Unidade 2

# **Aula 1**

# Android Studio e APIs do Android

Finalmente você começará a codificar um *app*, mas antes será necessário preparar seu ambiente de desenvolvimento e conhecer a interface básica do Android Studio. Além desse ambiente, é necessário levar em consideração a API mínima que seu *app* suportará, e tudo isso você vai aprender nesta aula.

## *Download* do Android Studio

Como para a maioria dos *softwares*, para desenvolver uma aplicação precisa-se de uma IDE (***I****ntegrated* ***D****evelopment* ***E****nvirorment*). No curso de POO, para aplicativos *desktop* em Java foi utilizado o Netbeans, que facilitou e agilizou o processo de desenvolvimento, mas para Android você utilizará a IDE Android Studio.

O Android Studio é a IDE oficial recomendada pela Google para desenvolvimento de aplicativos Android. Baseado na IDE IntelliJ IDEA da JetBrains, o Android Studio foi desenhado para fornecer novas ferramentas para o desenvolvimento de *apps* e para fornecer uma alternativa ao Eclipse, IDE utilizado anteriormente para desenvolvimento de aplicativos Android.

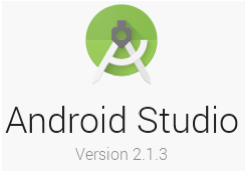


Figura : Logo do Android Studio

Baseado nos poderosos editores de texto e ferramentas de desenvolvedor da IntelliJ, o Android Studio oferece muitos recursos que aumentam a produtividade no desenvolvimento, como:

* um sistema de compilação baseado em Gradle (será abordado nas próximas aulas);
* um ambiente unificado no qual você pode desenvolver para qualquer equipamento com Android;
* execução instantânea para aplicar mudanças direto no aplicativo em execução sem a necessidade de compilar um novo APK;
* modelos de código e integração com o Github para auxiliar na construção de funcionalidades comuns de aplicativos e importação de códigos de exemplo;
* ferramentas e *frameworks* de testes;
* suporte a C++ e NDK;
* suporte ao Google Cloud Platform que facilita na integração com o Google CloudMenssaging e AppEngine.

### *Baixando o* Android Studio

Atualmente o Android Studio está em sua versão 2.1 e pode ser baixado direto do *site* de desenvolvedores Android (<https://developer.android.com/studio/index.html>) de forma gratuita.

Nessa página você encontrará um *link* para *download*, que dependendo do seu sistema operacional vai lhe direcionar para um arquivo diferente. O Android Studio está disponível para Windows, Mac e Linux.

