



## Projeto de Aplicações Android

Wagner Pereira Gomes

2018

## **Projeto de Aplicações Android**

Wagner Pereira Gomes

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

## Sumário

---

Capítulo 1. Android.....	9
1.1. Introdução .....	9
1.2. Instalação Android Studio .....	9
1.3. Apresentando IDE.....	10
1.3.1 Criando o Primeiro Projeto.....	11
1.4. Emulador Android .....	14
Capítulo 2. Conceitos Fundamentais.....	17
2.1 Estruturas do Projeto .....	17
2.2. Activity.....	18
2.3. Ciclo de Vida.....	18
2.4. AndroidManifest.xml .....	21
2.5. Pasta de recursos .....	21
Capítulo 3. Componentes de Tela.....	23
3.1. TextView.....	23
3.2. EditText.....	24
3.3. Button e Listeners .....	24
3.4. Imageview.....	26
3.5. Alert Dialog .....	27
Capítulo 4. Gerenciadores de Layout.....	29
4.1. ViewGroup .....	29

4.2. LinearLayout .....	30
4.2.1 Controles de alinhamento layout_gravity .....	31
4.3 TableLayout .....	33
4.4. RelativeLayout .....	34
4.5. Constraint Layout .....	36
4.6. Outros layouts .....	37
 Capítulo 5. Toolbar e AppCompatActivity v21 .....	39
5.1. O que há de novo na AppCompatActivity? .....	39
5.2. Temas .....	41
5.3. Widget tinting .....	42
5.4 Toolbar Widget .....	43
5.4.1. Modo Action Bar .....	44
5.4.2. Modo Standalone .....	45
5.5. Aplicando estilos .....	46
5.6. SearchView Widget .....	47
 Capítulo 6. RecyclerView e Cardview .....	49
6.1. RecyclerView: Introdução .....	49
6.2. CardView .....	51
6.3. RecyclerView: Adapter .....	53
 Capítulo 7. Layouts Adaptáveis e Fragments .....	58
7.1. O que são layouts single-panel e multi-panel? .....	58
7.2 O que são Fragments? .....	58
7.2.1. Ciclo de vida dos fragments .....	59
7.2.2. Fragmentos e Contexto de acesso .....	61

7.2.3. Vantagens de usar fragmentos .....	61
7.2.4. Como suportar diferentes tamanhos de tela com fragments.....	63
7.3 Definindo e usando fragmentos .....	63
7.3.1 Definindo fragmentos .....	63
7.3.2. Aplicando comunicação com os fragmentos.....	64
7.3.3. Adicionando fragmentos estáticamente ao arquivo de layout .....	66
7.3.4. Tratando dinamicidade em fragmentos.....	67
7.3.5. Verificar se um fragmento está presente no layout .....	69
7.4. Determinar se uma activity está em modo single/dual/multi .....	70
7.5. Adicionando transações de fragmento à backstack .....	71
7.6. Animações e transações para fragmentos.....	71
7.7. Persistindo dados em fragments.....	71
7.7.1. Persistindo dados entre restarts da aplicação .....	71
7.7.2. Persistindo dados entre trocas de configuração .....	72
7.8. Fragmentos para processamento em background.....	72
7.8.1. Fragmentos Headless .....	72
7.8.2. Retenção de fragmentos headless para lidar com as mudanças de configuração.....	73
Capítulo 8. Intents e Intent Filters.....	74
8.1. Tipos de Intents.....	74
8.2. Criando uma Intent .....	75
8.3. Exemplo de Intent Explícita.....	76
8.4. Exemplo de uma intent implícita .....	76
Capítulo 9. Sistema de Arquivos .....	79
9.1. Escolher entre armazenamento interno ou externo .....	79

9.2. Obter permissões para memória externa.....	80
9.3. Salvar um arquivo na memória interna .....	81
9.4. Salvar arquivos na memória externa .....	83
9.5. Verificando espaço disponível .....	87
9.6. Apagando um arquivo .....	88
 Capítulo 10. Bases de Dados SQLite .....	89
10.1. Definindo um Schema e um Contract .....	89
10.2. Criando uma base de dados usando SQL Helper.....	90
10.3. Inserir informações em uma base de dados .....	93
10.4. Ler informações de uma base de dados .....	94
10.5. Apagar informações da base de dados.....	96
10.6. Atualizar informações na base de dados .....	97
 Capítulo 11. Integração com Serviço Rest .....	98
11.1. JSON.....	98
11.2. Gson.....	99
11.3. Requisições HTTP .....	100
11.4. AsyncTask .....	103
 Capítulo 12. BroadcastReceiver e Service .....	107
12.1. Broadcast Receiver.....	107
12.2 Ciclo de Vida do BroadcastReceiver.....	109
12.3. Service.....	109
12.4. Intent Service .....	112
 Capítulo 13. Google Maps e GPS .....	114

13.1. Configurando o AndroidManifest.xml .....	115
13.2. Obtendo a chave dos mapas no Google APIs Console .....	116
13.3. Classe MapFragment.....	117
13.4. Classe GoogleMap.....	117
13.5. Tipo de Mapa .....	118
13.6 Localização do mapa .....	119
13.7. Marcadores .....	119
13.8. Polyline - Desenha uma linha no mapa.....	120
13.9. GPS.....	121
 Capítulo 14. Notificações.....	 124
14.1. Firebase Cloud Messaging .....	124
14.2. Criando um Firebase Project .....	125
14.3. FCM Client.....	125
14.4. Notifications .....	127
 Capítulo 15. Publicação na Google Play .....	 130
15.1. Preparação da Aplicação .....	130
15.2. Assinando a Aplicação.....	131
15.3. Conhecendo o Google Play Developer Console.....	133
15.3.1. All Applications.....	133
15.3.2. Multiple User Accounts .....	134
15.3.3. Store Listing Details .....	135
15.3.4 Upload and Instantly Publish.....	137
 Capítulo 16. Kotlin.....	 138
16.1. Conhecendo a nova linguagem oficial .....	138

16.2. Kotlin ou Java .....	139
16.3. Principais pontos de melhoria .....	140
16.5. Presente ou futuro?.....	144
Referências.....	145



## Capítulo 1. Android

---

### 1.1. Introdução

---

Android foi criada especialmente para dispositivos móveis. É uma plataforma composta de um sistema operacional, *middlewares* e um conjunto de aplicativos como contato, *browser*, telefone, e entre outros. A plataforma Android hoje não está mais limitada aos smartphones e tablets, também dá suporte a TVs e wearables (relógios inteligentes).

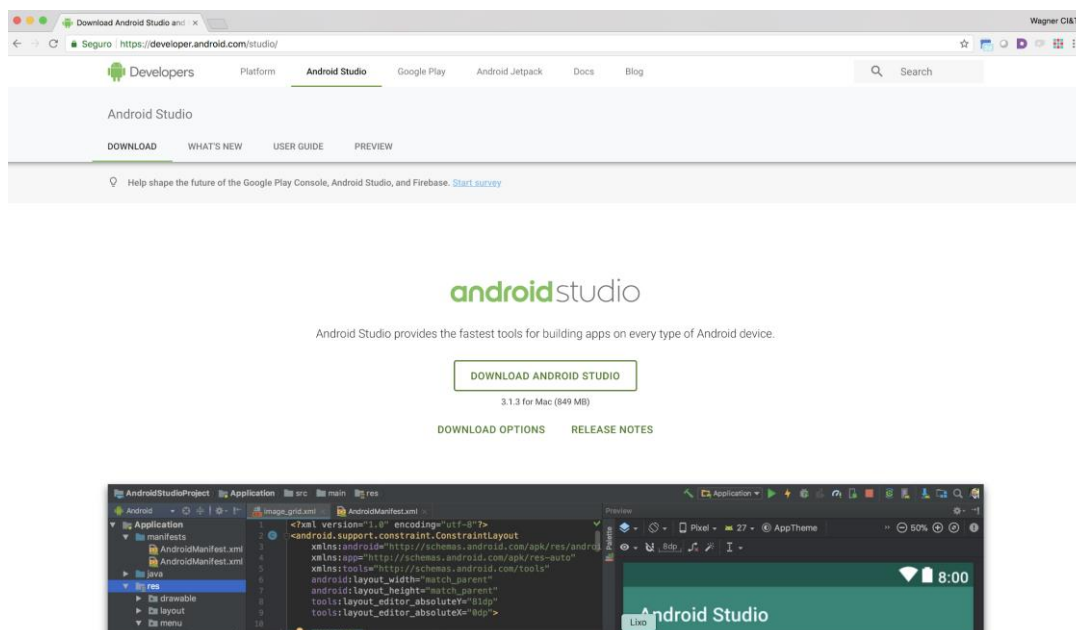
O Android teve seu desenvolvimento no início de 2003 pela Android Inc., e em 2005 foi adquirida pelo, Google que lidera o desenvolvimento até hoje. Em 2007 foi criada a Open Handset Alliance (<http://www.openhandsetalliance.com/>), que é uma associação de empresas de software, hardware e telecomunicações, cuja a missão é desenvolver uma plataforma para dispositivos móveis que seja completa, aberta e gratuita. Também em 2007 foi lançado a versão beta do primeiro SDK para Android, que hoje se encontra na versão 8.0 de codinome *Oreo*.

### 1.2. Instalação Android Studio

---

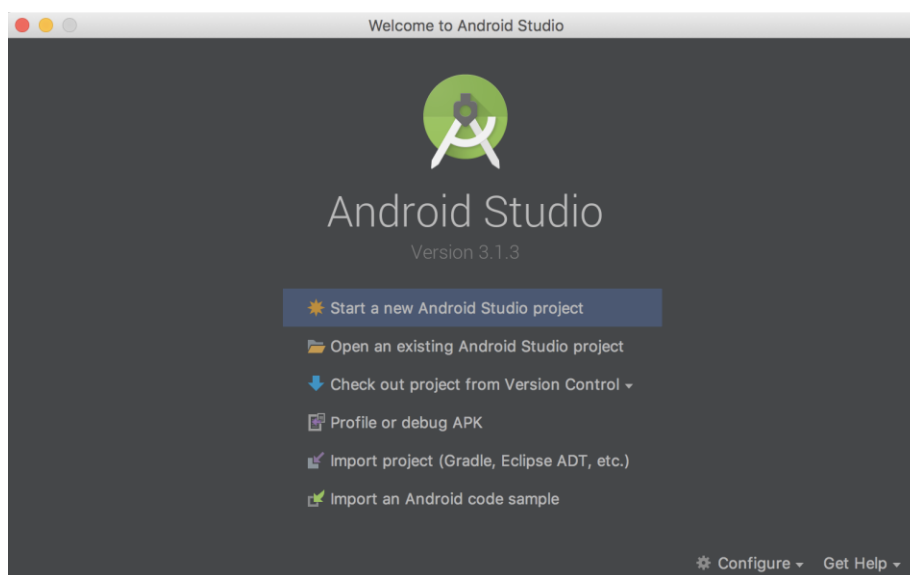
Em 2013 o Google anunciou o Android Studio sua IDE oficial para desenvolvimento Android baseada no IntelliJ IDEA. Para configurar nosso ambiente de desenvolvimento, vamos precisar, além do Android Studio, de instalar o Android SDK, o interessante é que o instalador do Android Studio já instala toda as ferramentas necessárias e pode ser baixado em <https://developer.android.com/studio/>.

É importante que o Java Development Kit (JDK) já esteja instalando, caso ainda não tenha, siga as instruções de instalação em <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.



### 1.3. Apresentando IDE

No momento da criação desta apostila, a versão atual do Android Studio é 3.1.3, e após ter instalado o Android Studio a interface apresentada será conforme a figura abaixo:



O Android Studio trabalha em conjunto com o Gradle, ferramenta utilizada para automatizar a construção, gestão de dependências, testes e publicação/deploy

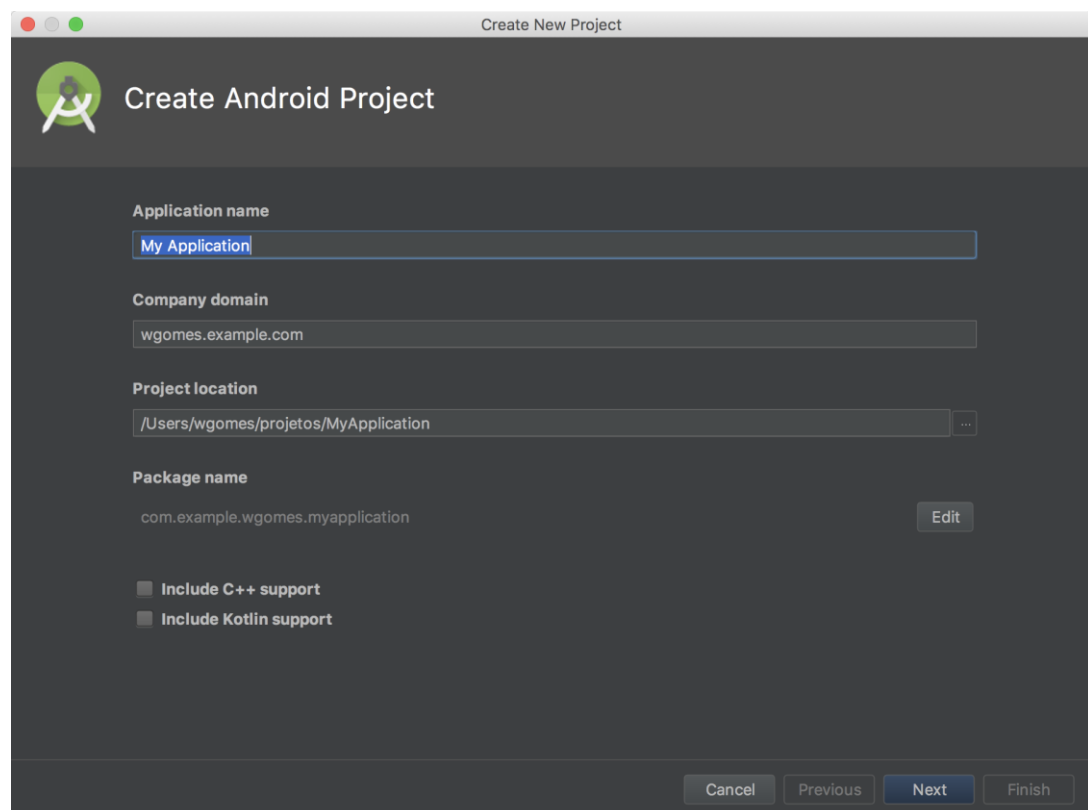
de aplicações. Além do Gradle, o Android Studio possui outras características interessantes:

- Suporte à ferramenta de análise estática de código, Lint.
- Assistente baseado em templates para fácil criação de modelos e componentes.
- Editor de layout para componentes UI (User Interface).

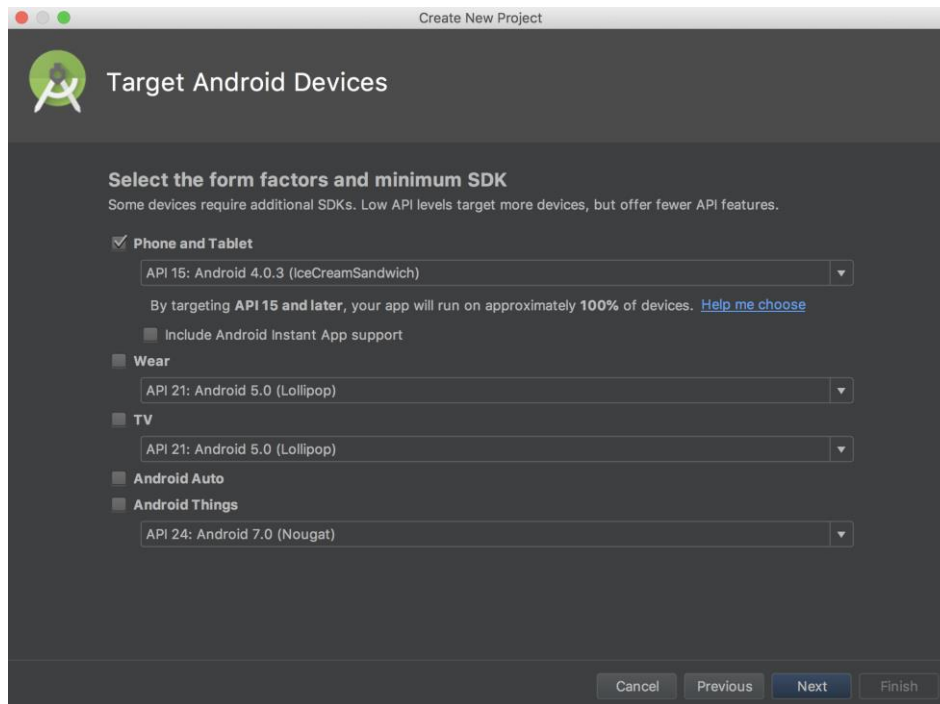
### 1.3.1 Criando o Primeiro Projeto

---

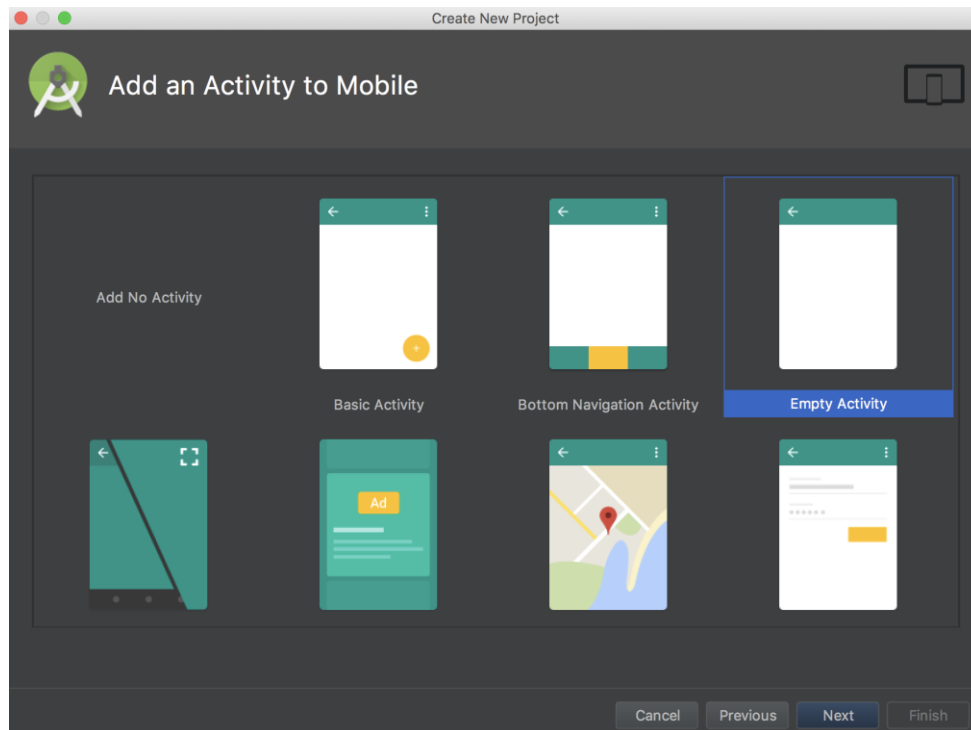
Para criar um novo projeto, clique em **Start a new Android Studio project**. Em seguida, preencha o formulário com o nome da aplicação e domínio e clique em Next:



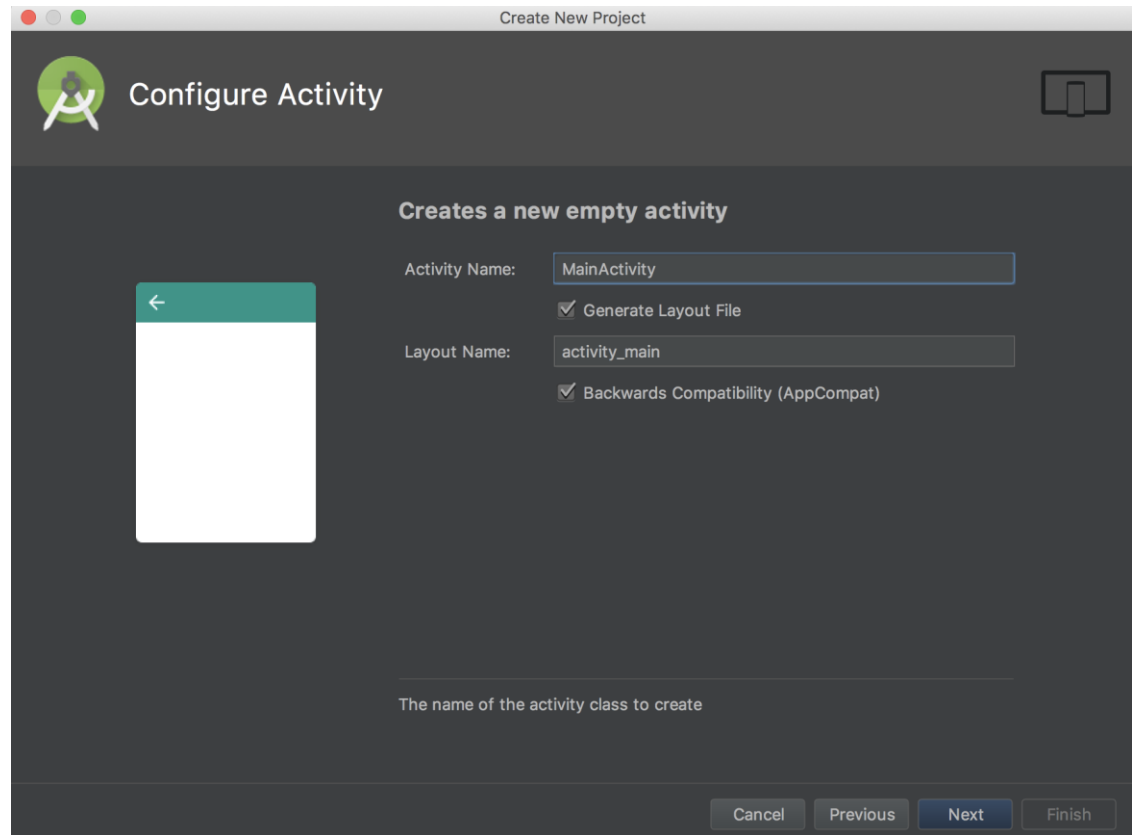
Em seguida vamos escolher para qual plataforma vamos trabalhar e a versão mínima do SDK:



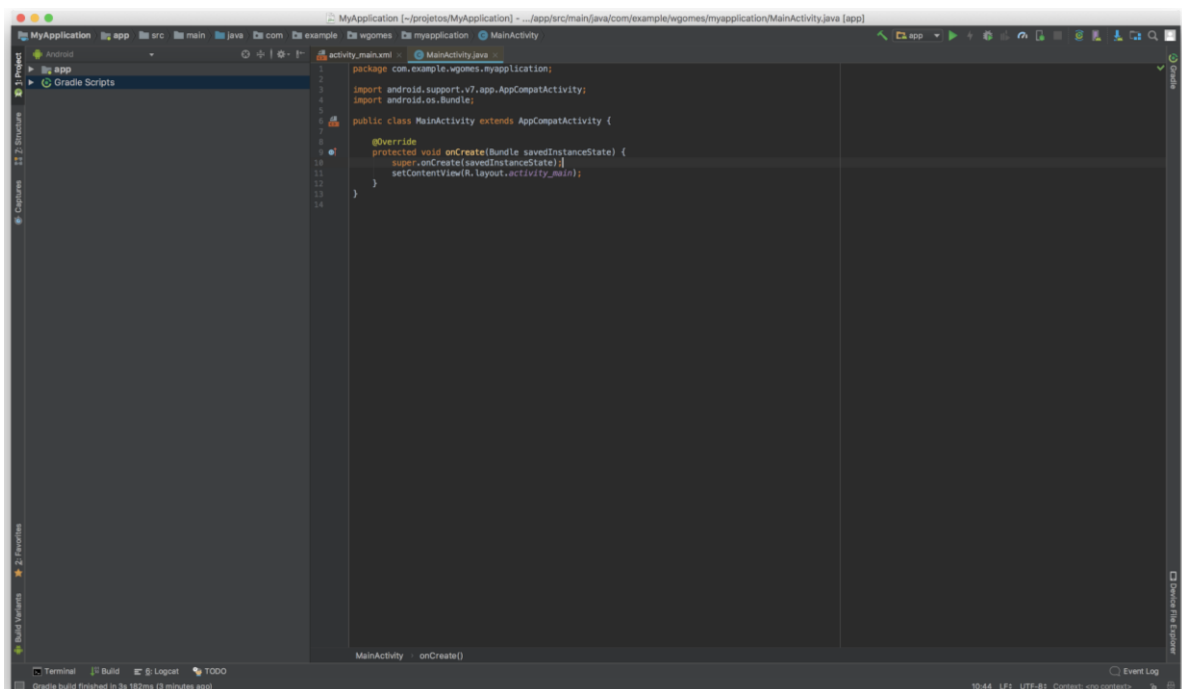
Na próxima tela podemos escolher um template para iniciar o nosso projeto; vamos escolher a Empty Activity, que já cria uma aplicação básica:



Agora no final você pode definir o nome da sua Activity principal:

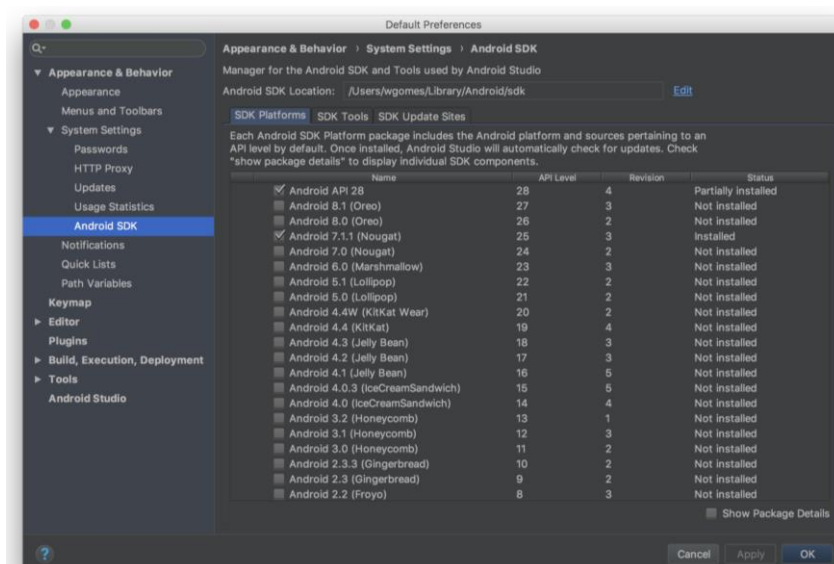


Depois do projeto criado, veja como é o ambiente de trabalho:



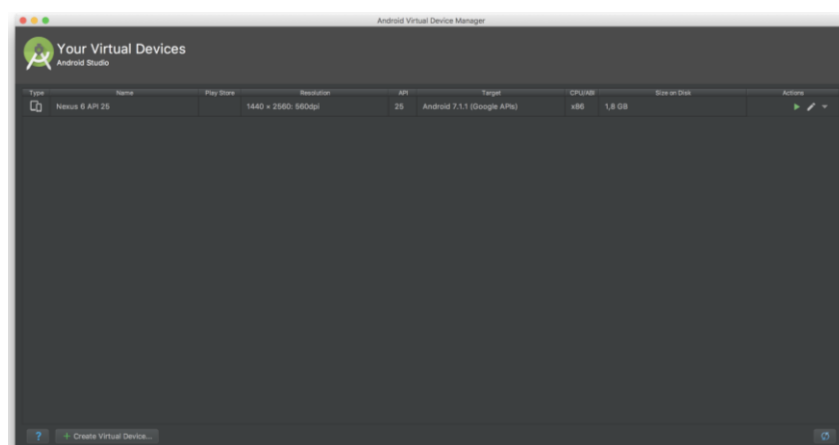
## 1.4. Emulador Android

Com o Android SDK você pode emular todas as versões disponíveis do Android e algumas versões de hardware. Mas antes de criarmos um Emulador, vamos conhecer o Android SDK Manager.

