo aluno, o Júlio:



Agora, quando clicarmos no “Salvar”, deve aparecer o “Aluno Júlio salvo”.



Conseguimos preparar o caminho para salvar o nome do aluno no banco de dados!

Conseguimos armazenar as informações preenchidas no formulário e jogá-las no objeto “Aluno”. Assim, quando acrescentamos um novo nome no formulário e damos um “check” o aplicativo reconhece a ação e dá uma resposta.

Agora, queremos que todos os novos formulários criados apareçam em nossa lista.

Isto é, que o nome de “Paulo” esteja junto aos demais!

Vamos começar a abrir o caminho para um banco de dados, que possibilitará armazenar as informações. Para fazer isso poderíamos modificar nossa Array, na ListaAlunosActivity.java. Mas, se fizermos isso “a mão”, sempre que desligarmos o celular qualquer novo aluno cadastrado será perdido.

Se tivermos as informações dos alunos em um arquivo ou banco de dados e a Array popular isso, não perderemos os alunos toda vez que a aplicação reiniciar. Portanto, para não perder mais as informações, vamos começar a colocar persistência no aplicativo!

Podemos inserir uma conexão para se ligar ao banco de dados tanto na ListaAlunosActivity.java quanto no FormularioActivity.java, mas as conexões nem sempre são códigos simples. A medida que formos acrescentando novas informações o código aumentaria e teríamos que espalhá-lo em diversos locais, o que complicaria um pouco o desenvolvimento.

Vamos fazer o que aprendemos anteriormente, que é isolar toda a parte da conexão em uma classe específica. Assim, sempre que quisermos manipular algo que diga respeito a um aluno, usaremos uma classe que nos ajude a fazer as modificações necessárias. Ela também irá abstrair toda a parte de acesso ao banco de dados, operações, conexão, etc. O nome do objeto que abstrai tudo o que acabamos de comentar é “DAO”.

Criaremos uma classe nova no *Android*, para isso, vamos na pasta do nosso pacote, clicamos com o botão direito e “New > Java Class”. Abrirá uma janela pedindo um Name, vamos nomear de “AlunoDAO”, damos um “Enter” e... Pronto! Criou o “AlunoDAO.java”.

Queremos colocar ele em outro pacote, para isso, vamos de novo no nosso pacote atual, clicamos com o botão direito e selecionamos “New > Package”. Abrirá uma janela pedindo para que digitemos o *Package name*. O nome do pacote que vamos criar será “dao”. Ele criará no canto esquerdo o “dao”, dentro do nosso pacote. Para trocar o “AlunoDAO” de lugar basta selecioná-lo e arrastá-lo para o novo package. Abrirá uma janela nova e nela é só pedir para refaturar o código selecionando o “Refactor”.

Criamos a nova classe, “AlunoDAO.java” e dentro dela vamos fazer a parte de conexão de bancos de dados. Imagine que, a medida que o tempo for passando, nosso objetivo seja introduzir uma nova coluna na nossa lista, por exemplo, a coluna foto dos alunos. Nesse caso, teríamos que fazer diversas verificações para que tudo ocorresse bem! Teríamos que pensar no caso de como o *Android* avisaria as pessoas que já tinham a aplicação baixada para atualizá-la, etc.

As coisas ficariam complicadas! Assim, vamos facilitar um pouco utilizando uma classe do *Android* que faz tudo isso por nós.

Vamos na aba ‘AlunoDAO.java’. O banco de dados que vamos utilizar no Android é o SQLite, uma versão simplificada do sql. Então, vamos transformar nossa classe em um SQLiteOpenHelper, para fazer isso digitaremos SQLiteOpenHelper na mesma linha do public class AlunoDAO. Vamos importar o SQLiteOpenHelper usando em cima dela o atalho «Alt+Enter».

Essa classe vai nos ajudar em todas as operações relativas ao banco de dados.

Mas, perceba que o que acabamos de digitar estará em vermelho, pois, existe um pequeno erro. O Android ainda requer um construtor uma vez que o SQLiteOpenHelper pede, obrigatoriamente, alguns parâmetros que são fornecidos, justamente, pelo construtor. Clicando em cima do SQLiteOpenHelper damos um

«Alt+Enter» e pedimos para que ele «Implement methods». Na hora que damos um

«Enter» ele vai pedir para que implementemos dois métodos, que são obrigatórios, o onCreatee o onUpgrade. Basta dar um “Ok”.

Nossa tela ficará da seguinte maneira:

```
public class AlunoDAO extends SQLiteOpenHelper{
    @Override
    public void onCreate (SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVer-
    sion)
    {
    }
}
```

Só que, ao fazer isso, o SQLiteOpenHelper continuará em vermelho, agora damos mais um «Alt+ Enter» e pedimos para ele gerar o seguinte construtor: Create constructor matching super. Uma nova janela será aberta e ele vai sugerir dois construtores, nós vamos utilizar apenas o primeiro, que selecionamos, e daremos “ok”

![mostrando a janela que é aberta e que sugere os construtores]<https://s3.amazonaws.com/caelum-online-public/Android+III/Android+I+Super+class+Constructor.png>“>

O que ele irá gerar é um construtor automático:

```
Context context, String name, SQLiteDatabase.CursorFactory factory, int version O código ficará assim:  
  
public class AlunoDAO extends SQLiteOpenHelper{  
    public AlunoDAO (Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {  
        super(context, name, factory, version);  
    }  
  
    @Override  
    public void onCreate (SQLiteDatabase db) {  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    }  
}
```

Veremos o que está sendo pedido no 'super'. Primeiro, o construtor pediu um contexto, que é a identificação para o java. Ele pede em seguida um name, mas com isso não precisamos nos preocupar, pois temos a opção de usar qualquer nome. Vamos tirar o primeiro parâmetro que colocamos no AlunoDAO, apagando o String name.

Ele irá pedir um factory, se quisermos customizar alguns parâmetros, mas como nem sempre precisamos fazer essas modificações, vamos apagar e substituir o factory por um null e apagaremos, por consequência o SQLiteDatabase.CursorFactory factory.

Por fim, ele pergunta qual é a versão do banco de dados. Como é a primeira, escreveremos apenas um 1 e apagamos o int version. No super ficará entre os parênteses o context, «Agenda», null, 1 e no public apenas Context context. Agora, temos tudo para que o construtor funcione. Ficaremos com:

```
public class AlunoDAO extends SQLiteOpenHelper {  
    public class AlunoDAO (Context context) {  
        super(context, "Agenda", null, 1);  
    }  
  
    @Override
```

```
public void onCreate (SQLiteDatabase db) {  
}  
  
}  
  
@Override  
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
}  
}
```

Perceba que ainda existem dois métodos. O primeiro é o onCreate, que é usado assim que o banco de dados é criado.

Para introduzir algo dentro do banco de dados, a primeira coisa a fazer é colocar uma tabela. Para criar uma tabela em sql vamos introduzir uma instrução, a String sql, e um comando para criar a tabela, o CREATE TABLE, e um nome para a tabela, p Alunos. Descreveremos entre os parênteses os nomes das colunas da tabela.

Colocaremos na primeira coluna um número de identificação, o id INTEGER PRIMARY KEY. Em seguida, inserimos em todos os campos do formulário o nome, o endereço, o telefone, o site e a nota. O dado de nome será acompanhado de TEXT, pois é um texto, portanto, não pode ser um campo sem nada. Aos demais campos acrescentaremos que são do tipo TEXT e a nota é do tipo REAL. Vamos fechar o parênteses e colocar um ;para dizer que nossa instrução acabou e mais um ; devido a linha java. Ficaremos com:

```
public class AlunoDAO extends SQLiteOpenHelper {  
    public class AlunoDAO (Context context) {  
        super(context, "Agenda", null, 1);  
    }  
  
    @Override  
    public void onCreate (SQLiteDatabase db) {  
        String sql = "CREATE TABLE Alunos (id INTEGER PRIMARY KEY, nome TEXT  
NOT NULL, endereço TEXT, telefone TEXT, site TEXT, nota REAL);";  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    }  
}
```

Depois da linha da String, damos um “Enter” e na próxima, para executar as instruções, acrescentamos o db.execSQL(sql); - usando a variável db que o Android passa como parâmetro. O método fica da seguinte maneira:

```
@Override  
public void onCreate (SQLiteDatabase db) {  
    String sql = "CREATE TABLE Alunos (id INTEGER PRIMARY KEY, nome TEXT  
NOT NULL, endereço TEXT, telefone TEXT, site TEXT, nota REAL);";  
    db.execSQL(sql);  
}
```

Vamos passar para o próximo método, o onUpgrade. Lembre-se que podemos ter duas situações, ou um novo banco é criado ou ele precisa ser atualizado. Toda vez que mudarmos algo no banco de dados vamos ter que alterar a versão para 2, 3, 4, etc.

Uma vez que o banco de dados é alterado e modificamos o número da versão é o onUpgrade que avisa que ela foi modificada e precisa ser atualizada. Assim, adicionamos algo que faça passar da versão 1 para a versão 2 e que avisa sobre a atualização. Como não temos uma atualização do banco de dados não precisaremos introduzir nada mas, vamos inserir um caminho para facilitar isso no futuro.

Vamos adicionar uma instrução para jogar fora a tabela, digitaremos o comando DROP TABLE IF EXISTS e o nome da tabela, no caso, «Alunos». Teremos, String sql = DROP TABLE IF EXISTS Alunos. Vamos pedir também para que ela seja executada, através de db.execSQL(sql).

Por que estamos fazendo isso?

Imagine que tivéssemos escrito o método onCreate errado, ao em vez de «nome», tivéssemos digitado «nom» e tivéssemos subido a aplicação. Essa seria nossa versão 1 e ao perceber o erro, voltaríamos para alterar isso. Mas, arrumando o Android, ele não chama nem o onCreate, nem o onUpgrade, pois a aplicação não mudou sua versão. Como erros iguais esse são comuns é bom deixar a instrução DROP TABLE IF EXISTS, pois dessa forma a tabela velha será substituída pela arrumada.

Também vamos pedir, na linha de baixo de db.execSQL(sql), para chamar o onCreate de novo, utilizando o onCreate(db). Sempre que fizermos alguma modificação, vamos alterar a versão (no caso do número 1 para 2, 3, 4, etc.) e a tabela antiga vai ser “jogada fora” e substituída por uma versão mais atualizada. Ficaremos com o seguinte método:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
String sql = "DROP TABLE IF EXISTS Alunos";
db.execSQL(sql); onCreate(db);
}
```

Por enquanto é isso que faremos! Estamos apenas preparando caminho para que o banco de dados comece a salvar os alunos. Na tela final ficaremos com:

```
public class AlunoDAO extends SQLiteOpenHelper { public class AlunoDAO
(Context context) {
super(context, "Agenda", null, 1);
}
@Override
public void onCreate (SQLiteDatabase db) {
String sql = "CREATE TABLE Alunos (id INTEGER PRIMARY KEY, nome TEXT
NOT NULL, endereço TEXT, telefone TEXT, site TEXT, nota REAL);";
db.execSQL(sql);
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
String sql = "DROP TABLE IF EXISTS Alunos";
db.execSQL(sql); onCreate(db);
}
}
```

Já preparamos parte do caminho para um banco de dados. Vamos continuar nesse percurso para fazer com que o aluno seja de fato selecionado quando clicarmos no check mark.



Vamos jogar o aluno no banco de dados?

Para alterar o comportamento do check mark, vamos voltar na FormularioActivity.

O que queremos fazer é simplesmente dizer para o código inserir o aluno, sem termos que nos preocupar com o banco de dados. Vamos na onOptionsItemSelected e teremos o seguinte:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) { switch (item.getItemId()) {  
    case R.id.menu_formulario_ok; Aluno aluno = helper.pegaAluno();  
    Toast.makeText(FormularioActivity.this, "Aluno " + aluno.getNome() + " salvo!", Toast.LENGTH_SHORT).show();  
    finish(); break;  
}  
return super.onOptionsItemSelected(item);  
}
```

Depois do Aluno aluno = helper.pegaAluno, damos um “Enter” e na linha de baixo vamos instanciar o objeto DAO, responsável por fazer essa abstração. Vamos criar uma referência para o DAO e quando acrescentarmos o AlunoDAO dao = new AlunoDAO ele pedirá como parâmetro o contexto, aquele que temos na AlunoDAO.java, no construtor, isto é, a identificação para o Android. O contexto vai ser acrescentado entre parênteses, digitaremos this e ficaremos com: AlunoDAO dao = new AlunoDAO(this).

Vamos inserir algo que fale para o dao que queremos que os novos nomes sejam adicionados. Digitaremos dao.insere e como parâmetro escreveremos aluno e dao.insere(aluno). Só que o método ainda não existe, portanto, daremos um “Alt+Enter” e “Create method insere”. Ele criará o método na aba AlunoDAO.java onde teremos o seguinte:

```
public class AlunoDAO extends SQLiteOpenHelper { public class AlunoDAO  
(Context context) {  
    super(context, "Agenda", null, 1);  
}  
  
@Override  
public void onCreate (SQLiteDatabase db) {  
    String sql = "CREATE TABLE Alunos (id INTEGER PRIMARY KEY, nome TEXT  
NOT NULL, endereco TEXT, telefone TEXT, site TEXT, nota REAL)";  
    db.execSQL(sql);  
}  
  
@Override  
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    String sql = "DROP TABLE IF EXISTS Alunos";  
    db.execSQL(sql); onCreate(db);  
}  
  
public void insere(Aluno aluno) {  
}  
}
```

Para inserir esse aluno podemos inserir uma instrução String e utilizar um INSERT INTO e discriminar as colunas e o respectivos valores. Teremos o seguinte:

```
public void insere(Aluno aluno) {  
    String sql = "INSERT INTO Alunos (nome, endereco, telefone, site, nota)  
VALUES (" + aluno.getNome())  
}
```

O que aconteceria se alguém fosse no nosso formulário e prenchesse com o nome de Paulo seguido de "); DROP TABLE Alunos;"?



Se o java concatenar o que foi escrito no campo nome, que é "Paulo"); DROP TABLE" teremos mais ou menos o seguinte:

```
public void insere(Aluno aluno) {  
    String sql = "INSERT INTO Alunos (nome, endereco, telefone, site, nota)  
VALUES ('Paulo'); + DROP TABLE Alunos;";  
}
```

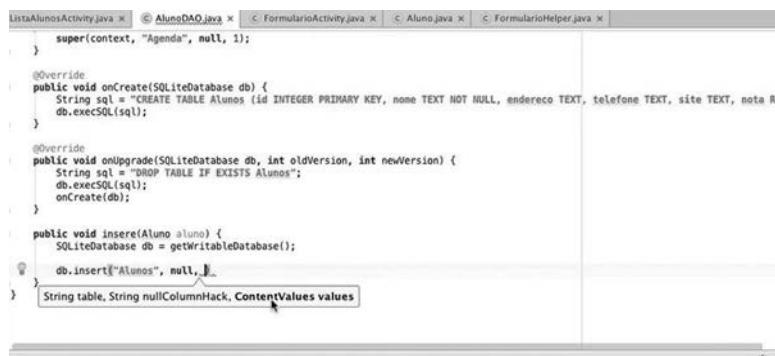
Executando a sql vai rodar a primeira instrução e depois a segunda, que é DROP TABLE Alunos e que joga fora a tabela inteira que construímos. Isto é, o banco de dados é manipulado! Essa prática é bem comum para quem gosta de destruir aplicações por aí...

Existem formas de cuidar para que isso não ocorra! Em vez de buscarmos essas ações vamos pedir para que a 'SQLiteOpenHelper' faça isso por nós, que ele faça a operação de inserção de novos alunos.

Para isso, vamos inserir no insere(Aluno aluno) um método getWritableDatabase cuja referência é o Database para que justamente possamos escrever nele. Vamos dar um

«Alt+Enter» e «Introduce local variable» e chamaremos um db. Agora, temos a referência, falta inserir um db.insert para introduzir algo, vamos colocar o primeiro parâmetro «Aluno»,

o segundo deixaremos como «null», pois não utilizaremos ele normalmente. O null é usado quando queremos uma linha em branco, mas não queremos isso aqui! Por fim, ele pede alguns valores, como um «Content Value».



```
public void insere(Aluno aluno) {
    SQLiteDatabase db = getWritableDatabase();

    ContentValues dados = new ContentValues();
    dados.put("nome", aluno.getNome());
    dados.put("endereco", aluno.getEndereco());
    dados.put("telefone", aluno.getTelefone());
    dados.put("site", aluno.getSite());
    dados.put("nota", aluno.getNota());

    db.insert("Alunos", null, dados);
}
```

Vamos criá-lo em cima do db.insert("Alunos", "null",). Vamos criar um ContentValues, chamar de dados e instancia-lo, teremos ContentValues dados = new ContentValues(). Agora, ele vai funcionar como uma espécie de mapping do java. Isto é, a ideia de chave e valor. Na chave nome, guardaremos o valor nome do aluno, na chave endereço, o endereço e assim por diante. Por isso, acrescentaremos abaixo de ContentValues o que queremos guardar, os dados, e put para colocar o que queremos colocar, o nome, aluno e getNome. Teremos, dados.put("nome", aluno.getNome()). Faremos o mesmo com o resto dos dados que queremos guardar, como o endereço, site, telefone e nota. E passamos

o ContentValues como parâmetro no insert.