1. Entre no site mencionado acima, clique no botão “**DOWNLOAD ANDROID STUDIO 2.1**”A.

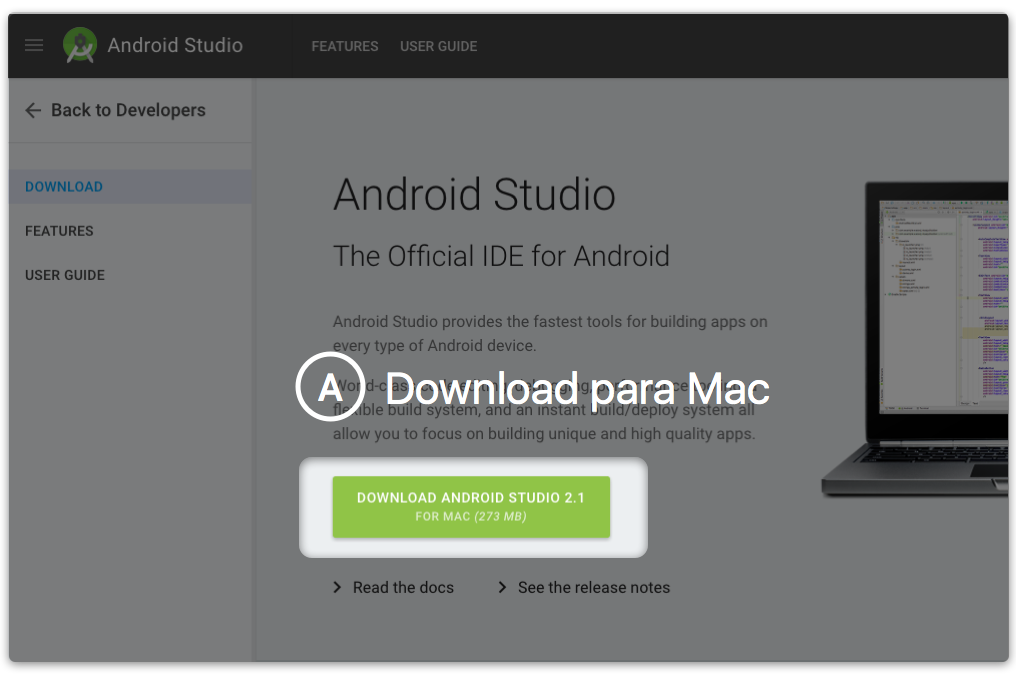


Figura : Iniciando o Download do Android Studio

1. Aceite os termos e condições de uso**B**, clique em **OK** e aguarde o *download*.

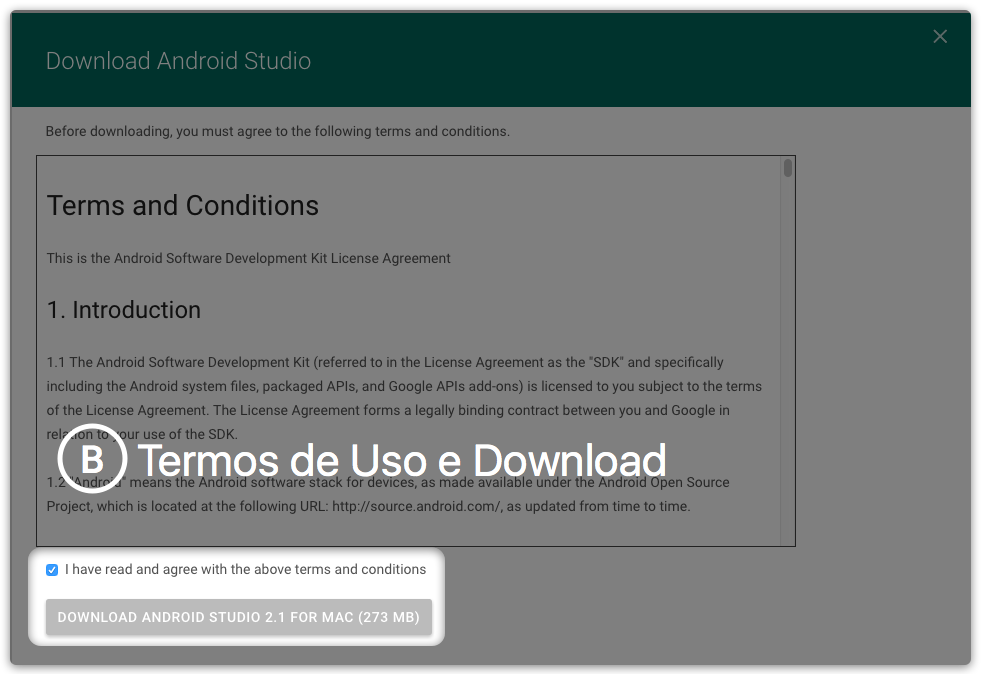


Figura : Aceitando os termos de uso do Android Studio

### Instalação do Android Studio

Lembra-se do Java SDK (JDK), mencionado no curso de Programação Orientada a Objetos (POO)? Enquanto o *download* está em andamento é necessário verificar se seu computador possui o JDK instalado. Para que o Android Studio funcione corretamente, é preciso que seu JDK esteja pelo menos na versão 1.6, mas a recomendada é a 1.8.

Primeiro verifique a versão do JDK, então abra seu terminal (caso esteja em Linux ou macOS) ou o CMD (caso esteja em Windows) e digite:

javac -version

O resultado esperado é um texto como este:

java version “1.8.0\_101”

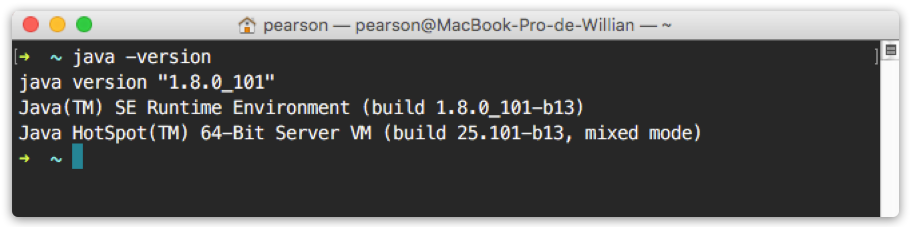


Figura : Terminal aberto mostrando a versão do Java instalada

Nesse caso, a saída apresentada indica que a versão instalada do SDK é 1.8.0. Se no seu terminal não apareceu uma versão anterior a esta, ou nem apresentou a versão, você deve baixar o JDK mais recente acessando esta página:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

