

Módulo 2 – Android Nativo com Java

Bootcamp: Desenvolvedor Mobile Apss

Bruno Augusto Teixeira

2020

Android Nativo com Java

Bootcamp: Desenvolvedor Mobile App

Bruno Augusto Teixeira

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

Sumário

Capítulo 1. Android.....	4
1.1 Introdução ao Android.....	4
1.2 Instalação.....	7
1.3 Estrutura do projeto Android	9
1.4 Configuração do Emulador Android (AVD).....	12
1.5 Android Manifest	16
Capítulo 2. Componentes do Android.....	17
2.1 Activities	17
2.2 Intents	19
2.3 Content Provider	21
2.4 Services	22
2.5 Broadcast Receiver	24
Capítulo 3. Interface do Usuário no Android.....	25
3.1 Layouts Estáticos	25
3.2 Layouts Dinâmicos.....	27
Capítulo 4. Armazenamento de Dados.....	29
4.1 Shared Preferences	29
4.2 File Systems.....	29
4.3 Banco de Dados.....	30
Capítulo 5. Notificações e Publicação	31
5.1 Notificações.....	31
5.2 Publicação.....	31
Referências.....	36

Capítulo 1. Android

1.1 Introdução ao Android

O Android é um sistema operacional desenvolvido em 2003 pela Android Inc., que em 2005 foi adquirida pela Google, que o mantém até o momento. Além de ser um sistema operacional, o Android é uma plataforma de desenvolvimento para dispositivos móveis, como celulares e tablets, que pode ser executada em dispositivos de diferentes fabricantes.

Em 2007, foi criada a *Open Handset Alliance* (<http://www.openhandsetalliance.com/>), que é um consórcio de várias empresas de software, hardware e telecomunicações para implementar uma plataforma de dispositivos móveis sob o Android que seja completa, aberta e gratuita. Neste mesmo período, foi lançada a primeira versão beta do Android Software Development Kit (SDK) para a comunidade desenvolver Apps. Atualmente, a plataforma suporta o desenvolvimento de TVs, Wearables (relógios inteligentes, óculos, etc.), Autos e dispositivos para IoT.

Arquitetura da plataforma

A arquitetura do Android é composta por camadas, descritas na Figura 1, que auxiliam no suporte às necessidades de diferentes dispositivos. O sistema operacional é baseado em Linux, com diversas bibliotecas em C e C++ disponibilizadas aos fabricantes.

Figura 1 - A pilha de software do Android.



Fonte: <https://developer.android.com/guide/platform>

A Figura 1 apresenta as camadas da arquitetura da plataforma, que podem ser identificadas da seguinte forma:

- **Linux Kernel** – Esta camada é a mais inferior, e é considerada o core da arquitetura Android, pois por meio dela é realizado o gerenciamento de todos os drivers (bluetooth, câmera, memória...) do dispositivo.
- **Hardware Abstraction Layer (HAL)** – Trata-se da camada de abstração entre o hardware do dispositivo e o restante das camadas. É responsável pelo

gerenciamento de memória, de energia, de dispositivos, de acesso a recursos, etc.

- **Android Runtime (ART)** – Android Runtime é uma camada que contém as principais bibliotecas em Java que compõe o aplicativo e, sobretudo, a máquina virtual (VM) Dalvik. A Dalvik Virtual Machine (DVM) é uma máquina virtual baseada em registro, assim como a Java Virtual Machine (JVM). Essa VM foi especialmente projetada e otimizada para o Android, de modo que seja garantido que um dispositivo possa executar várias instâncias com eficiência.
- **Native C/C++ Libraries** – Esta camada dispõe de bibliotecas dos componentes e serviços do ART e HAL, que são implementados por códigos nativos em linguagem nativa (C e C++). Entretanto, a plataforma fornece Frameworks API para utilização dessas bibliotecas em Java.
- **Java API Framework** – O Application Framework fornece as API's em Java que são utilizadas para criação dos aplicativos Android. Essas API's proveem frameworks com os recursos do SO, tais como sistemas de visualização, gerenciador de atividade, gerenciador de recursos, etc.
- **System Apps** - Camada superior da arquitetura Android que dispõe de aplicativos nativos e de terceiros, como contatos, e-mail, música, galeria, relógio, jogos, etc. A instalação de qualquer aplicação ocorre nesta camada.

Evolução do Android

As versões do Android foram evoluindo desde a sua primeira versão disponibilizada em 2008 para acompanhar as evoluções dos dispositivos. Cada nova versão recebia um nome de uma sobremesa, em ordem alfabética, até versão 9.0, conforme podemos visualizar na Figura 2. Atualmente, a plataforma está na versão

10, com a versão 11 Beta disponível para desenvolvedores. Porém, a partir da versão atual a Google abdicou dos nomes de sobremesa.

Figura 2 – Evolução das versões do Android.



Fonte: Adaptado do Google.

1.2 Instalação

Android Studio é uma ferramenta baseada no IntelliJ IDEA, e é o ambiente de desenvolvimento oficial para Android. O ambiente está disponível para diversas plataformas de sistemas operacionais, como Windows, Mac e Linux. A seguir, serão apresentados os passos para instalação da ferramenta no ambiente Windows.

Para a instalação, é necessário que o Java Development Kit (JDK) esteja instalado no seu ambiente. Os passos para instalação podem ser encontrados em <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Após a instalação do JDK, é necessário realizar o download do Android Studio em <https://developer.android.com/studio>