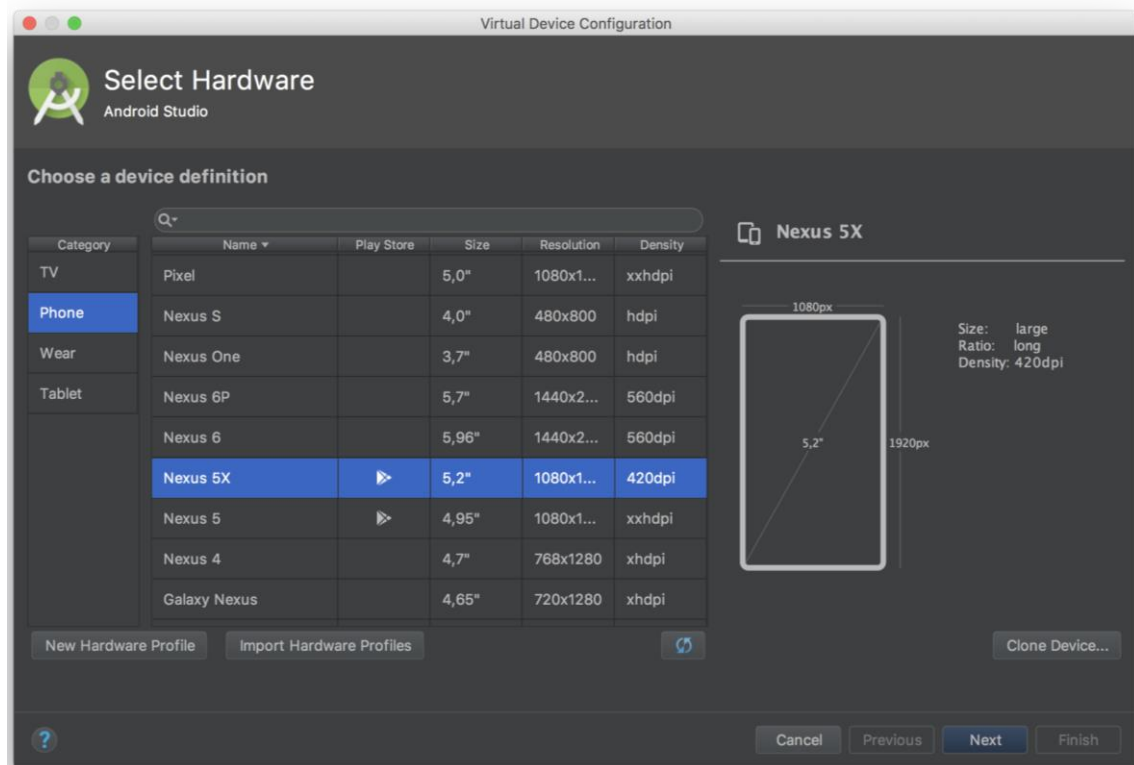


O Android SDK Manager é responsável por gerenciar o download das versões de Android, para criar os emuladores e ainda baixar as ferramentas de build e bibliotecas de suporte.

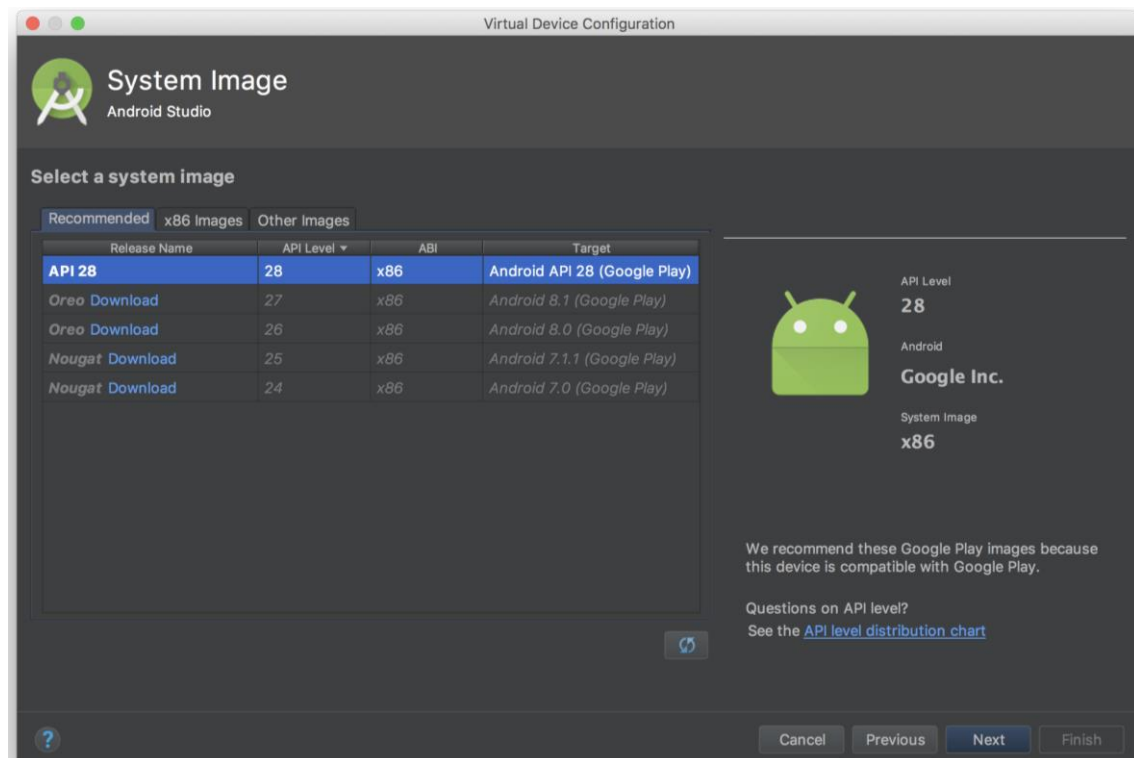
Agora vamos conhecer o AVD Manager responsável por criar os emuladores. Clique em Create Virtual Device:



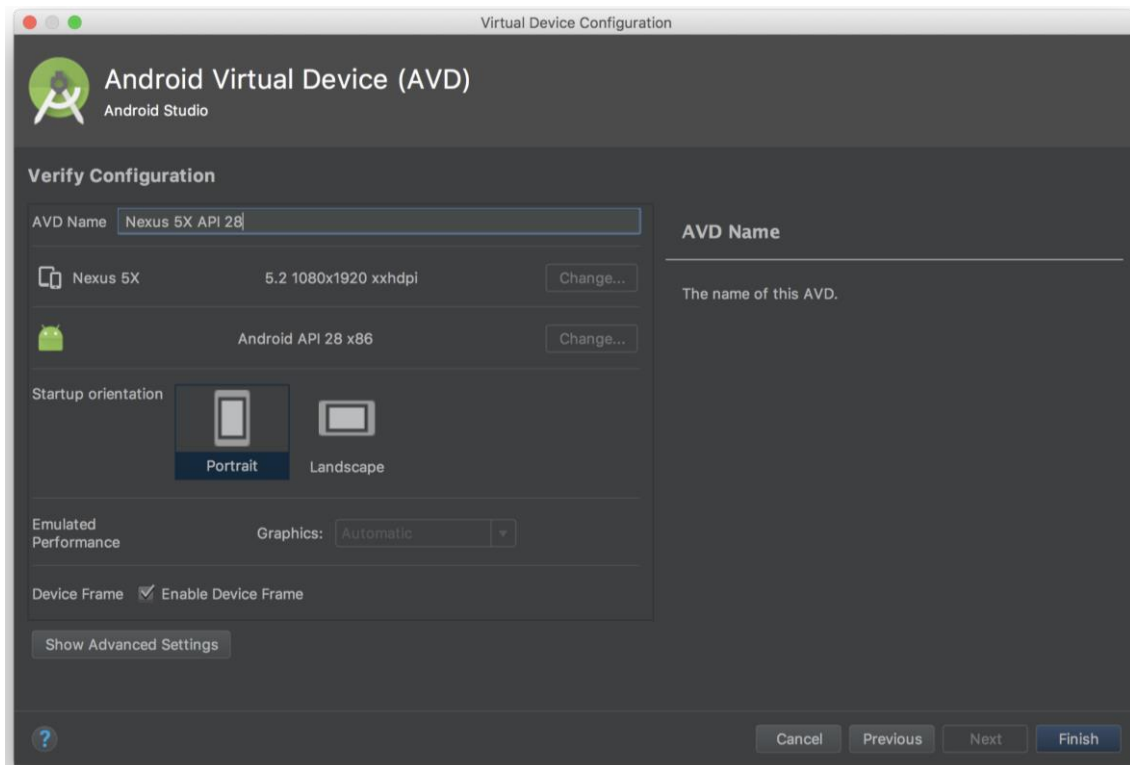
Escolha um modelo, ou crie um novo Hardware, e aperte em Next:



Agora precisamos escolher qual versão do Android queremos emular:



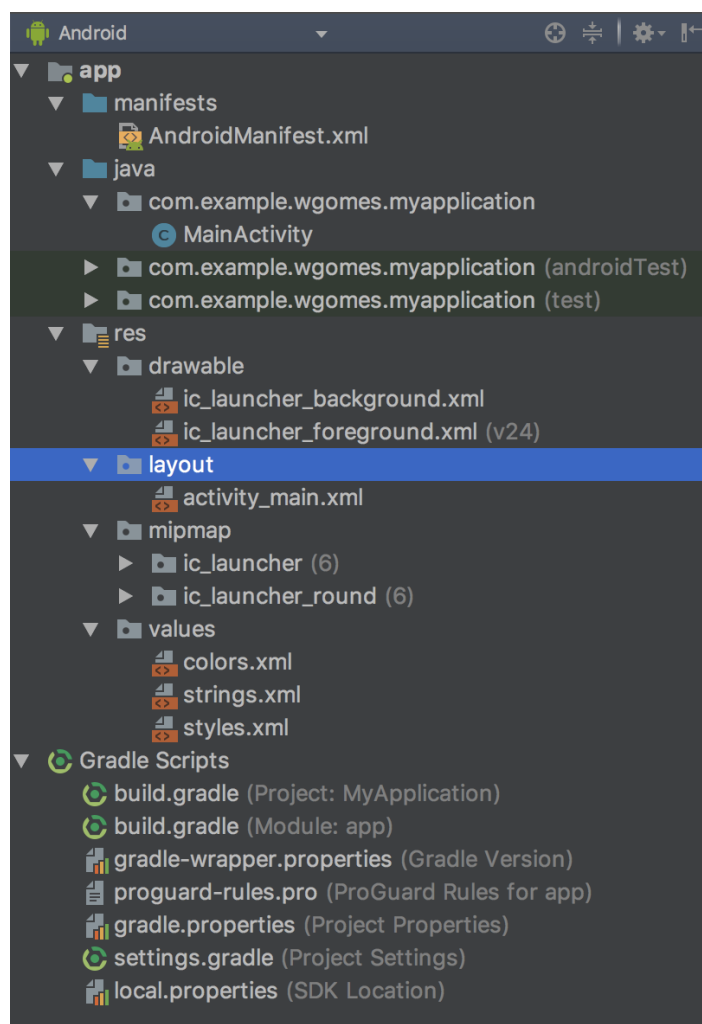
E por final, damos um nome para o nosso emulador e finalizamos:





## Capítulo 2. Conceitos Fundamentais

### 2.1 Estruturas do Projeto



1. **App:** diferente do eclipse o Android Studio trabalha com módulo, ou seja sua aplicação pode ter vários módulos. Por exemplo uma biblioteca que você está criando, ela poderia ter dois módulos: library e app.
2. **AndroidManifest:** é a base de uma aplicação Android. Ele é obrigatório e contém todas as configurações necessárias para executar a aplicação, como nome do pacote, declarações de cada *activity*, permissões, etc.
3. **Java:** diretório dedicado ao armazenamento dos código-fonte do projeto e será onde colocaremos as classes java.

4. **Res:** dedicado ao armazenamento de recurso (arquivos de layout, imagens, animações e xml contendo valores como strings, arrays, etc.), acessíveis através da classe R.
5. **Gradle:** novo sistema de build, possui os arquivos de configuração para geração de builds. Para saber mais consulte: <http://developer.android.com/sdk/installing/studio-build.html>.

## 2.2. Activity

---

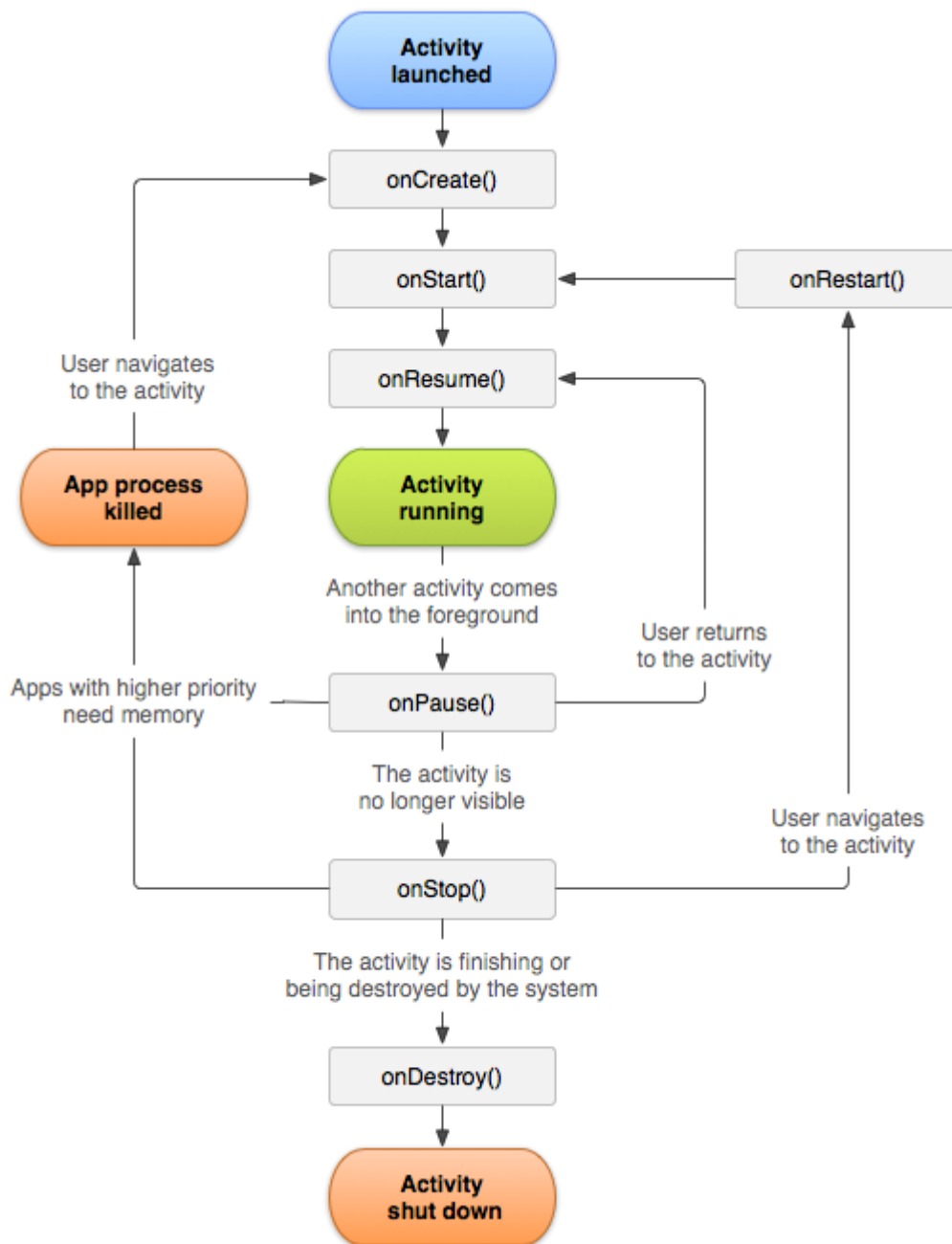
Activity representa uma tela com interface gráfica capaz de promover algum tipo de interação com o usuário. Uma aplicação Android pode ser composta de diversas activities que fornece um conjunto de funcionalidades para o usuário.

## 2.3. Ciclo de Vida

---

Activity é um componente de aplicação com um ciclo de vida, e quando o usuário acessa a aplicação, navega pelas telas, coloca a aplicação em segundo plano ou volta para o primeiro plano, as activities que a compõem passam por uma série de estados do ciclo de vida.

Entender como funciona é importante para preparar a aplicação para lidar com situações que podem interferir na sua execução de tarefas e interações do usuário. Abaixo se encontra um exemplo que ilustra o ciclo de vida da Activity.



Veja o que cada método representa:

**onCreate:** é chamado quando a activity é criada.

**onStart:** é chamado após o onCreate, e antes da activity se tornar visível para o usuário.

**onResume:** é chamado após o onStart, quando a activity se torna visível para o usuário.

**onPause:** é chamado após o onResume, quando a activity está para perder a visibilidade para outra activity.

**onStop:** a activity não está mais visível para o usuário.

**onDestroy:** a activity está prestes a ser destruída.

Agora veja a representação de uma classe que estende de Activity:

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
    }  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
    }  
  
    @Override  
    protected void onStop() {  
        super.onStop();  
    }  
  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
    }  
}
```

## 2.4. AndroidManifest.xml

*AndroidManifest.xml*, é o arquivo de configuração obrigatório para toda aplicação Android. Esse arquivo contém informações essenciais sobre a aplicação e sobre o que é necessário para executá-la. O nome do pacote principal escolhido durante a criação do projeto, por exemplo, é armazenado no manifesto.

O manifesto também descreve os componentes (activities, services, content providers e broadcast receivers) que fazem parte da aplicação, assim o sistema operacional é capaz de identificá-los e determinar quando serão executados.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.framework.helloworld" >

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

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

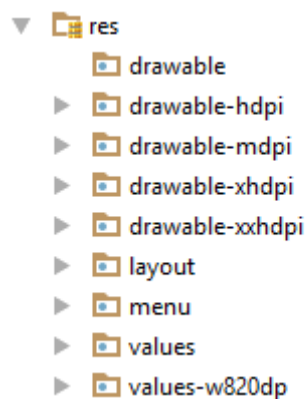
</manifest>
```

## 2.5. Pasta de recursos

A pasta de recurso conhecido como res é dedicado ao armazenamento de recursos (arquivos de layout, imagens, animações e xml contendo valores como strings, arrays e etc.), acessíveis através da classe R.

- *drawable* - e são destinadas a armazenar imagens que são usadas na aplicação, como ícones, por exemplo.
- *layout* - se destina a armazenar os arquivos XML que representam o layout das telas da aplicação.
- *Values* - também se destina a armazenar XMLs que serão usados na aplicação. Os XMLs podem conter arquivos de strings, tamanhos, cores, e etc.

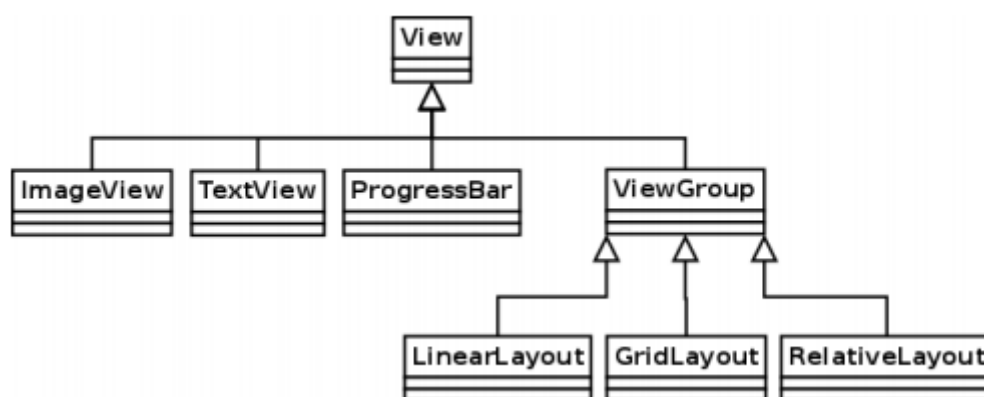
O interessante da plataforma Android é que ela foi criada para trabalhar com vários tamanhos de telas, idiomas e dispositivos. E para facilitar o trabalho dos desenvolvedores, o Google permite que coloquemos qualificadores nos nomes de diretórios do *res*. Então se precisamos trabalhar com várias resoluções de telas, podemos seguir o exemplo abaixo:



Neste exemplo, temos a pasta de *drawable* para cada resolução de tela que são qualificadas por MDPI, HDPI, XHDPI, XXHDPI e, além disso, temos também uma pasta *values* com o qualificado de w820dp que apresenta uma resolução de tablets.