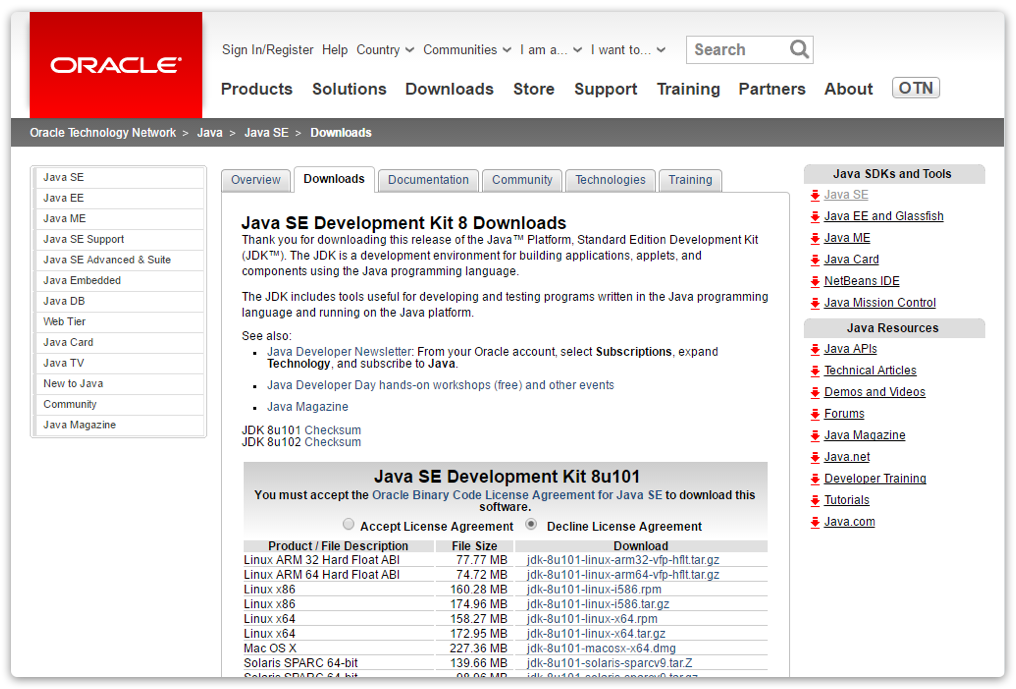


Figura : Página de download do Java SE

Aceite os termos **A** e o *download* de acordo com o seu sistema operacional **B** e instale-o.

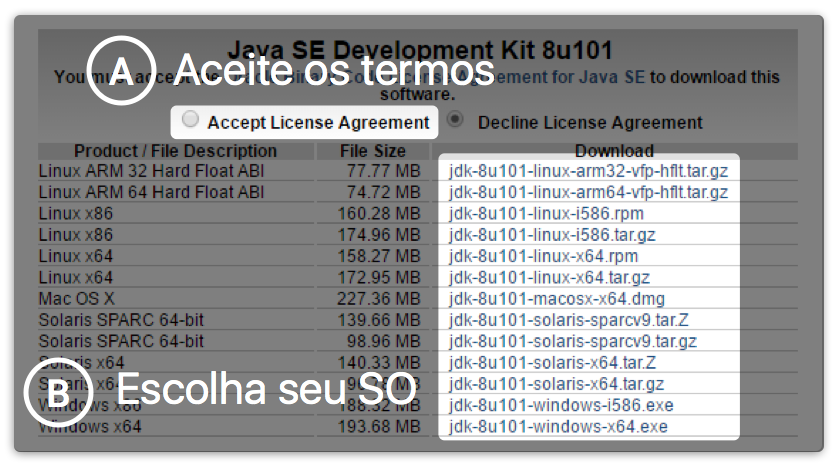


Figura : Passos para download do Java SE

#### Instruções para macOS

Após o término do download e de verificar que o JDK está instalado apropriadamente, realize o processo de instalação no macOS:

1. Execute o arquivo **.dmg** baixado.
2. Na nova tela que se abriu, clique e arraste o Android Studio para dentro da pasta **Aplicativos**, em seguida execute-o.