Temos o seguinte:

```
public void insere(Aluno aluno) {
    SQLiteDatabase db = getWritableDatabase();

    ContentValues dados = new ContentValues();
    dados.put("nome", aluno.getNome());
    dados.put("endereco", aluno.getEndereco());
    dados.put("telefone", aluno.getTelefone());
    dados.put("site", aluno.getSite());
    dados.put("nota", aluno.getNota());

    db.insert("Alunos", null, dados);
}
```

Vamos voltar no FormularioActivity.java para ver como ficou, instanciamos o dao e pedimos para ele inserir logo na linha abaixo. O dao não precisa saber nada do banco de dados. Ele simplesmente pede para inserir o «aluno». Quando fazemos uma operação com o dao é preciso lembrar de fechar a conexão com o banco de dados. Para isso, vamos utilizar um método close. Ficamos com, dao.close(). Lembrando que conseguimos chamar o método porque ele já está na AlunoDAO na SQLiteOpenHelper. Teremos o seguinte:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) { switch (item.getItemId()) {  
  
    case R.id.menu_formulario_ok:  
        Aluno aluno = helper.pegaAluno(); AlunoDAO dao = new AlunoDAO(this);  
        dao.insere(aluno);  
        dao.close();  
        Toast.makeText(FormularioActivity.this, "Aluno " + aluno.getNome() + " salvo!", Toast.LENGTH_SHORT).show();  
  
        finish(); break;  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```

Por enquanto, ainda não conseguimos visualizar o que fizemos através do emulador.

Ficaremos com esse início de caminho traçado.

Inserimos um método que adiciona novos alunos ao banco de dados, mas os nomes presentes na lista advêm de ListaAlunosActivity.java. Lembra-se que escrevemos os nomes “a mão” na String?

Em vez de trazer os alunos do código, vamos buscá-los a partir do banco de dados!

Para auxiliar nessa tarefa adicionamos, abaixo da linha do setContentView, o dao que abstrairá o que desejamos. Instanciamos o dao digitando AlunoDAO dao = new AlunoDAO, importamos a classe com um “Alt+Enter” e como ela pede um contexto, acrescentamos um this. Ficaremos com, AlunoDAO dao = new AlunoDAO(this). Vamos acrescentar na linha de baixo, também, um dao.buscaAlunos que trará todos os alunos do banco de dados.

Como o “lista alunos” deve devolver uma coleção de alunos, completaremos o dao.buscaAlunos com List<Aluno> alunos e ficaremos com List<Aluno> alunos = dao.buscaAlunos. A classe Aluno e List aparecerá em vermelho, dessa maneira, é preciso importar com um «Alt+Enter». Ainda, é preciso fechar o dao, para isso, digitaremos dao.close. Como a Array não é mais necessária podemos apagar sua linha. Falta alterar a Adapter, substituiremos a String por Aluno e faremos essa alteração em ambos os lados. Ficaremos com o seguinte:

```
public class ListaAlunosActivity extends AppCompatActivity { @Override  
    protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-  
vedInstanceState); setContentView(R.layout.activity_lista_alunos); AlunoDAO dao  
= new AlunoDAO(this);  
    List<Aluno> alunos = dao.buscaAlunos(); dao.close();  
    ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos); ArrayAdapter<Aluno>  
    adapter = new ArrayAdapter<Aluno>(this, android.R.layout.  
        simple_list_item_1, alunos);  
    listaAlunos.setAdapter(adapter);  
  
    Button novoAluno = (Button) findViewById(R.id.novo_aluno); novoAluno.set  
    OnClickListener((v) -> {  
        Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this, For  
        mularioActivity.class);  
        startActivityForResult(intentVaiProFormulario);  
    });  
}
```

Falta implementar um método de busca de alunos, para tanto, na linha da List, acima do buscaAlunos, damos um “Alt+Enter”. Será criado um método abaixo do SQLiteDatabase. Como no buscaAlunos queremos realizar uma busca, adicionaremos na linha abaixo do public class List<Aluno> buscaAluno uma String: String sql = “SELECT * FROM Aluno;”. Nessa String indicamos as colunas que devemos receber e oSelect e o From indicam qual é a tabela. Na linha de baixo inserimos um banco de dados e como é preciso realizar a leitura adicionamos o getReadableDatabase e SQLiteDatabase db = getReadableDatabase. Poderíamos inserir um execSQL, mas ele não devolveria nada para manipular.

Resolvemos esse problema inserindo a rawQuery, uma instrução sql, parâmetros e passando o null. Ficaremos com db.rawQuery(sql, null). O rawQuery devolve um resultado que a primeira vista pode parecer meio estranho. Esse resultado, entretanto, é um Cursor que importaremos («Alt+Enter»). Ficaremos com:

```
public void insere(Aluno aluno) { SQLiteDatabase db = getWritableDatabase();
    ContentValues dados = new ContentValues();
    dados.put("nome", aluno.getNome());
    dados.put("endereco", aluno.getEndereco());
    dados.put("telefone", aluno.getTelefone());
    dados.put("site", aluno.getSite());
    dados.put("nota", aluno.getNota());

    db.insert("Alunos", null, dados);
}

public List<Aluno> buscaAlunos() { String sql = "SELECT * FROM Alunos";
    SQLiteDataBase db = getReadableDatabase(); Cursor c = db.rawQuery(sql,
null);

    return null;
}
```

O Cursor é uma espécie de ponteiro que mostra os resultados da busca. A princípio o ponteiro aponta para uma linha vazia, dessa maneira é preciso que ele se move para a próxima linha. O c.moveToNext faz isso e acrescentamos ele na linha de baixo do Cursor. O moveToNext, além de mover o Cursor para a próxima linha também nos avisa se as linhas acabam, isto é, ele nos devolve um boolean. E, se desejamos percorrer todos os resultados da busca, podemos inserir o moveToNext em uma while.

Falta extraímos os resultados dessa linha, portanto, ao final do moveToNext damos um «Enter» e usamos o método c.getString. Esse método pede o índice da coluna que queremos recuperar, o que podemos simplesmente perguntar ao Cursor e fazemos isso digitando String nome = c.getString(c.getColumnIndex("nome")). Poderíamos repetir isso várias vezes adaptando apenas o campo. Mas, não estamos querendo gerar uma lista de alunos?

Vamos instanciar uma Lista que chamaremos “alunos”, ela deve ficar acima do moveToNext. Escolheremos uma implementação, no caso, a List<Aluno> alunos

= new ArrayList<Alunos>. Como cada linha do resultado representa um novo aluno, acrescentamos na linha de baixo do moveToText um Aluno aluno = new aluno e ao em vez de colocar cada um em uma String diferente, adicionamos apenas uma setNome para “aluno”. Teremos, aluno.setNome(c.getString(c.getColumnIndex("nome"))); e ao todo o seguinte:

```
public List<Aluno> buscaAlunos() {  
    String sql = "SELECT * FROM Alunos";  
    SQLiteDataBase db = getReadableDatabase();  
    Cursor c = db.rawQuery(sql, null);  
  
    List<Aluno> alunos = new ArrayList<Aluno>();  
    while (c.moveToNext()) {  
        Aluno aluno = new Aluno();  
        aluno.setNome(c.getString(c.getColumnIndex("nome")));  
    }  
  
    return null;  
}
```

Introduzindo o setNome falamos que o nome do aluno deverá coincidir com a coluna «nome». Devemos fazer o mesmo com o restante dos campos que queremos resgatar e, para isso, basta selecionarmos a linha que acabamos de escrever e copiá-la e colá-la cinco vezes. Alterando apenas o campo «nome» e o Id por endereço, telefone, site e nota.

Atenção!!!: O campo Id não é uma String e sim um Long, então, digitaremos, aluno.setId(c.getLong(c.getColumnIndex("id"))); E atente-se também ao campo nota que é uma Double.

Falta adicionar um add para que de fato o aluno seja acrescentado em nossa lista. Portanto, ao final de tudo digitamos alunos.add(aluno). E como queremos retornar um aluno, digitaremos return aluno. Para finalizar, o Android precisa saber quando terminamos a busca para liberar esse recurso do Cursor. Para dar o aviso evocamos o método close. Acrescentaremos, c.close(). Ficaremos com:

```
public List<Aluno> buscaAlunos() { String sql = "SELECT * FROM Alunos";  
SQLiteDataBase db = getReadableDatabase(); Cursor c = db.rawQuery(sql,  
null);  
  
List<Aluno> alunos = new ArrayList<Aluno>(); while (c.moveToNext()) {  
Aluno aluno = new Aluno(); aluno.setId(c.getLong(c.getColumnIndex("id")));  
aluno.setNome(c.getString(c.getColumnIndex("nome")));  
aluno.setEndereco(c.getString(c.getColumnIndex("endereco")));  
aluno.setTelefone(c.getString(c.getColumnIndex("telefone")));  
aluno.setSite(c.getString(c.getColumnIndex("site")));  
aluno.setNota(c.getDouble(c.getColumnIndex("nota")));  
  
alunos.add(aluno);  
  
}  
c.close();  
  
return alunos;  
}
```

Voltando na Activity, repare como ela está. Antes a ArrayAdapter tinha uma String e agora tem Alunos:

```
public class ListaAlunosActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-  
vedInstanceState); setContentView(R.layout.activity_lista_  
alunos);
```

```

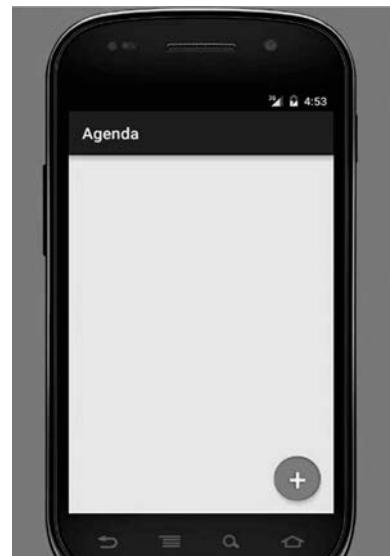
AlunoDAO dao = new AlunoDAO(this);
List<Aluno> alunos = dao.buscaAlunos(); dao.close();

ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos); ArrayAdapter<Aluno> adapter = new ArrayAdapter<Aluno>(this, android.R.layout.simple_list_item_1, alunos);
listaAlunos.setAdapter(adapter);

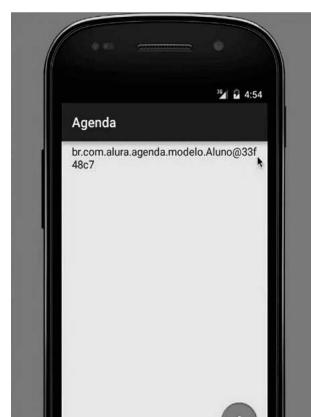
Button novoAluno = (Button) findViewById(R.id.novo_aluno); novoAluno.setOnClickListener((v) -> {
    Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this, FormularioActivity.class); startActivity(intentVaiProFormulario); });
}
}

```

Vamos salvar a aplicação e rodá-la:



O banco de dados está vazio, mas acrescentando um novo aluno ele não será adicionado à lista. O que ele mostra é uma espécie de código na tela, uma referência. O Java não sabe como mostrar o object, por isso ele traz a referência:

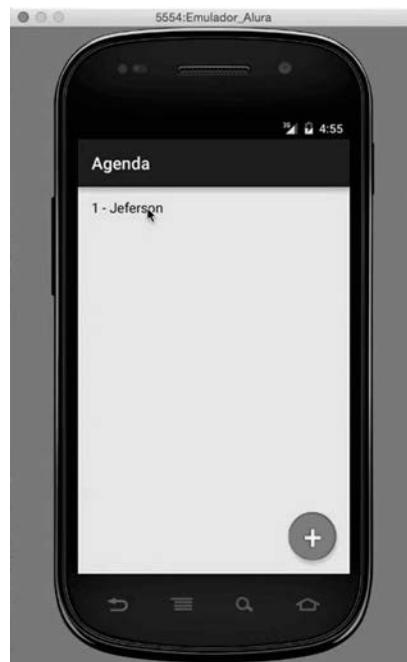


Como não existe um `toString` no `Aluno.java` ele usa a implementação padrão do `Object` e temos o que aparece na tela do celular. Para que algo seja mostrado é preciso sobrescrever o `toString` no `Aluno.java`. Ao final da tela, depois de `setNome`, digitamos `toString` e damos um «Enter» e isso será sobreescrito. Apagamos o `super.toString()` e falamos o que deve ser devolvido, no caso, o nome do aluno através do `getNome`. Teremos o seguinte, `return getId() + " - " + getNome();`.

Ficaremos com:

```
@Override  
public String toString() {  
    return getId() + " - " + getNome();  
}
```

Vamos rodar de novo e ver o que acontece?

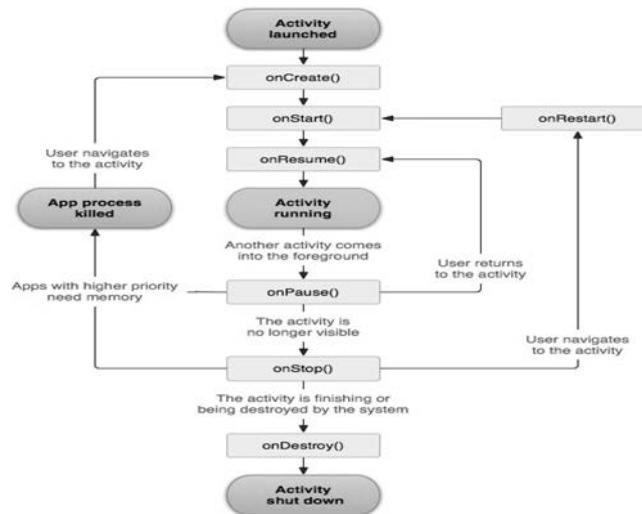


Agora, toda vez que sairmos da aplicação nosso aluno continuará armazenado! Entretanto... nossa lista não é atualizada automaticamente! Assim, inserindo um segundo aluno, ele não aparece na lista, apenas depois de sairmos e voltarmos a aplicação. Resolveremos esse problema na sequência!

Lembrando que conseguimos fazer com que um novo aluno aparecesse em nossa lista, mas ele só é adicionado quando saímos da aplicação. Vamos mudar isso!

Para fazer essa mudança temos que entender um pouco sobre o ciclo de vida das activities. Vamos observar o seguinte desenho disponível no site do Android(d. android.com). No link fazemos uma busca por activity e na página dos resultados clicamos em Reference, no menu do canto esquerdo.

Selecionamos o Activity| Android Developers que aparece logo no topo e vai abrir uma página na qual vemos o ciclo de vida das activities :



Para entender o Ciclo de vida das Activities

A activity, como vemos na imagem, evoca o método `onCreate`, que chama o `onStart` e o `onResume`, cujas funções são, respectivamente, iniciar a activity e continuá-la. Só depois de fazer isso ela começa a ser executada.

Dependendo do celular, se algum outro método entra na frente da Activity, ele chama o método `onPause`. Isso ocorre no caso dos "pop ups" que não tapam completamente a activity, ou seja, a activity é pausada temporariamente e apenas depois do «pop up» ser fechado é chamado o `onResume` para retornar a atividade.