## Capítulo 3. Componentes de Tela

O elemento fundamental de uma interface gráfica na plataforma Android é a View. A partir dela é que são derivados todos os demais elementos, como botões, imagens, checkboxes, campos para entrada e exibição de textos e também widgets mais complexos; como seletores de data, barras de progresso e de pesquisa, e até mesmo um widget para exibir páginas web: o WebView. A imagem abaixo mostra hierarquia de Views:



### 3.1. TextView

A primeira e mais simples das subclasses de View é o *android.widget.TextView*, que representa um label (Texto) na tela. Em Java pode-se comparar com o JLabel do Swing.

```

<TextView android:text="@string/hello_world" android:layout_width="wrap_content"
|   android:layout_height="wrap_content" />
    
```

### 3.2. EditText

A classe EditText é uma subclasse de TextView, que é utilizada para que os usuários possam digitar informações em um campo de texto.

Esta classe pode ser usada para a entrada de: texto normal, apenas números e formato de senha. A forma em que os atributos são utilizados é parecido com a da classe TextView. Vejam o exemplo a seguir:

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/hint_nome"
    android:inputType="textPersonName"/>
```

### 3.3. Button e Listeners

As classes android.widget.Button e android.Widget.ImageButton são utilizadas para criar um botão na tela. A classe Button permite criar botões para interação da aplicação.

A diferença principal entre estas duas classes são basicamente que a classe ImageButton permite que se use imagens para desenhar um botão, assim permitindo que se possa customizar o mesmo

```
<Button
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:text="OK"/>

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"/>
```

No Android, há mais de uma maneira para interceptar os eventos a partir da interação do usuário com sua aplicação. Ao considerar os eventos dentro de sua



interface com o usuário, a abordagem é para capturar os eventos que o usuário interage com algum objeto específico. A classe View fornece os meios para fazê-lo.

Dentro das diferentes classes de View que você vai usar para compor o seu layout, você pode observar vários métodos de retorno de chamada pública que parecem úteis para eventos UI. Esses métodos são chamados pelo framework Android quando a ação ocorre respectivamente neste objeto. Por exemplo, quando um View (no nosso caso um botão) é tocado, o método `onTouchEvent()` é chamado no objeto. No entanto, a fim de interceptar isso, você deve estender a classe e substituir o método. No entanto, estendendo-se cada objeto View a fim de lidar com um evento como esse não seria prático. É por isso que a classe View também contém uma coleção de interfaces aninhadas com callbacks que você pode facilmente definir. Estas interfaces, chamada ouvintes de eventos (listeners), são formas para capturar a interação do usuário com a interface.

Enquanto você vai mais comumente usar estes listeners para realizar a interação com o usuário, pode chegar um momento em que você tenha que estender uma classe View, a fim de construir um componente personalizado. Talvez você queira estender a classe Button para fazer algo em particular. Neste caso, você será capaz de definir o comportamento padrão de evento para sua classe usando os manipuladores de eventos de classe.

São estes os listeners utilizados para o retorno de uma chamada:

**onClick():** a partir do método [View.OnClickListener](#). É chamado quando o usuário toca o item (quando em modo de tocar), ou incide sobre o item com a navigation-keys ou trackball e pressiona a tecla "enter", ou pressiona para baixo no trackball.

**onLongClick():** a partir do método [View.OnLongClickListener](#). É chamado quando o usuário toca e detém o item (quando no modo de toque), ou incide sobre o item com a navigation-keys ou trackball e pressionar a tecla "enter", ou pressiona e mantém pressionada a trackball (por um segundo).

**onFocusChange():** a partir do método [View.OnFocusChangeListener](#). É chamado quando o usuário navega para longe do item, usando a navigation-keys ou trackball.

**onKey():** a partir do método [View.OnKeyListener](#). É chamado quando o usuário está com foco no item e pressiona ou solta uma tecla no dispositivo.

**onTouch():** a partir do método [View.OnTouchListener](#). É chamado quando o usuário executa uma ação qualificada como um evento de toque, inclusive se pressionar ou qualquer gesto de movimento na tela (dentro dos limites do item).

**onCreateContextMenu():** a partir do método [View.OnCreateContextMenuListener](#). É chamado quando um Context Menu está sendo construído (como o resultado de um "clique longo").

### 3.4. Imageview

---

Exibe uma imagem arbitrária, como um ícone. A classe `ImageView` pode carregar imagens de várias fontes (tais como recursos ou fornecedores de conteúdos), cuida de sua medição de computação a partir da imagem de modo que possa ser usado em qualquer gerenciador de layout, e oferece várias opções de tela como o dimensionamento e coloração. A classe `android.widget.ImageView` é bastante utilizada em efeitos visuais, construção de movimentos e troca de imagens de forma dinâmica.

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

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="fill_parent" android:layout_height="fill_parent"
```

```
        android:orientation="vertical">
```

```
        <ImageView android:src = "@drawable/feliz"
```

```
android:layout_width="wrap_content"  
  
android:layout_height="wrap_content" />  
  
</LinearLayout>
```

No atributo “**android.src**” é definido o caminho da imagem a ser exibida.

### 3.5. Alert Dialog

---

Uma caixa de diálogo (Modal Dialog) é geralmente uma pequena janela que aparece na frente da atividade atual. Modal Dialogs são normalmente utilizados para notificações que devem interromper o usuário e para executar tarefas curtas que se relacionam diretamente com a aplicação em curso. No nosso caso, vamos falar do Alert Dialog, que é basicamente uma mensagem de alerta que aparece quando se deseja informar algo para o usuário.

Código java correspondente:

```

public class Exemplo extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button b = (Button) findViewById(R.id.btnAlert);
        b.setOnClickListener(
            new OnClickListener() {
                @Override
                public void onClick(View v) {
                    AlertDialog.Builder alerta = new
AlertDialog.Builder(Exemplo.this);
                    alerta.setIcon(R.drawable.feliz);
                    alerta.setTitle("Você está na
Framework!");
                    alerta.setMessage("Parabéns, você
está na Framework, Você está feliz?");
                    alerta.setPositiveButton("Sim", new
DialogInterface.OnClickListener() {
                        @Override
                        public void
onClick(DialogInterface dialog, int which) {
                            Toast.makeText(Exemplo.this, "você clicou em Sim",
Toast.LENGTH_SHORT).show();
                        }
                    });
                    alerta.setNegativeButton("Não", new
DialogInterface.OnClickListener() {
                        @Override
                        public void
onClick(DialogInterface dialog, int which) {
                            Toast.makeText(Exemplo.this, "você clicou em Não",
Toast.LENGTH_SHORT).show();
                        }
                    });
                    alerta.show();
                }
            }
        );
    }
}

```

## Capítulo 4. Gerenciadores de Layout

---

Quando desenvolvemos para o Android, temos que ficar atentos ao conceito de Gerenciadores de Layouts. Este conceito é extremamente similar ao conceito de layouts do Java Swing. Quem já desenvolveu aplicações para Java Swing, não terá muitas dificuldades para criar seus layouts para o Android.

A classe `android.view.View` pode ser considerada a classe pai das demais classes de layouts e componentes visuais do Android. No Android, temos dois tipos de componentes gráficos: os widgets e os gerenciadores de layout.

Um widget irá herdar diretamente de `android.view.View` enquanto os gerenciadores de layout irão herdar de `android.view.ViewGroup`.

### 4.1. ViewGroup

---

Para organizarmos os componentes gráficos nas telas (Activities), teremos que utilizar os gerenciadores de layouts. Esta organização acontecerá de forma automática de acordo com a regra implementada pelo gerenciador de layout utilizado.

Esta abordagem é muito importante pelo simples fato de que existem inúmeros dispositivos móveis e, com isto, inúmeros displays diferentes. Desta forma é necessário que os layouts se ajustem corretamente a esta infinidade de variedades de tamanhos de telas.

Os principais gerenciadores de layouts disponíveis para o Android, são:

- **LinearLayout:** organiza os componentes na horizontal ou vertical.
- **TableLayout:** organiza os componentes em tabelas. Este layout é muito similar ao `table` do HTML.
- **FrameLayout:** layout mais simples de se utilizar. Este layout faz com que o componente ocupe a tela toda.

- **RelativeLayout:** organiza os componentes um em relação aos outros.

## 4.2. LinearLayout

---

O LinearLayout é o componente que organiza seus componentes filhos em uma única coluna ou uma única linha. A classe `android.widget.LinearLayout` é um dos gerenciadores de layout mais utilizados, sendo possível organizar uma sequência de componentes na horizontal (padrão) ou na vertical.

Pode-se observar que toda a classe de layout possui os atributos `android:layout_width` e `android:layout_height`, que são propriamente definidos na classe mãe `ViewGroup`. Na classe `LinearLayout` é possível configurar parâmetros para controlar componentes a serem exibidos verticalmente e horizontalmente. Para configurar a orientação, pode-se fazê-lo pelo atributo `android:orientation`. Você também pode especificar a `android:gravity` do componente, que define o alinhamento de todos os elementos filhos.

Quando definimos um layout, temos que tomar cuidado com o tamanho da tela, pois se o mesmo ocorrer o componente inserido será automaticamente cortado, assim, mostrando uma parte ou não exibindo o mesmo.

Exemplo:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent">

    <ImageView

        android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

```
android:src="@drawable/icon"/>
```

```
<ImageView
```

```
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

```
    android:src="@drawable/feliz" />
```

```
</LinearLayout>
```

Por padrão, a orientação é horizontal, mas se desejamos que os componentes fiquem de forma vertical, é só inserir o atributo `android:orientation="vertical"` na tag `LinearLayout` para que seja mudado a visualização, veja como fica com orientação vertical:

#### 4.2.1. Controles de alinhamento `layout_gravity`

---

Para definir o alinhamento dos componentes dentro de um layout, podemos utilizar o `layout_gravity`, onde podemos definir através da tag `android:layout_gravity`. Os valores válidos para este atributo são: `top` (para cima), `bottom` (para baixo), `left` (para a esquerda), `right` (para a direita), `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center` e `fill`.

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

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent"

    android:orientation="vertical">

    <ImageView

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:src="@drawable/icon" android:layout_gravity="left"/>

    <ImageView

        android:layout_width="wrap_content"

        android:layout_height="wrap_content" android:src="@drawable/feliz"

        android:layout_gravity="center"/>

    <ImageView android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:src="@drawable/icon"

        android:layout_gravity="right"/>

</LinearLayout>
```



### 4.3. TableLayout

A classe `android.widget.TableLayout` é um layout que organiza seus filhos em linhas e colunas. A classe `TableLayout` consiste em um número de objetos `TableRow`, cada um definindo uma linha (na verdade, você pode ter outros filhos também). Cada linha tem zero ou mais células, cada célula pode conter um objeto `View`. Esta classe é muito utilizada para a criação de algumas telas, como formulários. Quando utilizamos o `TableLayout`, cada linha da tabela é formada por um `android.widget.TableRow`, que é uma subclasse de `LinearLayout`, e assim pode conter outros componentes que representarão colunas na tabela.

O `TableLayout` possui dois atributos importantes, são eles:

- **android:stretchColumns:** faz com que as colunas especificadas ocupem o espaço disponível na tela, expandindo-as. Use esse atributo quando é necessário que uma coluna ocupe a linha inteira. Ele funciona basicamente como um "colspan" em uma página HTML.
- **android:shrinkColumns:** faz com que as colunas especificadas sejam sempre exibidas na tela. Caso o valor do texto seja muito grande e fique fora da tela, a linha é quebrada e o texto é exibido em várias linhas na mesma coluna.

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

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="fill_parent" android:layout_height="fill_parent"
```

```
        android:shrinkColumns="2">
```

```
<TableRow>
```

```
    <TextView android:text="Coluna 1" />
```

```
    <TextView android:text="Coluna 2" />
```

```
</TableRow>

<TableRow>

    <TextView android:text="Coluna 1" />

    <TextView android:text="Coluna 2" />

</TableRow>

<TableRow>

    <TextView android:text="Coluna 1" />

    <TextView android:text="Coluna 2" />

    </TableRow>

<TableRow>

    <TextView android:text="Coluna 1" />

    <TextView android:text="Coluna 2" />

    <TextView android:text="Coluna 3" />

</TableRow>
</TableLayout>
```

#### 4.4. RelativeLayout

---

O `android.widget.RelativeLayout` é um `ViewGroup` que exibe os componentes filhos em posições relativas. A posição pode ser especificada em relação aos elementos irmãos como à “esquerda de” ou “abaixo de” um determinado elemento), ou em posições relativas à área do `RelativeLayout` (como alinhados ao

fundo ou à esquerda do centro). Um RelativeLayout é um utilitário muito poderoso para a concepção de uma interface de usuário, porquê pode-se eliminar ViewGroups aninhadas. Se você se encontrar com vários grupos LinearLayout aninhados, você pode ser capaz de substituí-los com um único RelativeLayout.

<RelativeLayout ...>

<TextView

```
        android:id="@+id/novo_gasto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dp"
        android:layout_marginTop="80dp"
        android:clickable="true"
        android:drawableTop="@drawable/novo_gasto"
        android:onClick="selecionarOpcao"
        android:text="@string/novo_gasto"
        android:textColor="#FFFFFF"
        android:textStyle="bold" />
```

<TextView

```
        android:id="@+id/nova_viagem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:layout_alignParentRight="true"
        android:layout_alignTop="@id/novo_gasto"
        android:layout_marginRight="50dp"
        android:clickable="true"
        android:drawableTop="@drawable/nova_viagem"
        android:onClick="selecionarOpcao"
```

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

```
android:textColor="#FFFFFF"
```

```
android:textStyle="bold"
```

```
/>
```

```
</RelativeLayout>
```

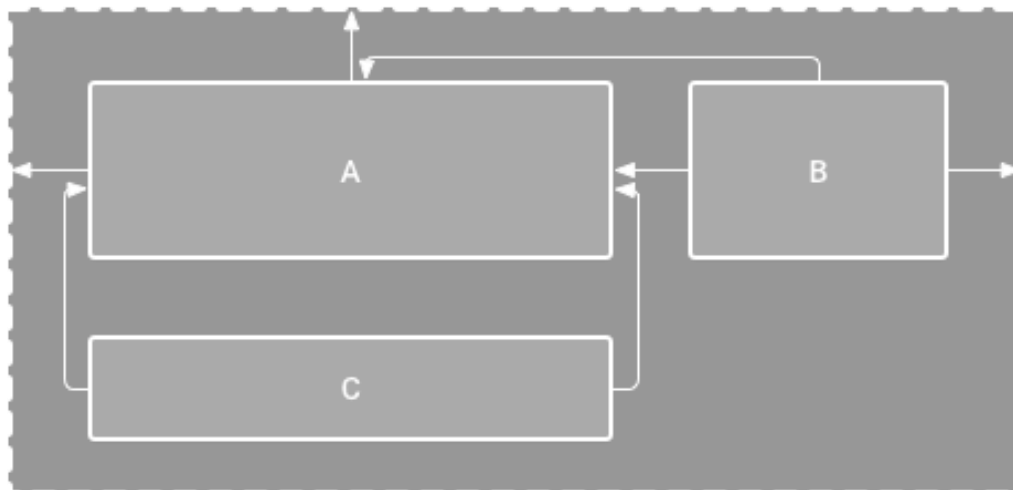
## 4.5. Constraint Layout

Constraints significa algo como “restrições” ou “limitações”, e são essas restrições que são o cerne por trás do funcionamento deste layout manager, sendo fundamentais para sua utilização entender elas primeiro.

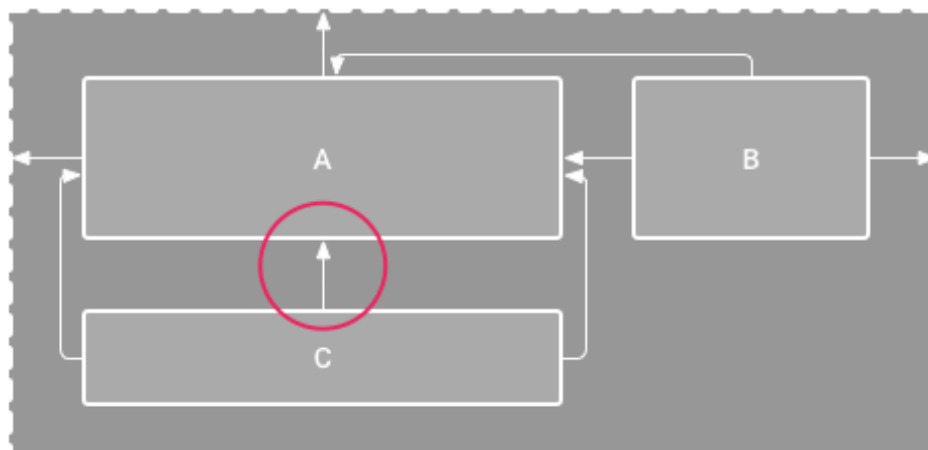
Para definir a posição de uma view no ConstraintLayout, você deve adicionar ao menos uma constraint horizontal e uma vertical para a view. Cada constraint representa uma conexão ou alinhamento em relação à outra view, o layout parent ou mesmo uma linha-guia invisível (??). Cada constraint define a posição da view a partir de seus eixos vertical e horizontal; motivo pelo qual temos de definir no mínimo essas duas constraints, embora seguidamente precisaremos de mais de duas para conseguir os comportamentos desejados.

Quando você arrasta e solta uma view no Layout Editor, ela fica exatamente onde você deixar ela, mesmo que não possua constraint alguma. No entanto, isso é apenas para tornar o seu trabalho mais fácil quando estiver posicionando os elementos; se uma view não possui constraints, ela ficará no canto superior esquerdo da tela automaticamente (0,0).

Na figura abaixo, o layout parece ok no editor, mas não há constraint vertical na view C. Quando este layout renderizar em um dispositivo, a view C se alinhará horizontalmente com as faces esquerda e direita da view A, mas irá aparecer no topo da tela porque não há constraint vertical.



O correto seria adicionar uma constraint vertical entre as views A e C, como mostra a figura abaixo:



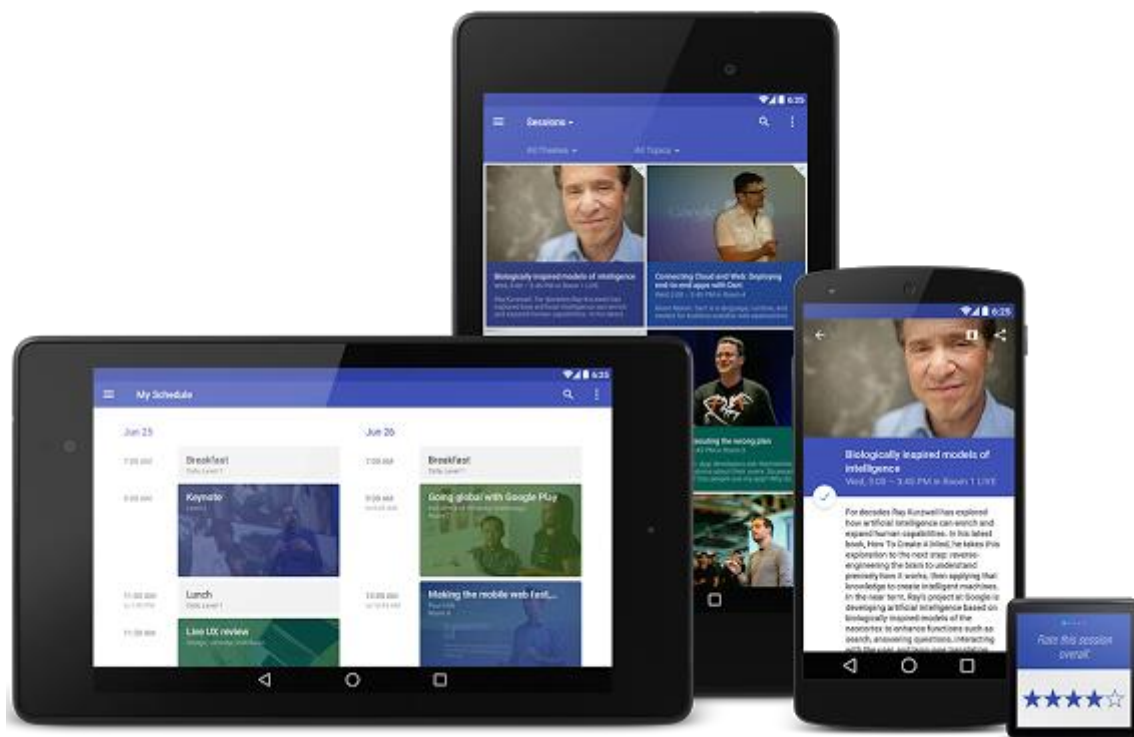
Embora a falta de uma constraint não cause um erro de compilação, o Layout Editor indicará a falta de constraints como um erro na toolbar. Para ajudar você a não esquecer de constraints, o Layout Editor pode automaticamente adicionar as constraints para você com os recursos Autoconnect e “infer constraints”, que são novidades na ferramenta.

#### 4.6. Outros layouts

Além dos layouts apresentados acima, existem outros tipos, no qual a semântica é parecida, mas possui suas peculiaridades. Entre eles podemos citar:

- **FrameLayout:** a classe `android.widget.FrameLayout` é a mais simples de todos os gerenciadores de layout do Android, é utilizada quando se deseja que a tela tenha apenas um componente que possa preencher a tela inteira, por exemplo, uma imagem de fundo, assim podemos definir que o `FrameLayout` é projetado para bloquear uma área na tela, para mostrar um único item. Geralmente, `FrameLayout` deve ser usado para manter uma visão única do componente filho, porque pode ser difícil de organizar visões de componentes de uma forma que é escalável para diferentes tamanhos de tela sem que os componentes filhos se sobreponham uns aos outros. Você pode, entretanto, adicionar vários filhos para um `FrameLayout` e controlar a posição de cada um dentro da `FrameLayout` atribuindo, a cada filho, usando o atributo `android:layout_gravity`.
- **ScrollView:** a classe `android.widget.ScrollView` deve ser utilizada para telas que contenha muitos elementos e nas quais seja necessário fazer a rolagem da tela, assim permitindo que ele possua uma exibição maior do que a exibição física. Um `ScrollView` é na verdade uma subclasse de `FrameLayout`, significando que ele suporta apenas um componente filho, que irá ocupar todo o tamanho da tela. Normalmente é adicionado outro layout dentro do `ScrollView`, como o `LinearLayout`, que permite receber outros componentes filhos para organizar na forma horizontal ou vertical. A classe `TextView` também cuida de sua própria rolagem, assim não necessita de um `ScrollView`, mas usando os dois juntos é possível obter o efeito de uma exibição de texto dentro de um recipiente maior. Vale lembrar que um `ScrollView` suporta apenas a rolagem vertical.
- **GridView:** um `android.widget.GridView` é um `ViewGroup` que exibe os itens em uma grade bidimensional, com rolagem. Os itens da grid são inseridos automaticamente no layout usando um `ListAdapter`.

## Capítulo 5. Toolbar e AppCompatActivity v21



A Google lançou ultimamente a SDK do Android 5.0 trazendo novos widgets e o tão comentado *Material Design*. Para que possamos trazer essas novas funcionalidades para as plataformas antigas do Android, a Google expandiu as bibliotecas de suporte, incluindo um grande update para **AppCompat**, bem como adição de novas bibliotecas como **RecyclerView**, **CardView** e **Palette**

### 5.1. O que há de novo na AppCompatActivity?

AppCompatActivity (ActionBar Compat) começou como um backport da ActionBar API do Android 4.0 para dispositivos android executando versões antigas do sistema. A biblioteca provê uma camada de API comum entre todos esses dispositivos. AppCompatActivity v21 entrega uma API e um conjunto de features que está totalmente atualizada com o novo Android 5.0.

Neste lançamento, o Android introduziu um novo widget chamado **Toolbar**. É uma generalização do padrão ActionBar que nos dá muito mais controle e flexibilidade. Toolbar é uma **view** na sua aplicação como qualquer outra, tornando fácil a interação dela com o resto das views, aplicação de animações e reação a eventos de scroll, por exemplo. Você também pode configurá-la como a ActionBar da sua activity, significando que suas opções padrão e ações de menu serão exibidas com ela. Os últimos updates da AppCompat já vêm sendo usados a algum tempo, incluindo a Google Play Store e a Google Banca.

**Configuração:** se você está usando Gradle, adicione appcompat como uma dependência em seu arquivo de build.gradle.