Figura : Início da instalação do Android Studio em Mac. Clique e arraste o app para a pasta Aplicativos

1. Se você já usou o Android Studio neste computador, você pode importar seus projetos se preferir.

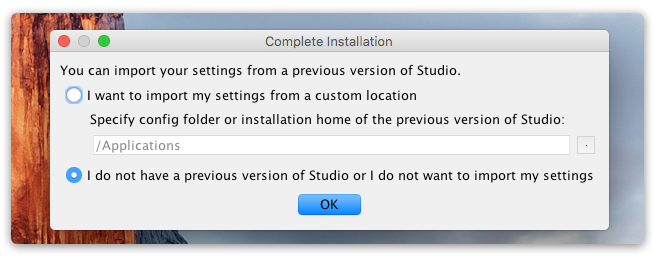


Figura : Completando a instalação - Restaurando configurações prévias

1. O *wizard* (ou assistente de instalação) do Android Studio lhe guiará até o final do processo de instalação, apenas escolha suas preferências e prossiga.

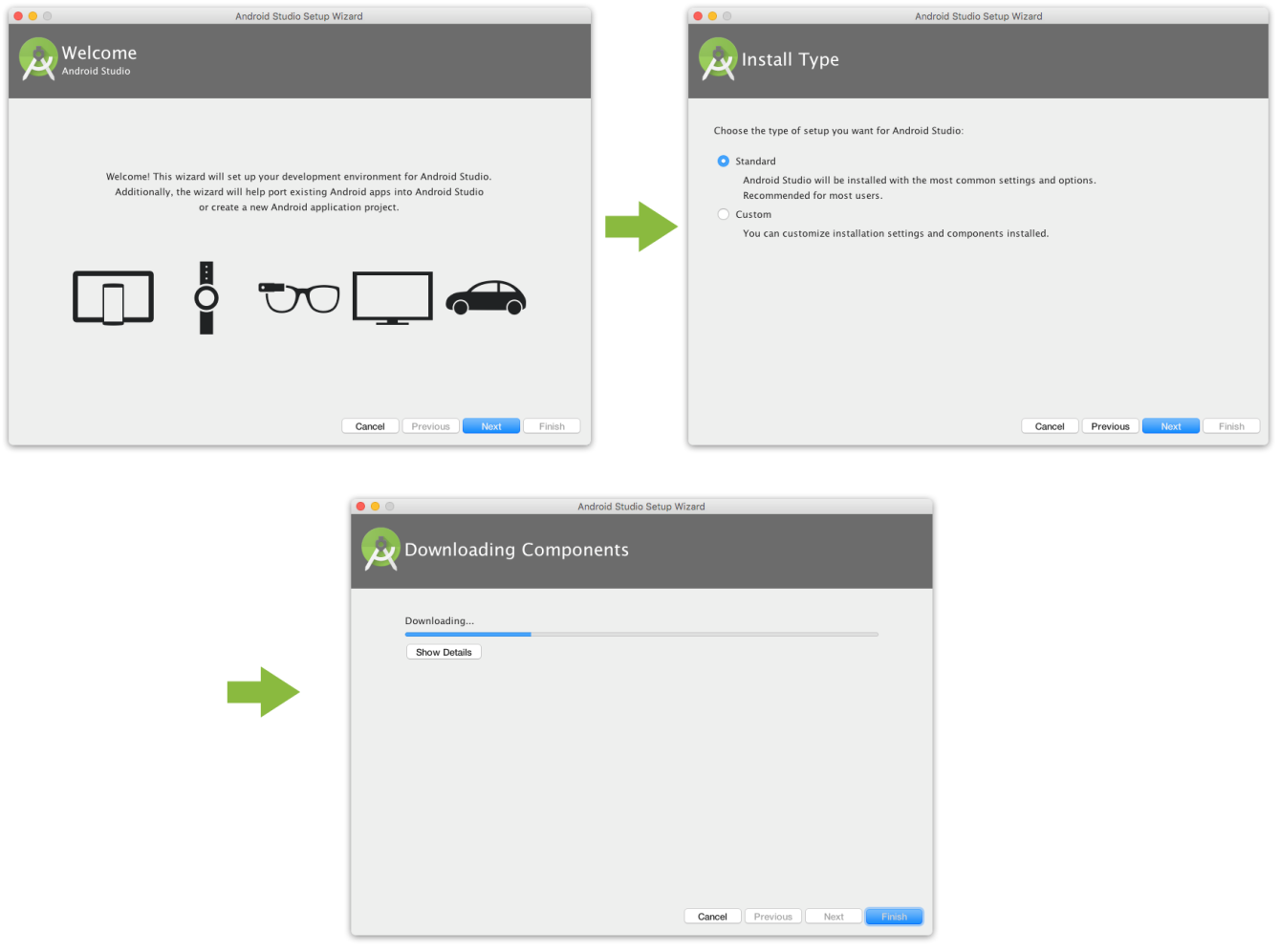


Figura : Sequência de telas para instalação do Android Studio

Isso é o bastante para a instalação no macOS.

#### Instruções para Linux

Desenvolver aplicativos utilizando o Android Studio é mais recomendado em ambiente Linux, pois além de ser um SO mais leve, a IDE foi feita para rodar principalmente nele, então você encontrará mais fluidez e menos travamentos utilizando o Linux. Para fazer a instalação, siga os seguintes passos:

1. Extraia o arquivo **.zip** baixado para uma localização apropriada para seus aplicativos, como **/usr/local** para usuário normal ou **/opt/** para usuários compartilhados.
2. Para executar o Android Studio, abra o terminal, navegue até o diretório **android-studio/bin** e execute o **arquivo.sh**.