Figura 3 – Download do Android Studio

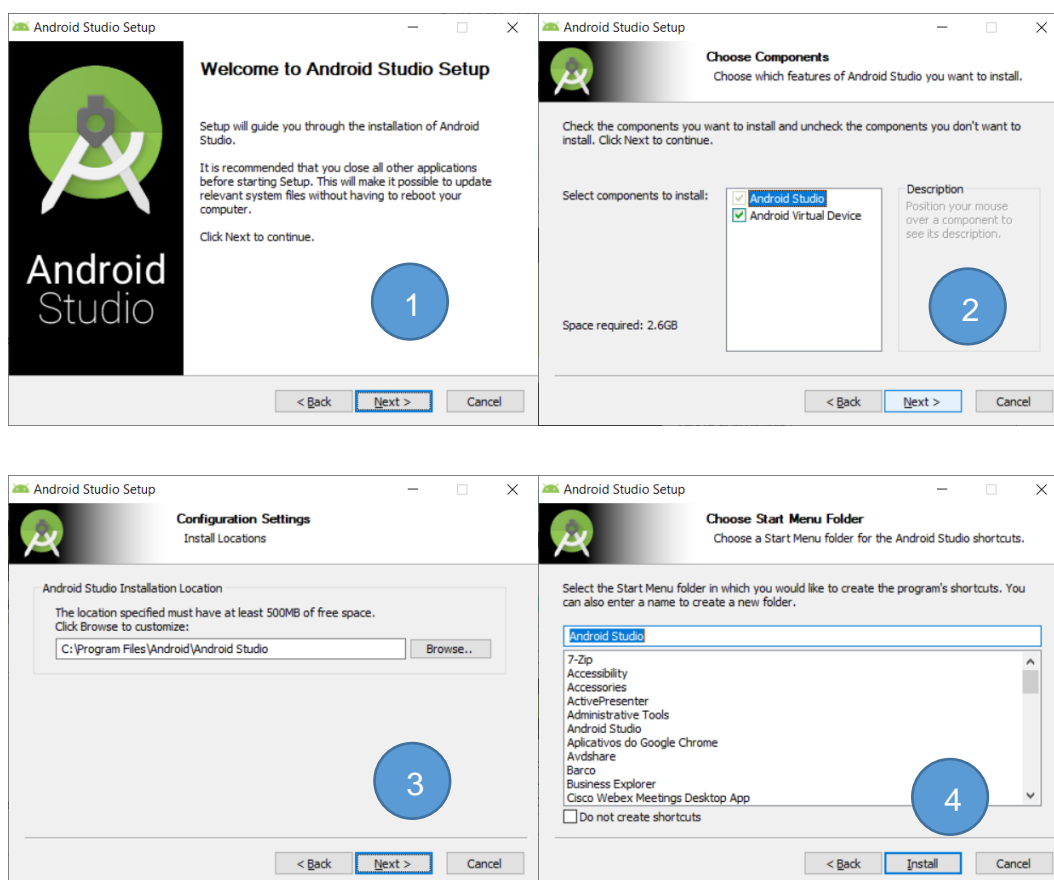


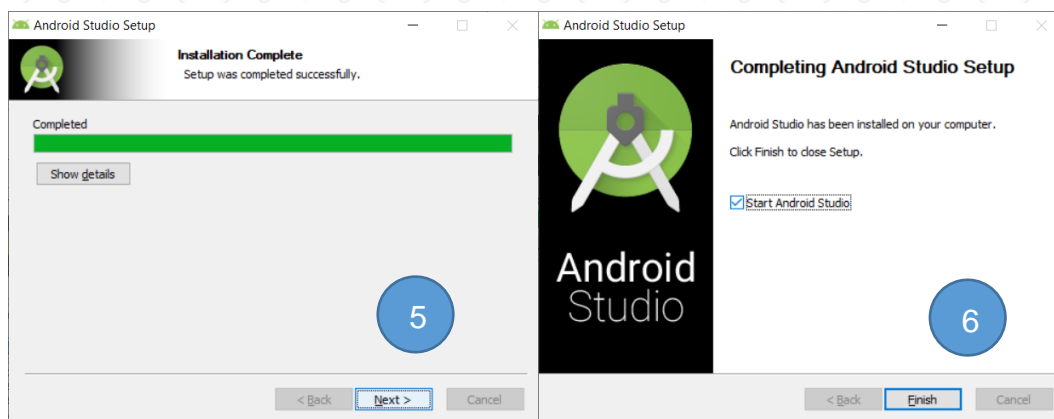
Android Studio provides the fastest tools for building apps on every type of Android device.

DOWNLOAD ANDROID STUDIO

4.0.1 for Windows 64-bit (871 MB)

A instalação é a padrão do instalador, conforme indicado nas imagens abaixo:

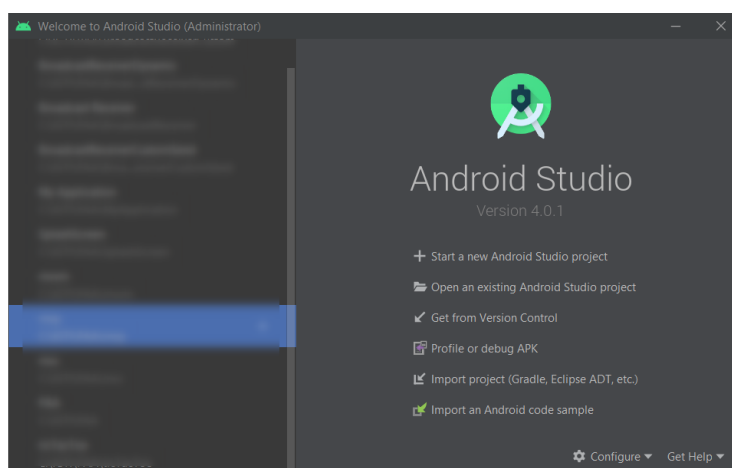




1.3 Estrutura do projeto Android

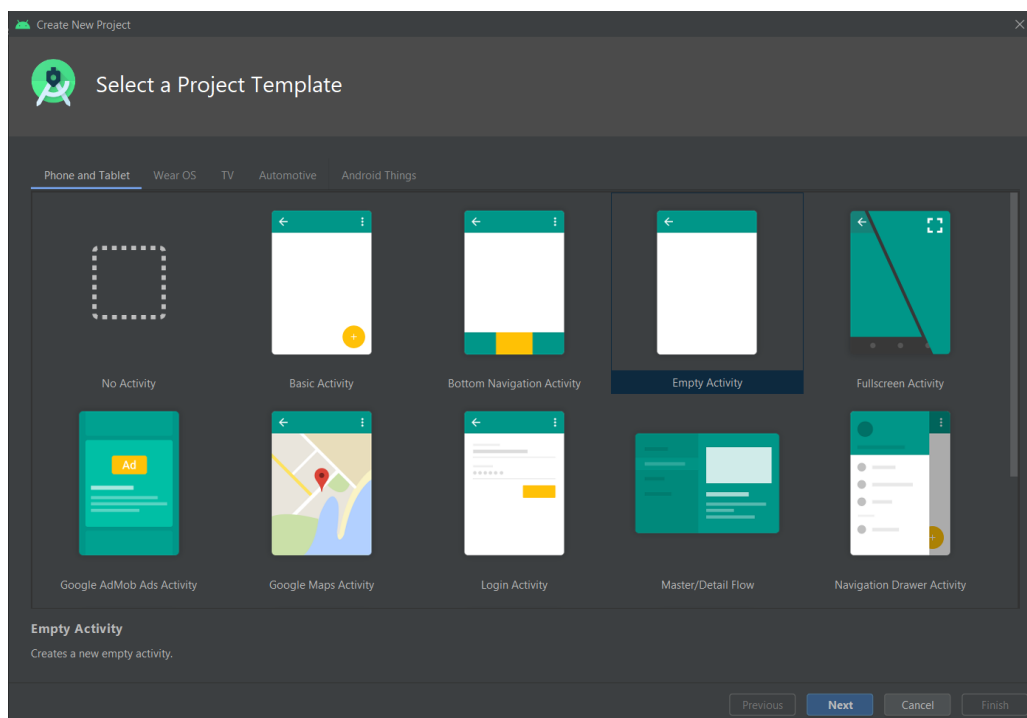
A abertura do Android Studio apresenta a interface ilustrada na Figura abaixo:

Figura 4 – Interface Android Studio



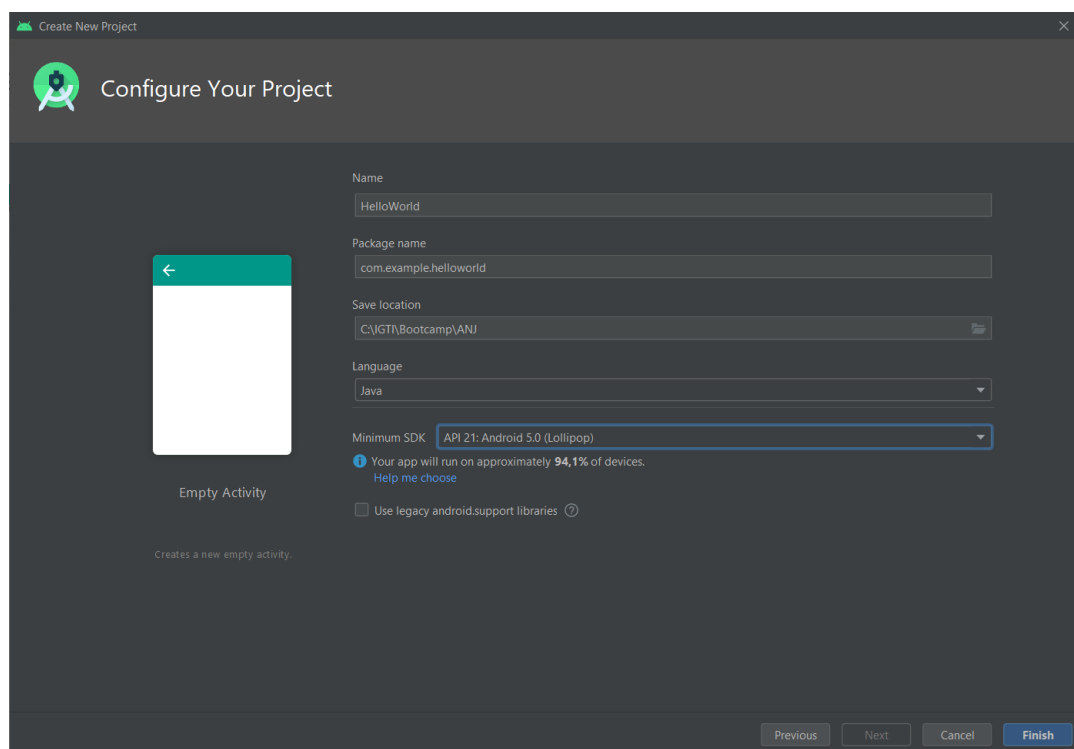
Para a criação do projeto Android, basta clicar em **Start a new Android Studio Project**. Uma nova tela será exibida com vários templates pré definidos, que podem ser utilizado com a estrutura da implementação inicial, conforme indicado pela Figura abaixo:

Figura 5 – Templates do projeto.