```
dependencies {  
  
    implementation 'com.android.support:appcompat-v7:28.0.+'  
  
}
```

### Nova Integração

Se você não está usando AppCompat ou está começando do zero, aqui está o que você deve fazer:

- Todas as suas Activities devem estender da **ActionBarActivity**, que estende da **FragmentActivity** da biblioteca de suporte v4, então você pode continuar a usar fragments.
- Todos os seus temas (que querem uma ActionBar/Toolbar) precisam herdar do **Theme.AppCompat**, há variantes disponíveis, incluindo **Light** e **NoActionBar**.
- Quando inflar qualquer coisa para ser exibida em uma actionBar (como um SpinnerAdapter para navegação em lista na toolbar), tenha certeza que você



está usando o contexto com temas aplicados da actionBar que pode ser obtido via **getSupportActionBar().getThemedContext()**.

- Você precisa usar os métodos estáticos em **MenuItemCompat** para qualquer ação relacionada às chamadas em um **MenuItem**.

## Migração de uma configuração já existente

Para a maioria dos apps, você precisará somente de uma declaração de tema em values:

values/themes.xml:

```
<style name="Theme.MyTheme" parent="Theme.AppCompat.Light">
<!-- Set AppCompat's actionBarStyle -->
<item name="actionBarStyle">@style/MyActionBarStyle</item>
<!-- Set AppCompat's color theming attrs -->
<item name="colorPrimary">@color/my_awesome_red</item>
<item name="colorPrimaryDark">@color/my_awesome_darker_red</item>
<!-- The rest of your attributes -->
</style>
```

Você agora pode remover todos os seus **values-v14+ Action Bar styles**.

## 5.2. Temas

AppCompat tem suporte aos atributos do novo **color palette theme**, que permite você customizar facilmente seu tema para estar de acordo com sua marca usando de cores primárias até acentuadas. Por exemplo:

values/themes.xml:

```

<style      name="Theme.MyTheme"      parent="Theme.AppCompat.Light">
<!--  colorPrimary  is  used  for  the  default  action  bar  background  -->
<item      name="colorPrimary">@color/my_awesome_color</item>

<!--  colorPrimaryDark  is  used  for  the  status  bar  -->
<item      name="colorPrimaryDark">@color/my_awesome_darker_color</item>

<!--  colorAccent  is  used  as  the  default  value  for  colorControlActivated,
      which  is  used  to  tint  widgets  -->
<item      name="colorAccent">@color/accent</item>

<!--  You  can  also  set  colorControlNormal,  colorControlActivated
      colorControlHighlight,  and  colorSwitchThumbNormal.  -->
</style>

```

Quando você configura estes atributos, a AppCompatActivity automaticamente propaga os seus valores para os atributos de sistema na API 21+. Isso colore automaticamente a barra de status.

Em plataformas antigas, AppCompatActivity emula o tema de cores onde é possível; no momento é limitado a colorir a Action Bar e alguns widgets

### 5.3. Widget tinting

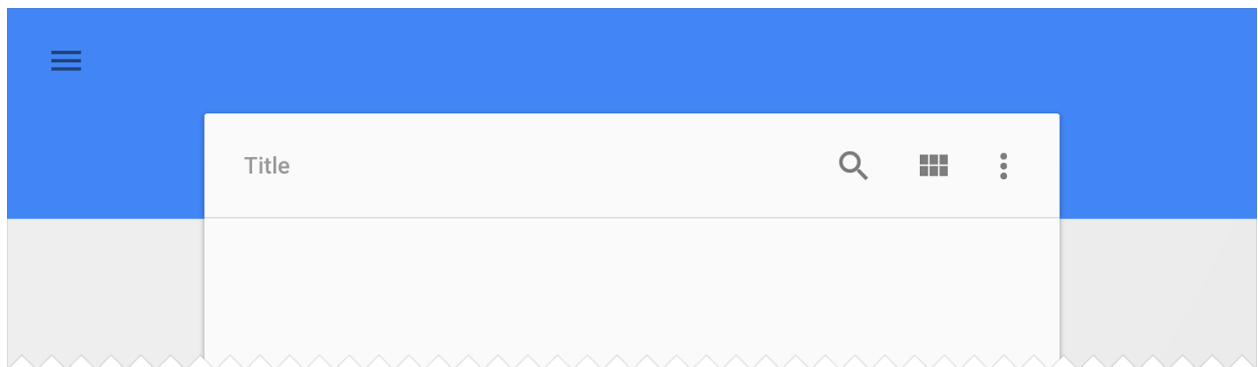
Quando seu app está executando em dispositivos com Android 5.0, todos os widgets são coloridos usando os atributos do tema de cores que explicamos anteriormente. Há duas características principais que permitem isso no Lollipop: coloração de **drawables** e referências a atributos de temas em drawables (na forma ?attr/foo).

AppCompatActivity provê comportamento similar para versões anteriores de Android para um subconjunto de widgets.

- Tudo que for provido pela toolbar AppCompatActivity (**action modes**, etc.).
- EditText.
- Spinner.
- CheckBox.
- RadioButton.
- Switch (use [android.support.v7.widget.SwitchCompat](https://developer.android.com/reference/android/support/v7/widget/SwitchCompat))
- CheckedTextView.

Você não precisa fazer nada de especial para fazer eles funcionarem, simplesmente use esses controles de forma usual e o AppCompatActivity fará o resto.

## 5.4 Toolbar Widget



Toolbar é totalmente suportada pela AppCompatActivity. É implementada na classe `android.support.v7.widget.Toolbar`. Há duas formas de se usar a Toolbar:

- Use uma Toolbar como uma ActionBar quando você quer usar as facilidades de uma ActionBar (como o **menu inflation**, **ActionBarDrawerToggle** e outros) mas quer ter mais controle sobre a aparência dela.
- Use uma Toolbar de forma isolada quando você precisar desse padrão em seu app, nas situações em que uma ActionBar não suporta. Por exemplo, mostrar múltiplas Toolbars na tela, expandir por somente uma parte da tela e outros.

### 5.4.1. Modo Action Bar

Para usar Toolbar como uma ActionBar, primeiro temos que desabilitar a ActionBar provida pelo sistema. A forma mais fácil é ter seu tema estendendo de **Theme.AppCompat.NoActionBar** (ou sua variante light). Segundo, criar uma instância do widget Toolbar via layout XML:

```
<android.support.v7.widget.Toolbar
    android:id="@+id/my_awesome_toolbar"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:minHeight="?attr/actionBarSize"
    android:background="?attr/colorPrimary" />
```

A largura, tamanho, background e outras configurações são totalmente ao seu critério. Estes são somente alguns exemplos. Uma Toolbar é somente uma ViewGroup, você pode estilizá-la e posicioná-la onde achar necessário. Então, em sua Activity ou Fragment, configure a Toolbar para se comportar como uma ActionBar:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.blah);
    Toolbar toolbar = (Toolbar) findViewById(R.id.my_awesome_toolbar);
    setSupportActionBar(toolbar);
}
```

A partir desse ponto, todos os menu items são exibidos na sua Toolbar, populados via chamada de função padrão de menu.

### 5.4.2. Modo Standalone

---

A diferença no modo standalone é que você não configura a Toolbar para se comportar como uma ActionBar. Por essa razão você pode usar qualquer tema AppCompatActivity e você não precisa desabilitar a ActionBar provida pelo sistema.

No modo standalone você precisa manualmente popular a Toolbar com **contents** ou **actions**. Por exemplo, se vc precisa exibir ações, você precisa inflar um menu em sua Toolbar:

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

    Toolbar toolbar = (Toolbar) findViewById(R.id.my_awesome_toolbar);

    // Set an OnMenuItemClickListener to handle menu item clicks
    toolbar.setOnMenuItemClickListener(new Toolbar.OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem item) {
            // Handle the menu item
            return true;
        }
    });

    // Inflate a menu to be displayed in the toolbar
    toolbar.inflateMenu(R.menu.your_toolbar_menu);
}
```

## 5.5. Aplicando estilos

Estilizar a Toolbar é feito de forma diferente de uma ActionBar padrão, e é configurada diretamente dentro da **view**. Aqui está uma configuração de estilo básica que você pode usar quando estiver usando a Toolbar como uma ActionBar:

```
<android.support.v7.widget.Toolbar

    android:layout_height="wrap_content"

    android:layout_width="match_parent"

    android:minHeight="?attr/actionBarSize"

    app:theme="@style/ThemeOverlay.AppCompat.ActionBar" />
```

A declaração **app:theme** assegurará que seus textos e itens estarão usando cores sólidas (por exemplo 100% branco opaco).

### DarkActionBar

Você pode estilizar instâncias da Toolbar diretamente usando atributos de layout. Para conseguir uma Toolbar que se parece com uma 'DarkActionBar' (conteúdo escuro, menu de overflow claro), você deve prover os atributos **theme** e **popupTheme**:

```
<android.support.v7.widget.Toolbar

    android:layout_height="wrap_content"

    android:layout_width="match_parent"

    android:minHeight="@dimen/triple_height_toolbar"
```

```
app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

## 5.6. SearchView Widget

AppCompat oferece a SearchView API atualizada do Android Lollipop, que é de longe mais customizável e estilizável. Nós agora usamos a estrutura de design do Lollipop ao invés de usar os antigos atributos de design SearchView.

Aqui está como você deve estilizar seu SearchView:

values/themes.xml:

```
<style          name="Theme.MyTheme"          parent="Theme.AppCompat">
  <item          name="searchViewStyle">@style/MySearchViewStyle</item>
</style>
<style          name="MySearchViewStyle"          parent="Widget.AppCompat.SearchView">
  <!-- Background for the search query section (e.g. EditText) -->
  <item          name="queryBackground">...</item>
  <!-- Background for the actions section (e.g. voice, submit) -->
  <item          name="submitBackground">...</item>
  <!-- Close button icon -->
  <item          name="closeIcon">...</item>
  <!-- Search button icon -->
  <item          name="searchIcon">...</item>
  <!-- Go/commit button icon -->
  <item          name="goIcon">...</item>
  <!-- Voice search button icon -->
  <item          name="voiceIcon">...</item>
  <!-- Commit icon shown in the query suggestion row -->
  <item          name="commitIcon">...</item>
  <!-- Layout for query suggestion rows -->
```

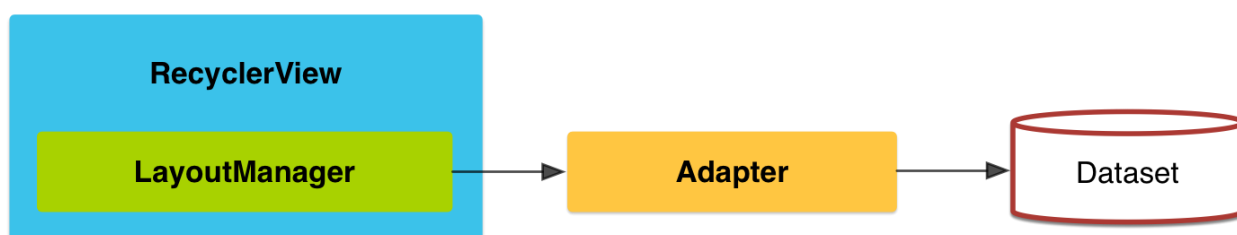
```
<item                                     name="suggestionRowLayout">...</item>
</style>
```

Você não precisa configurar todos, ou nenhum, desses; as configurações padrão irão funcionar para a maioria dos apps.



## Capítulo 6. RecyclerView e Cardview

A nova biblioteca de suporte AppCompat introduziu dois novos widgets: **RecyclerView** e **CardView**. A RecyclerView é uma versão mais avançada e mais flexível de uma ListView. Este novo componente é um grande passo, porque a ListView é um dos widgets mais usados. O widget CardView, por outro lado, é um novo componente. Neste capítulo explicaremos como usar esses dois widgets e mostrar como combinar eles. Vamos começar aprofundando no RecyclerView.



### 6.1. RecyclerView: Introdução

Como mencionado anteriormente, RecyclerView é mais flexível que a ListView, porém adiciona algumas complexidades. Todos nós sabemos como usar uma ListView em nosso app e nós sabemos que se queremos aumentar o desempenho de uma ListView podemos usar um padrão chamado ViewHolder. Este padrão consiste em uma classe simples que armazena as referências para os componentes de interface para cada uma das linhas da ListView. Este padrão evita a procura de componentes de interface toda hora que o sistema precisar exibir uma determinada linha da ListView na tela.

Mesmo esse padrão trazendo alguns benefícios, podemos implementar a ListView sem usar esse padrão. RecyclerView nos força a usar o padrão ViewHolder. Para mostrar como podemos usar a RecyclerView, nós podemos supor que queremos construir um app que mostra uma lista de cards de contatos. A primeira coisa que precisamos criar é o layout principal. RecyclerView é bem similar à ListView e nós podemos usar ela da mesma forma.

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MyActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/cardList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</RelativeLayout>

```

Podemos notar pelo layout mostrado acima, que a RecyclerView está disponível na biblioteca de suporte do Android, então temos que modificar o arquivo build.gradle e incluir a dependência:

```

dependencies {
    implementation 'com.android.support:recyclerview-v7:27.1.1'
}

```

Agora, no método **onCreate** nós podemos obter uma referência para nossa RecyclerView e configurá-la:

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my);
    RecyclerView recList = (RecyclerView) findViewById(R.id.cardList);
    recList.setHasFixedSize(true);
    LinearLayoutManager llm = new LinearLayoutManager(this);
}

```

```

llm.setOrientation(LinearLayoutManager.VERTICAL);
recList.setLayoutManager(llm);
}

```

Se você olhar para o código acima, você irá notar algumas diferenças entre a RecyclerView e a ListView. RecyclerView requer um **layout manager**, esse componente posiciona os itens dentro das linhas da lista e determina quando é tempo de reciclar as **views**. A biblioteca provê um gerenciador de layout padrão chamado **LinearLayoutManager**.

## 6.2. CardView

O componente **CardView** mostra informações dentro de card. Nós podemos customizar as suas bordas, a elevação e outras propriedades. Nós precisamos usar esse componente para mostrar informações sobre os contatos de nosso app. Esses cards serão linhas de nossa RecyclerView e veremos mais tarde como integrar os dois componentes. Por agora, definiremos o layout do card:

```

<android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    card_view:cardCornerRadius="4dp"
    android:layout_margin="5dp">
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<TextView
    android:id="@+id/title"
    android:layout_width="match_parent"
    android:layout_height="20dp"

```

```

        android:background="@color/bkg_card"
        android:text="contact"
        android:gravity="center_vertical"
        android:textColor="@android:color/white"
        android:textSize="14dp"/>

```

det"

<TextView

```

        android:id="@+id/txtName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Name"
        android:gravity="center_vertical"
        android:textSize="10dp"
        android:layout_below="@id/title"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="5dp"/>

```

<TextView

```

        android:id="@+id/txtSurname"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Surname"
        android:gravity="center_vertical"
        android:textSize="10dp"
        android:layout_below="@id/txtName"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="5dp"/>

```

<TextView

```

        android:id="@+id/txtEmail"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Email"
        android:textSize="10dp"
        android:layout_marginTop="10dp"
        android:layout_alignParentRight="true"
        android:layout_marginRight="150dp"

```

```

        android:layout_alignBaseline="@id/txtName"/>
    </RelativeLayout>

```

Como você pode ver, o `CardView` é muito simples de usar. Este componente está disponível em uma outra biblioteca de suporte e devemos adicionar outra linha às nossas dependências:

```

dependencies {

    implementation 'com.android.support:cardview-v7:27.1.1'
    implementation 'com.android.support:recyclerview-v7:27.1.1'

}

```

### 6.3. RecyclerView: Adapter

O **adapter** é um componente que se encontra entre o **modelo de dados** que queremos mostrar no app e o componente que mostrará a informação na tela. Em outras palavras, um adaptador guia a forma como os dados são mostrados na tela. Então se nós precisamos mostrar nossos contatos, precisamos de um adaptador para a `RecyclerView`. Este adaptador precisa estender uma classe chamada **`RecyclerView.Adapter`**, passando nossa classe que implementa o padrão **`ViewHolder`**:

```

public class MyAdapter extends RecyclerView.Adapter<MyHolder> { ..... }

```

Nós agora temos que sobrescrever dois métodos para implementar nossa lógica: **`onCreate ViewHolder`** é chamado sempre que uma nova instância de nossa classe `ViewHolder` é criada, e **`onBindViewHolder`** é chamada quando o sistema operacional anexa a view com os dados, ou em outras palavras, quando os dados são mostrados na tela. Neste caso, o adaptador nos ajuda a combinar a `RecyclerView` e a `CardView`. O layout que definimos antes para os cards será o layout da linha (que representa um contato) da nossa `RecyclerView` (que será uma lista de contatos). Antes de fazer isso, precisamos definir nosso modelo de dados

que será a base dos dados que serão exibidos na tela. Para esse propósito definimos uma classe simples:

```
public class ContactInfo {  
  
    protected String name;  
  
    protected String surname;  
  
    protected String email;  
  
    protected static final String NAME_PREFIX = "Name_";  
  
    protected static final String SURNAME_PREFIX = "Surname_";  
  
    protected static final String EMAIL_PREFIX = "email_";  
  
}
```

E finalmente nós estamos prontos para criar nosso adaptador. Se você lembra o que dissemos anteriormente sobre o padrão ViewHolder, nós temos que codificar nossa classe e implementá-la (Fonte: <https://dzone.com/articles/guide-android-recycler-and-cardview#printSource>):

```
public static class ContactViewHolder extends RecyclerView.ViewHolder {  
  
    protected TextView vName;  
  
    protected TextView vSurname;  
  
    protected TextView vEmail;  
  
    protected TextView vTitle;  
  
  
    public ContactViewHolder(View v) {  
  
        super(v);  
  
    }  
}
```

```
vName = (TextView) v.findViewById(R.id.txtName);

vSurname = (TextView) v.findViewById(R.id.txtSurname);

vEmail = (TextView) v.findViewById(R.id.txtEmail);

vTitle = (TextView) v.findViewById(R.id.title);

    }

}
```

Olhe o código, em nosso construtor nós pegamos a referência para as views definidas em nosso layout de card. Agora já é tempo de implementar nosso adaptador:

```
public class ContactAdapter extends

    RecyclerView.Adapter<ContactAdapter.ContactViewHolder> {

    private List<ContactInfo> contactList;

    public ContactAdapter(List<ContactInfo> contactList) {

        this.contactList = contactList;

    }

    @Override

    public int getItemCount() {

        return contactList.size();

    }

}
```

```
}
```

```
@Override
```

```
public void onBindViewHolder(ContactViewHolder contactViewHolder, int i)
```

```
{
```

```
    ContactInfo ci = contactList.get(i);
```

```
    contactViewHolder.vName.setText(ci.name);
```

```
    contactViewHolder.vSurname.setText(ci.surname);
```

```
    contactViewHolder.vEmail.setText(ci.email);
```

```
    contactViewHolder.vTitle.setText(ci.name + " " + ci.surname);
```

```
}
```

```
@Override
```

```
public ContactViewHolder onCreateViewHolder(ViewGroup viewGroup, int i)
```

```
{
```

```
    View itemView = LayoutInflater.from(viewGroup.getContext()).
```

```
    inflate(R.layout.card_layout, viewGroup, false);
```

```
    return new ContactViewHolder(itemView);
```

```
}
```

```
public static class ContactViewHolder extends RecyclerView.ViewHolder {
```

```
    ...
```



}

}

Em nossa implementação, nós sobrescrevemos **onBindViewHolder** onde nós anexamos os dados (nossa informação de contato) às views. Note que nós não manejamos os componentes de tela, mas simplesmente referências armazenadas em nosso **ContactViewHolder**. No **onCreateViewHolder** nós retornamos nosso **ContactViewHolder** inflando o layout da linha da lista (o CardView em nosso caso). Execute o app e veja os resultados:



## Capítulo 7. Layouts Adaptáveis e Fragments

### 7.1. O que são layouts single-panel e multi-panel?

Dispositivos Android existem em uma variedade de tamanhos e densidade de telas. Um painel representa uma parte de uma interface de usuário. O termo painel é o termo geral usado para descrever o conceito que múltiplas views são combinadas em uma view agrupada, dependendo do espaço atualmente disponível.

Se não existe espaço disponível, somente um painel estará visível. Isto é chamado tipicamente de layout **single-panel**.

Se mais espaço está disponível, então múltiplos painéis podem ser mostrados.

## One panel visible Two panels visible



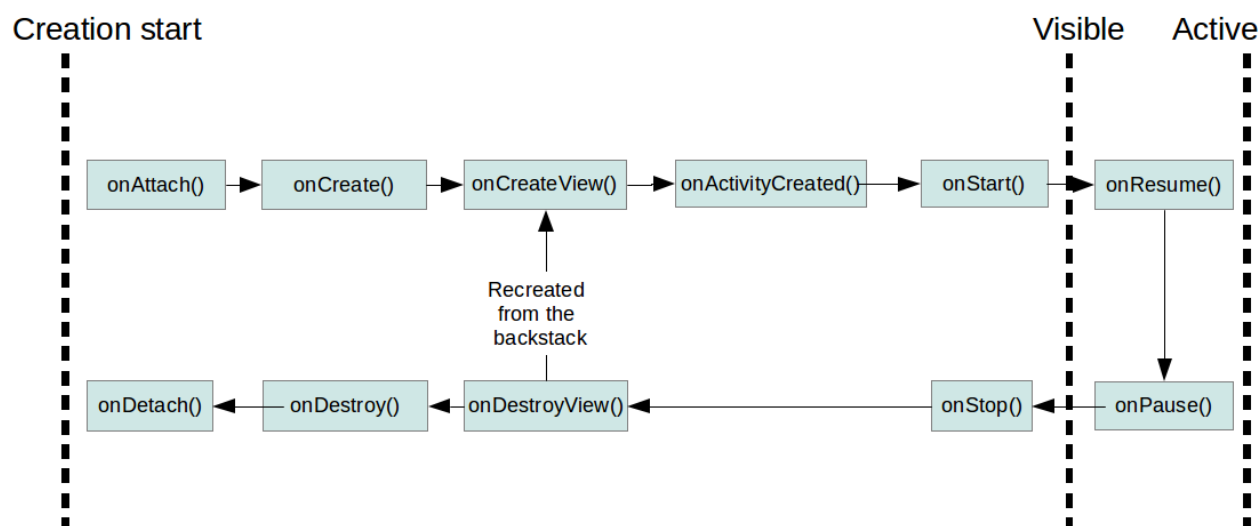
### 7.2 O que são Fragments?

Um **fragment** é um componente Android independente que pode ser usado por uma activity. O fragmento encapsula funcionalidades de modo que é simples de ser reusado com activities e layouts.

Um fragmento é executado no contexto de uma activity, mas tem seu próprio ciclo de vida e tipicamente sua própria interface de usuário. É possível definir fragmentos sem uma interface de usuário, por exemplo, **headless fragments**.

### 7.2.1. Ciclo de vida dos fragments

Um fragmento tem seu próprio ciclo de vida, mas ele é sempre conectado ao ciclo de vida da activity que usa aquele fragmento.



Se uma activity para, seus fragmentos são também parados; se uma activity é destruída, seus fragmentos também são destruídos.

**Tabela 1 - Ciclo de vida dos fragments.**

Método	Descrição
onAttach()	A instância do fragmento é associada com a instância da activity. O fragmento e a activity não estão competamente

	<p>inicializados. Tipicamente, neste método você obtém uma referência para a activity, que usa esse fragmento para fins de inicialização de algumas funcionalidades.</p>
<code>onCreate()</code>	<p>O fragmento é criado. O método <b>onCreate()</b> é chamado depois do método <b>onCreate()</b> da activity, mas antes do método <b>onCreateView()</b> do fragmento.</p>
<code>onCreateView()</code>	<p>A instância de fragmento cria sua hierarquia de views. No método <b>onCreateView()</b>, o fragmento cria sua interface de usuário. Aqui você pode inflar um layout utilizando o método <b>inflate()</b> do objeto <b>Inflator</b> passado como parâmetro para este método.</p> <p>Neste método você ainda não pode interagir com a activity, pois ela ainda não está totalmente inicializada.</p> <p>Não há necessidade de implementar esse método para fragmentos <b>headless</b>. As views infladas se tornam parte da hierarquia de view da activity pai.</p>
<code>onActivityCreated()</code>	<p>O método <b>onActivityCreated()</b> é chamado depois do método <b>onCreateView()</b>, quando a activity pai é criada. As instâncias de activity e fragmento foram criadas assim como a hierarquia de views da activity. Neste ponto, views podem ser acessadas com o método <b>findViewById()</b>.</p> <p>Neste método você pode instanciar objetos que requerem um objeto da classe <b>Context</b>.</p>

onStart()	O método <b>onStart()</b> é chamado assim que o fragmento se torna visível.
onResume()	O fragmento se torna ativo.
onPause()	O fragmento é visível, mas não fica ativo. Por exemplo, se outra activity é animada em cima da activity que contém o fragmento.
onStop()	O fragmento se torna não-visível.
onDestroyView()	Destrói a view do fragmento. Se o fragmento é recriado da backstack, este método é chamado e logo depois o método <b>onCreateView</b> .
onDestroy()	Não garantido de ser chamado pela plataforma Android.