**DICA**: Adicione o diretório **android-studio/bin/** para a sua variável de ambiente **PATH** para que você possa inicializar o Android Studio a partir de qualquer diretório.

1. Se você já havia utilizado o Android Studio neste computador, importe quaisquer configurações anteriores, se assim preferir, e clique em OK.
2. Da mesma forma que no macOS, o *wizard* do Android Studio v lhe guiará durante o processo de instalação, inclusive para instalações adicionais dos componentes do SDK que podem ser requeridos no desenvolvimento.

**DICA:** Se você está utilizando uma versão 64-bit do Ubuntu, você precisará instalar algumas bibliotecas de compatibilidade com 32-bit. Use o seguinte comando:

sudoapt-getinstall lib32z1 lib32ncurses5 lib32bz2-1.0 lib32stdc++6

Se você está utilizando uma versão 64-bit do Fedora, o comando é o seguinte:

sudoyuminstall zlib.i686 ncurses-libs.i686 bzip2-libs.i686

#### Instruções para Windows

Se você não está usando macOS nem Linux, então siga as instruções para o Windows:

1. Execute o arquivo**.exe** baixado.
2. Siga os passos do *wizard* para instalar o Android Studio e as ferramentas necessárias.

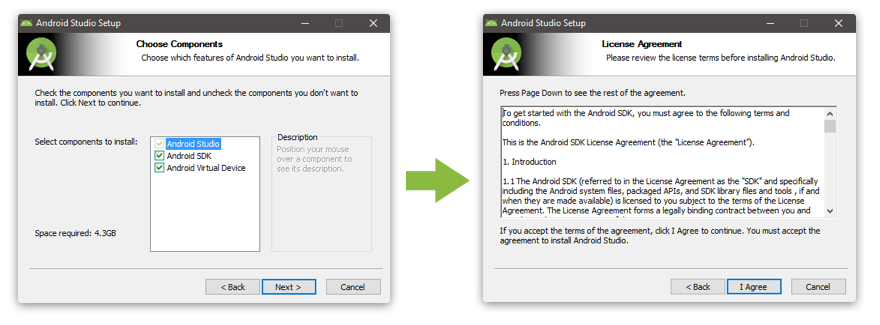


Figura : Sequência de telas para instalação do Android Studio em Windows - Parte 1