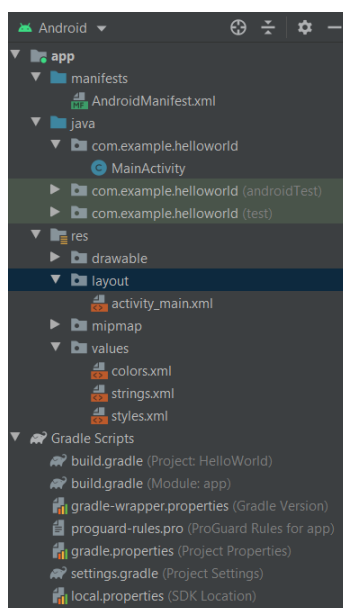
Após selecionar um dos templates, é exibida a próxima tela para as configurações básicas da aplicação, como o **nome**, a definição do **pacote**, o local para **armazenamento**, a **linguagem** (Java ou Kotlin) e a versão mínima da **API**.

Figura 6 – Configurações do projeto.



O Android Studio permite a visualização do projeto em diferentes estruturas, a partir da qual define os módulos que serão exibidos na estrutura do projeto. A figura abaixo apresenta a estrutura do projeto criado na visualização Android.

Figura 7 - Estrutura do projeto.

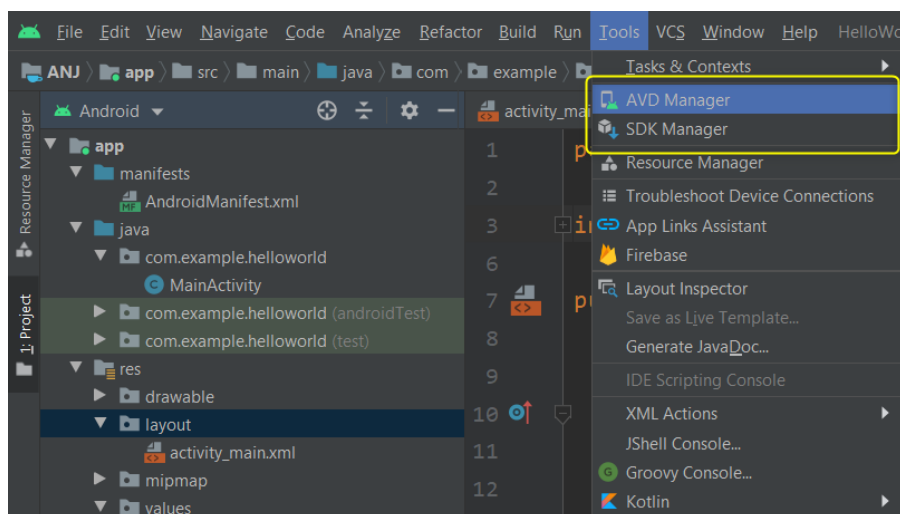


- **App** – Módulo da aplicação atual. Caso seu projeto possísse vários módulos, haveriam várias estruturas.
- **AndroidManifest** – Contém todas as configurações necessárias para executar a aplicação, como nome do pacote, declarações de cada activity, permissões, etc.
- **Java** – Contém os arquivos dos código-fonte do projeto, separados por nome de pacote, incluindo o código de teste JUnit.
- **Res** – Contém os arquivos relacionados aos recursos do aplicativo (arquivos de layout, imagens, animações e xml contendo valores como strings, arrays, etc.), acessíveis através da classe R.
- **Gradle** - Sistema de build do projeto one. É definida a configuração de compilação que se aplica a todos os módulos.

1.4 Configuração do Emulador Android (AVD)

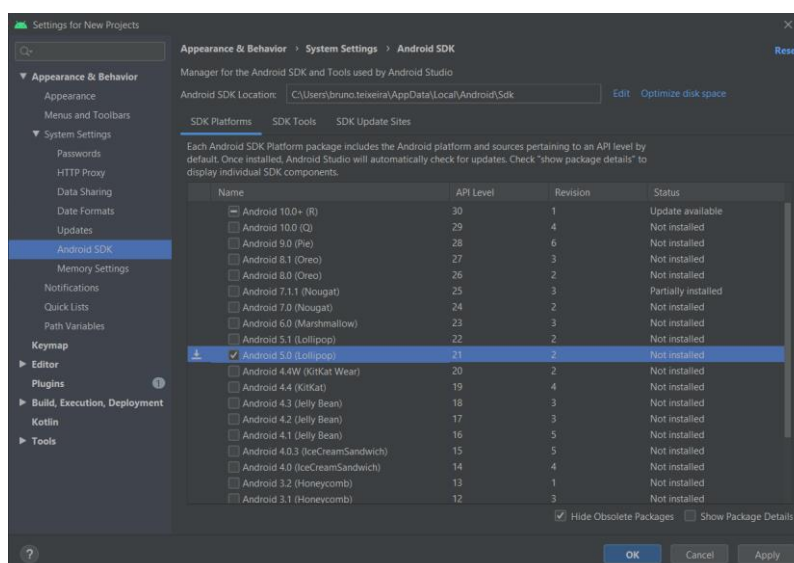
Android virtual device (AVD) é um emulador utilizado para replicar as funcionalidades de dispositivos como smartphone, tablet, android wear, TV, etc. No Android Studio é disponibilizado o AVD Manager, que permite emular todas as versões disponíveis do Android em algumas versões de hardware (dispositivos).

Figura 8 – AVD Manager

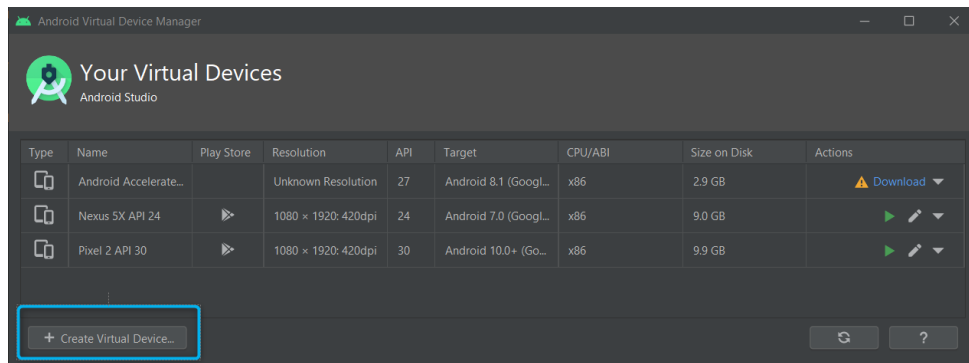


Para criar os emuladores em diferentes versões do Android, o Android Studio possui o SDK Manager, que realiza o download dessas diferentes versões de API, conforme demonstrado pela figura abaixo:

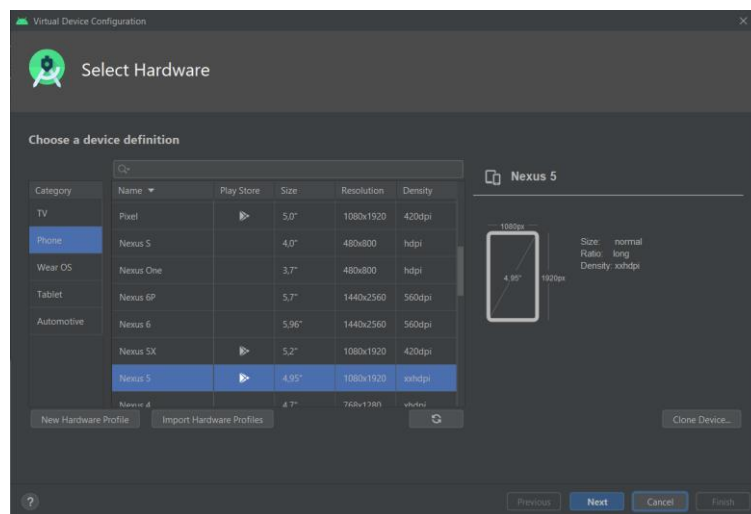
Figura 9 – SDK Manager



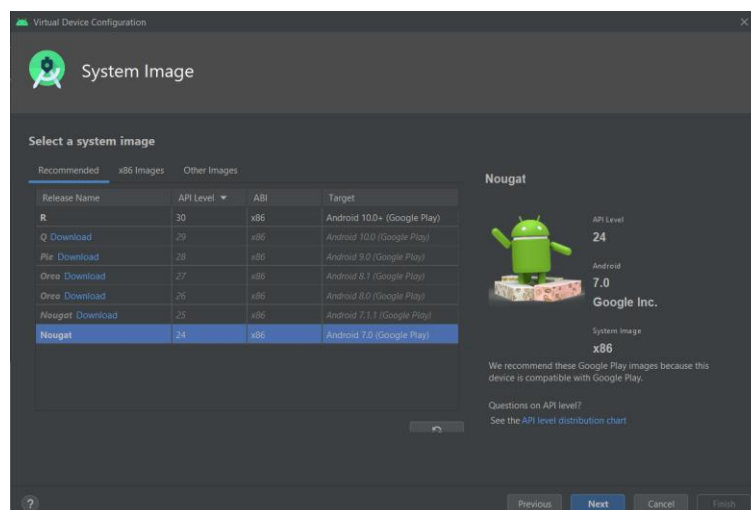
Após a instalação das versões do SDK, é possível configurar o emulador.



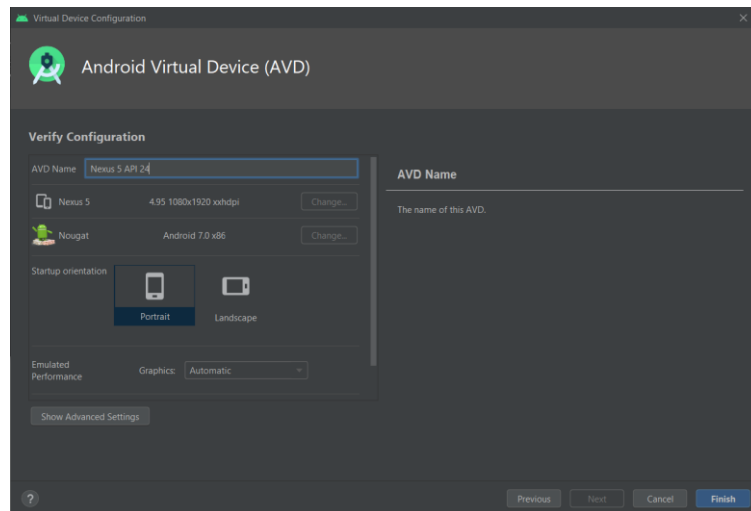
Defina o dispositivo que será utilizado no emulador:



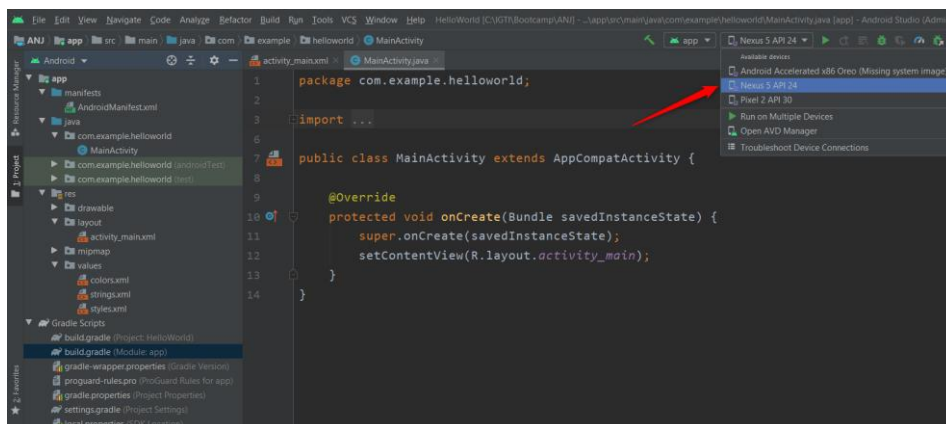
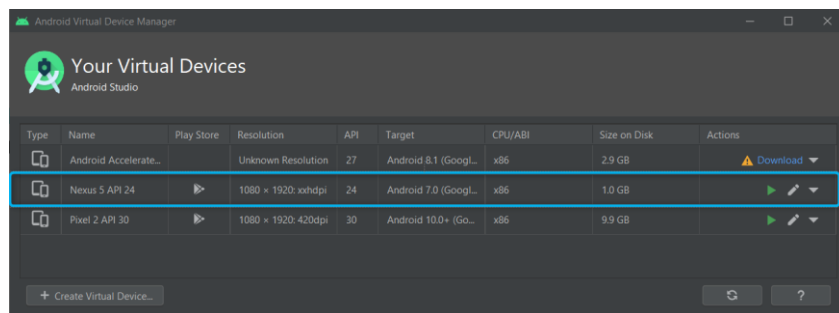
Defina a versão da API para o dispositivo:



Defina o nome do AVD e a definição de orientação da tela:



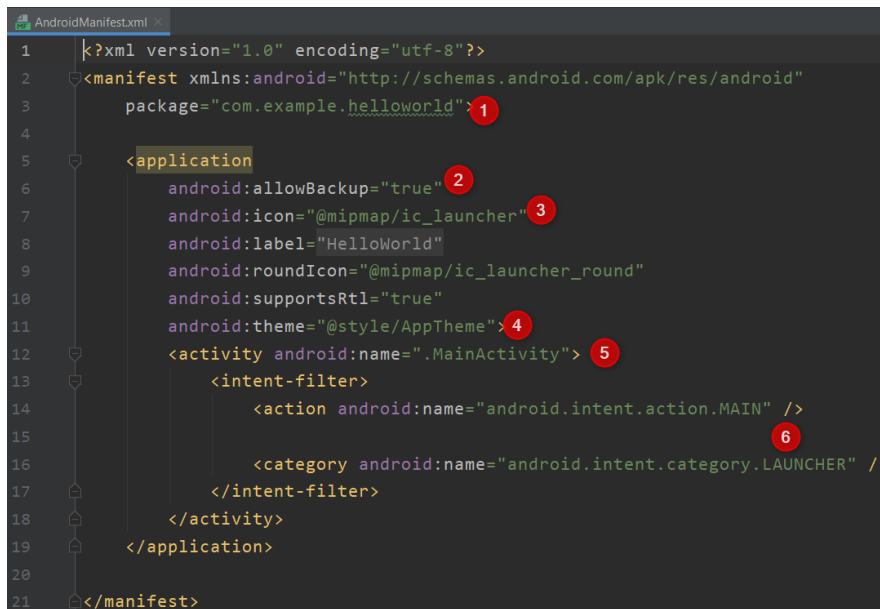
Com isso, o emulador está criado e pode ser inicializado pelo botão de play ou por meio do build da aplicação, conforme demonstrado pelas figuras abaixo:



1.5 Android Manifest

O AndroidManifest.xml é o arquivo principal de configuração das aplicações Android. Este arquivo contém todos os componentes do aplicativo, como as Activities.

Figura 10 – AndroidManifest.xml



A figura acima indica os principais itens do AndroidManifest.xml:

- 1- Apresenta o nome do pacote do aplicativo, que deve ser único. Seu nome não pode ser alterado após a publicação do aplicativo.
- 2- Habilita o backup e o restore dos dados do aplicativo automaticamente.
- 3- Define o ícone da aplicação.
- 4- Este atributo define o tema que é utilizado para a aparência do aplicativo nos componentes de interface de usuário.
- 5- Descrição das Activities disponíveis no aplicativo.
- 6- Define qual é a Activity de start do aplicativo.

Capítulo 2. Componentes do Android

Os componentes do Android são os blocos básicos disponibilizados pela plataforma para a construção de uma aplicativo. Todos esses componentes podem ser definidos no `AndroidManifest.xml`.