### 7.2.2. Fragmentos e Contexto de acesso

---

Fragmentos não são subclasses da classe Contexto. Portanto tem que usar o método getActivity() para obter a activity pai.

### 7.2.3. Vantagens de usar fragmentos

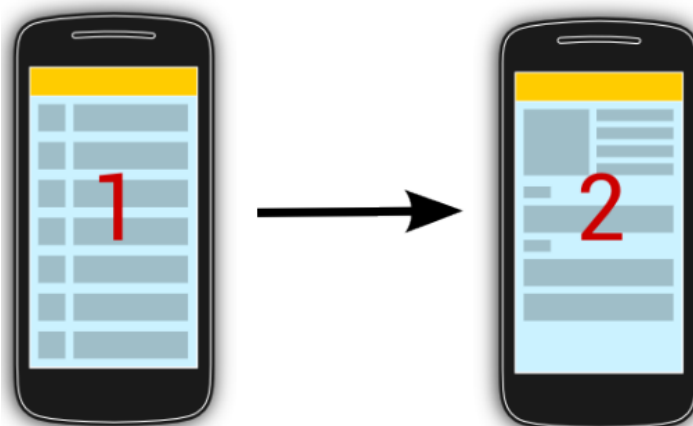
---

Fragments simplificam o reuso de componentes em diferentes layouts, por exemplo, você pode criar layouts single-panel para telefones móveis (handsets) e layouts multi-panel para tablets. Isso não é limitado aos tablets, por exemplo, você

pode usar fragments para diferentes layouts para orientação em retrato e paisagem do smartphone.

Como é possível adicionar e remover fragmentos dinamicamente de uma activity, o uso de fragmentos permite um design muito flexível das interfaces de usuário.

O exemplo típico é uma lista de itens em uma activity. Em um tablet você vê os detalhes imediatamente na mesma tela, do lado direito ao item clicado. Em um smartphone você é levado para uma nova tela de detalhes. Isso é explicado nas figuras seguintes:



A discussão que se segue assume que você tem dois fragments (main e detail), mas você pode ter mais. Nós temos também uma activity main e uma activity de detalhes. Em um tablet, a activity main contém os dois fragmentos no layout; em um smartphone, ela contém apenas o fragmento main.

A imagem que segue ilustra esse caso.



#### 7.2.4. Como suportar diferentes tamanhos de tela com fragments

---

É possível definir no arquivo de layout de uma activity, que ela contenha fragmentos (definição estática) ou modificar os fragmentos de uma activity em tempo de execução (definição dinâmica). Para exibir diferentes fragmentos em suas activities baseado no espaço disponível atual, você pode:

- Usar uma activity, que exibe dois fragments para tablets e em smartphones. Neste caso, mude em tempo de execução os fragmentos que serão exibidos pela activity onde for necessário. Neste cenário você tipicamente define instâncias da classe **FrameLayout** como um espaço reservado em seu arquivo de layout e adiciona os fragments em tempo de execução a ele.
- Use activities separadas para suportar cada fragment em um smartphone. Por exemplo, enquanto a interface do tablet usa dois fragmentos em uma activity, use a mesma activity para smartphones, mas forneça um layout alternativo que inclui somente um fragmento. Se o fragmento detalhado está ali, a activity main diz ao fragmento para se auto atualizar. Se o fragmento de detalhes não está disponível, a activity main inicia a activity detail em seu lugar.

Qual opção usar depende do seu caso. Tipicamente a troca dinâmica de fragmentos é mais flexível e um pouco mais difícil de implementar.

### 7.3 Definindo e usando fragmentos

#### 7.3.1 Definindo fragmentos

---

Para definir um novo fragmento, você pode tanto estender a classe **android.app.Fragment** ou uma de suas subclasses, por exemplo **ListFragment**, **DialogFragment**, **PreferenceFragment** ou **WebViewFragment**. O código que segue mostra um exemplo de implementação.

```

package com.example.android.rssfeed;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_rssitem_detail,
            container, false);
        return view;
    }

    public void setText(String item) {
        TextView view = (TextView) getView().findViewById(R.id.detailsText);
        view.setText(item);
    }
}

```

### 7.3.2. Aplicando comunicação com os fragmentos

Para aumentar o reúso dos fragmentos, eles não devem se comunicar diretamente um com o outro. Toda comunicação deve ser feita através da activity que hospeda esses fragmentos. Para este propósito, um fragmento deve definir uma interface como um tipo interno e requisitar que a activity que o usa implemente essa interface. Desta forma, você impede que o fragmento tenha qualquer conhecimento sobre a activity que o usa. Em seu método **onAttach()** ele pode verificar se a activity implementa corretamente esta interface.



Por exemplo, assuma que você tem um fragmento que precisa comunicar um valor para sua activity que o instanciou. Isso pode ser implementado da seguinte forma:

```

package com.example.android.rssfeed;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

public class MyListFragment extends Fragment {
    private OnItemSelectedListener listener;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_rsslist_overview,
            container, false);
        Button button = (Button) view.findViewById(R.id.button1);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                updateDetail();
            }
        });
        return view;
    }

    public interface OnItemSelectedListener {
        public void onRssItemSelected(String link);
    }

    @Override

```

```

public void onAttach(Activity activity) {
    super.onAttach(activity);
    if (activity instanceof OnItemSelectedListener) {
        listener = (OnItemSelectedListener) activity;
    } else {
        throw new ClassCastException(activity.toString()
            + " must implement MyListFragment.OnItemSelectedListener");
    }
}

@Override
public void onDetach() {
    listener = null;
}

// may also be triggered from the Activity
public void updateDetail() {
    // create a string just for testing
    String newTime = String.valueOf(System.currentTimeMillis());

    // inform the Activity about the change based
    // interface definition
    listener.onRssItemSelected(newTime);
}
}

```

### 7.3.3. Adicionando fragmentos estáticamente ao arquivo de layout

Para usar seu novo fragmento você pode estaticamente adicionar ele a um layout XML. Neste caso, o atributo **android:name** aponta para a classe correspondente, como demonstrado pelo código que segue:

```

        <?xml                                version="1.0"                                encoding="utf-8"?>
<LinearLayout                                xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:baselineAligned="false"
    android:orientation="horizontal"                                >
    <fragment
        android:id="@+id/listFragment"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="match_parent"
        class="com.example.android.rssfeed.MyListFragment"                                ></fragment>
    <fragment
        android:id="@+id/detailFragment"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"
        class="com.example.android.rssfeed.DetailFragment"                                >
    </fragment>
</LinearLayout>

```

Usar esse cenário faz sentido no caso de você possuir diferentes layouts estáticos para diferentes configurações de dispositivo.

### 7.3.4. Tratando dinamicidade em fragmentos

A classe que pode ser acessada na activity pelo método **getFragmentManager()** permite a você adicionar, remover e trocar fragmentos no layout de uma activity.

As modificações precisam ser feitas em transações pela classe **FragmentManager**.

Para modificar os fragmentos em uma activity, você tipicamente define um container **FrameLayout** no seu layout.

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

    <FrameLayout
        android:id="@+id/listcontainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"                                />

    <FrameLayout
        android:id="@+id/detailscontainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="gone"                                />

</LinearLayout>
```

Você usará a o **FragmentManager** para trocar o container pelo fragment.

```
//                                get                                fragment                                manager
FragmentManager                                fm                                =                                getFragmentManager();

//                                add
FragmentManager                                ft                                =                                fm.beginTransaction();
ft.add(R.id.your_placehodler,                                new                                YourFragment());
//                                alternatively                                add                                it                                with                                a                                tag
//                                trx.add(R.id.your_placehodler,                                new                                YourFragment(),                                "detail");
ft.commit();
```

```
// replace
FragmentTransaction ft = fm.beginTransaction();
ft.replace(R.id.your_placehodler, new YourFragment());
ft.commit();

// remove
Fragment fragment = fm.findFragmentByld(R.id.your_placehodler);
FragmentTransaction ft = fm.beginTransaction();
ft.remove(fragment);
ft.commit();
```

Um novo fragmento substitui um fragmento existente no container. Se você precisa adicionar a transação à backstack do android, você pode usar o método **addToBackStack()**. Isso irá adicionar a ação à pilha de histórico da activity, por exemplo, isso permitirá reverter as mudanças no fragmento pelo botão voltar do android.

### 7.3.5. Verificar se um fragmento está presente no layout

Para verificar se o fragmento é parte do seu layout você pode usar a classe **FragmentManager**. O método **isInLayout()** funciona se o fragmento foi adicionado à activity pelo layout.

```
DetailFragment fragment = (DetailFragment) getFragmentManager().
findFragmentByld(R.id.detail_frag);
if (fragment==null || ! fragment.isInLayout()) {
// start new Activity
}
else {
```

```
fragment.update(...);  
}
```

#### 7.4. Determinar se uma activity está em modo single/dual/multi

---

Como a lógica na activity depende do cenário (single/multi-panel), você tipicamente faz uma verificação antes de configurar o layout na activity para saber em qual modo você está. Há vários modos de conseguir isso. Uma forma é definir um arquivo de configuração na pasta de recurso **values** do seu projeto, com um par de chave/valor por definição falso e arquivos de configuração adicionais para diferentes configurações, atribuindo valor true a este campo.

Por exemplo, este é o arquivo de configuração *config.xml*.e:

```
<resources>  
  <item type="bool" name="dual_pane">false</item>  
</resources>
```

Por exemplo, na pasta *values-land* você coloca um *config.xml* com um valor diferente:

```
<resources>  
  <item type="bool" name="dual_pane">true</item>  
</resources>
```

No seu código, você pode acessar o estado do parâmetro seguindo este pedaço de código:

```
getResources().getBoolean(R.bool.dual_pane);
```

## 7.5. Adicionando transações de fragmento à backstack

---

Você pode adicionar uma transação de fragmento à backstack para permitir usar o botão de voltar para reverter a transação.

Para isso você pode usar o método **addToBackStack()** do objeto **FragmentManager**.

## 7.6. Animações e transações para fragmentos

---

Durante uma transação de fragmentos você pode definir animações que deverão ser usadas baseadas na **Property Animation API** pelo método **setCustomAnimations()**.

Você pode também usar várias das animações providas pelo Android, usando o método **setTransition()**. Este é definido por constantes, começando com **FragmentManager.TRANSIT\_FRAGMENT\_\***.

Ambos métodos permitem você definir uma animação de entrada e uma animação de saída.

## 7.7. Persistindo dados em fragments

### 7.7.1. Persistindo dados entre restarts da aplicação

---

Em fragmentos você também precisa guardar dados da aplicação. Para isso, você pode persistir os dados em um lugar centralizado. Por exemplo:

- Bases de dados SQLite;
- Arquivos;
- O objeto da aplicação (caso o objeto **Application** precise manejar o storage).

### 7.7.2. Persistindo dados entre trocas de configuração

---

Se você precisa persistir dados entre trocas de configuração, você pode também usar o objeto **application**.

Além disso, você pode usar o método **setRetainState(true)** do fragmento. Este retém instâncias de fragmento entre trocas de configuração, mas só funciona se os fragments não são adicionados à backstack. O uso desse método não é recomendado pela Google para fragmentos que possuem uma interface de usuário. Nestes casos, os dados podem ser armazenados como campos.

Se os dados que precisam ser salvos são suportados pela classe **Bundle**, você pode usar o **onSaveInstanceState()** para colocar os dados no Bundle, e recuperar esses dados no método **onActivityCreated()**.

## 7.8. Fragmentos para processamento em background

---

### 7.8.1. Fragmentos Headless

---

Fragmentos podem ser usados sem definir uma interface de usuário.

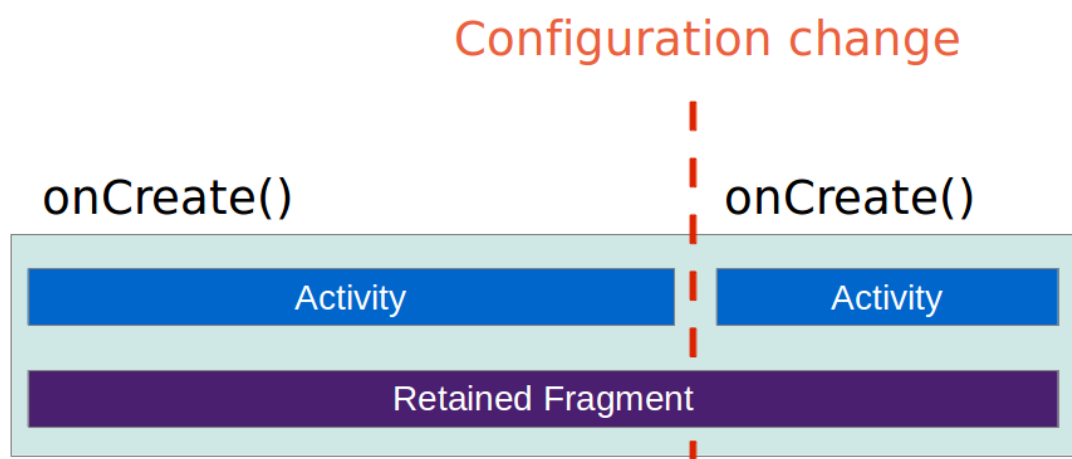
Para implementar um fragmento headless, simplesmente retorne **null** no método **onCreateView()** do seu fragmento.

**Dica:** é recomendado que se use fragmentos headless para seus processos em background, em combinação com o método **setRetainState()**. Dessa forma você não terá que lidar com as mudanças de configuração durante os processamentos assíncronos em si.



### 7.8.2. Retenção de fragmentos headless para lidar com as mudanças de configuração

Fragmentos headless são tipicamente usados para encapsular alguns estados entre mudanças de configuração ou para tarefas de background. Para esse propósito você pode configurar seu fragmento headless para ser retido. Um fragmento retido não é destruído durante trocas de configuração.



Para configurar seu fragmento para ser retido, chame o método **setRetainInstance()**.

Para adicionar esse fragmento à activity, você pode usar o método **add()** da classe **FragmentManager**. Se você precisar se referir ao fragmento depois, você precisa adicionar uma tag para ser capaz de procurar por ele via o método **findFragmentByTag()** do **FragmentManager**.

*Cuidado: o uso do método `onRetainNonConfigurationInstance()` está depreciado e deve ser trocado pelos fragmentos headless retidos que acabamos de ver.*

## Capítulo 8. Intents e Intent Filters

---

Uma Intent é um objeto de mensagem que você pode usar para requisitar uma ação para um componente de outro app. Embora Intents facilitem a comunicação entre componentes de diversas formas, há três casos de uso fundamentais em que se usam intents:

- **Para iniciar uma activity:** uma activity representa uma única tela de um app. Você pode inicializar uma instância de uma activity passando um intent para o método **startActivity()**, por exemplo.
- **Para iniciar um serviço:** um **Serviço** é um componente que executa operações em background sem uma interface de usuário. Você pode iniciar um serviço para executar uma operação um-para-um (como fazer download de um arquivo), passando uma intent para **startService()**.
- **Para entrega de um broadcast:** um broadcast é uma mensagem que qualquer app pode receber.

### 8.1. Tipos de Intents

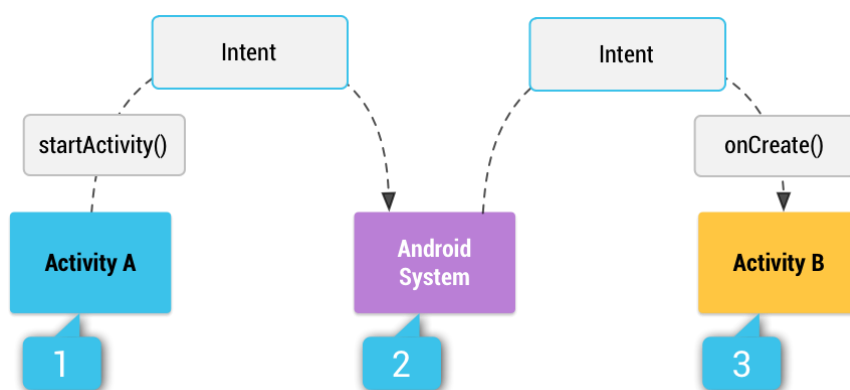
---

Há dois tipos de intents:

- **Intents Explícitas:** especificam o componente a ser iniciado pelo nome (nome de classe totalmente qualificado). Você irá tipicamente usar intents explícitas para iniciar um componente em seu próprio app, porque você sabe o nome de classe da activity ou serviço que você necessita iniciar. Por exemplo, iniciar uma nova activity em resposta a uma ação do usuário, ou iniciar um serviço para fazer download de um arquivo em background.
- **Intents Implícitas:** essas intents não nomeiam um componente específico, mas ao invés disso declaram uma ação geral a ser executada, o que permite um componente de outro app tratar essa ação. Por exemplo, se você precisa mostrar uma localização de usuário em um mapa, você pode

usar uma intent implícita para requisitar que outro app apresente a localização em um mapa.

Quando você cria uma intent explícita para iniciar uma activity, ou um serviço, o sistema imediatamente inicia o componente de app especificado no objeto de intent.



A figura acima é uma ilustração de como uma intent implícita é passada pelo sistema para iniciar outra activity: **[1]** *Activity A* cria uma Intent com uma descrição de ação e passa ela para o método `startActivity()`. **[2]** O sistema Android procura todos os apps por filtros de intent que casam com a intent. Quando uma correspondência é achada, **[3]** o sistema inicia a activity que atendeu aos requisitos (*Activity B*), invocando seu método `onCreate()` e passando pra ele o intent.

## 8.2. Criando uma Intent

Um objeto intent transporta as informações que o sistema android usa para determinar qual componente iniciar (pode ser um nome de componente exato ou uma categoria que deve receber a intent), almás informações que o componente recipiente usa para executar a ação de forma correta.

### 8.3. Exemplo de Intent Explícita

---

Uma intent explícita é uma das que você irá usar para inicializar um componente de de app específico, como uma activity em particular, ou um serviço do seu app. Para criar uma intent explícita, defina o nome de componente do objeto intent -- todas as outras propriedades da intent são opcionais.

Por exemplo, se você criar um serviço em seu app, chamado **DownloadService**, desenhado para fazer download de um arquivo da web, você pode iniciar ele com o código seguinte:

```
// Executed in an Activity, so 'this' is the Context

// The fileUrl is a string URL, such as "http://www.example.com/image.png"

Intent downloadIntent = new Intent(this, DownloadService.class);

downloadIntent.setData(Uri.parse(fileUrl));

startService(downloadIntent);
```

O construtor **Intent(Context,Class)** necessita do contexto do app e do componente como um objeto **Class**. Como tal, a intent inicia explicitamente o serviço **DownloadService** do app.

### 8.4. Exemplo de uma intent implícita

---

Uma intent implícita especifica uma ação que pode ser invocada por qualquer app no dispositivo capaz de executar a ação. Usar uma intent implícita é útil quando o seu app não pode executar uma ação, mas outros apps provavelmente podem e você quer deixar o usuário decidir qual app usar.

Por exemplo, se você tem um conteúdo que você quer que o usuário compartilhe com outras pessoas, crie um intent com a ação **ACTION\_SEND** e adiciona extras que especificam o conteúdo a ser compartilhado. Quando você chamar **startActivity()** com a intent, o usuário vai escolher um app pelo qual deseja compartilhar o conteúdo.

**Cuidado:** É possível que o usuário não tenha nenhum app para tratar a intent implícita que você enviou para **startActivity()**. Se isso acontecer, a chamada irá falhar e seu app irá falhar. Para verificar se uma activity irá receber o intent, chame **resolveActivity()** em seu objeto intent. se o resultado é não-null, então há pelo menos um app que pode tratar o intent e é seguro chamar **startActivity()**. Se o resultado é nulo, você deve usar a intent e, se possível, você deve desabilitar a funcionalidade que dispara a intent.

```
// Create the text message with a string
```

```
Intent sendIntent = new Intent();
```

```
sendIntent.setAction(Intent.ACTION_SEND);
```

```
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
```

```
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type
```

```
// Verify that the intent will resolve to an activity
```

```
if (sendIntent.resolveActivity(getPackageManager()) != null) {
```

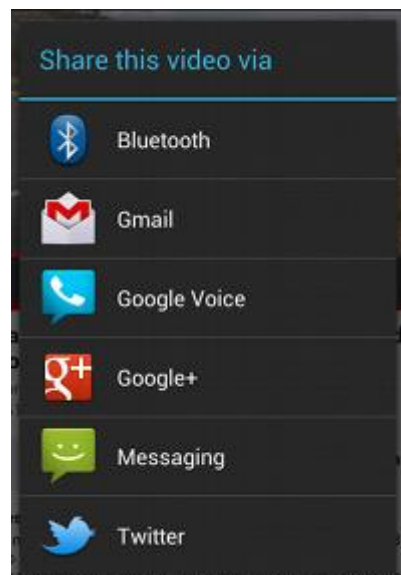
```
    startActivity(sendIntent);
```

```
}
```

**Nota:** Neste caso, a URI não é usada, mas o tipo de dados da intent é declarada para especificar o conteúdo carregado pelos extras.

Quando **startActivity()** é chamada, o sistema examina todos os apps instalados para determinar quais tratam esse tipo de intent ( uma intent com ação **ACTION\_SEND** e que carrega dados do tipo “text/plain” ). Se há somente um app que pode tratar ela, o app abre imediatamente e a ele é dada a intent. Se múltiplas activities aceitam o intent, o sistema mostra um diálogo para que o usuário escolha qual app usar.

**Figure 2 - Um diálogo de escolha de apps.**



## Capítulo 9. Sistema de Arquivos

---

O Android usa um sistema de arquivos que é similar ao sistemas baseados em disco de outras plataformas.

Um objeto **File** é adequado para a leitura ou escrita de grandes quantidades de dados em ordem do início ao fim (sequencial). Por exemplo, é bom para arquivos de imagem ou qualquer coisa trocadas através de uma rede.

Este capítulo mostrará como executar tarefas relacionadas a arquivos em seu app.

### 9.1. Escolher entre armazenamento interno ou externo

---

Todos os dispositivos Android possuem duas áreas de armazenamento: armazenamento **interno** e **externo**. Estes nomes vem desde o início do desenvolvimento do sistema android, quando a maioria dos dispositivos forneciam uma memória não-volátil ( armazenamento interno ), mais uma mídia de armazenamento como um cartão microSD ( armazenamento externo ). Alguns dispositivos dividem a área de armazenamento permanente em partições “**internas**” e “**externas**”, então mesmo sem uma mídia removível, há sempre dois espaços de armazenamento e o comportamento da API é o mesmo existindo uma mídia removível ou não. A lista que segue sumariza os fatos sobre cada um dos espaços de armazenamento.

#### **Armazenamento Interno:**

- Está sempre disponível.
- Arquivos salvos aqui são acessíveis somente para o seu app por padrão.
- Quando o usuário desinstala o app, o sistema remove todos os arquivos do app da memória interna.

Armazenamento interno é melhor quando você quer ter certeza que nem o usuário, nem outros apps possam acessar os arquivos que seu app criar.

### Armazenamento Externo:

- Não está sempre disponível, porque o usuário pode montar o armazenamento externo como USB e em alguns casos removê-lo do dispositivo.
- Armazenamento externo é legível para todos, então arquivos criados ali estão fora do seu controle.
- Quando o usuário desinstala o app, o sistema remove os arquivos criados pelo seu app somente se eles forem criados no diretório retornado pelo método **getExternalFilesDir()**.

Armazenamento externo é o melhor lugar para arquivos que não requerem restrições de acesso, para arquivos que você quer compartilhar com outros apps ou permitir que os usuários acessem pelo computador.

**Dica:** apesar de os apps serem instalados por padrão na memória interna, você pode especificar o atributo **android:installLocation** no seu arquivo de manifesto então seu app poderá ser instalado na memória externa. Usuários apreciam essa opção quando o tamanho do APK é muito grande e ele tem uma memória externa maior que memória interna.

## 9.2. Obter permissões para memória externa

---

Para escrever na memória externa, você precisa requisitar a permissão **WRITE\_EXTERNAL\_STORAGE** em seu arquivo de manifesto:

```
<manifest ...>

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

    ...