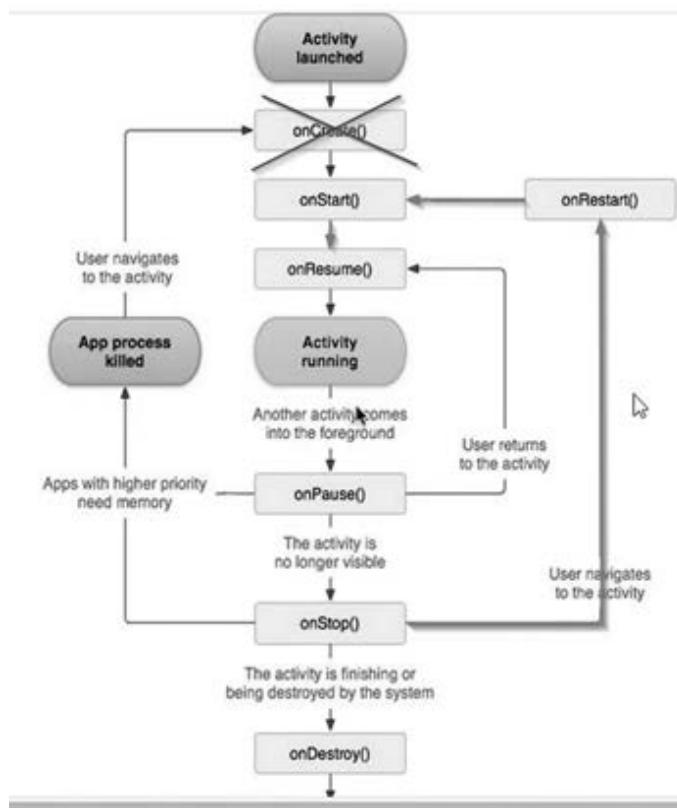
Outro caso é o das chamadas telefônicas, a aplicação nos avisa o recebimento da chamada eletrônica tomando a tela inteira. Nesse caso o Android chama o `onPause` e como a aplicação deixa de estar visível ele também chama o `onStop`, parando a activity. Esse processo pode matar nossa aplicação, isto é, seguindo o caminho de "App process killed". O que acontece é que o processo pode reiniciar sem salvar nada. Isso ocorre quando o Android precisa da memória, caso contrário ele retorna a activity através do `onRestart` e reiniciamos a aplicação a partir do `onStart`.

Agora que compreendemos um pouco a respeito da vida da activity vamos ver como isso se aplica ao que estamos fazendo!

Quando estamos no celular e aparece a tela da Lista de Alunos, já foram executados o onCreate, o onStart e o onResume. Quando passamos para o formulário estamos em uma activity que toma a tela inteira, assim, o Android chama o onStop. Quando apertarmos o botão de check mark a activity vai seguir o caminho da Activity shut down, como demonstra a imagem. Mas, quando acaba a activity do formulário voltamos para a lista de alunos que estará no onStop que seguirá o caminho onRestart e seguir no onStart normalmente.

Vamos voltar um pouquinho no Android Studio e verificar onde é feito o carregamento da lista, na ListaAlunosActivity.java.

O carregamento é feito no método onCreate e aí temos o problema! Pois o ciclo da activity depois do onStop dirige-se para o onRestart ou onResume e não passa pelo onCreate. É por isso que não conseguimos mostrar o aluno que acrescentamos na lista



Para resolver isso podemos acrescentar um onResume, pois todos os caminhos passam por ele. Se colocássemos um método onStart teríamos o problemas de que, caso houvesse uma pausa, não voltaríamos para o onStart e a lista não carregaria.

Para fazer isso, vamos na ListaAlunosActivity.java e escrevemos abaixo do onCreate um onResume, damos um "Enter". Teremos:

```

@Override
protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-
vedInstanceState); setContentView(R.layout.activity_lista_alunos);

AlunoDAO dao = new AlunoDAO(this);
List<Aluno> alunos = dao.buscaAlunos(); dao.close();

ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos); ArrayAdapter<Aluno> adapter = new ArrayAdapter<Aluno>(this,
        android.R.layout.simple_list_item_1, alunos); listaAlunos.setAdapter(adapter);

Button novoAluno = (Button) findViewById(R.id.novo_aluno); novoAluno.set
OnClickListener((v) -> {
    Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this, For-
mularioActivity.class); startActivity(intentVaiProFormulario);
});

}

@Override
protected void onResume() { super.onResume();
}

```

Lembre-se de chamar o super para manter o comportamento da AppCompatActivity!

O código que faz o carregamento vai do AlunoDAO até o fim do ListView. Para facilitar, vamos extrair o código que faz o carregamento da lista em um método. Selecioneimos com o mouse o seguinte:

```

AlunoDAO dao = new AlunoDAO(this);
List<Aluno> alunos = dao.buscaAlunos(); dao.close();

ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos); ArrayAdapter<Aluno> adapter = new ArrayAdapter<Aluno>(this,
        android.R.layout.simple_list_item_1, alunos);
listaAlunos.setAdapter(adapter);

```

E com o botão direito vamos em “Refactor> Extract> Method”. Clicamos no Method e uma janela é aberta pedindo para dar um nome ao método. Chamaremos de «carregalista» e ele será private. É só dar um “ok” e teremos o seguinte:

```
@Override  
protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-  
vedInstanceState); setContentView(R.layout.activity_lista_alunos);  
    carregaLista();  
    Button novoAluno = (Button) findViewById(R.id.novo_aluno); novoAluno.se-  
tOnClickListener((v) -> {  
        Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this, For-  
mularioActivity.class); startActivity(intentVaiProFormulario);  
    });  
}  
  
private void carregaLista() {  
    AlunoDAO dao = new AlunoDAO(this);  
    List<Aluno> alunos = dao.buscaAlunos(); dao.close();  
    ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos); ArrayAdapter<Aluno>  
    adapter = new ArrayAdapter<Aluno>(this,  
        android.R.layout.simple_list_item_1, alunos); listaAlunos.setAdapter(adapter);  
}  
  
@Override  
protected void onResume() { super.onResume();  
}
```

Mas, repare que o carregaLista está no Oncreate. Não queremos carregá-lo a partir desse método, senão o mesmo erro de antes ocorrerá. Vamos selecionar o carregaLista(); e dar um «Comand +X» ou «Ctrl+X» e no onResume, depois do super, damos um “Command+V” ou “Ctrl+V” e teremos:

```
@Override  
protected void onResume() { super.onResume(); carregaLista();  
}
```

Agora sim, carregamos a lista a partir do onResume. Vamos ver se deu certo? Vamos rodar o emulador e quando adicionarmos mais uma pessoa, salvando esse formulário, teremos o novo nome na lista!



Nossa aplicação já consegue armazenar os alunos. Suponhamos que precisamos deletar algum aluno. Como faríamos para apagar esse nome da lista?

O que faremos é criar um menu de contexto para cada nome, isto é, ao dar um clicklongo sobre um nome, abre-se um menu de contexto. Nele podemos escolher dentre as opções e uma delas será “Deletar” aluno.

Vamos voltar na aba ListaAlunosActivity.java e dar uma olhada em como está a tela.

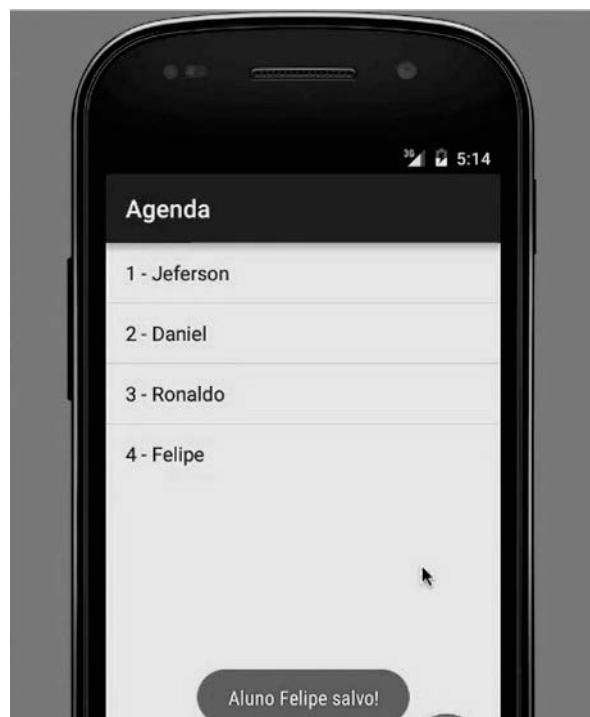
Agora, precisamos dizer qual componente vai possuir um menu de contexto. Vamos digitar abaixo de onCreate , um registerForContextMenu e acrescentarmos entre os parênteses listaAlunos. A listaAlunos ficará em vermelho, pois ela está sem referência, pois o findViewById da lista de alunos está no carregaLista. Para solucionar isso copiaremos a linha ListView listaAlunos = (ListView) findViewById (R.id.lista_alunos) através do «Command+X» e daremos um «Command+V» acima da linha Button novoAluno.

Para que a listaAlunos que acabamos de mover continue funcionando em ambos os lados, temos que transformá-la em atributo. Fazemos um «Alt+Enter» em cima dele e selecionamos «Create field». Agora, o listaAlunos virou um atributo que podemos acessar tanto no onCreate quanto no carregaLista. Ficaremos com:

```
@Override  
protected void onResume() { super.onResume(); carregaLista();  
}  
  
@Override
```

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {  
    menu.add("Deletar");  
}
```

Vamos rodar no emulador para ver o que acontece!



Agora, vamos introduzir um comportamento no “Deletar” para que ele avise que o aluno foi deletado.

Para fazer isso, podemos seguir o caminho que fizemos na FormularioActivity.java, por exemplo, colocar um comportamento similar ao onOptionsItemSelected, onContextItemSelected. Mas, ao fazer isso teremos que acrescentar um switch para filtrar, isto é, indicar o caminho a ser feito. Porém, inserir um Id é um problema, pois o item feito a mão não tem id, ele só possui um quando é criado na .xml.

Vamos seguir por outro caminho, vamos utilizar a ideia do Listener, assim, quando o botão é clicado o Android nos avisará.

Seguimos no onCreateContextMenu. Se queremos manipular um item do menu vamos ter que adicionar uma referência desse item. Vamos colocar essa referência na frente do menu.add. Ficaremos com MenuItem deletar = menu.add("Deletar"). Agora, acrescentaremos na linha de baixo o deletar.setOnMenuItemClickListener que significa que estamos interessados em escutar o evento de click no item do menu. Ele vai pedir, ainda, um «OnMenuItemClickListener» e poderemos preencher com new OnMenu. Dando um "Enter", o Android cria uma classe anônima, dentro da qual podemos fazer o que quisermos.

Estamos com o seguinte:

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v, Context-  
Menu.  
    ContextMenuInfo menuInfo) {  
    MenuItem deletar = menu.add("Deletar"); deletar.setOnMenuItemClickListener(  
        new MenuItem.OnMenuItemClickListener() {  
            @Override  
            public boolean onMenuItemClick(MenuItem item) {  
  
                return false;  
            }  
        });  
}
```

Lembrando que o MenuItem refere-se ao «Deletar» e não ao aluno número 5.

Vamos observar o onCreateContextMenu. Ele traz um ContextMenuInfo que contém a informação que necessitamos. É ele quem vai dizer qual item da lista foi clicado para gerar um ContextMenu. Para isso, precisaremos informar que o ContextMenuInfo é um adapter.

Na linha de baixo vamos digitar um adapter que chamaremos de info. Teremos AdapterView.AdapterContextMenuInfo info = menuInfo. Vamos falar para o Android que queremos nessa referência algo mais específico. Vamos fazer um Cast. Para isso basta dar um "Alt+Enter" em cima do menuInfo. Caso não ocorra essa sugestão podemos fazer "a mão" completando o Adapter. Teremos, AdapterView.AdapterContextMenuInfo info

= (AdapterView.AdapterContextMenuInfo) menuInfo. Como não conseguimos declarar o parâmetro da classe anônima, vamos apenas dizer no onCreate que é um final.

Dentro da info teremos um position, que nos informa qual posição da lista acabou de ser clicada. Basta informar ao Android que queremos que ele devolva o aluno que está nessa posição. Para fazer isso acrescentamos na linha de baixo do Adapter um método chamado getItemAtPosition que pedirá a posição a ser recuperada. Pediremos ao info que devolva um Aluno e, por fim, faremos um Cast nele através do «Alt+Enter» e ficaremos com Aluno aluno = (Aluno) listaAlunos.getItemAtPosition(info.position).

Para mostrar um comportamento após escolher “Deletar”, vamos fazer um Toast.makeText. Importamos o Toast e passamos o contexto, ListaAlunosActivity.this. Adicionamos a mensagem, “Deletar o aluno” + aluno.getNome, e acrescentamos sua duração, Toast.LENGTH_SHORT. Ficaremos com:

```
Toast.makeText(ListaAlunosActivity.this, "Deletar o aluno" + aluno.getNome(),  
Toast.  
LENGTH_SHORT)
```

No final evocamos um método show, digitando um .e show. Ficaremos com:

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v, final ContextMenu.  
onCreateContextMenuListener){  
    ContextMenuItem menuInfo = menu.add("Deletar");  
    menuInfo.setOnMenuItemClickListener(new MenuItem.OnMenuItemClickListener(){  
        @Override  
        public boolean onMenuItemClick(MenuItem item) {  
            AdapterView.AdapterView.  
            AdapterContextMenuInfo info = (AdapterView.  
            AdapterContextMenuInfo) menuInfo;  
            Aluno aluno = (Aluno) listaAlunos.getItemAtPosition(info.position);  
            Toast.makeText(ListaAlunosActivity.this, "Deletar o aluno" + aluno.  
            getNome(), Toast.LENGTH_SHORT).show();  
            return false;  
        }  
    });  
}
```

Vamos rodar no emulador para ver como ficou?



Ainda falta deletar o aluno. Para isso usaremos o dao. Vamos instanciar o AlunoDAO na linha de baixo do getItemAtPosition. Escreveremos, AlunoDAO dao = new AlunoDAO(ListaAlunosActivity.this). Na linha seguinte falaremos que o aluno selecionado deve ser deletado, digitaremos: dao.deleta(aluno).

Não esquecendo que se trata de um banco de dados e que é importante fechá-lo.

Utilizaremos o dao.close.

Podemos apagar o Toast porque veremos a operação acontecendo, então, não precisaremos mais de uma mensagem nos avisando que algo vai acontecer. Ficaremos com:

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v, final ContextMenu.  
ContextMenuItemInfo menuInfo) {  
    MenuItem deletar = menu.add("Deletar"); deletar.setOnMenuItemClickListener(  
newMenuItem.OnMenuItemClickListener()  
{  
    @Override  
    public boolean onMenuItemClick(MenuItem item) { AdapterView.Adapter-  
    ContextMenuItemInfo info = (AdapterView.  
    AdapterContextMenuInfo) menuInfo;  
    Aluno aluno = (Aluno) listaAlunos.getItemAtPosition(info.position);  
    AlunoDao dao = new AlunoDao(ListaAlunosActivity.this);  
    dao.deleta(aluno); dao.close();  
    return true;  
    }  
});
```

```
    return false;  
}  
});  
}
```

Falta implementar o “Deleta”, então, damos um “Alt+Enter” e “Create method”. Iremos para a aba AlunoDAO e o novo método aparecerá depois do while. Poderíamos fazer a operação a mão, mas vamos pedir para que o SQLiteOpenHelper faça isso por nós. Vamos pegar uma referência do banco de dados e digitá-la na linha de baixo do public void deleta(Aluno aluno). Vamos escrever o seguinte: SQLiteDatabase db = getWritableDatabase. O getWritableDatabase serve para manipular o banco de dados.

Embaixo disso, chamamos o método db.delete que pedirá como parâmetros, a tabela e a cláusula where. Preenchemos também com todos os alunos que desejamos remover, cujo id seja getId. Teremos db.delete(“Aluno”, “id = “ + aluno.getId()).

Observem que de novo estamos concatenando. Mas, sempre que quisermos concatenar algum valor no parâmetro, ao em vez de fazer isso, deixaremos um espaço reservado. Vamos marcar o espaço com uma interrogação e informaremos na frente da interrogação o parâmetro, digitaremos params. O params pede uma Array de String, então, adicionaremos na linha anterior o que o params pede, que são os valores. Acrescentaremos o Id do aluno e lembrando que como o Id é uma Long , ele pede conversão para String. TeremosString[] params = {String.valueOf(aluno.getId())}. No todo teremos:

```
//...  
public void deleta(Aluno aluno) { SQLiteDatabase db = getWritableDatabase();  
    String [] params = {String.valueOf(aluno.getId())}; db.delete("Alunos", "id = ?",  
    params);  
}
```

Rodando o emulador, vemos que o aluno só é deletado quando saímos da aplicação. Vamos tentar entender!

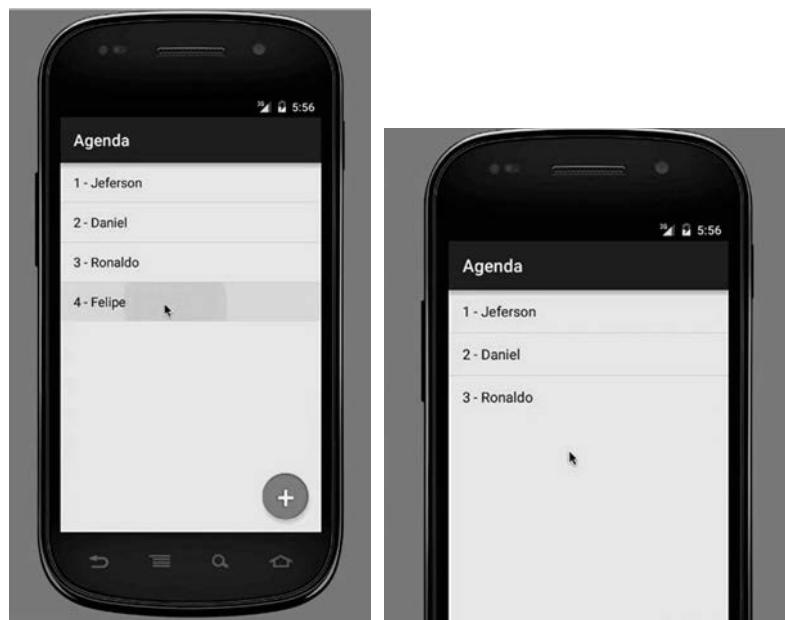
Quando selecionamos a opção de “Deletar”, observe que a activity não sai do primeiro plano, ela fica aparente, no plano de fundo do botão «Deletar». A activity continua a rodar e por isso a ação não passa pelo onResume do AlunoDAO.java. Assim, a lista não é carregada. Como não saímos da Activity, não podemos depender apenas do ciclo de vida da activity, teremos que chamar o método carregaLista.