No Android, os seguintes componentes são o core da plataforma:

1. Activities
2. Content Providers
3. Broadcast Receiver
4. Services

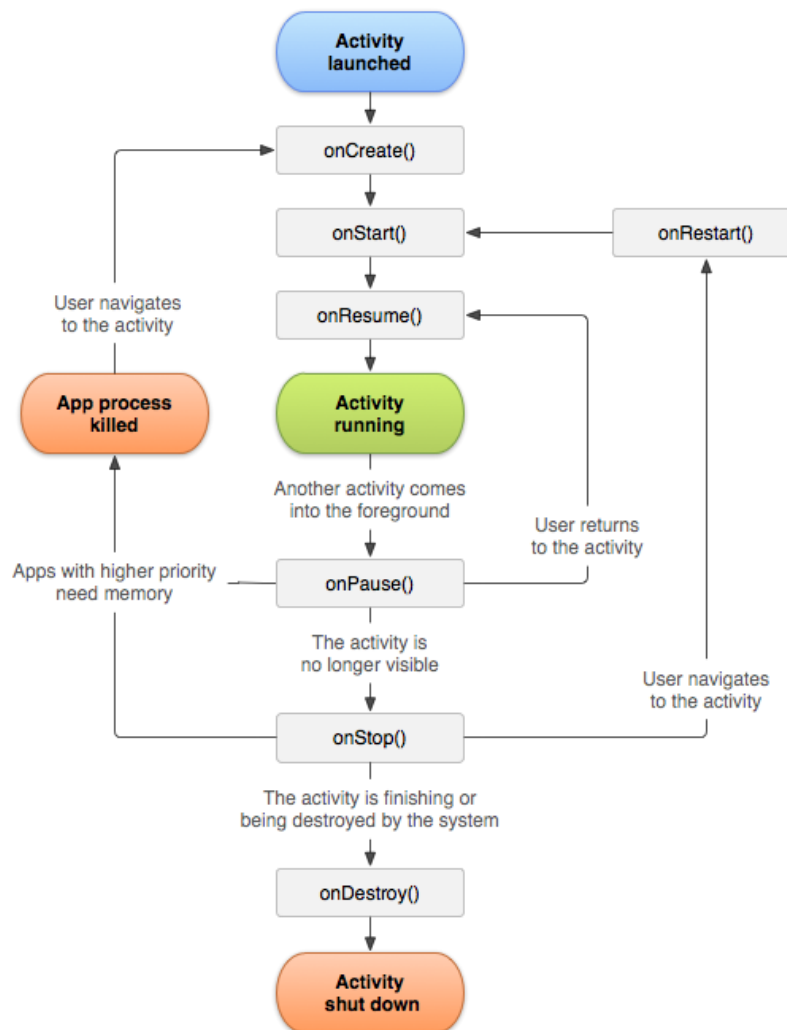
2.1 Activities

A Activity é o conceito mais importante no desenvolvimento de aplicativos Android, pois ela geralmente representa uma tela com interface gráfica que, na maioria das vezes, define o ponto de interação do usuário com o aplicativo. Um aplicativo Android pode ser composto por diversas Activities que se interligam e que fornecem um conjunto de funcionalidades para os usuários. Apesar de terem uma integração para compor o aplicativo, cada Activity é independente uma da outra. Assim, é possível que um aplicativo inicie uma Activity em outro aplicativo, e vice-versa.

Diferentemente de outras linguagens de programação, nas quais as aplicações se iniciam pelo método **main()**, no Android, a Activity é onde o aplicativo inicia seu processo. Elas possuem um ciclo de vida associado a partir do seu início e, para criar um aplicativo Android sem erros, é importante compreender o ciclo de vida de uma Activity.

Na Figura 11, temos o ciclo completo de uma Activity:

Figura 11 – Ciclo de Vida da Activity.



Fonte: <https://developer.android.com/>

O ciclo de vida de uma Activity é composto por sete métodos:

- **onCreate:** quando o usuário inicia uma Activity pela primeira vez, o primeiro método instanciado é o `onCreate`.
- **onStart:** método instanciado quando uma Activity se torna visível para o usuário e é instanciado após o `onCreate`.
- **onResume:** método instanciado um pouco antes do usuário conseguir interagir com o aplicativo.

- **onPause:** método instanciado após o onResume, quando a Activity está para perder a visibilidade para outra Activity
- **onStop:** é instanciado quando a Activity não está mais visível para o usuário.
- **onDestroy:** é instanciado quando a Activity está prestes a ser destruída, ou seja, quando a Activity é excluída da pilha de activities do aplicativo.
- **onRestart:** instanciado quando a Activity está no estado onStop e está prestes a iniciar novamente.

2.2 Intents

As Intents são objetos de mensagens assíncronas que permitem que os componentes do Android solicitem uma ação de outro componente. Elas podem ser utilizadas para as seguintes operações no aplicativo:

- Iniciar uma Activity (Activities).
- Iniciar um Serviço (Services).
- Transmitir uma mensagem (Broadcast Receiver).
- Iniciar aplicativos nativos.

As intents são utilizadas pelo aplicativo, mas também podem ser disparadas pelo sistema para notificar a aplicação de eventos específicos, como a chegada de uma mensagem de texto.

Tipos de Intents

Existem dois tipos de Intents disponíveis no Android: **Intents implícitas** e **Intents explícitas**.

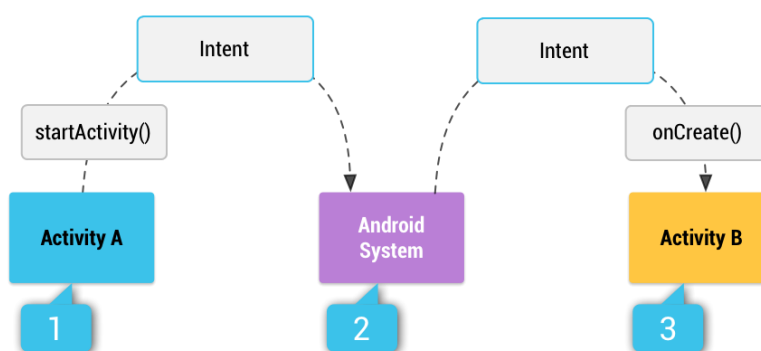
- **Intents Implícitas:** Essas intents não especificam o componente que irá ser acionado. Elas definem uma ação geral a ser executada, e o sistema Android corresponde à solicitação ou com uma Activity ou outro componente que possa

lidar com a ação requerida. Dessa forma, um componente de outro aplicativo pode processar uma Intent do seu aplicativo

- **Intents Explícitas:** Essas intents especificam a Activity ou componente que será acionado dentro do próprio aplicativo com o nome da classe.

A Figura 12 exemplifica o fluxo básico de uma Intent implícita dentro do aplicativo, na qual a **Activity A** cria uma Intent com uma descrição de ação e inicia pelo método `startActivity()`. A partir desse momento, o sistema Android busca todos os apps por filtros de Intent que correspondem à Intent em questão. Quando é encontrada uma correspondência, o sistema inicia a Activity que atendeu os requisitos invocando o método `onCreate()` da Activity.

Figura 12 – Fluxo de ações de uma Intent Implícita.



Fonte: <https://developer.android.com/>

Propriedades das Intents

Conforme citado acima, uma Activity ou uma ação podem ser disparadas pelas Intents. Elas são definidas nas informações presentes na Intent (implícita ou explícita): o sistema avalia as propriedades presentes nas Intents e, com base nessas informações, decide o que será disparado.

Algumas das propriedades básicas que um Intent contém são:

- **Component Name:** Nome do componente que deve ser iniciado. Caso ele não seja informado, a Intent será implícita.
- **Action:** Define a ação a ser executada por uma Activity específica.
- **Data:** Define os dados que podem ser transmitidos, bem como os seus respectivos tipos.
- **Category:** Utilizado para as Intenções explícitas, nas quais é necessário especificar o tipo de componente que será utilizado para executar a ação

Além das propriedades definidas acima, uma intent pode carregar informações adicionais que não afetam o modo com que é tratado um componente do aplicativo, sendo ela:

- **Extras:** é possível adicionar dados extras a um Intent na forma de pares de valores-chave. Essas informações extras podem ser passadas de uma Activity para outra.