</manifest>
```



**Cuidado:** atualmente, todos os apps tem a habilidade de ler da memória externa sem uma permissão especial. No entanto, isso irá mudar no futuro. Se seu app precisa ler a memória externa ( mas não escrever nela ), então você precisa declarar a permissão **READ\_EXTERNAL\_STORAGE**. Para garantir que seu app continuará a funcionar como esperado, você precisa declarar esta permissão agora, antes que as alterações tenham efeito.

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

No entanto, se seu app usa a permissão **WRITE\_EXTERNAL\_STORAGE**, então ele tem a permissão para ler a memória externa também.

Você não precisa nenhuma permissão para salvar arquivos na memória interna. Seus aplicativos sempre tem permissão para ler e escrever arquivos na sua memória interna.

### 9.3. Salvar um arquivo na memória interna

---

Quando salvar um arquivo na memória interna, você poderá obter o diretório apropriado como um objeto **File** chamando um dos dois métodos:

1. **getFilesDir()**: retorna um objeto **File** representando um diretório interno do seu app.
2. **getCacheDir()**: retorna um objeto **File** representando o diretório responsável pelos arquivos de cache temporários do sistema. Esteja certo de deletar cada arquivo assim que ele não for mais necessário e implemente um limite de tamanho razoável para a quantidade de memória que usará em um dado

tempo, algo como 1MB. Se o sistema começar a ficar sem memória, ele pode deletar seus arquivos de cache sem qualquer aviso.

Para criar um novo arquivo em um desses diretórios, você usa o construtor do objeto **File()**, passando o objeto **File** provido por um dos métodos que especificados anteriormente. Por exemplo:

```
File file = new File(context.getFilesDir(), filename);
```

Alternativamente, você pode chamar o método **openFileOutput()** para obter um **FileOutputStream** que escreve em um arquivo em seu diretório na memória interna. Por exemplo, aqui está como escrever algum texto em um arquivo:

```
String filename = "myfile";
```

```
String string = "Hello world!";
```

```
FileOutputStream outputStream;
```

```
try {  
  
    outputStream = openFileOutput(filename, Context.MODE_PRIVATE);  
  
    outputStream.write(string.getBytes());  
  
    outputStream.close();  
  
} catch (Exception e) {  
  
    e.printStackTrace();  
  
}
```

Se você precisa fazer cache de alguns arquivos, você pode, ao invés, chamar o método **createTempFile()**. Por exemplo, o método seguinte extrai o nome de arquivo de uma URL e cria um arquivo com esse nome no diretório de cache interno do seu app:

```
public File getTempFile(Context context, String url) {  
  
    File file;  
  
    try {  
  
        String fileName = Uri.parse(url).getLastPathSegment();  
  
        file = File.createTempFile(fileName, null, context.getCacheDir());  
  
        catch (IOException e) {  
  
            // Error while creating file  
  
        }  
  
        return file;  
  
    }  
}
```

#### 9.4. Salvar arquivos na memória externa

---

Como o armazenamento externo pode estar indisponível -- como quando o usuário montou o armazenamento em um PC, ou removeu o cartão SD que provê o armazenamento externo -- você deve sempre verificar que o volume está disponível antes de acessá-lo. Você pode consultar o estado do armazenamento externo chamando o método **getExternalStorageState()**.

Se o estado retornado é igual a **MEDIA\_MOUNTED**, então você pode ler e escrever seus arquivos. Por exemplo, o métodos seguintes são úteis para determinar a disponibilidade do armazenamento:

*/\* Checks if external storage is available for read and write \*/*

```
public boolean isExternalStorageWritable() {  
  
    String state = Environment.getExternalStorageState();  
  
    if (Environment.MEDIA_MOUNTED.equals(state)) {  
  
        return true;  
  
    }  
  
    return false;  
  
}
```

*/\* Checks if external storage is available to at least read \*/*

```
public boolean isExternalStorageReadable() {  
  
    String state = Environment.getExternalStorageState();  
  
    if (Environment.MEDIA_MOUNTED.equals(state) ||  
  
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {  
  
        return true;  
  
    }  
  
    return false;  
  
}
```

Embora o armazenamento externo seja modificável pelo usuários e outros apps, há duas categorias de arquivos que você precisa salvar lá:

### Arquivos públicos:

Arquivos que precisam estar disponíveis de forma livre para outros apps e para o usuário. Quando o usuário desinstala o app, estes arquivos devem permanecer disponíveis para o usuário.

Por exemplo, fotos capturadas pelo seu app ou arquivos que foram baixados.

### Arquivos privados:

Arquivos que são legitimamente pertencentes ao seu app e devem ser deletados quando o usuário desinstala seu app. Embora esses arquivos sejam tecnicamente acessíveis pelos usuários e outros apps porquê eles estão na memória externa, eles são arquivos que realmente não fornecem valor para o usuário fora do seu app. Quando o usuário desinstala seu app, o sistema deleta todos os arquivos do diretório privado do app na memória externa.

Por exemplo, recursos adicionais baixados pelo seu app ou arquivos de mídia temporários.

Se você precisa salvar arquivos públicos no armazenamento externo, use o método **getExternalStoragePublicDirectory()** para obter uma representação do diretório apropriado na memória externa através de um objeto **File**. O método recebe um argumento especificando o tipo de arquivo que você quer salvar para que eles sejam logicamente organizados nas pastas públicas, como **DIRECTORY\_MUSIC** ou **DIRECTORY\_PICTURES**. Por exemplo:

```
public File getAlbumStorageDir(String albumName) {  
  
    // Get the directory for the user's public pictures directory.  
  
    File file = new File(Environment.getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES), albumName);
```

```
if (!file.mkdirs()) {  
  
    Log.e(LOG_TAG, "Directory not created");  
  
}  
  
return file;  
  
}
```

Se você quer salvar arquivos que são privados para seu app, você pode obter o diretório apropriado chamando **getExternalFilesDir()** e passando para ele um nome indicando o tipo de diretório que você gostaria. Cada diretório criado dessa forma é adicionado a um diretório pai que encapsula todos os arquivos do seu app na memória externa, que o sistema irá apagar quando o usuário desinstalar seu app.

Por exemplo, aqui está um método que você pode usar para criar um diretório para um album de fotos de um usuário:

```
public File getAlbumStorageDir(Context context, String albumName) {  
  
    // Get the directory for the app's private pictures directory.  
  
    File file = new File(context.getExternalFilesDir(  
  
        Environment.DIRECTORY_PICTURES), albumName);  
  
    if (!file.mkdirs()) {  
  
        Log.e(LOG_TAG, "Directory not created");  
  
    }  
  
    return file;  
  
}
```

Se nenhum dos nomes de diretórios pré-definidos atender seus arquivos, você pode chamar o método **getExternalFilesDir()** e passar **null**. Isto retorna o diretório raiz do armazenamento privado do seu app na memória externa.

Lembre-se que **getExternalFilesDir()** cria um diretório dentro de um diretório que é apagado quando o usuário desinstala seu app. Se os arquivos que você está salvando devem permanecer disponíveis depois do usuário desinstalar seu app -- como quando seu app é a câmera e o usuário deseja manter as fotos -- você deve usar o método **getExternalStoragePublicDirectory()**.

Independente se você usa **getExternalStoragePublicDirectory()** para arquivos compartilhados ou **getExternalFilesDir()** para arquivos que são privados para seu app, é importante que você use nomes de diretórios providos pelas constantes da API como **DIRECTORY\_PICTURES**. Estes nomes de diretórios garantem que os arquivos são tratados de forma correta pelo sistema. Por exemplo, arquivos salvos no diretório **DIRECTORY\_RINGTONES** são categorizados pelo leitor de mídia do sistema como ringtones ao invés de serem tratados como música

## 9.5. Verificando espaço disponível

---

Se você sabe de antemão o quanto você irá salvar de dados no sistema, você pode descobrir se há espaço disponível no sistema sem causar uma **IOException**, chamando as funções **getFreeSpace()** ou **getTotalSpace()**. Esses métodos proveem o espaço disponível e o total de espaço no volume de armazenamento respectivamente. Esta informação também é útil para evitar encher o volume de armazenamento acima de um certo limiar.

No entanto, o sistema não garante que você poderá escrever o número de bytes que é indicado pela função **getFreeSpace()**. Se o número retornado é poucos MB maior que o arquivo que você deseja salvar, ou se o sistema está menos de 90% cheio, então é provavelmente seguro prosseguir. De outra forma, você provavelmente não deve escrever na memória do dispositivo.

**Nota:** você não é requisitado a verificar o tanto de memória disponível antes de salvar seu arquivo. Você pode, ao invés disso, tentar escrever o arquivo da forma usual, e então capturar uma **IOException** caso ela ocorra. Você pode precisar fazer isso se você não sabe ao certo quando espaço você precisa. Por exemplo, se você mudar a codificação do arquivo antes de salvá-lo, convertendo ele de PNG para JPEG, você não saberá o tamanho do arquivo de antemão.

## 9.6. Apagando um arquivo

---

Você deve sempre apagar um arquivo quando ele não for mais necessário. A forma mais direta de apagar um arquivo é ter o arquivo aberto e chamar o método **delete()** no próprio arquivo.

```
myFile.delete();
```

Se o arquivo é salvo na memória interna, você pode também pedir ao objeto **Context** que localize e apague o arquivo chamando o método **deleteFile()**:

```
myContext.deleteFile(fileName);
```



## Capítulo 10. Bases de Dados SQLite

---

Salvar dados para uma base de dados é ideal para dados estruturados ou repetitivos, como informações de contato. Esse capítulo assume que você é familiar com bases de dados SQL em geral. As APIs que você precisa para usar bases de dados no Android estão disponíveis no pacote **android.database.sqlite**.

### 10.1. Definindo um Schema e um Contract

---

Um dos princípios básicos das bases de dados SQL é o esquema: uma declaração formal de como a base de dados é organizada. O esquema é refletido nas declarações SQL que você usa para criar sua base de dados. Você pode achar ele útil para criar uma **classe associada**, chamada de **classe contrato** que especifica explicitamente o layout do seu esquema de uma forma sistemática e bem documentada.

Uma classe contrato é um container para constantes que definem nomes para URIs, tabelas e colunas. A classe contrato permite você usar as mesmas constantes dentro todas as suas classes no mesmo pacote. Isso permite que você troque o nome de uma coluna em um único lugar e essa alteração seja propagada através do código.

Uma boa forma de organizar uma classe contrato é colocar as definições que são globais para toda a sua base de dados no nível raiz da classe. então crie uma classe interna para cada tabela que enumera as colunas daquela tabela.

**Nota:** Ao implementar a interface **BaseColumns**, sua classe interna pode herdar um campo de chave primária chamada **\_ID** que algumas classes Android, como **CursorAdapter** vai esperar que ele tenha. Isso não é requerido, mas isso pode ajudar sua base de dados a funcionar de forma harmoniosa com o framework Android.

Por exemplo, esse pedaço de código define um nome de tabela e um nome de coluna para uma única tabela:

```
public final class FeedReaderContract {

    // To prevent someone from accidentally instantiating the contract class,

    // give it an empty constructor.

    public FeedReaderContract() {}

    /* Inner class that defines the table contents */

    public static abstract class FeedEntry implements BaseColumns {

        public static final String TABLE_NAME = "entry";

        public static final String COLUMN_NAME_ENTRY_ID = "entryid";

        public static final String COLUMN_NAME_TITLE = "title";

        public static final String COLUMN_NAME_SUBTITLE = "subtitle";

        ...

    }

}
```

## 10.2. Criando uma base de dados usando SQL Helper

Assim que você definiu como sua base de dados será, você deve implementar métodos que criarão e farão a manutenção das suas bases de dados e tabelas. Aqui estão rotinas típicas que criarão e apagarão uma tabela.

```
private static final String TEXT_TYPE = "TEXT";
```

```
private static final String COMMA_SEP = ",";

private static final String SQL_CREATE_ENTRIES =

    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +

    FeedEntry._ID + " INTEGER PRIMARY KEY," +

    FeedEntry.COLUMN_NAME_ENTRY_ID + TEXT_TYPE + COMMA_SEP

+

    FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE + COMMA_SEP +

    ... // Any other options for the CREATE command

    ")";
```

```
private static final String SQL_DELETE_ENTRIES =

    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```

Assim como arquivos que você salva na memória interna, o android armazena suas bases de dados num espaço de disco privado que é associado à sua aplicação. Seus dados estão seguros, porque, por padrão essa área não é acessível por outros aplicativos

Um conjunto útil de APIs estão disponíveis na classe **SQLiteOpenHelper**. Quando você usa essa classe para obter referências para sua base de dados, o sistema executa tarefas potencialmente demoradas como criar e atualizar a base de dados somente quando é preciso e não durante a inicialização do app. Tudo que você precisa fazer é chamar os métodos **getWritableDatabase()** ou **getReadableDatabase()**.

**Nota:** porque as operações podem ser demoradas, chame sempre o método **getWritableDatabase()** em uma thread de background, como **AsyncTask** ou **IntentService**.

Para usar **SQLiteOpenHelper**, cria uma subclasse que sobrescreve os métodos **onCreate()**, **onUpgrade()** e **onOpen()**. Você pode também precisar implementar **onDowngrade()**, mas isso não é requerido.

Por exemplo, aqui está uma implementação de **SQLiteOpenHelper** que usa alguns dos comandos mostrados anteriormente:

```
public class FeedReaderDbHelper extends SQLiteOpenHelper {

    // If you change the database schema, you must increment the database
    version.

    public static final int DATABASE_VERSION = 1;

    public static final String DATABASE_NAME = "FeedReader.db";

    public FeedReaderDbHelper(Context context) {

        super(context, DATABASE_NAME, null, DATABASE_VERSION);

    }

    public void onCreate(SQLiteDatabase db) {

        db.execSQL(SQL_CREATE_ENTRIES);

    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {

        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over

        db.execSQL(SQL_DELETE_ENTRIES);

        onCreate(db);

    }

}
```

```
}  
  
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int  
newVersion) {  
  
        onUpgrade(db, oldVersion, newVersion);  
  
    }  
  
}
```

Para acessar sua base de dados, instancie uma subclasse de **SQLiteOpenHelper**:

```
    FeedReaderDbHelper        mDbHelper        =        new  
FeedReaderDbHelper(getContext());
```

### 10.3. Inserir informações em uma base de dados

---

Insira dados em uma base de dados passando um objeto **ContentValues** para o método **insert()**:

```
// Gets the data repository in write mode
```

```
SQLiteDatabase db = mDbHelper.getWritableDatabase();
```

```
// Create a new map of values, where column names are the keys
```

```
ContentValues values = new ContentValues();
```

```
values.put(FeedEntry.COLUMN_NAME_ENTRY_ID, id);
```

```
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
```

```
values.put(FeedEntry.COLUMN_NAME_CONTENT, content);
```

```
// Insert the new row, returning the primary key value of the new row
```

```
long newRowId;
```

```
newRowId = db.insert(
```

```
    FeedEntry.TABLE_NAME,
```

```
    FeedEntry.COLUMN_NAME_NULLABLE,
```

```
    values);
```

O primeiro argumento para o **insert()** é simplesmente o nome de tabela. O segundo argumento provê o nome de uma coluna na qual o framework pode inserir **NULL** no evento que o **ContentValue** é vazio (se você configurar esse para **null**, então o framework não irá inserir uma linha onde não há valores).

#### 10.4. Ler informações de uma base de dados

---

Para ler de uma base de dados, use o método **query()**, passando para ele seu critério de seleção e as colunas desejadas. O método combina elementos de **insert()** e **update()**, exceto a lista de colunas define os dados que você deseja obter, so invés de inserir. Os resultados da consulta são restornados para você em um objeto **Cursor**.

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();
```

```
// Define a projection that specifies which columns from the database
```

```
// you will actually use after this query.
```

```
String[] projection = {

    FeedEntry._ID,

    FeedEntry.COLUMN_NAME_TITLE,

    FeedEntry.COLUMN_NAME_UPDATED,

    ...

};

// How you want the results sorted in the resulting Cursor

String sortOrder =

    FeedEntry.COLUMN_NAME_UPDATED + " DESC";

Cursor c = db.query(

    FeedEntry.TABLE_NAME, // The table to query

    projection,           // The columns to return

    selection,             // The columns for the WHERE clause

    selectionArgs,         // The values for the WHERE clause

    null,                  // don't group the rows

    null,                  // don't filter by row groups

    sortOrder              // The sort order

);
```

Para acessar uma linha no cursor, use um dos métodos **move** do **Cursor**, que você deve sempre chamar antes de começar a ler valores. Geralmente, você deve começar chamando **moveToFirst()**, que coloca a posição de leitura na primeira entrada dos resultados. Para cada linha, você pode ler um valor de coluna chamando um dos métodos **get** do cursor, como **getString()** ou **getLong()**.

Para cada um dos métodos **get**, você precisa passar a posição da coluna que você deseja, que você pode obter chamando os métodos **getColumnIndex()** ou **getColumnIndexOrThrow()**. Por exemplo:

```
cursor.moveToFirst();

long itemId = cursor.getLong(

    cursor.getColumnIndexOrThrow(FeedEntry._ID)

);
```

### 10.5. Apagar informações da base de dados

---

Para apagar linhas de uma tabela, você precisa prover um critério de seleção que identifica as linhas. A API de bases de dados provê um mecanismo para criar um critério de seleção que protege contra SQL injection. O mecanismo divide a especificação da seleção em uma cláusula de seleção e nos argumentos de seleção. A cláusula define as colunas que você está procurando, e também permite você combinar testes de colunas. Os argumentos são valores a testar que são anexados à cláusula. Porque o resultado não é tratado do mesmo modo que um SQL regular, ele é imune a SQL injection.

```
// Define 'where' part of query.
```

```
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
```

```
// Specify arguments in placeholder order.
```



```
String[] selectionArgs = { String.valueOf(rowId) };
```

```
// Issue SQL statement.
```

```
db.delete(table_name, selection, selectionArgs);
```

## 10.6. Atualizar informações na base de dados

---

Quando você precisa modificar um subconjunto dos valores dos valores da sua base de dados, use o método **update()**.

Atualizar os dados da tabela combina a sintaxe de **insert()** com a sintaxe do **delete()**.

```
SQLiteDatabase db = dbHelper.getReadableDatabase();
```

```
// New value for one column
```

```
ContentValues values = new ContentValues();
```

```
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
```

```
// Which row to update, based on the ID
```

```
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
```

```
String[] selectionArgs = { String.valueOf(rowId) };
```

```
int count = db.update(
```

```
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,
```

```
    values,
```

```
    selection,selectionArgs);
```

## Capítulo 11. Integração com Serviço Rest

---

REST é um conjunto de princípios que definem como Web Standards como HTTP e URIs devem ser usados:

- Utilizar os métodos do protocolo HTTP para representar as operações que pode ser executadas pelo serviço;
- Expor as informações através de URLs representativas, similar a uma estrutura de diretórios;
- O serviço não deve armazenar estado entre requisições;
- Transmitir os dados em formato XML e/ou JSON;
- O uso de *hypermedia* para representar possíveis transições.

Nosso foco não é falar sobre REST, então para saber mais você pode consultar [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm).

### 11.1. JSON

---

JavaScript Object Notation é um formato simples utilizado para representar dados, voltado principalmente para a conversão de dados estruturados para a forma textual. Esse formato é capaz de representa quatro tipos primitivos (números, strings, booleanos e null) e dois tipos estruturados (objetos e arrays). Exemplo:

```
{  
  
  "nome": "Wagner Gomes",  
  
  "idade": 31,  
  
  "telefones": [  
  
    "(31)0000-0000",  
  
    "(33)1111-111"
```

]

}

O Android possui classes nativas para trabalhar com JSON, e uma delas é o *JSONObject*. Só que para facilitar o nosso trabalho vamos utilizar uma biblioteca chamada Gson que é do próprio Google.

## 11.2. Gson

Para facilitar o trabalho do desenvolvedor o Google criou o Gson com os principais objetivos:

- Prover uma interface simples para ler e exportar no formato JSON.
- Permitir que objetos pré-existent e que não possam ser alterados sejam convertidos para e partir de JSON.
- Suporte ao generics do Java.
- Representação customizada de objetos.
- Suporte a tipos complexos de objetos.

E veja como é simples converter uma String JSON em um objeto:

```
public User loadUserFromJSONGson(String jsonString) {

    Gson gson = new Gson();

    User user = gson.fromJson(jsonString, User.class);

    return user;

}
```

Para saber mais, consulte: <https://code.google.com/p/google-gson/>.

### 11.3. Requisições HTTP

O Android já tem integrado a biblioteca HttpClient do Apache Commons, é o que vamos usar na nossa aplicação. A seguir temos um exemplo de uma requisição GET:

```
public List<Job> requestGet() {

    List<Job> list = null;

    try {

        HttpClient httpClient = new DefaultHttpClient();

        HttpGet httpGet = new
HttpGet("http://arimateia.info:3001/jobs");

        HttpResponse httpResponse = httpClient.execute(httpGet);

        if (httpResponse != null) {

            if (httpResponse.getStatusLine().getStatusCode() == 200) {

                String result =
convertInputStreamToString(httpResponse.getEntity().getContent());

                Gson gson = new Gson();
```

```

        list = gson.fromJson(result, new
TypeToken<List<Job>>().getType());

    }

}

} catch (IOException e) {

    Log.d("HTTP_GET", e.getMessage(), e);

}

return list;

}

```

Vamos ver outro exemplo de uma requisição POST:

```

public boolean requestPOST(Job job) {

    boolean result = false;

    Gson gson = new Gson();

    String json = gson.toJson(job);

    try {

        HttpClient httpClient = new DefaultHttpClient();

        HttpPost httpPost = new
HttpPost("http://wpgomes.info:3001/jobs/new");

```

```
httpPost.setEntity(new StringEntity(json, "UTF-8"));

httpPost.setHeader("Content-type", "application/json");

HttpResponse httpResponse = httpClient.execute(httpPost);

if (httpResponse != null) {

    if (httpResponse.getStatusLine().getStatusCode() == 200) {

        result = true;

    }

}

} catch (IOException e) {

    Log.d("HTTP_POST", e.getMessage(), e);

}

return result;

}
```

Para fazer requisições HTTP é importante saber que o Android restringem qualquer acesso a Internet na thread principal, para isso temos duas soluções executar as requisições através de uma AsyncTask que veremos a seguir ou através de um serviço que veremos no próximo capítulo.

## 11.4. AsyncTask

*AsyncTask* permite o uso adequado e fácil de threads. Ela permite realizar operações em background e publicar resultados na UI Thread, sem ter que manipular threads e handlers. A *AsyncTask* possui três métodos importantes:

Método	Descrição
<code>onPreExecute()</code>	Método executado antes de a thread iniciar, sendo uma boa oportunidade para exibir uma janela de progresso ao usuário ou uma mensagem de "por favor, aguarde".
<code>doInBackground()</code>	Método executado sobre uma thread separada, devendo conter todo o processamento pesado. Ele pode retornar um objeto qualquer, o qual será passado como parâmetro para o método <i>onPostExecute()</i> . É aqui que a thread executa, mas isso é feito automaticamente para você.
<code>onPostExecute()</code>	Método executado na thread principal, a famosa thread de interface UI Thread, podendo atualizar os componentes da tela. Ele é chamado internamente, utilizando um handler, encapsulando esse trabalho.

Vejo o exemplo de uma requisição Delete, com uso da AsyncTask:

```
public class DeleteJobTask extends AsyncTask<Job, Void, Boolean> {

    private DeleteJobListener deleteJobListener;

    public DeleteJobTask(DeleteJobListener deleteJobListener) {

        this.deleteJobListener = deleteJobListener;

    }

    @Override

    protected Boolean doInBackground(Job... jobs) {

        Boolean result = false;

        Job job = jobs[0];

        try {

            HttpClient httpClient = new DefaultHttpClient();

            HttpDelete httpDelete = new
            HttpDelete("http://wpgomes.info:3001/jobs/"+ job.getId());
```



```
httpDelete.setHeader("Content-type", "application/json");
```

```
HttpResponse httpResponse = httpClient.execute(httpDelete);
```

```
if (httpResponse != null) {
```

```
    if (httpResponse.getStatusLine().getStatusCode() == 200) {
```

```
        result = true;
```

```
    }
```

```
}
```

```
} catch (Exception e) {
```

```
    Log.d("HTTP_DELETE", e.getMessage(), e);
```

```
}
```

```
return result;
```

```
}
```

```
@Override
```

```
protected void onPostExecute(Boolean result) {
```

```
    super.onPostExecute(result);
```

```
if (deleteJobListener != null) {
```

```
deleteJobListener.onDeleteListener(result);
```

```
}
```

```
}
```

```
public interface DeleteJobListener {
```

```
    public void onDeleteListener(boolean success);
```

```
}
```

```
}
```

## Capítulo 12. BroadcastReceiver e Service

---

### 12.1. Broadcast Receiver

---

A classe `android.content.BroadcastReceiver` é utilizada para responder a determinados eventos enviados por uma `intent`, como executar determinada aplicação ao receber uma mensagem SMS, uma ligação telefônica ou qualquer outra ação customizada da aplicação, por exemplo.