Bom, vamos voltar na ListaAlunosActivity.java, no onCreateContextMenu. Na linha seguinte do dao.close vamos carregar a lista de alunos, para isso, introduzimos um carregaLista. Isso serve para atualizar nossa lista. Ficaremos com:

```
public class ListaAlunosActivity extends AppCompatActivity { private ListView listaAlunos;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) { super.onCreate(-  
        savedInstanceState); setContentView(R.layout.activity_lista_alunos); listaAlunos =  
        (ListView) findViewById(R.id.lista_alunos);  
        Button novoAluno = (Button) findViewById(R.id.novo_aluno); novoAluno.set-  
        tOnClickListener((v) -> {  
            Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this, Formu-  
            larioActivity.class); startActivity(intentVaiProFormulario);  
        });  
        registerForContextMenu(listaAlunos);  
  
    }  
    private void carregaLista() {  
        AlunoDAO dao = new AlunoDAO(this); List<Aluno> alunos = dao.buscaAlu-  
        nos(); dao.close();  
        ArrayAdapter<Aluno> adapter= new ArrayAdapter<Aluno>(this, an-  
        droid.R.layout.simple_list_item_1, alunos);  
        listaAlunos.setAdapter(adapter);  
  
    }  
    @Override  
    protected void onResume() { super.onResume();  
  
        carregaLista();  
    }  
  
    @Override  
    public void onCreateContextMenu(ContextMenu menu, View v, final Con-  
    textMenu.ContextMenuItemInfo menuInfo) {  
        MenuItem deletar = menu.add("Deletar");  
        deletar.setOnMenuItemClickListener(new MenuItem.OnMenuItemCli-  
        ckListener())  
        {  
            @Override
```

```
public boolean onMenuItemClick(MenuItem item) { AdapterView.Adapter-  
ContextMenuItemInfo info = (AdapterView.  
AdapterContextMenuInfo) menuInfo;  
Aluno aluno = (Aluno) listaAlunos.getItemAtPosition(info.position);  
  
AlunoDAO dao = new AlunoDAO(ListaAlunosActivity.this); dao.delete(alu-  
no);  
dao.close();  
  
carregaLista(); return false;  
}  
});  
}  
}
```

Vamos rodar o emulador e verificar se as alterações funcionaram! Vamos selecionar na tela o aluno Felipe para que ele seja deletado:



Pronto! Conseguimos deletar os alunos!

Já conseguimos inserir e remover alunos, agora, vamos adicionar um meio para editá-los. Assim, caso acrescentemos uma informação errada podemos modificá-la quando quisermos.

Poderíamos inserir um novo MenuContext para abrir a opção de «Editar» junto a de

«Deletar». Mas, vamos aprender algo novo! Vamos iniciar o caminho clicando no nome do aluno e logo em seguida sendo direcionados para o formulário que editaremos.

Para aplicar um comportamento no clique sobre o nome do aluno, voltamos na ListaAlunosActivity.java. Vamos na onCreate onde acabamos de recuperar a lista. Poderíamos inserir um setOnClickListener, como já fizemos anteriormente, mas ele devolveria uma View, que nesse caso é a lista inteira. Como queremos os subcomponentes da lista, isto é, apenas os nomes dos alunos que clicarmos, não usaremos o setOnClickListener. Usaremos um Listener diferente, o setOnItemClickListener, que manifesta que estamos interessados apenas no item clicado e não na lista inteira. Vamos completar com new OnItemClick e ficaremos com listaAlunos.setOnItemClickListener(new OnItemClick). Agora, é só dar um “Enter” e criar a classe, ela será anônima. Teremos o seguinte:

```
@Override  
protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-  
vedInstanceState); setContentView(R.layout.activity_lista_alunos);  
  
listaAlunos = (ListView) findViewById(R.id.lista_alunos);  
  
listaAlunos.setOnItemClickListener(new AdapterView.OnItemClickListener{  
    @Override  
    public void onItemClick(AdapterView<?> parent, View view, int position,  
    long id) {  
    }  
});  
}
```

Vamos implementar o método onItemClick. Aproveitando para explicar algumas coisas:

o parent é justamente a lista na qual clicamos, vamos alterar o nome para `lista; a View é o item que clicamos, então renomeamos para View item;

o int position é a posição do item

o long id é o id do item

Para pegar o aluno basta usar o método que já tínhamos utilizado no onOptionsItemSelected, o getItemAtPosition. Informamos para esse método uma posição e ele retorna o aluno.

Vamos acrescentar no setOnItemClickListener, na linha de baixo do onItemClick, um Aluno aluno = listaAlunos.getItemAtPosition(position) e fazemos um Cast. Para imprimir o nome do aluno acrescentamos um Toast seguido do contexto e do nome do aluno clicado, a duração e o show para mostrar. Teremos:

```
Toast.makeText(ListaAlunosActivity.this, "Aluno " + aluno.getNome() + " clicado!", Toast.LENGTH_SHORT).show()
```

Ficaremos com:

```
@Override  
public void onItemClick(AdapterView<?> parent, View item, int position,  
long id {  
    Aluno aluno = (Aluno) listaAlunos.getItemAtPosition(position);  
    Toast.makeText(ListaAlunosActivity.this, "Aluno " + aluno.getNome() + " clicado!", Toast.LENGTH_SHORT).show();  
}
```

Vamos rodar a aplicação e verificar o que temos!



Agora o clique no aluno possui
uma interação!

Observação: Podemos usar outro tipo de Listener, o setOnItemLongClick, que está relacionado ao clique longo na lista. Se quiser criar um setOnItemLongClick basta se guiar nos parâmetros que já fizemos. As únicas pontos que mudarão é o public boolean e return false. Ao rodar a aplicação veremos que ele mostrará o Toast do clique longo e o MenuContext.

Conseguimos estabelecer uma interação quando o aluno é clicado. O que queremos fazer depois de clicarmos no nome do aluno é que a aplicação nos leve de volta ao formulário para que possamos editá-lo. Para fazer isso vamos acessar o comportamento da classe, no *ListaAlunosActivity.java*.

Lembrando que inserimos um setOnClickListener e imprimimos através do Toast uma mensagem de que o aluno foi clicado. Vamos apagar a linha do Toast, pois ela não será mais necessária:

```
Toast.makeText(ListaAlunosActivity.this, "Aluno " + aluno.getNome() + " clicado!", Toast.LENGTH_SHORT).show()
```

O que queremos é, ao clicar no aluno, ir para a tela do formulário. Para isso usaremos uma intent na qual declararemos ao Android que queremos mudar de tela. Escreveremos a intent na última linha da classe setOnItemClickListener. Lembrando que a intent pede duas informações: um contexto, que será ListaAlunosActivity.this, e a classe do formulário, FormularioActivity.class. Ficaremos com, Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this, FormularioActivity.class). Para abrir a tela completaremos com um startActivity. Teremos startActivity(intentVaiProFormulario). Nossa código estará da seguinte maneira:

```
@Override  
protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-  
vedInstanceState); setContentView(R.layout.activity_lista_alunos);  
  
listaAlunos = (ListView) findViewById(R.id.lista_alunos);  
  
listaAlunos.setOnItemClickListener(new AdapterView.OnItemClickListener{  
    @Override  
    public void onItemClick(AdapterView<?> lista, View item, int position, long  
    id) {  
        Aluno aluno = (Aluno) listaAlunos.getItemAtPosition(position);  
        Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this, Formu-  
        larioActivity.class); startActivity(intentVaiProFormulario);  
    }  
});...//}
```

Vamos ver o que acontece no emulador?



O formulário abriu, mas falta recuperar os dados do aluno para que apareçam no formulário.

Vamos voltar na *ListaAlunosActivity.java* e como já recuperamos o aluno, Aluno aluno, em um momento anterior, vamos transmiti-lo para o *FormularioActivity.java*.

A intent é usada para demonstrar intenção. Ao criarmos a intent, o *Android* interpreta que a *ListaAlunosActivity* pede para mudar de lugar, isto é, que deseja ser transposta para o *FormularioActivity*. Feita essa leitura o *Android* fecha a intent e instancia o *FormularioActivity**.

Ao colocar algo na intent, como ela vai para o *Android* e depois para o *FormularioActivity*, recuperamos o que penduramos. Portanto, vamos pendurar o aluno na intent e assim conseguiremos passar ele para o *FormularioActivity*.

Para isso, na linha de baixo de Intent intentVaiProFormulario, digitaremos intentVaiProForulario e inserimos um método, o putExtra. Dentro dos parênteses colocamos aquilo que desejamos que a nossa intent leve para o *FormularioActivity*. No caso, o aluno, e aproveitamos para adicionar uma identificação, uma espécie de etiqueta. Ficaremos com, intentVaiProFormulario.putExtra("aluno", aluno). Ao todo teremos:

```
@Override  
public void onItemClick(AdapterView<?> lista, View item, int position, long id) { Aluno aluno = (Aluno) listaAlunos.getItemAtPosition(position);  
    Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this,  
    FormularioActivity.class);  
    intentVaiProFormulario.putExtra("aluno", aluno) startActivity(intentVaiPro-  
    Formulario);  
}
```

Perceba que o aluno que inserimos continua com problemas, pois o Android não consegue reconhecê-lo. Teremos que «serializar» esse aluno, isto é, transformá-lo em binário.

Para transformar a classe em algo que pode ser serializado, vamos na aba Aluno.java. Na mesma linha da public class e logo no topo da página, acrescentamos que a classe Aluno implementa a Serializable, teremos: public class Aluno implements Serializable. Vamos importar o Serializable através de um «Alt+Enter». Agora, o Java saberá como converter as String, o Long e o Double em binários. Encontraremos nossa tela da seguinte maneira:

```
public class Aluno implements Serializable { private Long id;  
    private String nome;  
    private String endereço;  
    private String telefone;  
    private String site;  
    private Double nota;  
}
```

Vamos voltar no ListaAlunosActivity. Repare que mais nada atrapalha o aluno de estar pendurado na intent. Fizemos a parte de enviar o aluno para o FormularioActivity.java. Vamos partir para a parte do quando os dados já estão lá.

Voltamos na aba FormularioActivity.java. No momento em que estamos criando nosso formulário, na onCreate se algo estiver pendurado na intent teremos que recuperar essa intent e popular isso na OnCreate. Para fazer isso vamos recuperar a Intent, digitando Intent intent = getIntent. Vamos importar isso através do «Alt+Enter». Agora, vamos recuperar o que está pendurado na intent, no caso, o aluno. Para isso, digitaremos intent.getStringExtra e como temos vários tipos de getExtra, usaremos o Serializable. Ficaremos com, intent.getSerializableExtra.

Como ainda pede um nome, vamos usar a identificação, a etiqueta usada anteriormente, e digitaremos “aluno”. Como o getSerializableExtra nos devolve o aluno, vamos falar que Aluno aluno = intent.getSerializableExtra. Damos um «Alt+Enter» para fazer o “Cast” do aluno.

Falta trazer o conteúdo do aluno e jogar na tela do formulário. Trazemos o aluno da intent que está no ListaAlunos.Activity e temos que lembrar de verificar se esse aluno é diferente de nulo e se ele for nulo, teremos que preencher o formulário. Para fazer essa verificação acrescentamos if (aluno != null).

Quando queríamos pegar os dados do formulário, utilizamos o Helper, por isso, vamos utilizá-lo novamente. Vamos dar um “Ctrl+C” e “Ctrl+V” na linha helper

= new FormularioHelper(this); e deslocá-la para dentro das chaves, abaixo da linha do setContentView.

Abaixo da linha if (aluno != null) digitamos helper.preencheFormulario e completamos com aluno. Ficaremos com, helper.preencheFormulario(aluno). Vamos dar um “Alt+Enter” em cima do preencheFormulario e escolhemos «Create Method». O método será criado ao final da página FormularioHelper.java. Teremos na FormularioActivity.java o seguinte:

```
@Override  
protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-  
vedInstanceState); setContentView(R.layout.activity_  
formulario);  
  
helper = new FormularioHelper(this);  
  
Intent intent = getIntent();  
Aluno aluno = (Aluno) intent.getSerializableExtra("aluno"); if (aluno != null) {  
helper.preencheFormulario(aluno);  
}  
  
}
```

Vamos acessar a aba FormularioHelper.java. Falta preencher os campos, basta acrescentar na linha seguinte o campoNome.setText(aluno.getNome()) e os demais dados. Lembrando que a nota recebe um setProgress e também um intValue. Tere-mos:

```
public void preencheFormulario(Aluno aluno) {  
campoNome.setText(aluno.getNome());  
campoEndereco.setText(aluno.getEndereco());  
campoTelefone.setText(aluno.getTelefone());  
campoSite.setText(aluno.getSite());  
campoNota.setProgress(aluno.getNota().intValue());  
}
```

Perceba que, os campos já foram todos introduzidos no FormularioHelper, isto é, estão prontos para uso, o que facilita muito as coisas!

Atenção para um detalhe!

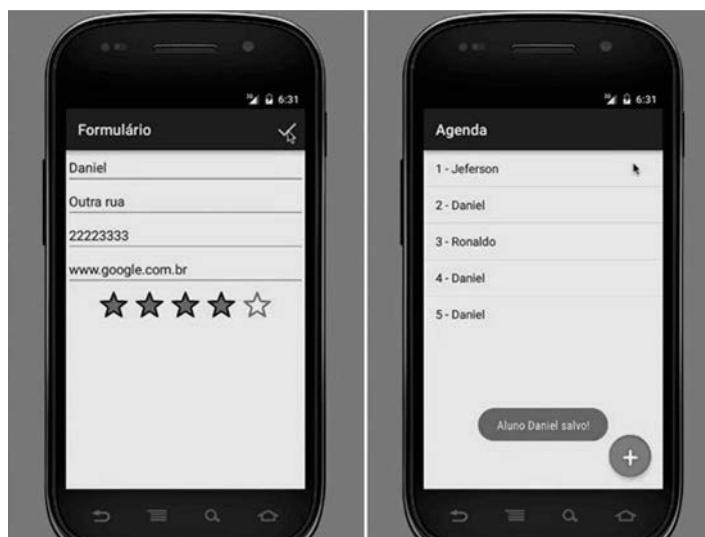
Salvamos o nome, o telefone, o site e a nota. Falta salvar o Id, caso contrário, perderemos ele e isso complicará as coisas quando precisarmos da identificação para alterar o banco. Então, salvaremos o aluno inteiro, criando um atributo inteiro. Digitaremos no topo da página do FormularioHelper.java, após o private final RatingBar campoNota, um private Aluno aluno. No final do FormularioHelper criaremos, aluno = new Aluno. Agora, podemos apagar no Aluno pegaAluno a linha Aluno aluno = new Aluno, pois ele só precisa estar instanciado uma vez. Ficaremos no FormularioHelper com:

```
public FormularioHelper(FormularioActivity activity) {  
    campoNome = (EditText) activity.findViewById(R.id.formulario_nome);  
    campoEndereco = (EditText) activity.findViewById(R.id.formulario_endere-  
    ço);  
    campoTelefone = (EditText) activity.findViewById(R.id.formulario_Telefone);  
    campoNota = (RatingBar) activity.findViewById(R.id.formulario_nota);  
    aluno = new Aluno();  
}
```

Agora, no preencheFormulario vamos guardar o aluno inteiro no atributo. Digitaremos ao final dele this.aluno =aluno. Ficaremos com:

```
public void preencheFormulario(Aluno aluno) {  
    campoNome.setText(aluno.getNome());  
    campoEndereco.setText(aluno.getEndereco());  
    campoTelefone.setText(aluno.getTelefone());  
    campoSite.setText(aluno.getSite());  
    campoNota.setProgress(aluno.getNota().intValue());  
    this.aluno = aluno;  
}
```

Como salvamos o aluno no preencheFormulario é desse local que vai advir o id. Vamos verificar se isso está funcionando no código! Vamos salvar e rodar o emulador.