2.3 Content Provider

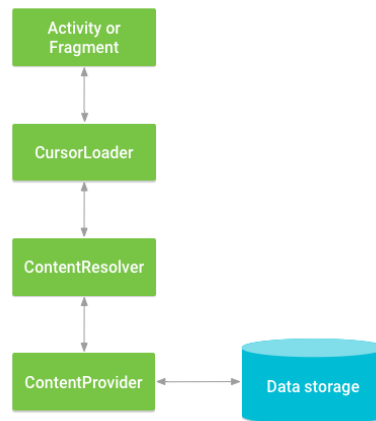
Os Contents Providers (Provedores de Conteúdo) permitem o compartilhamento de dados de um aplicativo Android com outro. Assim, os provedores de conteúdo são utilizados em dois cenários: ou quando é necessário implementar um código para acessar um provedor de conteúdo atual em outro aplicativo, ou quando é necessário criar um provedor no aplicativo para compartilhar dados com outros aplicativos.

Um exemplo são os detalhes dos contatos armazenados no dispositivo, que com a ajuda do aplicativo Contatos são compartilhados com outros aplicativos, como WhatsApp, Facebook, etc. Nesse caso, o aplicativo Contatos é o provedor, e o WhatsApp e o Facebook são os clientes do provedor.

Assim, o Content Provider gerencia o acesso a um repositório central de dados que são disponibilizados a aplicativos externos na forma de uma ou mais

tabelas, de forma semelhante aos bancos de dados relacionais, conforme indicado pela Figura 13. O uso dos content providers são importantes se o aplicativo necessita de compartilhar dados entre múltiplos aplicativos. Caso contrário, um banco de dados é o mais recomendado.

Figura 13 – Diagrama de acesso aos dados por ContentProvider.



Fonte: <https://developer.android.com/>

2.4 Services

O Service (serviço) é um componente que é executado em segundo plano para realizar operações sem interação com o usuário. Os Services geralmente são utilizados para atualizar o Content Provider, realizar o download da internet, processar os dados, acionar intenções e acionar as notificações.

Existem três tipos diferentes de utilização de serviços:

1. **Primeiro Plano:** o serviço em primeiro plano realiza uma ação que é perceptível ao usuário. Por exemplo, um aplicativo de áudio usaria um serviço em primeiro plano para reproduzir uma faixa de áudio.
2. **Segundo Plano:** o serviço em segundo plano realiza uma operação que não é perceptível ao usuário. Por exemplo, enquanto enviamos algumas imagens

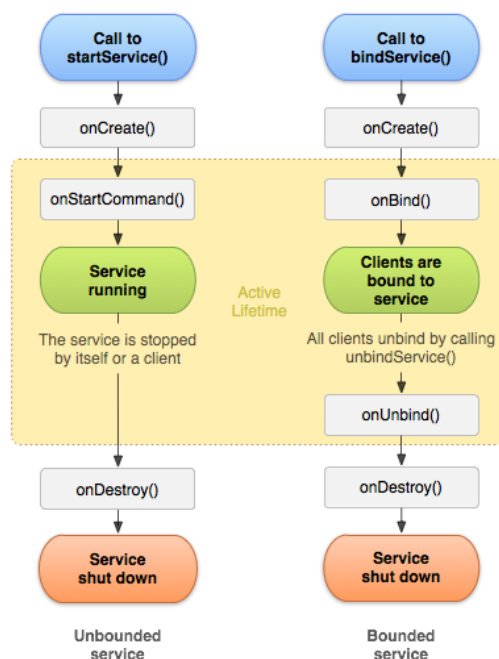
pelo Whatsapp, o aplicativo realiza a compactação da imagem para reduzir o tamanho, porém a tarefa é executada em segundo plano e o usuário não tem visibilidade dessa ação.

3. **Vinculado:** o serviço é vinculado quando um ou mais componente(s) do aplicativo vinculam o Serviço. Se os aplicativos o desvincularem, o serviço será destruído.

Ciclo de Vida

Assim como as Activities, os Services possuem um ciclo de vida, porém muito mais simples. Conforme indicado pela Figura 14, o ciclo de vida de um serviço pode ser Iniciado ou Vinculado. O **Serviço Iniciado** é criado quando o componente chama o `startService()` e permanece em execução até a chamada do método `stopSelf()`, enquanto o **Serviço Vinculado** é criado quando um componente chama o método `bindService()` e finaliza quando o último componente vinculado ao serviço faz a chamada do `unbindService()`.

Figura 14 – Ciclo de Vida dos Services.



Fonte: <https://developer.android.com/>

2.5 Broadcast Receiver

Um Broadcast Receiver é um dos componentes Android que possibilita assinar os eventos do sistema ou de aplicativos. O Android Runtime (ART) notifica todos os receptores assinados quando o evento ocorre. Por exemplo, um aplicativo pode registrar os eventos para mensagem ou chamada telefônica recebida, etc. O Broadcast Receiver possui um ciclo de vida que é válido somente durante a chamada do método `onReceive(context, intent)`. Após isso, o sistema operacional encerra seu processo para liberar memória.

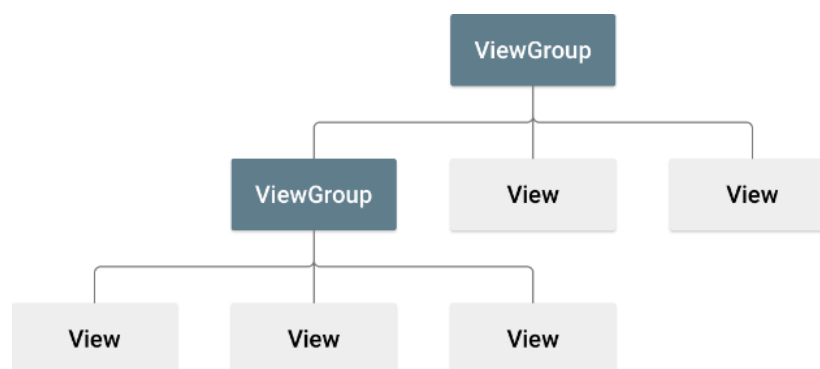
O Android permite que um BroadcastReceiver seja executado no máximo em 10 segundos. Então, é recomendado que alguns tipos de Broadcast iniciem um Service para processamento.

Capítulo 3. Interface de Usuário no Android

As interfaces de usuário nos aplicativos Android são construídas nos layouts, e o elemento fundamental desses layouts é a View. A partir da View, são derivados os componentes que o usuário consegue ver e interagir, 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.

Os elementos de tela oriundos da View são organizados pelos containers de ViewGroup dentro dos Layouts. A Figura 15 apresenta a hierarquia e organização de um layout.

Figura 15 – Hierarquia e organização do componente View no Layout



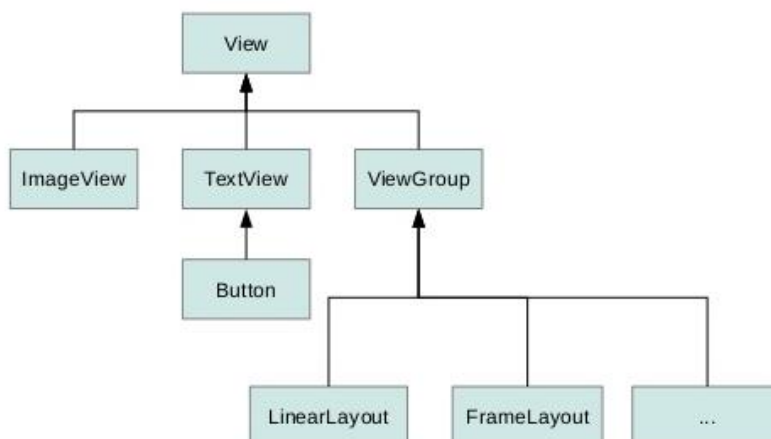
Fonte: <https://developer.android.com/>

Os objetos View para composição das telas são geralmente denominados “**widgets**”, e os objetos de ViewGroup para gerenciar o layout são denominados “**layouts**”.

3.1 Layouts Estáticos

Como citado acima, a View possui elementos estáticos que auxiliam na construção das interfaces de usuário do aplicativo. A Figura 16 apresenta uma hierarquia simplificada do componente View para os componentes estáticos.

Figura 16 – Hierarquia do componente View



Fonte: Adaptado do Google.