```
public class OpenActivityReceiver extends BroadcastReceiver
{

    @Override

    public void onReceive(Context context, Intent intent) {

        Intent intentHello = new Intent(context,
        HelloActivity.class);

        context.startActivity(intentHello);

    }

}
```

Existem duas formas de se configurar um `BroadcastReceiver`, a forma estática que fica no arquivo `AndroidManifest.xml` ou de forma dinâmica pela API chamando o método `registerReceiver(receiver, filter)`. A seguir podemos visualizar as duas formas:

1. Configurar o arquivo `AndroidManifest.xml` e inserir uma tag `<receiver>`. A tag `<intent-filter>` é utilizada para configurar para qual ação e categoria a classe deve ser executada.

```

        <receiverandroid:name=".BootReceiver"
        android:enabled="true">

            <intent-filter>

                <action
                android:name="android.intent.action.BOOT_COMPLETED"/>

                <category
                android:name="android.intent.category.DEFAULT"/>

            </intent-filter>

        </receiver>

```

- Utilizar os método `context.registerReceiver(receiver, filter)` dentro do código para registrar dinamicamente o `BroadcastReceiver`. O primeiro parâmetro chamado de receiver é uma instância de uma classe-fila de `BroadcastReceiver`, e o segundo é uma instância de uma classe `IntentFilter`, que por sua vez tem a configuração da ação e categoria.

```

private void registerReceiver() {

    receiver = new OpenActivityReceiver();

    IntentFilter intentFilter = new
    IntentFilter("ABRIR_HELLO_ACTIVITY");

    registerReceiver(receiver, intentFilter);

}

```

Registrar o receiver dinamicamente deve ser utilizado usando é necessário que ele esteja apto a executar somente quando determinada tela da aplicação estiver executando, caso contrário, não. Já se o receiver for definido de forma estática no arquivo `AndroidManifest.xml` ele pode executar a qualquer momento que

receber uma mensagem (intent), mesmo que a aplicação não esteja executando. Ele pode ser até configurado como o ponto de partida da aplicação, para executá-la assim que receber determinados eventos.

Sua aplicação também pode disparar mensagens para algum receiver da seguinte forma:

```
private void sendBroadcast() {  
  
    sendBroadcast(new Intent("ABRIR_HELLO_ACTIVITY"));  
  
}
```

## 12.2 Ciclo de Vida do BroadcastReceiver

---

Um *BroadcastReceiver* é válido somente durante a chamada ao método *onReceive(context, intent)*. Depois disso o sistema operacional encerrará seu processo para liberar memória. O Android permite que um *BroadcastReceiver* seja executado no máximo em 10 segundos, então é recomendado que mensagem seja processada por um *Service* por exemplo.

## 12.3. Service

---

A classe *android.app.Service* é utilizada para executar um serviço em segundo plano, geralmente vinculado a algum processo que deve executar por tempo indeterminado e tem um alto consumo de recursos, memória e CPU.

```
public class MyService extends Service {  
  
    public MyService() {  
  
    }  
  
}
```

@Override

```
public IBinder onBind(Intent intent) {
```

```
    // TODO: Return the communication channel to the
    service.
```

```
    throw new UnsupportedOperationException("Not yet
    implemented");
```

```
}
```

@Override

```
public int onStartCommand(Intent intent, int flags, int startId)
```

```
{
```

```
    return super.onStartCommand(intent, flags, startId);
```

```
}
```

```
}
```

Assim como o *BroadcastReceiver*, o *Service* precisa ser declarado no arquivo *AndroidManifest.xml*:

```
<service android:name=".MyService" android:enabled="true"
    android:exported="false" >
```

```
</service>
```

Há dois métodos que pode ser usado para iniciar um serviço:

Método	Descrição
<code>startService(intent)</code>	Método utilizado para iniciar um serviço que fixa executando por tempo indeterminado, até que o método <i>stopService(intent)</i> seja chamado ou até que o próprio serviço termine sua própria execução chamando o método <i>stopSelf()</i> ;
<code>bindService(intent, conn, flags)</code>	Este método pode iniciar um serviço se ele ainda não está executando ou simplesmente conectar-se a ele. Ao estabelecer a conexão é possível recuperar uma referência de uma classe ou interface que pode manipular o serviço.

Quando um *Service* é criado com o método *bindService(intent, conn, flags)* o processo do serviço está vinculado a quem se conectou a ele, por exemplo, uma *activity*. Então se a *activity* for encerrada o serviço também será. Isso não corre quando o método *startService(intent)* é chamado, que por sua vez deixa o serviço executando por tempo indeterminado. Vamos executar o projeto de exemplo desse capítulo, para entender tudo como funciona.

## 12.4. Intent Service

Diferente do *Service* que precisa chamar o método *stopSelf()* para avisar ao Android que já acabou sua execução, o *IntentService* faz isso de forma automática. O trecho de código a seguir demonstra um template simples de como funciona:

```
public class MyIntentService extends IntentService {

    /**
     * Creates an IntentService. Invoked by your subclass's constructor.
     *
     * @param name Used to name the worker thread, important only for
    debugging.
     */

    public MyIntentService(String name) {

        super(name);

    }

    /** Este método exxecuta em uma thread.

     * Quando ele terminar, o método stopSelf() será chamado
    automaticamente
     *
     */

    @Override
```



```
protected void onHandleIntent(Intent intent) {  
  
    //O processamento precisar ser feito aqui.  
  
}  
  
}
```

## Capítulo 13. Google Maps e GPS

Em dezembro de 2012, o Google lançou a nova API de mapas, chamada de Google Maps API v2, o *framework* de maps foi criado utilizando vetores para suportar as visualizações 2D e 3D, o que também possibilita um ganho significativo no desempenho.

Para utilizar o Google Maps Android API v2, é necessário adicionar a dependência do Google Play Service, que contém as classes da API de mapas.

```
dependencies {

    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation 'com.android.support:appcompat-
v7:21.0.3'

    implementation 'com.google.android.gms:play-
services-maps:15.0.1'

}
```

Depois de adicionado a dependência do Google Play Service precisamos adicionar uma *meta-data* dentro da tag `<application>` para que o Google Play Service identifique em qual versão do mesmo, nosso projeto foi compilado.

```
<meta-data

    android:name="com.google.android.gms.version"

    android:value="@integer/google_play_services_version" />
```

O Google Play Service foi uma solução encontrada pelo Google para resolver o problema de atualizações dos seus principais serviços. Através do

Google Play Service todas as versões do Android podem contar os as novidades dos serviços do Google.

### 13.1. Configurando o AndroidManifest.xml

---

Para utilizar os Google Maps, é necessário adicionar algumas permissões ao AndroidManifest.xml, como vemos abaixo:

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

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

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

<!-- The following two permissions are not required to use
Google Maps Android API v2, but are recommended. -->

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

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

O OpenGL ES v2 é requisito para funcionar o Google Maps, é um recurso para possibilitar os uso de Maps em 3D.

```
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
```

Por fim precisamos adicionar a chave de acesso do Google Maps, a chave é baseada no certificado digital que é assinado o projeto. No próximos tópico, aprenderemos a gerá-la.

## 13.2. Obtendo a chave dos mapas no Google APIs Console

---

Para obter a chave primeiramente precisamos criar um projeto no Google Developer Console, conforme os passos abaixo:

1. Abra o Google Developers Console.
2. Se você não tiver criado um projeto de API ainda, clique em **Criar Projeto**.
3. Forneça um nome para o projeto e clique em **Criar**.
4. Uma vez que o projeto foi criado, aparecerá uma página que exibe o ID do projeto e Número do projeto.

Agora é preciso habilitar o **Google Maps Android API v2**:

1. Na barra lateral à esquerda, selecione **APIs e autenticação**.
2. Na lista exibida de **APIs**, habilite **Google Maps Android API v2**.

Para finalizar, só precisamos obter uma API Key:

1. Na barra lateral à esquerda, selecione **APIs e autenticação > Credenciais**.
2. Em **Acesso público à API**, clique em **Criar nova chave**.
3. Uma nova caixa de diálogo, clique em **Create new Andorid key...**
4. Na janela de configuração, precisamos fornecer o SHA-1 fingerprints do certificado de assinatura da aplicação, que pode ser obtido da seguinte forma:

```
keytool -list -v -keystore ~/.android/debug.keystore -alias  
androiddebugkey -storepass android -keypass android
```

5. Em seguida devemos adicionar um ponto e vírgula, e o pacote principal da aplicação.
6. Clique em **Criar**. Na página atualizada, você deve copiar a **Chave de API**.
7. Agora só adicionar a chave na tag <application> no AndroidManifest.xml

```
<meta-data android:name="com.google.android.maps.v2.API_KEY"  
android:value="API_KEY"/>
```

### 13.3. Classe MapFragment

---

A classe *MapFragment* é responsável por renderizar o mapa na tela, é importante frisar que a classe *MapFramet* é compatível com a API Level 12 ou superior. Caso o projeto suporte uma versão anterior, é necessário utilizar a classe *SupportMapFragment*. Agora veja uma exemplo de como usamos a classe *MapFragment*.

```
<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"
android:paddingLeft="@dimen/activity_horizontal_margin"
tools:context=".MainActivity">
```

```
<fragment
```

```
    android:id="@+id/mapFragment"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

```
    android:name="com.google.android.gms.maps.MapFragment"/>
```

```
</FrameLayout>
```

### 13.4. Classe GoogleMap

---

A classe *com.google.android.gms.maps.GoogleMap* representa o mapa do Google e, por meio dela, podemos controlar a visualização do mapa e definir o

zoom, aparência, localização, inserir marcadores, etc. Veja um exemplo de como recuperar um objeto do tipo GoogleMap:

```
//Recuperando fragment do Maps através do ID
```

```
MapFragment mapFragment =  
(MapFragment)getFragmentManager().findFragmentById(R.id.mapFra  
gment);
```

```
//Recuperando o objeto GoogleMap
```

```
mapFragment.getMapAsync(new OnMapReadyCallback()  
{  
  
    @Override  
  
    public void onMapReady(GoogleMap googleMap) {  
  
        mGoogleMap = googleMap;  
  
    }  
  
});
```

Deste ponto em diante vamos ver pequenos exemplos de algumas funções importantes do mapa:

### 13.5. Tipo de Mapa

---

GoogleMap.MAP\_TYPE\_NONE - Modo de visualização mais simples do mapa, de forma que nenhuma informação extra é exibida.

GoogleMap.MAP\_TYPE\_NORMAL - Modo de visualização-padrão dos mapas, onde podemos visualizar as ruas, estradas e rios.

GoogleMap.MAP\_TYPE\_SATELLITE - Modo de visualização com os dados do satélite.

GoogleMap.MAP\_TYPE\_HYBRID - Modo de visualização com os dados fotográficos do satélite, com os mapas das ruas, Este é o modo de satélite mais detalhado.

GoogleMap.MAP\_TYPE\_TERRAIN - Modo de visualizaçnao que exhibe os dados topográficos do mapa.

```
mGoogleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

### 13.6 Localização do mapa

---

Para centralizar o mapa em uma determinada região é preciso utilizar a classe CameraUpdate, e através da Latitude e Longitude podemos movimentar o mapa para região desejada.

```
LatLng latLng = new LatLng(-19.9178713,-43.9603116);

CameraUpdate update =
CameraUpdateFactory.newLatLngZoom(latLng, 15); //O número 15 é o
zoom

mGoogleMap.moveCamera(update);
```

### 13.7. Marcadores

---

Os marcadores permitem adicionar figuras ao mapa, para marcar os locais de interesse, a fim de exibir informações importantes e interagir com o usuário. O marcador é representado pela classe *Marker*, veja o exemplo:

```

private Marker adicionarMarcador(GoogleMap googleMap, LatLng
latLng) {

    MarkerOptions markerOptions = new MarkerOptions();

    markerOptions.position(latLng).title("IGTI").snippet("Rua Roma,
561 - Santa Lúcia - Belo Horizonte/MG");

    markerOptions.icon(BitmapDescriptorFactory.fromResource(R.drawable.ic_pi
n));

    Marker marker = googleMap.addMarker(markerOptions);

    return marker;

}

```

### 13.8. Polyline - Desenha uma linha no mapa

A classe Polyline permite desenhar uma linha no mapa, que, na verdade, é uma sequência de coordenadas.

```

private void adicionaPolyline(GoogleMap map, LatLng latLng1, LatLng
latLng2) {

    PolylineOptions line = new PolylineOptions();

    line.add(latLng1);

    line.add(latLng2);

    line.color(Color.BLUE);

    Polyline polyline = map.addPolyline(line);

    polyline.setGeodesic(true);
}

```



}

### 13.9. GPS

Para trabalhar com a API de Localização do Android é preciso adicionar as permissões `ACCESS_FINE_LOCATION` e `ACCESS_COARSE_LOCATION` ao arquivo `AndroidManifest.xml`, para permitir que a aplicação utilize o GPS por hardware ou triangulação.

Para obter a localização do GPS, basta criar uma classe que implemente a interface `android.location.LocationListener`. Geralmente, essa interface é implementada pela própria activity que está criando o mapa. Há vários métodos nessa interface, mas, na prática utilizaremos apenas o método `onLocationChanged(location)`, que é chamado automaticamente pelo Android sempre que a localização do GPS for alterada.

```
private void requestLocationUpdate() {
```

```
    LocationManager locationManager =  
    (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

```
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  
    0, 0, new LocationListener() {
```

```
        @Override
```

```
        public void onLocationChanged(Location location) {
```

```
            mMyLocation = location;
```

```
        }
```

@Override

```
public void onStatusChanged(String provider, int status,  
Bundle extras) {  
  
}
```

@Override

```
public void onProviderEnabled(String provider) {  
  
}
```

@Override

```
public void onProviderDisabled(String provider) {  
  
}  
}
```

A seguir, veja a explicação de cada parâmetro do método *requestLocationUpdates()*:

Parâmetro	Descrição
String provider	Nome do provedor de localização, que possui as seguintes constantes <i>LocationManager.GPS_PROVIDER</i> ou <i>LocationManager.NETWORK_PROVIDER</i> .
long minTime	Tempo em milissegundos que representa o intervalo mínimo de tempo em que a aplicação de receber atualização sobre as localizações. Se o valor for 0 (zero), o método será chamado sempre que a localização for alterada.
long minDistance	Distância em metros que representa a distância mínima necessária a ser percorrida para a aplicação receber atualizações sobre as localizações. Se o valor for 0 (zero), o método será chamado sempre que a localização for alterada.
LocationListener listener	Implementação da interface <i>LocationListener</i> que receberá as atualizações das coordenadas pelo método <i>onLocationChanged(location)</i>

## Capítulo 14. Notificações

### 14.1. Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) é um serviço gratuito que ajuda os desenvolvedores a enviarem mensagens dos servidores para suas aplicações. As mensagens podem ter o tamanho de até 4KB de dados, então com isso você consegue enviar mensagens apenas dizendo que tem novidades na sua aplicação, ou até mesmo enviar pequenas atualizações de conteúdo.

O FCM é a evolução das APIs do Google Cloud Messaging.

A implementação do FCM inclui um *Firebase Cloud Messaging*, um servidor de aplicações 3rd-Party que interage com o servidor do Google (Firebase) e o aplicativo Android (ou IOs e até mesmo web) habilitado para usar FM, imagem abaixo exemplifica bem esse conceito:



## 14.2. Criando um Firebase Project

---

Para iniciar o uso do FCM primeiramente precisamos criar um projeto no Firebase Console, conforme os passos abaixo:

5. Abra o Firebase Console (<http://firebase.google.com>).
6. Se você não tiver criado um projeto de API ainda, clique em **Adicionar Projeto**.
7. Forneça um nome para o projeto e clique em **Criar**.
8. Uma vez que o projeto foi criado, aparecerá uma página que exibe o projeto e a opção para integração. No nosso caso seleciona e integração com Android.
9. Registre o App em questão, faça o download do json fornecido e configurações necessárias no gradle.build do projeto.

Agora é preciso habilitar o FCM Service:

3. Na barra lateral à esquerda, selecione **Ampliar** → **Cloud Messaging**
4. Você poderá enviar uma mensagem teste utilizando a plataforma do Firebase.

## 14.3. FCM Client

---

Agora que já criamos nosso projeto no Google Developer Console, precisamos configurar nossa aplicação para receber as notificações, segua os passos abaixo:

1. Adicione o Google Play Service como dependência:

```
dependencies {  
  
    implementation 'com.google.firebase:firebase-messaging:17.0.0'  
  
}
```

2. Adicione as seguintes permissões no Manifest:

```

<service

    android:name=".MyFirebaseMessagingService">

    <intent-filter>

        <action
        android:name="com.google.firebase.MESSAGING_EVENT"/>

    </intent-filter>

</service>

```

### 3. Verificar se tem Google Play Service:

```

private boolean checkPlayServices() {

    int                resultCode                =
    GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);

    if (resultCode != ConnectionResult.SUCCESS) {

        if (GooglePlayServicesUtil.isUserRecoverableError(resultCode))

        {

            GooglePlayServicesUtil.getErrorDialog(resultCode, this,

                PLAY_SERVICES_RESOLUTION_REQUEST).show();

        } else {

            Log.i(TAG, "This device is not supported.");

            finish();

        }

        return false;

    }
}

```

```

return true;

}

```

#### 4. Registrar device no FCM:

```

public class MyFirebaseInstanceIdService extends FirebaseInstanceIdService {

    public void onTokenRefresh() {
        // Get updated InstanceID token.
        String refreshedToken = FirebaseInstanceId.getInstance().getToken();
        Log.d( tag: "FIREBASEINSTANCEID", msg: "Refreshed token: " + refreshedToken);
        // TODO: Implement this method to send any registration to your app's servers.
        sendRegistrationToServer(refreshedToken);
    }

    private void sendRegistrationToServer(String refreshedToken) {

    }

}

```

#### 5. Agora precisamos criar as classes responsáveis por receber a mensagem e criar a notificação. Lembrando que o `FirebaseMessagingService` também precisa ser declarado no `AndroidManifest`.

```

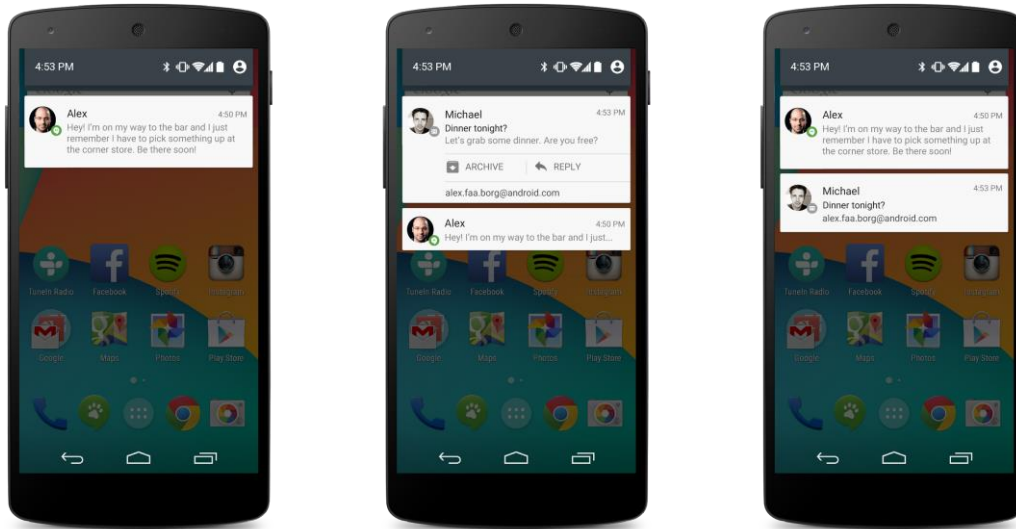
<service android:name=".fcm.MyFirebaseMessagingService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>

```

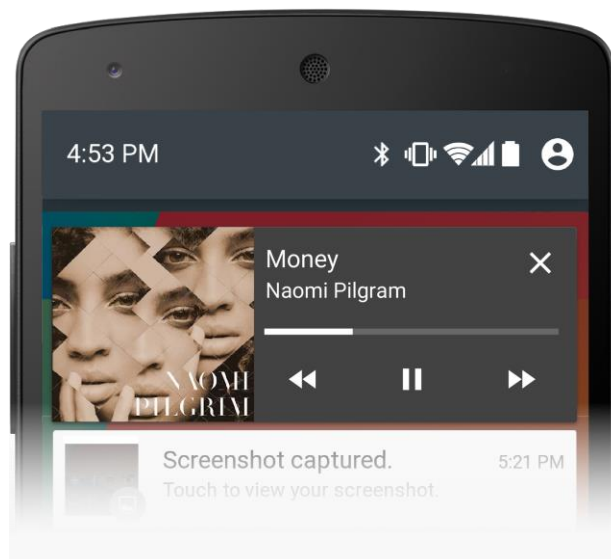
### 14.4. Notifications

As notificações podem ser usadas para alerta os usuários sobre as novidades do seu aplicativo, ou até mesmo para avisá-lo que tem novas mensagens para serem lidas em um aplicativo de bate-papo.

A cada versão do Android as notificações foram passando por transformações e ganhando importância no ecossistema Android. O exemplo disso é as novas funcionalidades que foram adicionadas no Android 5.0, que é demonstrado nas imagens abaixo:



Notificações com ações e mensagens extendidas.



Notificações para aplicativos de música.

Para manter a compatibilidade com as versões antigas o Google disponibilizou um conjunto de classes dentro do *Support Library*, que podemos ver no exemplo a seguir:

```
NotificationCompat.Builder mBuilder = new
NotificationCompat.Builder(this)

.setSmallIcon(R.drawable.notification_icon)
```



```
.setContentTitle("My notification").setContentText("Hello World!");
```

```
// Sets an ID for the notification
```

```
int mNotificationId = 001;
```

```
// Gets an instance of the NotificationManager service
```

```
NotificationManager mNotifyMgr =
```

```
    (NotificationManager)
```

```
    getSystemService(NOTIFICATION_SERVICE);
```

```
// Builds the notification and issues it.
```

```
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```

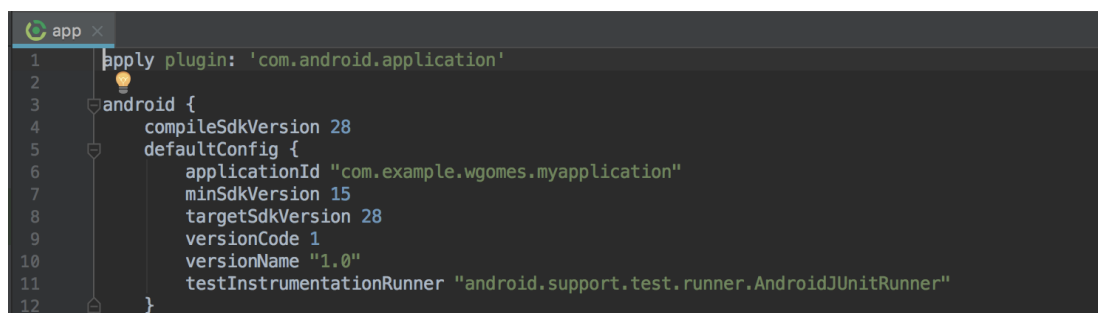
Os campos *small icon*, *title* e *content text* são os campos mínimos obrigatório, para lançar uma notificação. E para saber mais sobre o assunto acesse: <http://developer.android.com/training/notify-user/index.html>

## Capítulo 15. Publicação na Google Play

Para submeter aplicativos na Google Play, você irá precisar de uma conta do Google e de um cartão de crédito internacional para o pagamento da taxa de U\$ 25,00. Para saber mais sobre esse registro acesse: <http://developer.android.com/distribute/googleplay/developer-console.html>.

### 15.1. Preparação da Aplicação

O primeiro item que devemos conferir são os atributos *applicationId*, *versionCode* e *versionName*, que se encontram no arquivo **build.gradle** do módulo *app*.