Conseguimos trazer para o formulário todas as informações de “Daniel”, mas repare que se alterarmos e salvarmos o formulário estaremos apenas acrescentando mais “Daniels” na nossa lista. Esse problema resolveremos na sequência!

Conseguimos recuperar os dados de um aluno clicado no formulário. O problema é que quando alterarmos os dados desse aluno no formulário e salvarmos, a aplicação adiciona um novo aluno, em vez de simplesmente salvar as edições. Isso resultará em nomes duplicados na lista.

Vamos modificar o código para salvar as alterações ao em vez de gerar um novo aluno. Se estamos falando do checkmark, estamos falando de um clique no botão do menu. Então, vamos no `FormularioActivity.java`, no método `onOptionsItemSelected`, que fala que um item do menu foi selecionado. Encontramos o seguinte:

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_formulario_ok:
            Aluno aluno = helper.pegaAluno();
            AlunoDAO dao = new AlunoDAO(this);
            dao.insere(aluno);
            dao.close();
            Toast.makeText(FormularioActivity.this, "Aluno " + aluno.getNome() + " salvo!", Toast.LENGTH_SHORT).show();

            finish();
            break;
    }

    return super.onOptionsItemSelected(item);
}

```

Repare que dentro dele fizemos uma verificação para saber qual item do menu foi clicado. Então, falamos que caso clicássemos no botão menu_formulario_ok, teríamos um comportamento que pega o aluno do formulário e com esse aluno instanciávamos o dao e chamávamos o insere. Agora, não é sempre que vamos querer chamar o insere, pois, podemos estar simplesmente fazendo uma alteração e não querendo inserir um novo aluno.

Para resolver esse problema podemos fazer outro tipo de verificação. Se o aluno tiver um id e se não for nulo, significa que estamos simplesmente alterando esse aluno. Acrecentaremos duas linhas abaixo de Aluno aluno = helper.pegaAluno() uma verificação if (aluno.getId() != null). E com isso dizemos que se é diferente de nulo, deve ser alterado, assim, digitamos dao.altera(aluno). Importante que o Aluno dao = new AlunoDAO(this) fique acima da verificação.

Caso o id do aluno seja nulo, acrescentamos else e embaixo dele copiamos e colamos o dao.insere(aluno). O que acabamos de modificar ficará na linha de baixo do dao.altera. Teremos:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_formulario_ok;  
            Aluno aluno = helper.pegaAluno();  
            AlunoDAO dao = new AlunoDAO(this);  
            if (aluno.getId() != null) {  
                dao.altera(aluno);  
            } else {  
                dao.insere(aluno);  
            }  
            dao.close();  
            Toast.makeText(FormularioActivity.this, "Aluno " + aluno.getNome() + " salvo!", Toast.LENGTH_SHORT).show();  
            finish(); break;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

Só falta implementar o método altera. Vamos dar um “Alt+Enter” em cima dele e nosso método será criado no AlunoDAO.java, no final da página. Ficaremos com:

```
public void altera(Aluno aluno) {  
}
```

Vamos inserir o comportamento desse método. Primeiro, precisamos de um banco de dados que possa ser alterado, assim, acrescentamos o SQLiteDatabase db = getWritableDatabase(). Uma vez que temos o WritableDatabase, precisamos de um método que faça a atualização, mas fazer isso a mão seria extremamente complicado. Portanto, usaremos o db.update e para completá-lo usamos a tabela, «Alunos».

```
public void altera(Aluno aluno) { SQLiteDatabase db = getWritableDatabase();  
    db.update("Alunos");  
}
```

Para seguir completando o db.update vamos inserir os dados que usaremos, primeiro, o ContentValues. Como teremos que criar o ContentValues, na linha de cima digitaremos ContentValues dados = new ContentValues. Na linha de baixo disso podemos acrescentar o dados.put("nome", aluno.getNome()), mas lembra que já fizemos isso antes? Se você rolar sua barrinha para cima vai ver que já temos tudo isso feito no insere(Aluno aluno). O que faremos é extrair o método.

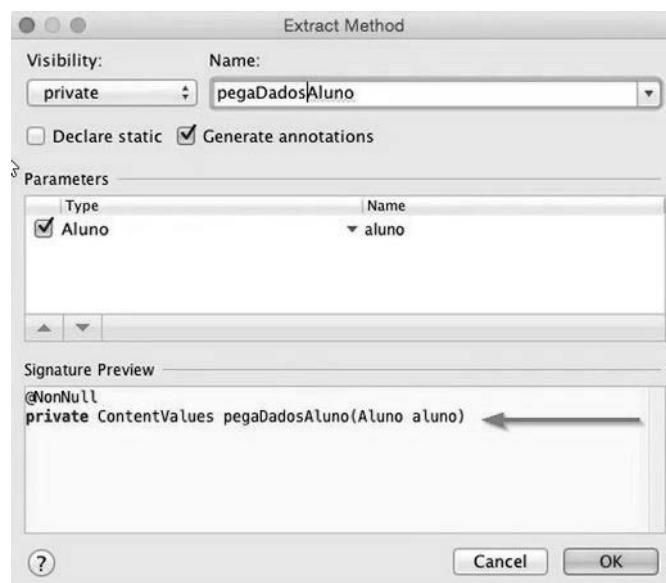
Lembrando que temos o seguinte no insere(Aluno aluno):

```
public void insere(Aluno aluno) {  
    SQLiteDatabase db = getWritableDatabase();  
    ContentValues dados = new ContentValues();  
    dados.put("nome", aluno.getNome());  
    dados.put("endereco", aluno.  
        getEndereco());  
    dados.put("telefone", aluno.getTelefone());  
    dados.put("site", aluno.getSite());  
    dados.put("nota", aluno.getNota());  
    db.insert("Alunos", null, dados);  
}
```

Selecionamos com o mouse, no insere(Aluno aluno), o seguinte:

```
ContentValues dados = new ContentValues();
dados.put("nome", aluno.getNome());
dados.put("endereco", aluno.
getEndereco());
dados.put("telefone", aluno.getTelefone());
dados.put("site", aluno.getSite());
dados.put("nota", aluno.getNota());
```

Clicamos com o botão direito e seguimos por Refactor> Extract> Method. Abrirá uma janela pedindo para nomear o método. No campo Name, chamaremos de pegaDadosDoAluno. Perceba que na *signature preview* conseguimos ver que ele criará um método que recebe um (Aluno aluno) e devolve um ContentValues.



Agora, vamos voltar no altera(Aluno aluno) e completar o ContentValues com aluno.

Teremos ContentValues dados = pegaDadosDoAluno(aluno).

Vamos acessar o db.update para completar o restante dos parâmetros do parênteses. Em vez de concatenar o parâmetro, colocaremos um ? e esse será substituído pelo parâmetro seguinte, que é uma Array de String, um params. Teremos update("Aluno", dados, "id=?", params).

Na linha de cima do update vamos construir uma Array de String, uma String[] params

= {}. Dentro das chaves colocaremos os valores que entrarão no lugar da interrogação, no caso, o id do aluno. Acrescentaremos no parenteses aluno.getId e como é um Long temos que converte-lo usando a .toString. Digitaremos String[] params = {aluno.getId().toString()}. Ficaremos com:

```
public void altera(Aluno aluno) {
    SQLiteDatabase db = getWritableDatabase();
    ContentValues dados = pegaDadosDoAluno(aluno);

    String[] params = {aluno.getId().toString()};
    db.update("Aluno", dados, "id = ?", params);
}
```

Só falta testar! Vamos salvar e dar um play.

Vamos editar o aluno “Jeferson”, acrescentando nele o sobrenome “Silva” e ao fazer isso constatamos que a aplicação altera o aluno.

Chegamos ao fim! Todos os passos básicos da nossa Agenda estão prontos!



Google Play Services

Se você é usuário da plataforma Android, já deve ter observado que em seu smartphone e/ou tablet existe um aplicativo pré-instalado chamado Google Play Services. Ele é o mecanismo oficial de distribuição de aplicativos android. Publicar seu aplicativo na loja permite que ele seja baixado, instalado e usado por toda a comunidade usuária do android.

Para o desenvolvedor é um tremendo facilitador pois reduzi a necessidade de desenvolver o mesmo aplicativo para as dezenas de versões diferentes do Android. Inclusive o Google está criando ou movendo algumas APIs para o Google Play Services. O objetivo é que as melhorias ou bugs possam ser corrigidos atualizando o aplicativo do Google Play Services pela loja.

O usuário também pode avaliar seu aplicativo e deixar comentários sobre ele, o que ajuda a identificar as possíveis tendências de uso e as áreas que apresentam problemas. O Google Play Services é usado para atualizar apps do Google e apps do Google Play, fornece também funcionalidades essenciais, como autenticação para seus serviços do Google, contatos sincronizados, acesso a todas as configurações de privacidade do usuário mais recentes e serviços baseados na localização com qualidade mais alta e menor consumo de recursos.

O Google Play Services também aprimora sua experiência com apps. Acelera pesquisas off-line, fornece mapas mais imersivos e aprimora experiências de jogo. Os apps podem não funcionar caso você desinstale o Google Play Services.

Para usar a Google Maps Android API, é necessário registrar o projeto do aplicativo no Google Developers Console e obter uma chave de API do Google para adicioná-la ao aplicativo. Isto pode ser feito na página <https://console.developers.google.com/>.



Criando um Arquivo de distribuição

Seu aplicativo está finalizado e pronto para ser publicado, a primeira coisa a se fazer é empacotar seu app para que ele possa ser colocado em seus dispositivos. Para tanto você precisa criar um arquivo de pacote Android ou arquivo APK.

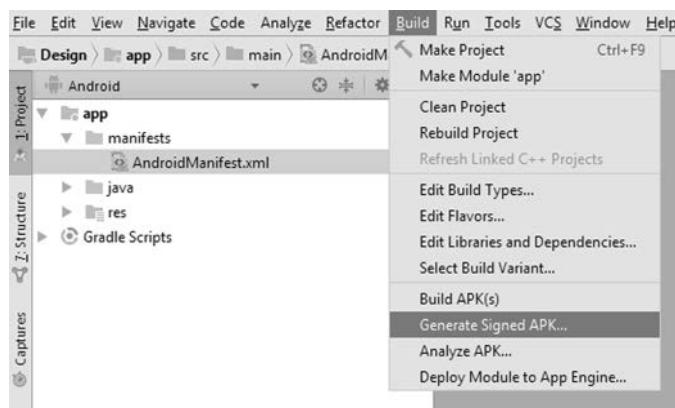
O android requer que todos os aplicativos instalados seja assinados digitalmente com um certificado que contém um par de chaves pública/privada. A chave privada é mantida pelo desenvolvedor. O certificado usado para assinar digitalmente o aplicativo identifica o desenvolvedor e estabelece as relações de confiança entre os apps.

Essa chave portanto é de grande importância para identificar seu aplicativo na Google Play Store, faça backup de seu armazenamento de chaves em um local seguro por que se você perder, não poderá mais atualizar seu aplicativo na loja.

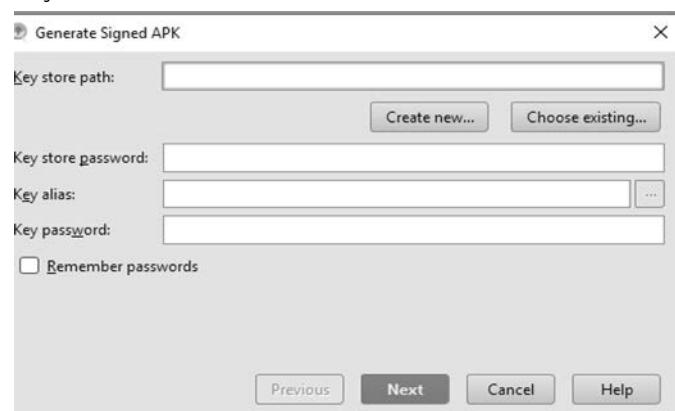


Criando um arquivo APK:

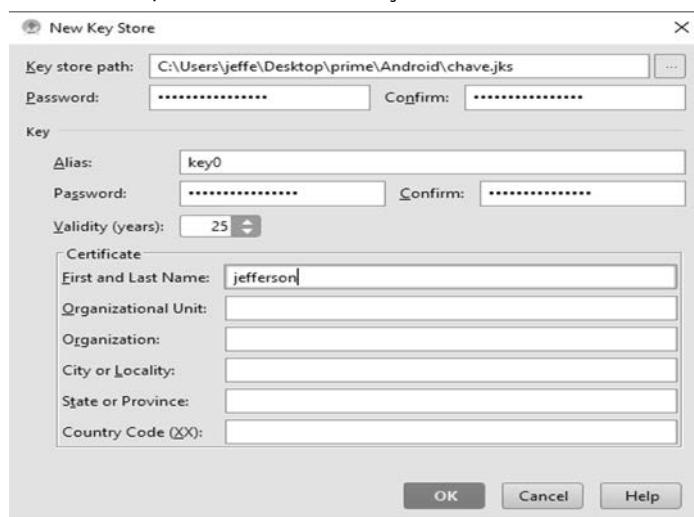
Para criar seu primeiro apk, siga estas etapas: Abra um projeto no seu Android Studio. Clique na opção buid e depois clique em Generate signed APK.



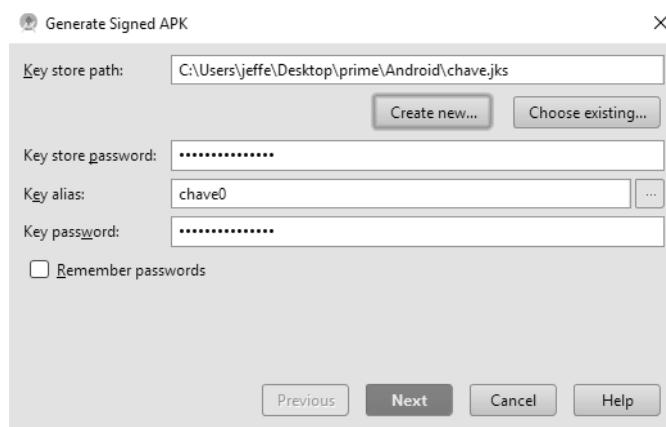
Agora vamos indicar ao Android Studio aonde ele deve armazenar a chave da assinatura digital escolha o caminho aonde ela está salva no campo Key store path. Se você não possuir nenhuma chave, clique no botão Create new para que o android studio te ajude na criação de uma nova chave.



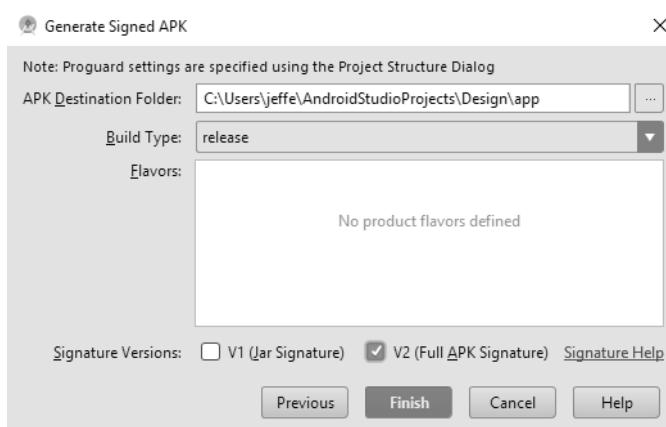
Agora vamos criar nossa chave, clique nos três pontos escolha um local que a sua chave ficará armazenada e digite uma senha, repita a senha no campo alias e preemcha ao menos um campo na área certificação.



Agora todos os campos estão preenchidos clique em next.



Selecione o tipo de assinatura V2 pois ela te dá mais segurança e proporciona instalações mais rápidas do app, após isso selecione o botão Finish e aguarde.



Publicando seu aplicativo na Google Play Store

Após ter criado seu arquivo APK, você poderá lança-lo na Google Play Store, mas primeiro é preciso que você crie um perfil de desenvolvedor google. Para a criação desse perfil você necessitara de uma conta Google.

Para criar seu perfil de desenvolvedor, siga estas etapas:

Abra seu navegador Web e navegue para <https://play.google.com/apps/publish>/ Clique no botão continuar pagamento.

Coloque seus dados e realize a compra, são 25 dólares mais impostos.

Forneça os detalhes de sua conta e após a confirmação de pagamento você terá acesso ao Google Play Store Developer Site.

Pronto agora você tem o seu APK e sua licença de desenvolvedor agora entre no site <https://play.google.com/apps/publish> e você verá a seguinte imagem.



Para publicar o app faça o seguinte:

Clique em criar app.

Após isso coloque o título do app.

Criar app

Idioma padrão *

Português (Brasil) – pt-BR

Título *

0/50

CANCELAR CRIAR

Coloque os dados que a próxima página te pede, será necessário que você insira: o caminho em que o arquivo está salvo no seu computador, as Screenshots do seu aplicativo, Título, descrição, texto promocional, defina o tipo de aplicativo, defina a categoria do aplicativo, selecione o local onde seu aplicativo deve ficar visível, dados de web site e e-mail.

Detalhes do app

Posto Fácil
Rascunho

PORTUGUÊS (BRASIL) – pt-BR

Gerenciar traduções

Os campos marcados com * devem ser preenchidos antes de publicar.

Título *

Português (Brasil) – pt-BR

Posto Fácil

11/50

Breve descrição *

Português (Brasil) – pt-BR

0/80

Descrição completa *

Português (Brasil) – pt-BR

0/80

SALVAR RASCUNHO

Se tudo estiver conforme as diretrizes da loja, clique em salvar rascunho que o site irá carregar automaticamente suas configurações e salva-las.

Seu aplicativo está criado, publicado e depois de alguns dias os usuários poderão instala-lo. Com essas facilidades o google promove um ciclo de lançamento rápido e eficiente.

Agora você poderá acompanhar suas avaliações, comentários, relatório de erros, número de instalações ativas e muito mais.



Exercícios Extras

Projeto cardápio

Com tudo que nós aprendemos até agora, vamos implementar um projeto de um simples cardápio onde o atendente irá marcar os pedidos do cliente. Para isso usaremos checkbuttons, e um alertDialog para avisar ao usuário quanto seu pedido custará. Crie um novo projeto com o nome de Cardápio. Não se esqueça de nomear a activity com um nome de sua preferencia.

Vamos aos dados do projeto: Nome do projeto: Projeto cardapio.

Api usada: 15.

Nome da Activity: Principal

Após criar o projeto, iremos definir um atributo de tamanho de fonte padrão no arquivo responsável pelo tema do aplicativo para que possamos evitar redundância de código. O código no arquivo Styles.xml ficara assim:

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
        <item name="android:textSize">18sp</item>
    </style>

</resources>
```