- **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.
- **EditText:** A classe `EditText` é uma subclasse de `TextView` utilizada para os usuário digitarem informações em um campo de texto. Ela pode ser usada para a entrada de: texto normal, apenas números e formato de senha.
- **ImageView:** A classe `android.widget.ImageView` exibe uma imagem arbitrária como um ícone.
- **Button e ImageButton:** 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.

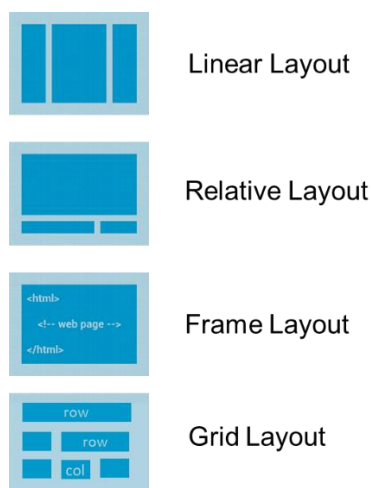
Além dos *widgets* listados, existem diversos outros disponíveis que herdam da classe `View` para utilização nas aplicações.

A organização desses componentes na tela se dá pelos gerenciadores de layout, que automaticamente os organizam de acordo com a regra implementada pelo

gerenciador de layout utilizado. Os principais gerenciadores de layouts disponíveis para o Android são:

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

Figura 17 – Gerenciadores de Layout (ViewGroup)



Fonte: Adaptado de <https://developer.android.com/>.

3.2 Layouts Dinâmicos

Os componentes estáticos são bastante utilizados na construção das aplicações. Porém, existem também componentes dinâmicos que se ajustam por diversos fatores, como pela quantidade de informações a serem exibidas, pelo

formato da tela, etc. Dentre esses componentes dinâmicos, podemos citar a RecyclerView, a CardView, etc.

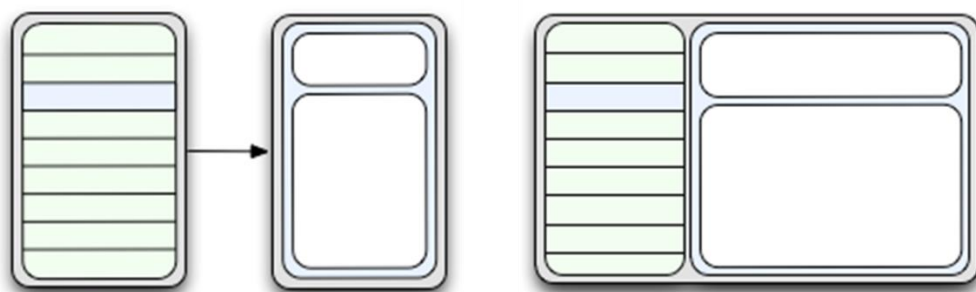
RecyclerView

A RecyclerView é uma versão mais avançada e mais flexível de uma ListView. As visualizações na lista são representadas por objetos fixadores de visualização, onde cada fixador é responsável por exibir um único item com uma visualização. Por exemplo, se sua lista mostra as mensagens do seu WhatsApp, cada fixador de visualização pode representar uma conversa. O RecyclerView cria fixadores de visualização necessários para exibir a parte da tela, além de alguns extras.

Fragments

Um Fragment é um componente Android independente que pode ser usado por uma Activity, ou seja, ele encapsula funcionalidades, de modo que é simples de incorporar nas Activities e nos layouts. Devido à sua modularização, a utilização de Fragments permite criar layouts flexíveis que podem ser ajustados com base no tamanho da tela do dispositivo, como tablets e smartphones. Um exemplo clássico, indicado pela Figura 18, é o aplicativo do Gmail. Se o aplicativo está aberto num smartphone, apenas a lista de emails é exibida na tela, ao passo que, se está aberto num tablet, a lista de emails é exibida juntamente com os detalhes do email.

Figura 18 – Exemplo de Fragments



Fonte: Adaptado do Google.

Capítulo 4. Armazenamento de Dados

O Android fornece várias opções para salvar os dados do aplicativo. A solução a ser utilizada depende das necessidades específicas de cada aplicativo, principalmente por duas questões: se os dados devem ser privados ou acessíveis a outros aplicativos (e ao usuário), e quanto espaço é necessário. A seguir, serão apresentadas algumas soluções de armazenamento de dados.

4.1 Shared Preferences

As Shared Preferences são recomendadas quando não se tem uma estrutura complexa e nem muitos registros para serem armazenados. As estruturas de informações armazenadas são em chave-valor, que são gravados em arquivos XML que ficam nas sessões do usuário mesmo que o aplicativo seja excluído. Os dados podem ser acessados por dois métodos principais da classe SharedPreferences: o `getPreferences()`, caso haja um único arquivo para salvar as preferências, e o `getSharedPreferences()`, caso haja vários arquivos.

4.2 File Systems

O Android usa um sistema de arquivos similar ao sistemas baseados em disco de outras plataformas: a classe File possui os métodos para leitura e escrita dos dados seguindo a mesma lógica.

Todos os dispositivos Android possuem duas áreas de armazenamento: armazenamento **interno** e **externo**. O armazenamento interno deriva das memórias não voláteis disponibilizadas pelos dispositivos, enquanto o armazenamento externo está relacionado à mídia de armazenamento num cartão microSD. 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.

Armazenamento Interno:

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

Armazenamento interno é recomendado quando se tem 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 podem ser compartilhados com outros apps ou caso seja permitido que os usuários os acessem pelo computador.

4.3 Banco de Dados

Os armazenamentos em Shared Preferences, arquivos internos e externos são praticados para um volume menor de dados. Porém, caso o conjunto de dados seja maior e mais estruturado, os bancos de dados são a melhor opção. Os bancos de dados são coleções organizadas que armazenam qualquer tipo de dados usando algum sistema de gerenciamento. O banco de dados SQLite é um exemplo de banco de dados no Android que pode ser utilizado por meio das APIs disponíveis no pacote `android.database.sqlite`.

Capítulo 5. Notificações e Publicação

5.1. Notificações

Notificações são mensagens utilizadas para alertar o usuário sobre algum evento que ocorreu no aplicativo. Geralmente, essas notificações sempre são exibidas fora das interfaces de usuário e são alertadas sem interromper as activities em execução.

Uma das formas de implementar notificações para os aplicativos é utilizando a plataforma do Firebase. O 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 é possível enviar mensagens apenas dizendo que tem novidades na aplicação, ou até mesmo enviar pequenas atualizações de conteúdo.

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.

5.2. Publicação

Após a concluir o desenvolvimento de um aplicativo Android, é possível disponibilizar o aplicativo para os usuários, sendo esse processo conhecido como **Publicação**. Para realiza-lo, existem duas principais tarefas que devem ser realizadas.

Preparação da aplicação

Nesta etapa o aplicativo deve ser gerado seguindo alguns passos: o primeiro item que devemos conferir são os atributos **applicationId**, **versionCode** e

versionName, que se encontram no arquivo **build.gradle** do módulo app, conforme figura abaixo:

```

apply plugin: 'com.android.application'

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.0"

    defaultConfig {
        applicationId "com.example.helloworld"
        minSdkVersion 21
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
}