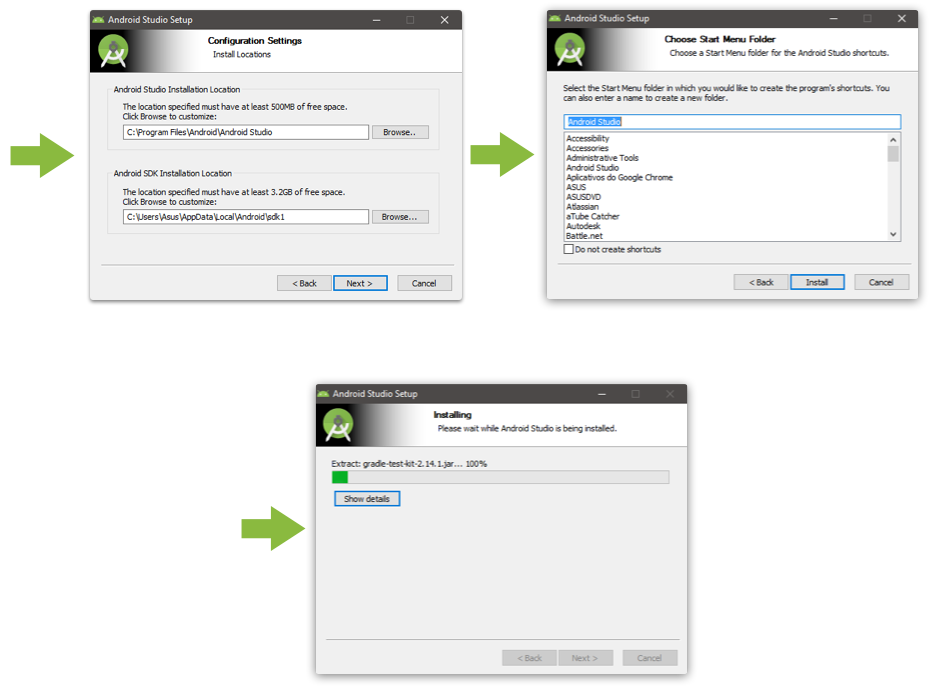


Figura : Sequência de telas para instalação do Android Studio em Windows – Parte 2

1. Alguns sistemas Windows poderão não reconhecer que o JDK está instalado e não conseguirão executá-lo quando o Android Studio o requerer. Se você está encontrando esse problema, você precisa ajustar uma variável de ambiente para indicar a localização correta do JDK.

Em uma instalação comum, selecione o **Menu Iniciar > Meu Computador > Propriedades > Propriedades Avançadas de Sistema**. Então abra a aba **Avançado > Variáveis de Ambiente** e adicione uma nova nomeando-a de JAVA\_HOME, que deverá apontar para o diretório do seu JDK, que neste caso está em **C:\Program Files\Java\jdk1.8.0\_101**.

A instalação para Windows pode ser um pouco mais complicada do que nos demais sistemas, então peça ajuda ao educador em caso de dúvidas.

Após esses passos, o Android Studio estará pronto para uso. Agora você vai conhecê-lo melhor.

## API mínima e *target*

As versões do Android, mencionadas na parte 1.4.4 da aula 1 da unidade 1, tem outros nomes quando os desenvolvedores as mencionam.

Essas versões, em termos técnicos, são chamadas API (*Application Programming Interface*) e cada versão é representada por um nível.

API é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades em aplicações que não pretendem entrar em detalhes da sua implementação, mas apenas consumi-las.

O Android possui diversas APIs, uma para cada versão, e cada uma delas apresenta um conjunto de funcionalidades específicas para essas versões. A cada novo nível, novas funcionalidades são incrementadas, algumas depreciadas e outras removidas. Isso implica na forma e na complexidade de desenvolvimento de um *app*. Quanto mais baixo o nível da API, mais *devices* seu aplicativo abrangerá, porém suas funcionalidades e seus padrões estarão cada vez mais limitados e desatualizados em relação aos *apps* mais atuais.

Fazendo um comparativo com o Java SE, utilizado no curso de POO, o que o difere do Android SDK é a sua API. Não seria possível desenvolver um *app* Android no Java SE por si só, pois é necessário adicionar as APIs para que se tenha acesso a este conjunto de funcionalidades que compõe um *app* Android. Veja os níveis de API do Android:

|  |  |  |
| --- | --- | --- |
| **Nível da API** | **Versão** | **Codinome** |
| 1 | 1.0 | *Alpha* |
| 2 | 1.1 | *Beta* |
| 3 | 1.5 | *Cupcake* |
| 4 | 1.6 | *Donut* |
| 5 | 2.0 | *Eclair* |
| 6 | 2.0.1 |
| 7 | 2.1 |
| 8 | 2.2 - 2.2.1 - 2.2.2 - 2.2.3 | *Froyo* |
| 9 | 2.3 | *Gingerbread* |
| 10 | 2.3.3 - 2.3.4 - 2.3.5 - 2.3.6 - 2.3.7 |
| 11 | 3.0 | *Honeycomb* |
| 12 | 3.1 |
| 13 | 3.2 - 3.2.1 - 3.2.2 |
| 14 | 4.0.1 - 4.0.2 | *Ice Cream Sandwich* |
| 15 | 4.0.3 - 4.0.4 |
| 16 | 4.1.1 - 4.1.2 | *JellyBean* |
| 17 | 4.2 - 4.2.1 - 4.2.2 |
| 18 | 4.3 |
| 19 | 4.4.2 - 4.4.3 - 4.4.4 | *KitKat* |
| 20 | 4.4W.2 |
| 21 | 5.0 - 5.0.1 - 5.0.2 | *Lollipop* |
| 22 | 5.1 - 5.1.1 |
| 23 | 6.0 - 6.0.1 | *Marshmellow* |
| 24 | 7.0 | *Nougat* |

### API mínima

Antes de começar seu projeto, você deve levar em consideração a API mínima do seu aplicativo.

Agora que você viu a tabela das APIs e suas versões, você deve sempre levar em consideração o público que seu *app* atingirá. A lógica é simples: quanto menor for o nível da API escolhida, maior será a quantidade de *devices* compatíveis com seu *app*; em contrapartida, maior será a limitação de funcionalidades disponibilizadas pela API em comparação com as mais altas. Veja a tabela abaixo tirada do *site* do Android.

|  |  |  |  |
| --- | --- | --- | --- |
| **Versão** | **Codinome** | **Nível da API** | **Distribuição** |
| [2.2](https://developer.android.com/about/versions/android-2.2.html) | *Froyo* | 8 | 0.1% |
| [2.3.3 -](https://developer.android.com/about/versions/android-2.3.3.html)  [2.3.7](https://developer.android.com/about/versions/android-2.3.3.html) | *Gingerbread* | 10 | 1.7% |
| [4.0.3 -](https://developer.android.com/about/versions/android-4.0.html)  [4.0.4](https://developer.android.com/about/versions/android-4.0.html) | *Ice Cream Sandwich* | 15 | 1.6% |
| [4.1.x](https://developer.android.com/about/versions/android-4.1.html) | *JellyBean* | 16 | 6.0% |
| [4.2.x](https://developer.android.com/about/versions/android-4.2.html) | 17 | 8.3% |
| [4.3](https://developer.android.com/about/versions/android-4.3.html) | 18 | 2.4% |
| [4.4](https://developer.android.com/about/versions/android-4.4.html) | *KitKat* | 19 | 29.2% |
| [5.0](https://developer.android.com/about/versions/android-5.0.html) | *Lollipop* | 21 | 14.1% |
| [5.1](https://developer.android.com/about/versions/android-5.1.html) | 22 | 21.4% |
| [6.0](https://developer.android.com/about/versions/marshmallow/index.html) | *Marshmallow* | 23 | 15.2% |

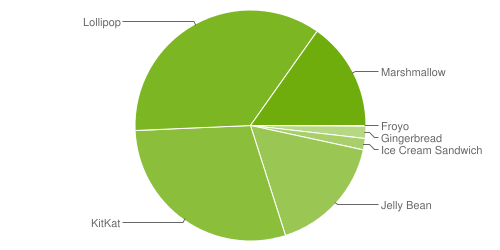


Figura : Gráfico de proporção das versões Android instaladas

Observando os dados, vê-se que a versão *KitKat* é a mais adepta pelos usuários e as versões menores do que 4.0.3 não atingem nem 2% dos usuários.

O que muitos profissionais indicam é que se desenvolva a partir da API 15, pois abrangerá mais de 98% de usuários, mas na prática existirão mais dificuldades e aplicativos com aparência ultrapassada.

Uma API ideal é a 4.4 (*KitKat*), pois nela se cobre aproximadamente 80% dos usuários e ela não é tão defasada. Apesar de as grandes mudanças visuais que chegaram na API 21 não estarem presentes nas anteriores, a Google lançou bibliotecas de suporte para versões mais antigas conseguirem utilizar alguns recursos do *Material Design*. A API 19 recebeu essa biblioteca.