Agora vamos implementar o layout do nosso aplicativo, segue o código a seguir:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    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="com.example.jeffe.cardapiotitanic.MainActivity">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Clique no pedido do Cliente"
        android:textSize="25sp"
        android:id="@+id/textView"/>

    <CheckBox
        android:text="Hamburguer (R$ 7,00)"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="40dp"
        android:layout_marginStart="40dp"
        android:layout_marginTop="38dp"
        android:id="@+id/checkBox"
        android:layout_below="@+id/textView" />

    <CheckBox
        android:text="Cachorro-Quente (R$ 3,00)"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/checkBox4"
        android:layout_below="@+id/checkBox3"
        android:layout_alignLeft="@+id/checkBox3"
        android:layout_alignStart="@+id/checkBox3" />

    <CheckBox
        android:text="Coca-cola lata (R$ 3,50)"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:id="@+id/checkBox5"
        android:layout_below="@+id/checkBox4"
        android:layout_alignLeft="@+id/checkBox4"
        android:layout_alignStart="@+id/checkBox4" />

    <CheckBox
        android:text="X-Egg-Bacon (R$ 6,00)"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/checkBox3"
        android:layout_below="@+id/checkBox2"
        android:layout_alignLeft="@+id/checkBox2"
        android:layout_alignStart="@+id/checkBox2" />

    <CheckBox
        android:text="X-Salada (R$ 5,00)"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/checkBox2"
        android:layout_below="@+id/checkBox"
        android:layout_alignLeft="@+id/checkBox"
        android:layout_alignStart="@+id/checkBox" />

    <Button
        android:text="Calcular"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/checkBox5"
        android:layout_alignRight="@+id/checkBox2"
        android:layout_alignEnd="@+id/checkBox2"
        android:layout_marginTop="37dp"
        android:id="@+id/button"
        android:background="#a2caf0"
        android:textStyle="bold"
        android:textSize="20sp"
        android:textColor="#ffffffff"
        android:padding="4sp"/>

</RelativeLayout>
```

Repare que utilizamos um novo gerenciador de layout, o RelativeLayout, por isso foi necessário acrescentar alguns novos atributos. Abaixo explicarei esses novos atributos, para isso vou pegar parte do código e colocar logo abaixo:

```
<TextView android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Clique no pedido do Cliente"  
    android:textSize="25sp" android:id="@+id/textView"/>  
  
<CheckBox  
    android:text="Hamburguer (R$ 7,00)"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginLeft="40dp"  
    android:layout_marginStart="40dp"  
    android:layout_marginTop="38dp"  
    android:id="@+id/checkBox"  
    android:layout_below="@+id/textView" />  
  
<CheckBox  
    android:text="Cachorro-Quente (R$ 3,00)"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/checkBox4"  
    android:layout_below="@+id/checkBox3"  
    android:layout_alignLeft="@+id/checkBox3"  
    android:layout_alignStart="@+id/checkBox3" />
```

Observe que logo abaixo do textView temos um checkBox e temos uma propriedade nova dentro desse checkbox que é essa: android:layout_below="@+id/textView". Ela é responsável por colocar o checkbox logo abaixo do textView. Logo após temos outro checkbox e a propriedade acima citada aparece de novo só que agora ela coloca esse novo checkbox abaixo do checkBox3. Depois encontramos duas propriedades são elas: android:layout_alignLeft="@+id/checkBox3" e android:layout_alignStart="@+id/checkBox3" as duas são responsáveis por alinhar a esquerda o checkbox4 com o checkbox3. A diferença entre as duas é que a primeira só serve para as APIs abaixo da 16, para androids mais modernos se usa a segunda.

Agora vamos modificar os códigos em java. A primeira coisa a se fazer é declarar os objetos e logo em seguida fazer as referencias devidas dos objetos com os widgets no layout. Feito isso é hora de implementar um listener no botão para que ele pegue o clique do usuário calcule o pedido e mostre a ele.

Segue o código abaixo:

```
public class MainActivity extends AppCompatActivity {

    CheckBox cBombado,cXSalada,cXEggBacon,cCachorro,cCoca; Button button;
    double total;

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

        cBombado = (CheckBox) findViewById(R.id.checkBox);
        cXSalada = (CheckBox) findViewById(R.id.checkBox2);
        cXEggBacon = (CheckBox) findViewById(R.id.checkBox3);
        cCachorro = (CheckBox) findViewById(R.id.checkBox4);
        cCoca = (CheckBox) findViewById(R.id.checkBox5);

        button = (Button) findViewById(R.id.button);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                if (cBombado.isChecked()){ total += 7;
                }
                if (cXSalada.isChecked()){ total += 5;
                }
                if (cXEggBacon.isChecked()){ total += 6;
                }
                if (cCachorro.isChecked()){ total += 3;
                }
                if (cCoca.isChecked()){ total += 3.5;
                }
            }
        });
    }
}
```

```
AlertDialog.Builder alerta = new AlertDialog.Builder(MainActivity.this); alerta.setTitle("Total do pedido");
    alerta.setMessage("O total do pedido é de: R$"+total); alerta.setNeutralButton("OK",null);
    alerta.show(); total = 0;
}
});

}

}
```

Repare que não existe nenhuma novidade no código, só devemos ficar atentos no método onClick ele funciona da seguinte forma:

```
public void onClick(View v) {

    if (cBombado.isChecked()){ total += 7;
    }
    if (cXSalada.isChecked()){ total += 5;
    }
    if (cXEggBacon.isChecked()){ total += 6;
    }
    if (cCachorro.isChecked()){ total += 3;
    }
    if (cCoca.isChecked()){ total += 3.5;
    }
}
```

Se o checkbox estiver marcado ele entra no if e coloca o valor numérico dentro da variável total, se mais de um checkbox estiverem marcados a variável total irá receber a soma dos valores correspondentes.

Ao final do código é criado um AlertDialog que irá informar ao usuário sobre o total do pedido de seu cliente.

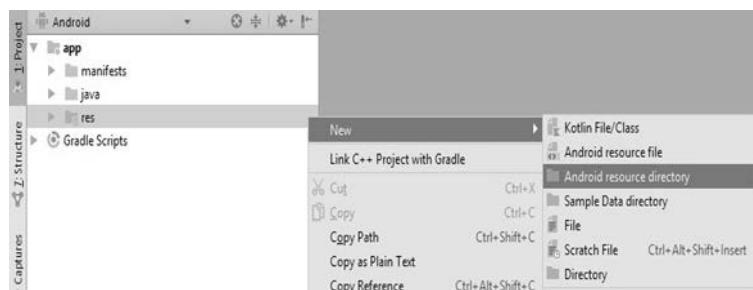


Para que o aplicativo funcione comece a funcionar marque as opções e clique em calcular, aparecerá a seguinte tela:

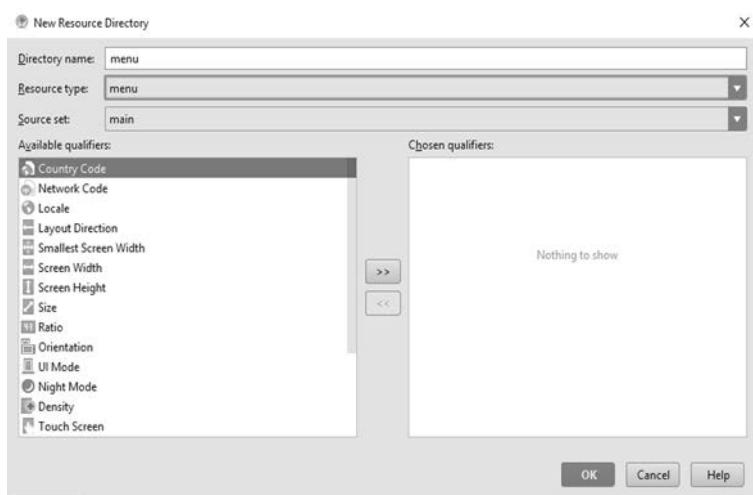


Agora vamos implementar um menu com uma lixeira para que todos os checkbox sejam desmarcados rapidamente com um só clique.

Para isso devemos primeiramente criar uma pasta menu dentro da pasta res, para isso clique com o botão direito em cima da pasta res logo após clique em new e depois android resource directory. confira as imagens:

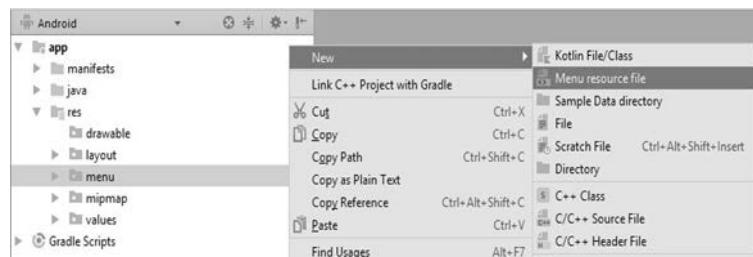


Após esses passos irá aparecer essa seguinte tela:

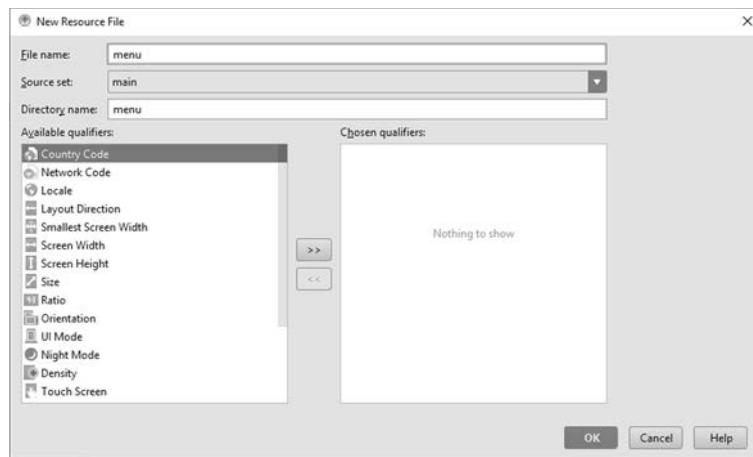


Observe que em Resource type devemos selecionar menu e depois pressionar o botão ok.

Logo após clique em cima da pasta menu com o botão direito e clique na opção Menu resource file para criar o seu arquivo de menu.

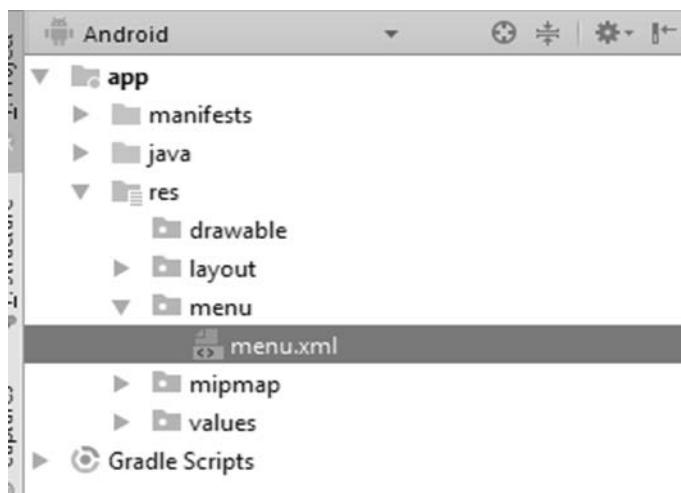


Depois que você clicar em Menu resource file a tela a seguir deverá aparecer:



Lembre-se de nomear seu menu no campo File name.

Depois de executar esses passos o seu arquivo de menu deve aparecer na tela do Android Studio dessa forma



Dentro do arquivo menu.xml digite o seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:title="Apagar"
        android:id="@+id/apagar"
        android:icon="@android:drawable/ic_menu_delete"
        app:showAsAction="always" />
```

```
    android:checkable="false" />
</menu>
```

Agora vamos aos códigos java. Primeiro temos que implementar o método onCreateOptionsMenu que é responsável por criar o menu. Dentro desse método temos que inflar o layout menu que acabamos de criar, para que o android associe os códigos java com o layout.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //linha responsável por inflar o menu getMenuInflater().inflate(R.menu.
menu, menu);

    return super.onCreateOptionsMenu(menu);
}
```

Depois temos que implementar o método onOptionsItemSelected que será responsável por gerenciar os itens do nosso menu. Ele irá ser capaz de captar o clique e tratar de acordo com a ação que programamos para cada item.

Segue o código:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    //Linha responsável por atribuir a variável id, o item do menu clicado int id =
item.getItemId();

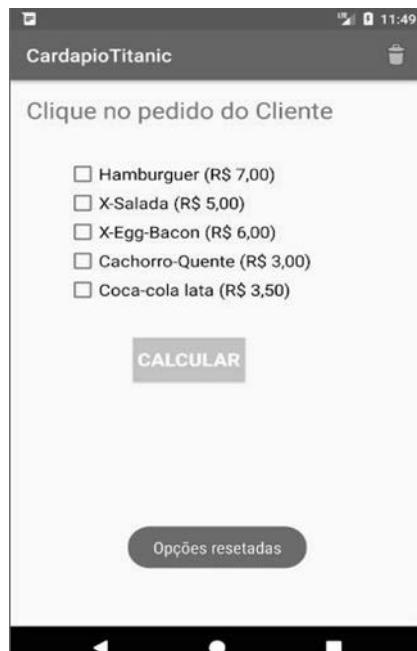
    //Linha responsável por fazer uma verificação no item do menu para ver se
    //é o mesmo id da nossa variável
    if (id == R.id.apagar){

        //métodos para desmarcar o checkBox cBombado.setChecked(false); cXSa-
        //lada.setChecked(false);
        cCoca.setChecked(false);
        cXEggBacon.setChecked(false);
        cCachorro.setChecked(false);

        //mensagem que informará ao usuário que as opções foram resetadas.
        Toast toast = Toast.makeText(MainActivity.this, "Opções resetadas",Toast.
LENGTH_SHORT);
    }
}
```

```
toast.show();  
  
}  
return super.onOptionsItemSelected(item);  
}
```

O que fizemos primeiro foi criar uma variável id para pegar o item de menu clicado, essa operação foi feita através do atributo item em conjunto com o método getItemId(), esse dois retornam exatamente o id do item clicado. Depois foi usada uma verificação com a estrutura de dados if para saber se o botão clicado foi o apagar, depois de verificado, usamos o método setChecked(false) para desmarcar nossos checkboxes e ao final mostramos um pequeno aviso ao usuário informando que as opções foram resetadas. Rode seu projeto e você verá um resultado parecido com esse ao clicar na lixeira:



Projeto CursosPraxis

Nesse projeto vamos criar um catálogo de cursos onde o usuário poderá escolher entre as modalidades presencial ou EAD. Logo depois ele escolherá os cursos que irá se matricular e se o usuário digitar o cupom de promoção terá um abate no preço dos cursos. Ao final o app irá informar ao usuário quanto ficará o custo de todos os cursos selecionados. Para que essas funcionalidades sejam implementadas utilizaremos os widgets RadioButton, EditText, Checkbox e Button.