```

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

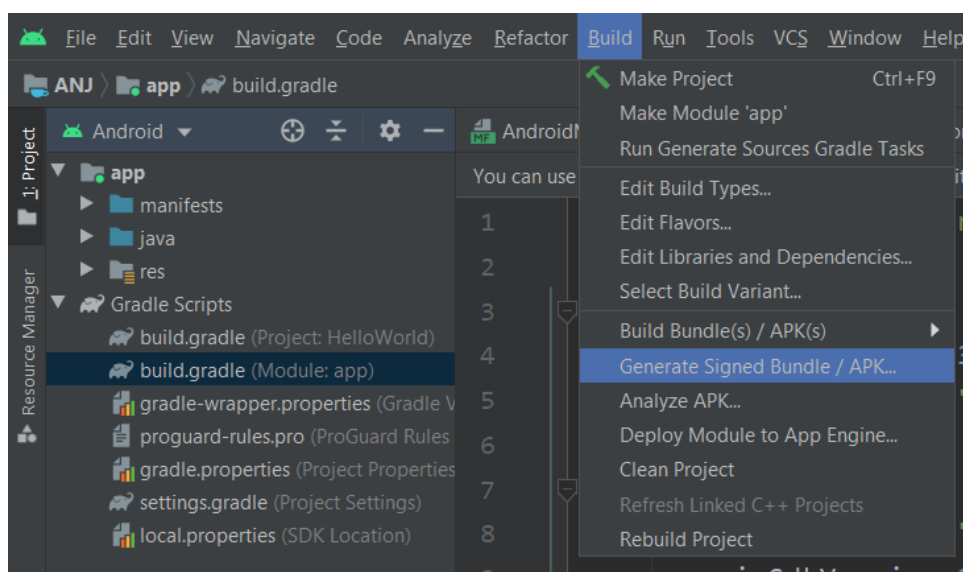
Assinando a aplicação

O segundo item é **assinar a aplicação**. Para submeter uma aplicação, a Google Play exige que toda aplicação seja assinada digitalmente, uma forma de identificar o autor da aplicação. A aplicação precisa ser assinada sempre com a mesma chave, assim, as atualizações ficam de forma transparente para o usuário. Quando é submetida uma nova versão para o Google Play, o mesmo valida se o apk foi assinado com a chave da primeira versão, caso contrário, não será possível realizar a atualização. Enquanto estávamos desenvolvendo nossa aplicação, o

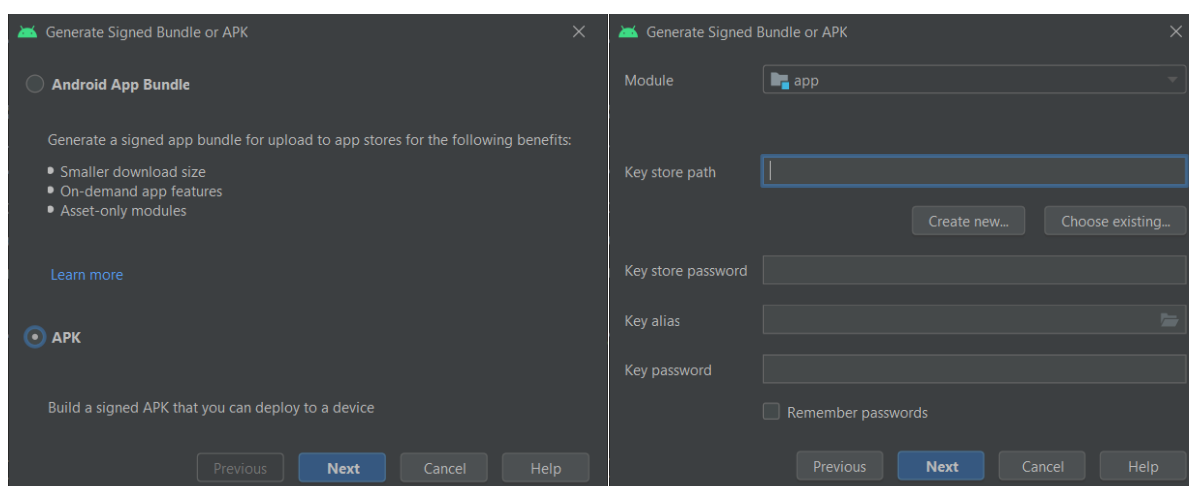
Android Studio já assina a aplicação com uma chave de debug, por isso conseguimos executar a aplicação no smartphone ou tablet.

Existem duas formas de assinar a aplicação: a convencional, onde é assinado o APK gerado, e a App Bundle, onde é disponibilizado o pacote do aplicativo para o Google Play disponibilizar o APK otimizado para cada configuração de dispositivo. A seguir, veja uma das formas de assinar a aplicação e exportar o apk assinado:

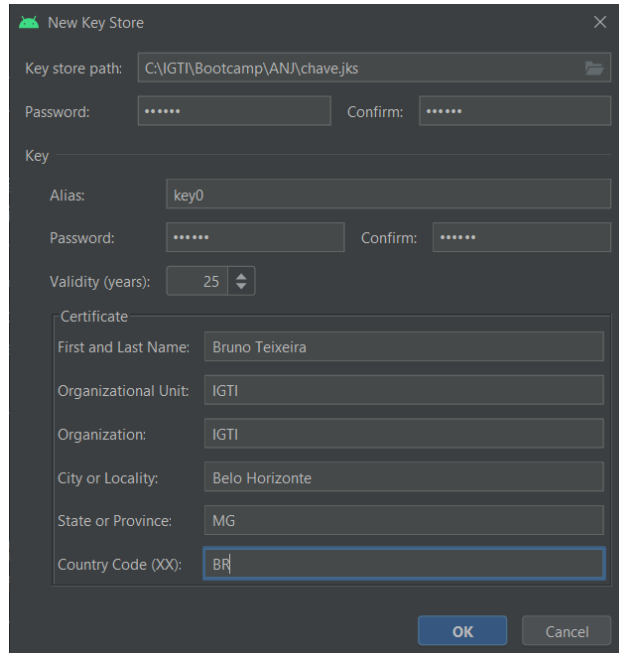
1. Na barra de navegação, vá em Build > Generate Signed APK.



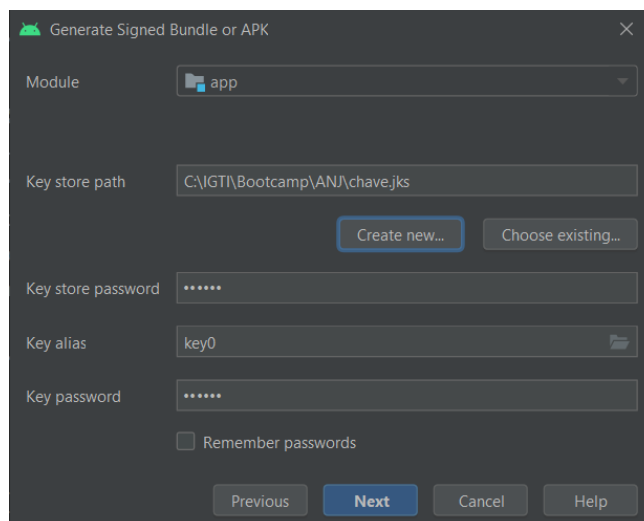
2. Selecione APK:



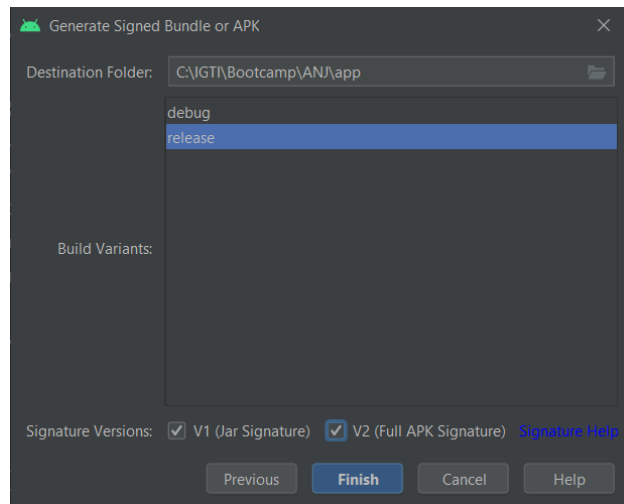
3. Crie uma chave que pode ser utilizada para assinar os seus aplicativos em **Create New** e preencha todos os campos conforme a figura abaixo. É recomendado que a chave tenha uma vida útil de no mínimo 25 anos, para que possa assinar atualizações.



4. Após a geração da chave, os valores são carregados e o próximo passo pode ser seguido.



5. O próximo passo é selecionar o variant de Build e o local onde será salvo o APK.



Liberar na Google Play

O próximo passo é subir a versão do aplicativo no Google Play Developer Console, para ter acesso à ferramenta, é necessária uma conta no Google, com um cartão de crédito internacional vinculado para o pagamento da taxa de U\$25,00.

Informações para o registro:
<http://developer.android.com/distribute/googleplay/developer-console.html>.

Referências

ANDROID. *Home*. Disponível em: <<https://developer.android.com/>>. Acesso em: 06 ago. 2020.

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

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.