```

1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 28
5      defaultConfig {
6          applicationId "com.example.wgomes.myapplication"
7          minSdkVersion 15
8          targetSdkVersion 28
9          versionCode 1
10         versionName "1.0"
11         testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
12     }

```

O *applicationId* deve ser único para identificar a sua aplicação no Google Play por isso é comum utilizar o *package* da sua aplicação. Uma vez publicado, não será possível alterar o *applicationId*. O atributo *versionCode* é um atributo importante que identifica a versão atual da sua aplicação. Com ele a Google Play consegue verificar se há atualizações a serem realizadas no aplicativo.

A cada nova versão o *applicationId* deve ser incrementado. Já o atributo *versionName* é de uso livre para o desenvolvedor nomear a versão da forma que quiser. Existe atributo é exibido no Google Play e também no próprio dispositivo do usuário.

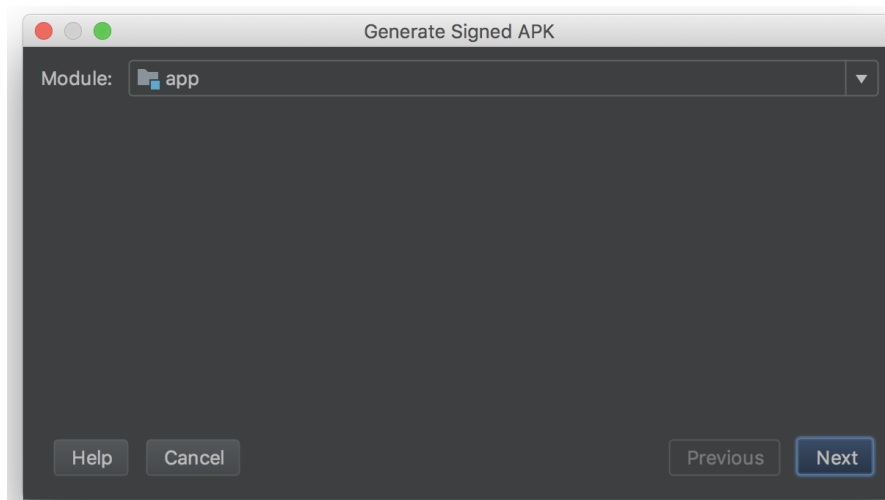
## 15.2. Assinando a Aplicação

Para submeter uma aplicação a Google Play exige que toda aplicação seja assinada digitalmente, que é uma forma de identificar o aturo da aplicação. Enquanto estávamos desenvolvendo nossa aplicação o Android Studio já assina aplicação com uma chave de debug, por isso conseguimos roda nossa aplicação no smartphone ou tablet.

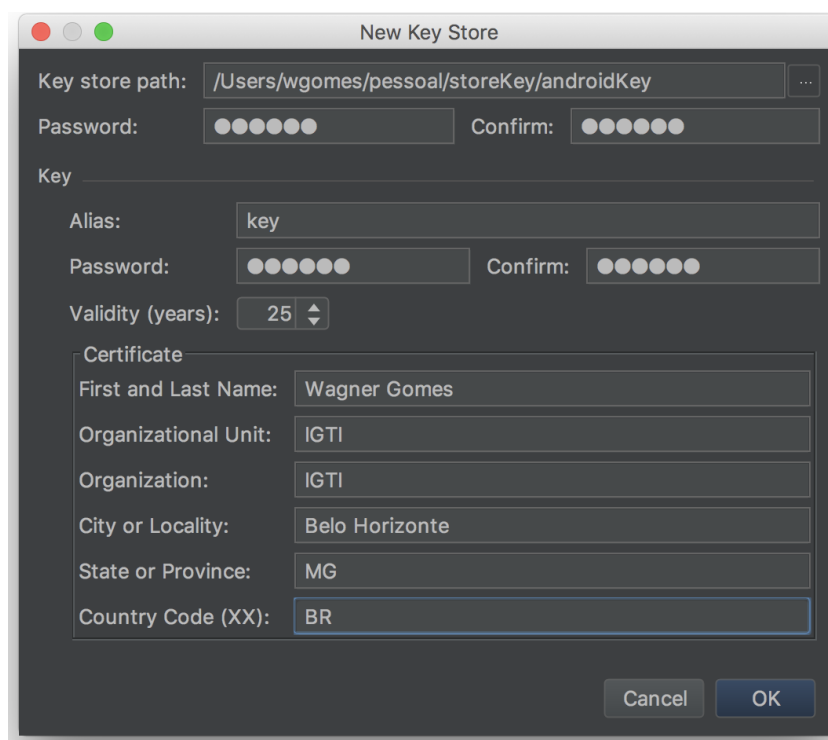
Outro ponto importante sobre o certificado, é que a aplicação precisa ser assinada sempre com a mesma chave. Assim as atualizações ficam de forma transparente para o usuário. Quando você submete uma versão nova para a Google Play o mesmo valida se o *apk* foi assinado com a chave da primeira versão. Caso contrário, não será possível realizar a atualização.

A seguir, veja como assinar sua aplicação e exporta o *apk* assinado:

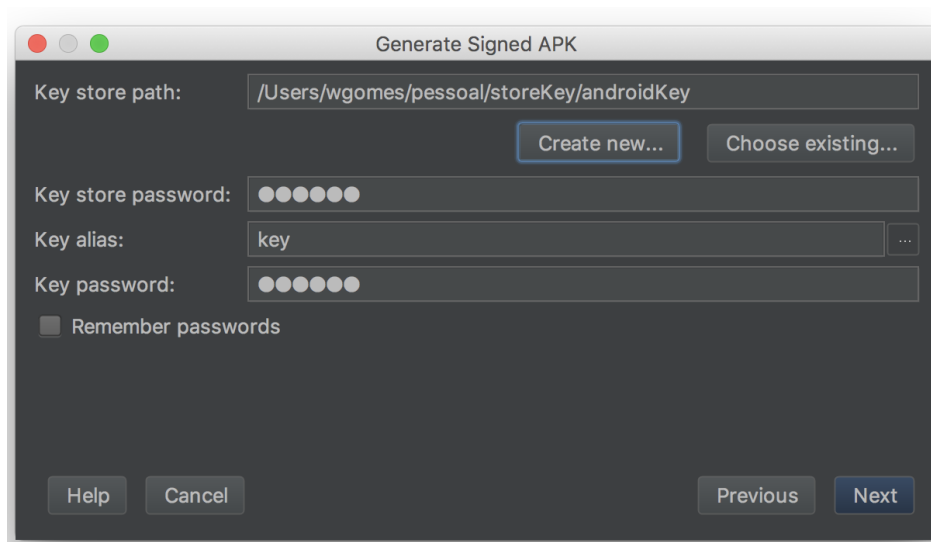
1. Na barra de navegação vá em Build > Generate Signed APK.
2. Na janela Generate Signed APK Wizard, escolha o módulo que deseja assinar clique em Next.



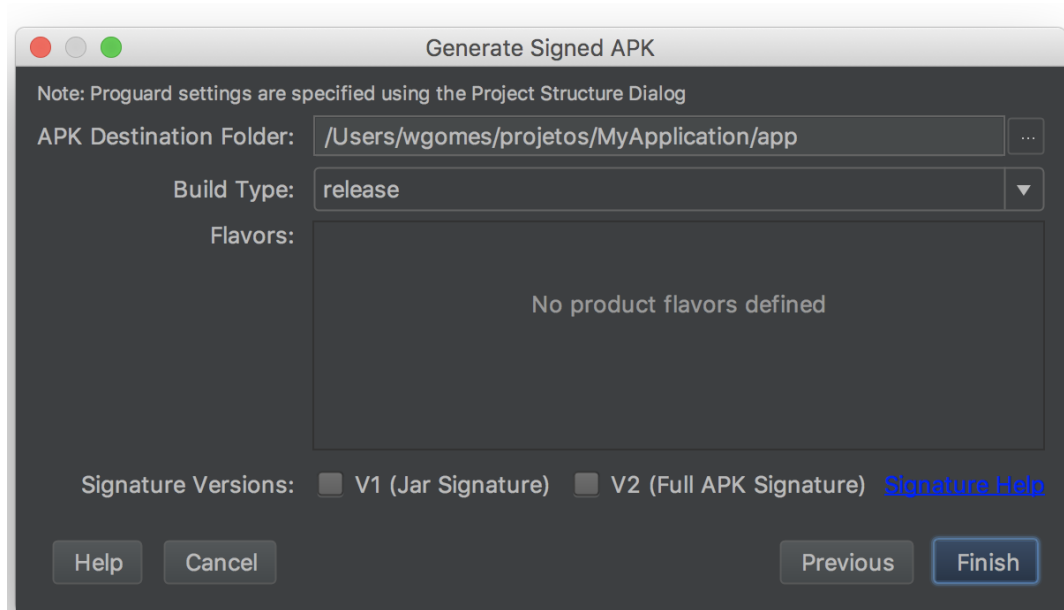
3. Agora clique em **Create New**, e preencha todos os campos conforme figura abaixo. Sua chave deve ter uma vida útil de no mínimo 25 anos, para que possa assinar atualizações.



4. Depois de criado a sua chave, vamos selecioná-la na janela Generate Signed APK Wizard, e informar a senha para key store e key alias, conforme imagem abaixo.



5. Por último precisamos escolher qual tipo de build vamos gerar e onde vamos salvar o nosso apk.






### 15.3. Conhecendo o Google Play Developer Console

Para ter acesso ao Google Play Developer Console é preciso pagar uma taxa de U\$25 através do Google Wallet. Mais informações você pode encontrar em <http://developer.android.com/distribute/googleplay/start.html>.

Através do Developer Console você pode publicar apps, configurar preços, região de distribuição, e etc. Outro item importante é o gerenciador de as fases do projeto, no Developer Console você pode distribuir sua aplicação nas fases Alpha, Beta e Produção. Como veremos nos próximos tópicos.

#### 15.3.1. All Applications

Aqui você tem uma visão geral de todos os seus aplicativos, onde você pode escolher o detalhe de cada aplicativo ou fazer o upload de novas aplicações.

<div>  Google play Developer Console <input type="text"/> <span>@gmail.com Sign out Give us feedback</span> </div>						
<div> <div> <div>☰</div> <div>All applications</div> </div> <div> <div>📊</div> <div>Financial reports</div> </div> <div> <div>⚙️</div> <div>Settings</div> </div> <div> <div>📢</div> <div>Announcements</div> </div> </div> <div> <div>ALL APPLICATIONS</div> <div>+ Add new application</div> </div>						
APP NAME	PRICE	ACTIVE / TOTAL INSTALLS ?	AVG. RATING / TOTAL #	CRASHES & ANRS ?	LAST UPDATE	STATUS
 <b>Animal Translator 1.1</b>	Free	12,078 / 185,410	★ 3.29 / 566		Apr 21, 2010	Published
 <b>Earthquake! 3.8</b>	Free	71,426 / 785,829	★ 4.15 / 6,212		Feb 1, 2013	Published
Page 1 of 1						

### 15.3.2. Multiple User Accounts

Aqui você pode convidar membros do seu time para ter acesso ao Developer Console, dando permissão apenas de algumas funcionalidades.

#### INVITE A NEW USER

Email address

Choose a role for this user

Product lead ▼

- ☒ Create & edit draft apps
- ☒ Edit store listing, pricing & distribution
- ☐ Manage Production APKs
- ☒ Manage Alpha & Beta APKs
- ☒ Manage Alpha & Beta users
- ☐ Create apps distributed only to Google Apps domain
- ☐ Change distribution restriction
- ☐ View financial reports
- ☐ Reply to reviews
- ☒ Edit games
- ☐ Publish games


Send Invitation

Cancel

### 15.3.3. Store Listin Details

---

O Store Listin Details é onde você deve cadastrar todas as informações que são visualizadas pelos usuários, como imagens, descrição do aplicativo, e etc.


**EARTHQUAKE!** – com.radioactiveyak.earthquake
 

Published ▾

STORE LISTING

Saved

Fields marked with \* need to be filled before publishing.

PRODUCT DETAILS

English (United States)

Add translations

**Title \***  
 English (United States)
 

Earthquake!

 11 of 30 characters

**Description \***  
 English (United States)
 

Get a head start on the apocalypse with Earthquake!  
  
 Last 24hrs of earthquakes, with damage and rumble areas shown on an interactive map. Features notifications and vibration to indicate quake magnitude, and a dynamic widget.  
  
 Now optimized for tablets!

**Promo text**  
 English (United States)
 

Get a head start on the apocalypse! 24h of quakes mapped.

**Recent changes**  
 English (United States)
 

Fixed force close on refresh bug introduced in last update (sorry!)

GRAPHIC ASSETS


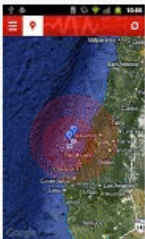


If you haven't added localized graphics for each language, graphics for your default language will be used.  
[Learn more about graphic assets.](#)

Screenshots \*

Default – English (United States)

320 x 480 or 480 x 800 or 480 x 854 or 1280 x 720 or 1280 x 800. JPG or 24-bit PNG (no alpha)

Drag to reorder. At least two are required.







High-res icon \*

Default – English (United States)

512 x 512

32-bit PNG (with alpha)




Feature Graphic

Default – English (United States)

1024 w x 500 h

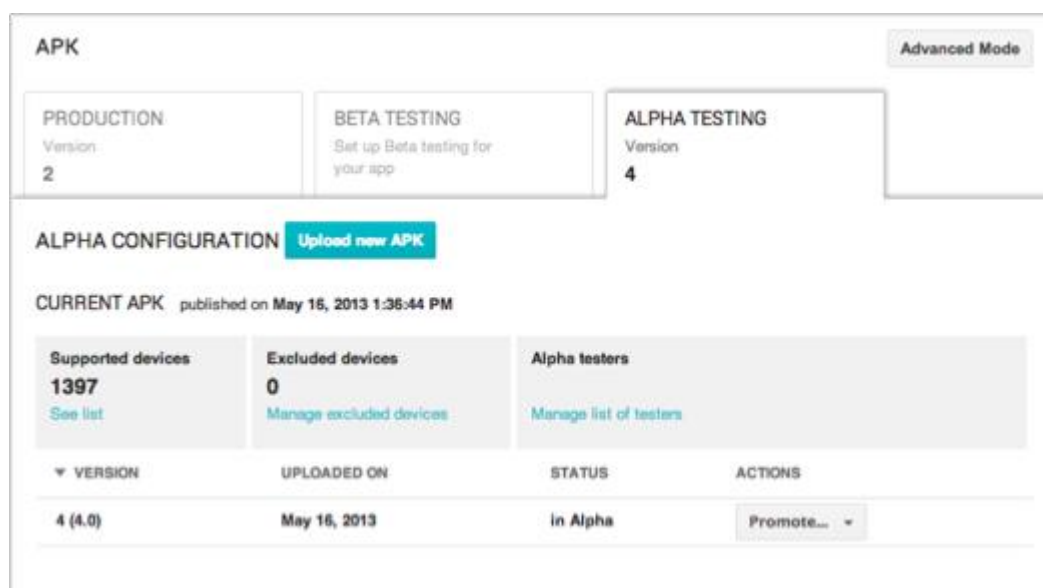
JPG or 24-bit PNG (no alpha)





### 15.3.4 Upload and Instantly Publish

E finalmente chegamos na área responsável pelo upload do apk, lembrando que tem disponível três níveis para se fazer o upload.



- Production - é quando sua aplicação está disponível para todos os usuários Android.
- Beta e Alpha - o aplicativo fica visível a uma lista de testers que é cadastrada por você através do Google Groups ou uma comunidade no Google Plus.

O Google Play Developer Console é um gerenciador muito robusto e abrangente. Aqui citamos algumas pontes importantes para se iniciar no mundo Android, mas você pode consultar a sessão de Distribute do site de developer (<http://developer.android.com/index.html>) para obter mais informações.

## Capítulo 16. Kotlin

---

### 16.1. Conhecendo a nova linguagem oficial

---

O Kotlin é uma linguagem estaticamente tipada, desenvolvida pela JetBrains, cuja sintaxe é mais expressiva e concisa do que a do Java. Com recursos como expressões lambda, sobrecarga de operadores, templates de strings e muito mais.

Como o Java e o Kotlin são altamente interoperáveis, elas podem ser usadas juntas no mesmo projeto. Inclusive, o Kotlin tem como base, e é executado na Máquina Virtual do próprio Java.

O Kotlin é muito intuitivo e fácil de aprender para desenvolvedores Java. A maioria das partes da linguagem são muito semelhantes ao que já sabemos, e as diferenças nos conceitos básicos podem ser aprendidas rapidamente.

Se você é um programador Java experiente, você será capaz de aprender o Kotlin de forma muito fácil, caso contrário, será como aprender qualquer outra linguagem.

Algumas características interessantes da linguagem são:

- **Expressiva:**

Esta é uma das suas qualidades mais importantes, você pode escrever mais com muito menos código.

- **Segura:**

O Kotlin tem seguro contra valores nulos, o que significa pe validado situações de valores nulos em tempo de compilação, para evitar exceções em tempo de execução.

- **Funcional:**

O Kotlin é uma linguagem orientada a objetos, não uma linguagem funcional pura.

No entanto, como muitas outras linguagens modernas, ela usa muitos conceitos funcionais, como expressões lambda, para resolver alguns problemas de uma maneira muito mais fácil.

Outra característica interessante é a maneira como lida com coleções.

- **Funções de Extensão:**

Isso significa que podemos estender qualquer classe e adicionar novos recursos, mesmo se não tivermos acesso ao código-fonte.

- **Interoperável:**

Você pode continuar usando a maioria das bibliotecas e códigos escritos em Java, porque a interoperabilidade entre elas é excelente. É até possível criar projetos mistos, com os arquivos Kotlin e Java coexistindo.

## 16.2. Kotlin ou Java

---

Pode ser empolgante, mas também um pouco assustador para qualquer programador trocar de uma linguagem que você já tem uma certa experiência, por algo totalmente novo.

Nas comunidades de desenvolvimento Android, o **Kotlin** está sendo comparado ao **Java** da mesma maneira que o **Swift** foi comparado ao **Objective-C**.

- **A forma de aprender:**

A curva de aprendizado para usar do Kotlin é bastante baixa: qualquer desenvolvedor Java experiente pode aprender em poucas horas. A documentação oficial da JetBrains é completa e muito bem feita.

Desenvolvedores C# também vão se sentir em casa trabalhando com a linguagem, uma vez que as duas línguas compartilham algumas características.

- **Documentação e referências:**

Para desenvolvimento Android utilizando o Android Studio, existe um suporte muito bom, já que o Android Studio é baseado no IntelliJ, que também foi construído pela JetBrains.

Como já falado, ele também interage perfeitamente com o Java podendo utilizar arquivos das duas linguagens no mesmo projeto.

O uso de bibliotecas Java em projetos Kotlin também é simples: ele simplesmente funciona.

O inverso também é verdade: você pode facilmente incluir qualquer biblioteca escrita em Kotlin dentro do código Java.

### 16.3. Principais pontos de melhoria

---

Talvez a vantagem mais importante do Kotlin sobre o Java seja o conjunto de recursos. O Kotlin adiciona novas capacidades importantes nos projetos Android que ainda não é possível encontrar em Java.

Algumas capacidades incluem.

### ▪ Proteção Contra Nulo (Null Safety):

Essa proteção elimina a maioria dos problemas referências nulas, tornando todos os tipos não nulos por padrão – o que significa que o compilador não permitirá que você use uma variável não inicializada, ou com possibilidade de nulo.

Se você precisar de uma variável nula, você deve declarar o tipo como “anulável”, adicionando um ponto de interrogação após o tipo.

1. **var** nonNullable: String = "My string"
2. **var** nullable: String?

### ▪ Funções Estendidas:

Você pode adicionar comportamentos a uma classe sem estendê-la diretamente. Através de uma função de extensão, você pode chamar uma função de um objeto como se fosse parte de sua classe.

Poderíamos criar uma função de extensão para String chamada novaFuncao e chamá-la de minhaString.novaFuncao(). Como sabemos, novaFuncao não é uma função de String, mas sintaticamente ela é.

Então, como isso funciona? Você só precisa adicionar a função a classe String conforme exemplo abaixo:

1. StringExtensions.kt
2. **fun** String.novaFuncao(): String {
3.     *// implementacao*
4. }

### ▪ Lambdas:

Se você estiver familiarizado com JavaScript (ou C# e muitos outros), você provavelmente já conhece sobre as funções lambdas.

Uma função lambda recebe funções como parâmetros ou retorna uma função. As funções podem ser armazenadas em variáveis para uso posterior, passadas como parâmetro ou criadas dentro de outra função.

```
1. // Criando a função
2. fun networkCall(onSuccess: (ResultType) -> Unit,
3.   onError: (Throwable) -> Unit) {
4.   try {
5.     onSuccess(myResult)
6.   } catch(e: Throwable) {
7.     onError(e)
8.   }
9. }
10.
11. // Chamando a função
12. networkCall(result -> {
13.   // use successful result
14. }, error -> {
15.   // handle error
16. });
```

#### ▪ Classes de Dados (Data Classes):

Este recurso traz uma grande economia de tempo. Pois a maioria das nossas aplicações são orientadas por dados, muitas vezes precisamos criar classes com apenas propriedades e campos para armazenar dados.

Em Java, isso é muito custoso, exigindo os métodos get/set para cada campo. Com o Kotlin, podemos declarar a classe e todas as suas propriedades em uma única linha. O compilador irá gerar todos os getters e setters.

```
1. data class User(var name: String, var age: Int)
```

- **Imutabilidade:**

Uma grande preocupação que os desenvolvedores precisam ter ao desenvolver aplicativos multithread é a administração do estado. Se uma variável é mutável, ela pode ser alterada por qualquer thread que possa acessá-la.

Isso significa que, se os dados puderem ser alterados por várias fontes, você precisará implementar manualmente a sincronização, o que evita a perda de dados, mas aumenta a complexidade do código e o tempo de execução. Se os dados nunca podem ser alterados, eles podem ser acessados por várias threads sem problema, uma vez que são imutáveis.

No Kotlin, você pode declarar variáveis com as palavras-chave `var` e `val`. O primeiro declara uma variável que pode ser alterada; O último uma variável que uma vez atribuído um valor nunca pode mudar.

Isso dá ao desenvolvedor e ao compilador confiança de que a variável não pode ser alterada. Em Java temos funcionalidade semelhante com palavra-chave `final`.

A biblioteca padrão do Kotlin inclui uma série de interfaces de coleções imutáveis e funções para ajudá-lo a escrever um código imutável. Por exemplo, a função `listOf()` cria uma lista imutável.

- **Co-rotinas:**

Normalmente, quando os desenvolvedores precisam executar uma tarefa de longa execução, como uma chamada externa de rede ou carregar um arquivo do disco, a thread de chamada fica bloqueada aguardando a conclusão da operação.

As co-rotinas nos permitem executar esses tipos de operações sem bloquear seu código. Em vez disso, é utilizada uma operação mais leve chamada co-rotina.

Podemos escrever um código aparentemente síncrono que é, na verdade, assíncrono e durante a compilação, o código será transformado para assíncrono. Isso significa que essa técnica não depende da máquina virtual ou do sistema operacional.

### 16.5. Presente ou futuro?

---

Como vimos, o Kotlin roda em cima da Máquina Virtual do Java e facilita muitas coisas para o desenvolvedor Android, então basicamente, ele ajuda a escrever códigos mais limpos e de forma mais fácil.

Em um futuro a médio prazo, é provável que a maioria dos aplicativos Android sejam desenvolvidos utilizando apenas o Kotlin, mas a chance disso acontecer em um futuro próximo é muito pequena, logo não precisamos sair de forma desesperada aprendendo uma nova linguagem e realizando a migração de aplicativos já lançados.



## Referências

---

BURNETT, Ed. Hello, Android: Introducing Google's Mobile Development Platform. 2. Ed. Pragmatic Bookshelf, 2009.

BURTON, Michael; FELKER, Donn. *Android Application Development For Dummies*. 2. Ed. For Dummies, 2012.

CORDEIRO, Filipe. *Kotlin: Introdução e Primeiros Passos*. Disponível em: <<https://www.androidpro.com.br/blog/kotlin/kotlin/>>. Acesso em: 25 out. 2018.

Developers. *Develop Android apps with Kotlin*. Disponível em: <<https://developer.android.com/kotlin/>>. Acesso em: 25 out. 2018.

Developers. Documentação oficial do Android. Disponível em <<https://developer.android.com/>>. Acesso em: 25 out. 2018.

Developers. Introdução ao Android. Disponível em: <<http://developer.android.com/guide/index.html>>. Acesso em: 25 out. 2018

LECHETA, Ricardo R. *Android Essencial com Kotlin*. São Paulo: Novatec, 2017.

LECHETA, Ricardo R. *Android Essencial*. São Paulo: Novatec, 2016.

LECHETA, Ricardo R. *Google Android*. 5. Ed. São Paulo: Novatec, 2017.

MIKKONEN, Tommi. *Programming Mobile Devices An Introduction For Practitioners*. 1. Ed. Wiley, 2007.

ZIGURD, Mednieks et al. *Programando o Android*. São Paulo: Novatec, 2012.