Vamos começar, crie um projeto chamado CursosPraxis e faça toda a configuração inicial, seguindo os dados do projeto:

Vamos aos dados do projeto:

Nome do projeto: CursosPraxis. Api usada: 15.

Nome da Activity: Principal

Logo após criar sua primeira activity vamos implementar o nosso layout. Para isso abra o seu arquivo responsável pelo layout e digite o seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    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="com.example.jeffe.cursospraxis.MainActivity">

    <TextView android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Escolha a modalidade dos cursos."
        android:textSize="19sp" android:layout_centerHorizontal="true"/>
    <RadioGroup android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView"
        android:layout_alignLeft="@+id/textView"
        android:layout_marginTop="30dp" android:orientation="horizontal" android:id="@+id/rgopcoes">

        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="18sp"
```

```
        android:text="Presencial"
        android:id="@+id/rbPresencial"
    />

    <RadioButton android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp" android:text="EAD"
        android:id="@+id/rbEAD"
        android:layout_marginLeft="105dp"/>
    </RadioGroup>

    <CheckBox android:text="Java"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/rgopcoes"
        android:layout_alignLeft="@+id/rgopcoes"
        android:layout_marginTop="53dp"
        android:id="@+id/cbJava" />

    <CheckBox android:text="Android"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/cbJava"
        android:layout_alignLeft="@+id/cbJava"
        android:layout_alignStart="@+id/cbJava"
        android:layout_marginTop="24dp"
        android:id="@+id/cbAndroid" />

    <CheckBox android:text="PHP"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/cbAndroid"
        android:layout_alignLeft="@+id/cbAndroid"
        android:layout_marginTop="21dp"
        android:id="@+id/cbPhp" />

    <CheckBox android:text="Python"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/cbPython"
```

```
        android:layout_alignBaseline="@+id/cbAndroid"
        android:layout_alignRight="@+id/textView"
    />

    <CheckBox android:text="Ruby"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/cbRuby"
        android:layout_above="@+id/cbAndroid"
        android:layout_alignLeft="@+id/cbPython"
        android:layout_alignStart="@+id/cbPython"
        android:layout_alignRight="@+id/textView"/>

    <CheckBox android:text="HTML + CSS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/cbHtml"
        android:layout_alignBaseline="@+id/cbPhp"
        android:layout_alignBottom="@+id/cbPhp"
        android:layout_alignLeft="@+id/cbPython"
        android:layout_alignStart="@+id/cbPython" />

    <Button
        android:text="Total" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button" android:layout_below="@+id/edtCupon"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="12dp" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:hint="Digite o cupom de desconto"
        android:ems="10"
        android:layout_below="@+id/cbHtml"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="24dp"
        android:id="@+id/edtCupon"
        android:textAlignment="center"/>

</RelativeLayout>
```

Reforçando que o RadioGroup e o RadioButton funcionam da seguinte forma, o primeiro serve para criar um container aonde os RadioButton se agruparão, dessa forma, com eles agrupados o usuário só poderá escolher dentre uma opção. O funcionamento do checkBox já é totalmente diferente pois o usuário poderá marcar a todos se desejar.

Logo após implementar o nosso layout, vamos agora implementar os nossos códigos java, para isso abra o arquivo java de sua activity e comece a fazer as seguintes modificações:

```
public class MainActivity extends AppCompatActivity {  
  
    RadioGroup opcoes;  
    CheckBox java,android,php,ruby,python,html;  
    EditText cupon;  
    Button button;  
  
    Double total;  
    String desconto = "boravencer";  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        opcoes = (RadioGroup) findViewById(R.id.rgopcoes);  
  
        java = (CheckBox) findViewById(R.id.cbJava);  
        android = (CheckBox) findViewById(R.id.cbAndroid);  
        php = (CheckBox) findViewById(R.id.cbPhp);  
        ruby = (CheckBox) findViewById(R.id.cbRuby);  
        python = (CheckBox) findViewById(R.id.cbPython); html = (CheckBox) findViewById(R.id.cbHtml);  
  
        cupon = (EditText) findViewById(R.id.edtCupon);  
  
        button = (Button) findViewById(R.id.button);  
  
        button.setOnClickListener(new View.OnClickListener() {  
            @Override
```

```
public void onClick(View v) { total = 0.0;
int op = opcoes.getCheckedRadioButtonId();

if (op == R.id.rbEAD){ if(java.isChecked()){ total += 50;

}

if(android.isChecked()){ total += 50;
}

if(php.isChecked()){ total += 50;
}

if(ruby.isChecked()){ total += 50;
}

if(python.isChecked()){ total += 50;
}

if(html.isChecked()){ total += 50;
}

}

}else if(op == R.id.rbPresencial){

if(java.isChecked()){

total += 100;

}

if(android.isChecked()){ total +=100;
}

if(php.isChecked()){ total += 100;
}

if(ruby.isChecked()){ total += 100;
}

}

if(python.isChecked()){ total += 100;
}

}

if(html.isChecked()){ total += 100;
}

}

if(cupon.getText().toString().equalsIgnoreCase(desconto)) {
AlertDialog.Builder alerta = new AlertDialog.Builder(MainActivity.this);
alerta.setTitle("Resultado");
alerta.setMessage("O Total a ser pago é de: R$" + (total - (total * 0.3)));
}
```

```

        alerta.setNeutralButton("OK", null);
        alerta.show();
    }else {
        AlertDialog.Builder alerta = new AlertDialog.Builder(MainActivity.this);
        alerta.setTitle("Resultado");
        alerta.setMessage("O Total a ser pago é de: R$" + total);
        alerta.setNeutralButton("OK", null);
        alerta.show();

    }
}
});
}
}

```

Vamos entender o código.

No primeiro momento foram feitas as declarações dos objetos e depois dentro do método onCreate foram feitas as referencias com os widgets do layout. Logo após isso implementamos o Listener no botão para que o clique do usuário fosse captado. Dentro desse método criamos uma variável total que irá receber e somar os valores de cada curso, e também criamos outra variável op que irá monitorar o radioGroup e conseguir identificar qual radioButton está ativado.

Feito isso começam as verificações para saber qual RadioButton está ativado e quais os Checkboxes também estão ativados. Ao final verifica-se se o usuário possui um cumpom de desconto e o resultado final da compra aparece para o usuário. Segue as telas da app rodando:



Depois de selecionar a modalidade e os cursos clique no botão total e veja a próxima tela



Agora vamos implementar o menu para resetar todos os RadioButtons e Checkbox. Primeiro temos que criar a pasta Menu dentro da pasta res do nosso projeto, lembre-se que esse procedimento já foi feito em projetos anteriores. Depois de criada a pasta menu, crie um arquivo xml de menu dentro dela. Para ver as imagens desse procedimento basta voltar ao projeto do cardápio.

Feito isso digite o seguinte código no arquivo de menu:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:title="Item" android:id="@+id/límpar"
          android:icon="@android:drawable/ic_menu_delete"
          app:showAsAction="always" />
</menu>
```

Agora vamos para os códigos java. Primeiro crie um novo método na nossa activity chamado onCreateOptionsMenu, ele será responsável por inflar o layout do nosso menu. Segue o código:

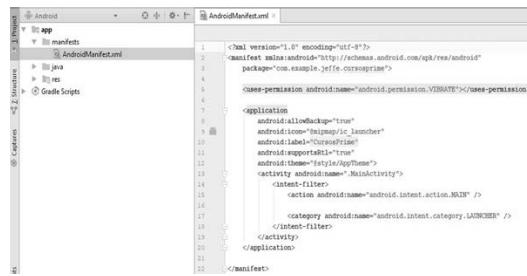
```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    //Linha responsável por inflar o menu getMenuInflater().inflate(R.menu.
    menu,menu);
    //Linha que irá retornar o nosso menu
    return super.onCreateOptionsMenu(menu);
}
```

Feito isso agora vamos criar o método que gerencia cada item do nosso menu que é o onOptionsItemSelected. Nele vamos implementar alguns métodos em nossos widgets que irão resetar a todos eles. Segue o código:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
  
    //Linha que desmarca os RadioButtons dentro do RadioGroup opcoes.clearCheck();  
  
    //Bloco de Código que desmarca os CheckBoxs java.setChecked(false);  
    android.setChecked(false);  
    php.setChecked(false);  
    python.setChecked(false);  
    html.setChecked(false);  
    ruby.setChecked(false);  
  
    //Linha que apaga o texto do cupom cupon.getText().clear();  
  
    //Mensagem que aparece ao usuário quando ele reseta doas as opções  
    Toast toast = Toast.makeText(MainActivity.this,"Opções resetadas",Toast.LENGTH_SHORT);  
    toast.show();  
  
    //Retorna o item de menu clicado  
    return super.onOptionsItemSelected(item);  
}  
}
```

Pronto o menu está totalmente funcional, mas agora vamos implementar outra novidade que é uma pequena vibração que o android dispara quando o usuário clica no botão total. Para que nos possamos fazer isso primeiramente devemos abrir o nosso arquivo manifest e pedir uma autorização especial para habilitar a funcionalidade de vibração no nosso projeto. Olhe o exemplo na imagem a seguir:



Repare no código uses-permission na imagem, ele que é o responsável por pedir ao usuário acesso ao recurso de vibração no celular dele.

Segue o código para que você coloque dentro do seu arquivo AndroidManifest.xml:

```

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

```

Agora vamos voltar para nossa Activity e vamos criar um método responsável por controlar a vibração do botão que não poderá ser excessiva para não prejudicar a experiência do nosso usuário com o app. Nesse método precisamos de um objeto do tipo Vibrator que é responsável por gerenciar a vibração do android, depois vamos criar um tempo de vibração para esse objeto. O código fica assim:

```

public void vibrar(){

    //Criando o objeto vibrator
    Vibrator vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);

    //Criando o tempo de vibração long tempo = 30;

    //Executando a vibração com o método vibrate e setando o nosso tempo
    //nele. vibrator.vibrate(tempo);
}

```

Agora chame o método vibrar dentro do método onCreate de sua classe. O código completo de nosso projeto está logo a seguir:

```

public class MainActivity extends AppCompatActivity {

    RadioGroup opcoes;
    CheckBox java, android, php, ruby, python, html; EditText cupon;

```

```
Button button; Double total;  
String desconto = "boravencer";  
  
@Override  
protected void onCreate(Bundle savedInstanceState) { super.onCreate(sa-  
vedInstanceState); setContentView(R.layout.activity_main);  
  
opcoes = (RadioGroup) findViewById(R.id.rgopcoes);  
  
java = (CheckBox) findViewById(R.id.cbJava);  
android = (CheckBox) findViewById(R.id.cbAndroid);  
php = (CheckBox) findViewById(R.id.cbPhp);  
ruby = (CheckBox) findViewById(R.id.cbRuby);  
python = (CheckBox) findViewById(R.id.cbPython);  
html = (CheckBox) findViewById(R.id.cbHtml);  
  
cupon = (EditText) findViewById(R.id.edtCupon);  
  
button = (Button) findViewById(R.id.button);  
  
button.setOnClickListener(new View.OnClickListener() { @Override  
public void onClick(View v) {  
  
vibrar(); total = 0.0;  
int op = opcoes.getCheckedRadioButtonId();  
if (op == R.id.rbEAD){ if(java.isChecked()){ total += 50;  
}  
if(android.isChecked()){ total += 50;  
}  
if(php.isChecked()){ total += 50;  
}  
if(ruby.isChecked()){ total += 50;  
}  
if(python.isChecked()){ total += 50;  
}  
if(html.isChecked()){ total += 50;  
}  
  
}else if(op == R.id.rbPresencial){ if(java.isChecked()){  
total += 100;  
}  
}});
```

```

    }

    if(android.isChecked()){ total += 100;
    }

    if/php.isChecked(){ total += 100;
    }

    if(ruby.isChecked()){ total += 100;
    }

    if(python.isChecked()){ total += 100;
    }

    if(html.isChecked()){ total += 100;
    }

}

if(cupon.getText().toString().equalsIgnoreCase(desconto)) {
    AlertDialog.Builder alerta = new AlertDialog.Builder(MainActivity.this);
    alerta.setTitle("Resultado");
    alerta.setMessage("O Total a ser pago é de: R$" + (total - (total * 0.3)));
    alerta.setNeutralButton("OK", null);
    alerta.show();
} else {
    AlertDialog.Builder alerta = new AlertDialog.Builder(MainActivity.this);
    alerta.setTitle("Resultado");
    alerta.setMessage("O Total a ser pago é de: R$" + total);
    alerta.setNeutralButton("OK", null);
    alerta.show();
}
}

});

}

public void vibrar(){
    Vibrator vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
    long tempo = 30;
    vibrator.vibrate(tempo);

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {

```

```

getMenuInflater().inflate(R.menu.menu,menu);
return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
opcoes.clearCheck();

java.setChecked(false);
android.setChecked(false);
php.setChecked(false);
python.setChecked(false);
html.setChecked(false);
ruby.setChecked(false);

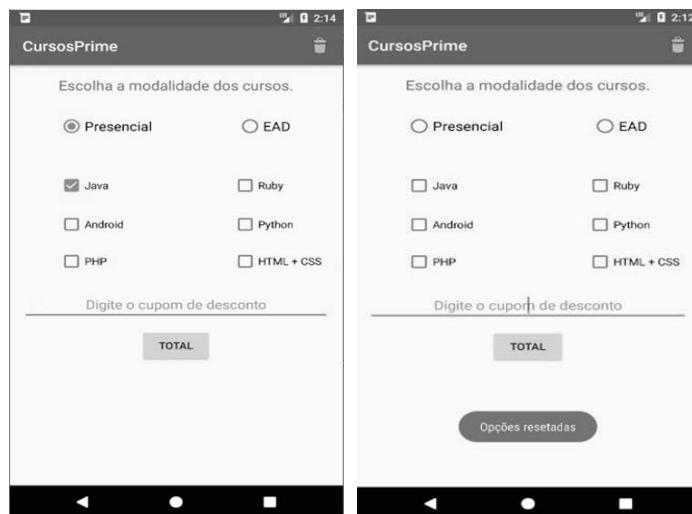
cupon.getText().clear();

Toast toast = Toast.makeText(MainActivity.this,"Opções resetadas",Toast.LENGTH_SHORT);
toast.show();

return super.onOptionsItemSelected(item);
}
}

```

Execute seu projeto, e faça os testes devidos. Lembre-se a funcionalidade de vibrar só poderar ser testada em um dispositivo real.





Projeto Posto Fácil

Nesse projeto o usuário poderá selecionar entre álcool ou gasolina, colocar o preço do combustível o valor que você desejará abastecer, o app calculará e informará ao usuário a quantidade de litros total. Para isso usaremos um RadioGroup com dois RadioButtons, um EditText, uma seekBar, e mais dois textView.

A novidade nesse projeto é que para começar os cálculos não será necessário que o usuário pressione qualquer botão, assim que ele inserir os dados o app começa a calcular automaticamente.

Dados do projeto:

Nome do projeto: Posto Fácil. Api usada: 15.

Nome da Activity: Principal Nome do layout: activity_principal

Primeiro vamos nos concentrar no arquivo de layout, abra o arquivo activity_principal e digite o seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    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"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.example.jeffe.design.TelaPrincipal"
    tools:showIn="@layout/activity_tela_principal">

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView android:id="@+id/TextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Escolha o tipo de combustível"
            android:layout_gravity="center"
            android:textStyle="bold"
            android:textSize="20sp"
            android:layout_marginTop="10dp"

            android:textColor="@color/fonte" />

        <RadioGroup android:id="@+id/rgrupo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:orientation="vertical">

            <RadioButton android:id="@+id/checkbox1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Álcool" />

            <RadioButton android:id="@+id/checkbox2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Gasolina" />

        </RadioGroup>
    </LinearLayout>
</ScrollView>
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_marginTop="15dp"
        android:layout_gravity="center"
        android:background="@drawable/fundo_ativado"
        android:paddingTop="2dp"
        android:paddingBottom="2dp"
        android:paddingLeft="7dp"
        android:paddingRight="7dp">

    <RadioButton android:id="@+id/rbGasolina"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Gasolina"
        android:layout_marginRight="40dp"
        android:textColor="@color/primary_text"
        android:textSize="18sp"
        android:buttonTint="@color/colorPrimary"/>

    <RadioButton
        android:id="@+id/rbAlcool"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Álcool"

        android:textColor="@color/primary_text"
        android:textSize="18sp"
        android:buttonTint="@color/colorPrimary"/>

</RadioGroup>

<TextView android:id="@+id/TextView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Valor do Combustível"
        android:layout_gravity="center"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_marginTop="15dp"
        android:textColor="@color/fonte"/>

<EditText android:id="@+id/edtLitro"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:background="@drawable/fundo"
        android:padding="4dp"
        android:textAlignment="center"
        android:textColor="#6b6565"
```

```
        android:textStyle="bold"

        android:inputType="numberDecimal"
        android:hint="toque aqui"
        android:enabled="false"/>

    <TextView android:id="@+id/TextView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Quanto você deseja abastecer?"
        android:textSize="20sp" android:textStyle="bold"
        android:layout_gravity="center"
        android:layout_marginTop="15dp"
        android:textColor="@color/fonte"/>

    <SeekBar android:id="@+id/skbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="15dp"
        android:layout_gravity="center"
        android:progressTint="@color/primary_light"/>
    <LinearLayout android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="center">
        <TextView android:id="@+id/TextView5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="18sp" android:text="R$"

            android:layout_marginRight="10dp"/>
        <TextView android:id="@+id/tvmseek"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:textSize="18sp"
            android:text="0"
            android:textColor="@color/colorAccent"
            android:textStyle="bold"/>

    </LinearLayout>
    <View
        android:layout_width="match_parent"
        android:layout_height="2dp"
        android:background="#e9e8e8"
        android:layout_marginTop="40dp"/>

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

```

    android:layout_marginTop="5dp"
    android:layout_gravity="center">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=" Quantidade de Litros:"
        android:layout_marginRight="60dp"
        android:layout_marginTop="20dp"
        android:textSize="20sp"
        android:textColor="@color/fonte"/>

    <TextView android:id="@+id/tvfCombustivel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0"
        android:layout_marginTop="20dp"
        android:textSize="20sp"
        android:textStyle="bold"
        android:textColor="@color/colorPrimaryDark"/>

</LinearLayout>
<View
    android:layout_width="match_parent"
    android:layout_height="2dp"
    android:background="#e9e8e8"
    android:layout_marginTop="20dp"/>
</LinearLayout>

</ScrollView>

```

Repare que no layout a grande novidade é o uso do ScrollView, ele é responsável por criar uma barra de rolagem para que o nosso usuário possa navegar por nossa aplicação. Ele deve ser o layout pai para funcionar corretamente. Usamos também o widget view que insere linhas horizontais que servem para criar divisões visuais no nosso layout.

Agora abra a activity Principal e digite o seguinte código java:

```

public class TelaPrincipal extends AppCompatActivity {
    //Aqui são criados os objetos e variáveis RadioGroup rGrupo;
    EditText edtLitro; SeekBar skbar; TextView mseek;
    TextView tvfcombustivel; int qtdfinal;
    double fcombustivel; double pcombustivel;
    DecimalFormat df = new DecimalFormat("###,##0.00");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tela_principal);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);

```

```

setSupportActionBar(toolbar);

//Aqui é feita as referencias entre objetos e widgets do layout rGrupo = (RadioGroup) findViewById(R.id.rgrupo);
edtLitro = (EditText) findViewById(R.id.edtLitro);
skbar = (SeekBar) findViewById(R.id.skbar);
mseek = (TextView) findViewById(R.id.tvmseek);
tvfcombustivel = (TextView) findViewById(R.id.tvfCombustivel);

app

//Essa linha é responsável por esconder o teclado numérico na inicialização do
this.getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_
INPUT_STATE_HIDDEN);

//Aqui é instanciado o método que irá bloquear a interação do usuário com a
seekbar
skbar.setOnTouchListener(new View.OnTouchListener() { @Override
public boolean onTouch(View v, MotionEvent event) { return true;
}
});

controle(); setseek();

}

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

//Nesse método é gerenciado os itens do menu @Override
public boolean onOptionsItemSelected(MenuItem item) {
// Handle action bar item clicks here. The action bar will

// automatically handle clicks on the Home/Up button, so long
// as you specify a parent activity in AndroidManifest.xml. int id = item.getItemId();

//noinspection SimplifiableIfStatement if (id == R.id.action_delete) {

rGrupo.clearCheck(); edtLitro.setText(""); skbar.setProgress(0); tvfcombustivel.setText("0");

return true;
}

return super.onOptionsItemSelected(item);
}

public void controle() {

```

```

rGrupo.setOnCheckedChangeListener(new RadioGroup.
OnCheckedChangeListener() {
@Override

public void onCheckedChanged(RadioGroup group, @IdRes int checkedId) {
edtLitro.setEnabled(true);

edtLitro.setBackground(ContextCompat.getDrawable(TelaPrincipal.
this,R.drawable.fundo_ativado));
skbar.setThumb(ContextCompat.getDrawable(TelaPrincipal.this,R.
drawable.thumb));

mseek.setTextColor(ContextCompat.getColor(TelaPrincipal.this,R.
color.colorPrimaryDark));
skbar.setOnTouchListener(new View.OnTouchListener() { @Override
public boolean onTouch(View v, MotionEvent event) { return false;
}
});

}

});

}

public void setseek() {

skbar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener()
{
int progresso; @Override
public void onProgressChanged(SeekBar seekBar, int progress, boolean
fromUser) {

progresso = progress; mseek.setText(""+progress); calculos();
}

@Override
public void onStartTrackingTouch(SeekBar seekBar) {
}

@Override
public void onStopTrackingTouch(SeekBar seekBar) {
mseek.setText(""+progresso);
calculos();

}

});

}

public void calculos(){ if(!(TextUtils.isEmpty(edtLitro.getText().toString()))){


```

```
qtdfinal = skbar.getProgress();
pcombustivel = Double.parseDouble(edtLitro.getText().toString()); fcombustivel = qtdfinal /
pcombustivel; tvfcombustivel.setText(String.valueOf(df.format(fcombustivel)));

}
}
}
```



Projeto LembraTexto

Vamos agora trabalhar com um dos modos de persistência de dados com o android que é o sharedPreferences.

Nesse projeto o usuário digitará um simples texto e assim guardaremos esse texto para depois mostrar ao usuário. Para isso utilizaremos um EditText e dois Buttons.

Vamos aos dados do projeto:

Nome do projeto: LembraTexto. Api usada: 15.

Nome da Activity: Principal Nome do layout: activity_principal

Vamos começar implementando o nosso layout. No arquivo activity_principal
digite o seguinte código:

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

    <TextView
        android:text="Shared Preferences"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"

        android:textStyle="bold"/>

    <EditText android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/tvTitulo"
        android:layout_margin="20dp"
        android:id="@+id/texto"/>
```

```

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Salvar" android:id="@+id(btnSalvar"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Mostrar"
    android:id="@+id	btnMostrar" />

</LinearLayout>

```

Logo após o termino do layout, vamos implementar os códigos responsáveis por fazer nosso app ser funcional. Abra a activity Principal e vamos começar a codificar. Segue o código:

```

public class Principal extends AppCompatActivity {

    //Criando os objetos. EditText texto;
    Button btnSalvar, btnMostrar;

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

        //Aqui são feitas as referências entre os objetos criados e seus widgets no layout texto =
        findViewById(R.id.texto);
        btnSalvar = findViewById(R.id.btnSalvar); btnMostrar = findViewById(R.id.btnMostrar);

        botão

        //Aqui é colocado um listenner no botão para que o android capture o clique no
        btnMostrar.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {

                //Instanciamos o objeto sharpref
                SharedPreferences sharpref = getPreferences(Principal.MODE_PRIVATE);

                // colocamos uma String nele para que possa ser identificado. String valor = sharpref.getString("MeuTexto",null);

                //mostramos ao usuário o texto que ele digitou
                Toast.makeText(getApplicationContext(), "Texto : "+valor,Toast.LENGTH_LONG).show();

            }
        });
    }
}

```

```
btnSalvar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        SharedPreferences sharpref = getPreferences(Principal.MODE_PRIVATE);

        //Estamos preparando o objeto para receber um valor digitado pelo usuário.
        SharedPreferences.Editor editor = sharpref.edit(); editor.putString("MeuTexto", texto.getText().toString()); editor.commit();

    }
});
```

Seu app deverá rodar assim:





Projeto Memes

Nesse projeto vamos utilizar nossos conhecimentos em relação a classe MediaPlayer e faremos um aplicativo que irá tocar sons. Você terá que fazer o download de um som em mp3 e uma imagem qualquer.

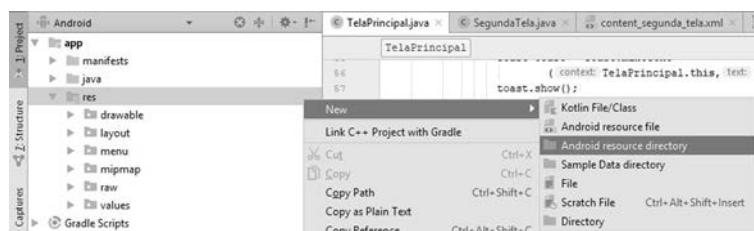
Vamos aos dados do projeto:

Nome do projeto: Memes. Api usada: 15.

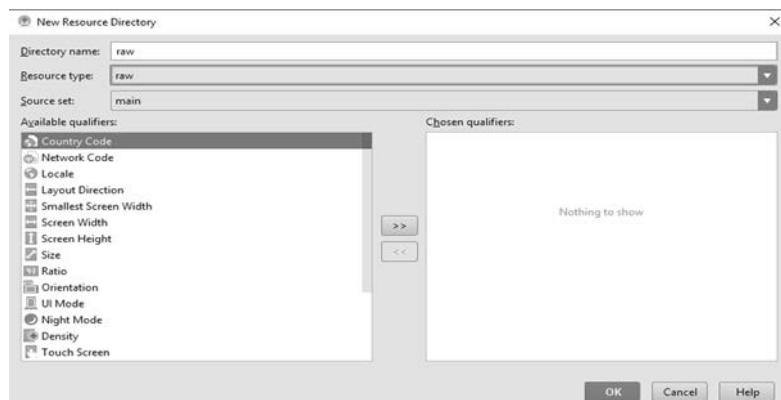
Nome da Activity: Principal Nome do layout: activity_principal

O som ficará dentro da pasta raw, e a imagem na pasta drawable, provavelmente a pasta raw não é criada junto com o projeto você terá que criar da seguinte forma:

Clique com o botão direito na pasta res, depois em new/ Android resource directory.



Logo após crie a pasta raw e arraste seu arquivo .mp3 lá para dentro



Feito esses processos vamos ao layout, segue o código:

```
<android.support.constraint.ConstraintLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
```

```
tools:context="com.example.jeffe.ogais.TelaPrincipal"
tools:showIn="@layout/activity_tela_principal">
<ImageButton android:id="@+id/gais"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:background="@drawable/gas"/>

</android.support.constraint.ConstraintLayout>
```

Segue os códigos java:

```
public class TelaPrincipal extends AppCompatActivity {

    ImageButton gais; MediaPlayer mpgais;

    //Primeiro instanciamos dois objetos, um do tipo ImageButton e outro do tipo MediaPlayer.

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tela_principal);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);

        setSupportActionBar(toolbar);

        //Agora vamos fazer as referencias entre os objetos e os widgets do layout. Note que o mediaPlayer não possui id, ele fica dentro da pasta raw do projeto.

        gais = (ImageButton) findViewById(R.id.gais); mpgais = MediaPlayer.create(this,R.raw
        ogas);

        //Aqui implementamos um listener no ImageButton para o clique ser captado

        gais.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                //Dentro do método que gerencia o clique do botão é feita uma verificação para
                //saber se o som já está tocando

                if(!mpgais.isPlaying()){

                    //Se não estiver, o som começa a tocar
                    mpgais.start();

                    //Se estiver tocando, o som é pausado, parado e resetado
                }else if (mpgais.isPlaying()){ mpgais.pause(); mpgais.stop(); mpgais.prepareAsync();}
```

```
}

}

});

}

//Nesse método é criado o menu @Override
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.menu_tela_principal, menu); return true;
}

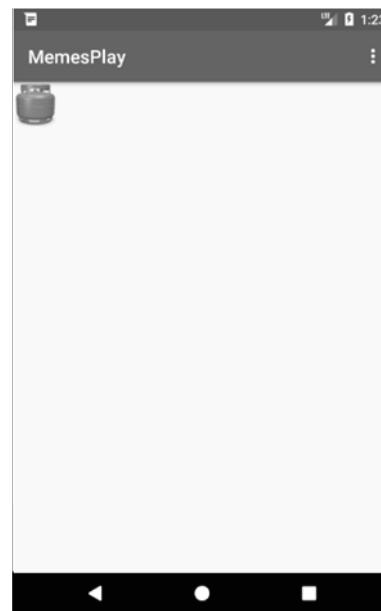
// Nesse método é gerenciado os itens do menu @Override
public boolean onOptionsItemSelected(MenuItem item) {

int id = item.getItemId();

if (id == R.id.action_settings) { return true;
}

return super.onOptionsItemSelected(item);
}
}
```

Executando o projeto ele ficará assim:



Clique na imagem e escute o som que você inseriu dentro do app.



Projeto CompartilhaText

Nesse projeto iremos montar uma aplicação capaz de compartilhar textos com outros aplicativos, para isso usaremos alguns recursos da classe Intent. Nosso layout será simples, apenas com dois EditText, um para o título e outro para o texto. No projeto existirá um item de menu que quando pressionado abre uma aba com opções para qual aplicativo externo você deseja enviar seu título e texto.

Vamos aos dados do projeto: Nome do projeto: Compartilha text. Api usada: 15.

Nome da Activity: Principal Nome do layout: activity_principal

Primeiramente vamos começar com o layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.jeffe.compartilhatext.MainActivity"
    android:orientation="vertical">

    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Compartilhe seu Texto"
        android:textSize="30dp"
        android:layout_gravity="center"
        android:textStyle="bold"/>

    <EditText android:id="@+id/titulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Escreva Seu Título" android:layout_gravity="center"
        android:textSize="25dp" android:ems="10"
        android:textAlignment="center"
        android:background="#E0E0E0"/>

    <EditText android:layout_marginTop="4dp"
        android:id="@+id/texto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@null"
        android:hint="digite seu texto"
        android:inputType="textMultiLine"
        android:textAlignment="textStart"
        android:gravity="start"/>

</LinearLayout>
```

O editText com o id texto é capaz de receber diversas linhas pois usa o atributo android:inputType="textMultiLine", ele é capaz também de fazer uma rolagem automática e ele tomara quase que a tela por completo.

Agora em seguida no seu arquivo de activity, vamos para os códigos java:

```
public class MainActivity extends AppCompatActivity {

    //Instanciamento de objetos TextView texto,titulo;
    String textS;

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

        //Referências
        texto = findViewById(R.id.texto);
        titulo = findViewById(R.id.titulo);
    }

    private void shareTextUrl(String msg) {

        //Aqui é instanciado uma intente que irá enviar uma mensagem para outras aplicações do tipo texto
        Intent share = new Intent(Intent.ACTION_SEND);
        share.setType("text/plain");
        share.putExtra(Intent.EXTRA_TEXT, msg);

        startActivity(Intent.createChooser(share, "Compartilhar link"));
    }

    // Criação do menu @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.menu,menu);
        return super.onCreateOptionsMenu(menu);
    }

    //Gerenciamento do item de menu @Override

    public boolean onOptionsItemSelected(MenuItem item) {

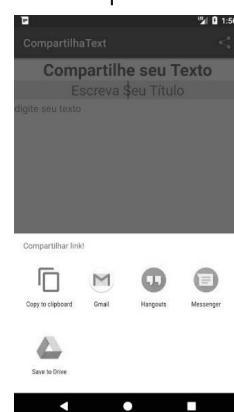
        switch (item.getItemId()){
            case R.id.comp:

                //Essa linha pega o que o usuário digitou e é chamado o método para que ela seja compartilhada
                this.textS = texto.getText().toString(); shareTextUrl(textS);
            }
            return super.onOptionsItemSelected(item);
        }
    }
}
```

Mande executar seu projeto ele ficará da seguinte forma:



Digite o texto e mande compartilhar que ele irá apresentar essa tela:





Referencial Bibliográfico

BRITO, Robson Cris, Android para iniciantes com Eclipse - Passo a Passo, 1^a edição, Editora Ciência Moderna, 2015.

ABLESON, Frank W., SEN, Robi, KING, Chris, ORTIZ, C. Enrique -vAndroid em ação, 3^a edição, 2012.

ANSELMO, Fernando - Android Em 50 Projetos, Editora Visual Books, 2012.

DARWIN, Ian F.- Android Cookbook - Problemas e soluções para desenvolvedores Android, Editora Novatec, 2012.

DEITEL, Paul J. - DEITEL, Harvey M., Android para Programadores Uma Abordagem Baseada em Aplicativos, Editoria Bookman, 2013.

NUDELMAN, Greg - Padrões de Projeto para o Android, Editora Novatec, 2013.
Deitel,Paul. - Android:como programar, Editora Bookman, 2015

Burton,Michael – Desenvolvimento de aplicativos Android para leigos, Editora Alta Books, 2014.