Então, a melhor escolha ainda é o Android *KitKat*.

### *Targets*

Já foi explicado o que é uma API e já foi citada a necessidade de se escolher bem uma API mínima, mas o que são as *targets*?

Segundo a Google, a *target* indica qual será a API principal na qual o *app* será desenvolvido, ou seja, a *target* define a API máxima e o app não rodará em *devices* com versões posteriores à versão *target*.

**Qual é a target recomendável?**

Quando se utilizava a IDE Eclipse no lugar da Android Studio, a especificação da target era obrigatória. Hoje com o Android Studio 2.1, é assumido na criação do projeto que a API *target* será a mais alta, por padrão.

Quando se utiliza a API mais alta, isso implica que as bibliotecas suporte deverão acompanhar as funcionalidades para esta API, pois será disposto para você utilizar o que há de mais novo na API do Android. Então é necessário sempre cuidar para que as APIs mais baixas que a target (porém superiores com a API mínima) mantenham-se compatíveis.

## Atualizando o SDK do Android

Pelo menos uma vez por ano é lançada uma versão nova do Android. Além disso, constantemente são lançadas melhorias e ferramentas novas para auxiliar no desenvolvimento.

Em meio a tantas versões do Android sendo lançadas rapidamente, deve-se de tempo em tempo testar o *app* para verificar se ele ainda funciona nessa nova versão.

Para lhe notificar de novas atualizações, sejam elas novas APIs, atualizações na IDE ou ferramentas de desenvolvimento novas, o Android Studio lhe mostrará numa caixa de diálogo (notificação em forma de balão) que existem novas atualizações. Mas como atualizar?

Desde antes do surgimento do Android Studio, quando se desenvolvia para Android na IDE Eclipse, existia o chamado SDK Manager. Esse *manager* é uma central que oferece ferramentas, plataformas e outros componentes do SDK necessários para desenvolver aplicativos.

Para atualizar o SDK do Android, abra o Android Studio e siga os seguintes passos:



1. Com o Android Studio aberto, localize e clique no botão (**SDK Manager)** na barra de ferramentas.

**DICA**: Uma segunda alternativa para abrir o Android SDK Manager é clicar no menu **Tools >Android> SDK Manager**.

Ao abrir esta nova tela, estarão listadas todas as versões/APIs disponíveis do Android.

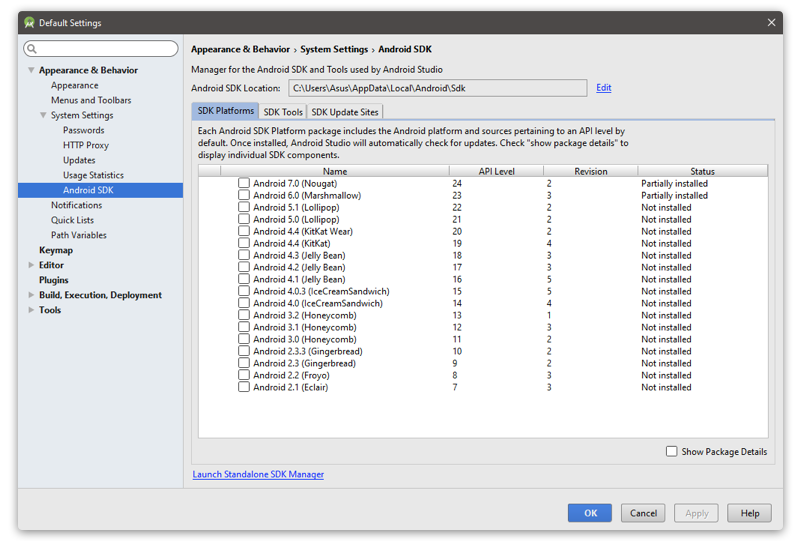


Figura : Tela ADB

1. Para instalar novas APIs, basta selecioná-las na lista e clicar em **OK.** Por enquanto não instale nada;
2. Se você deseja instalar ferramentas adicionais ao SDK ou outro meio de instalar novas APIs, com o Android Studio aberto, localize e clique no botão **SDK Manager** na barra de ferramentas;
3. Ao abrir esta tela, que é a mesma aberta no passo 1, localize a opção **Launch Standalone SDK Manager.** Uma nova tela se abrirá:

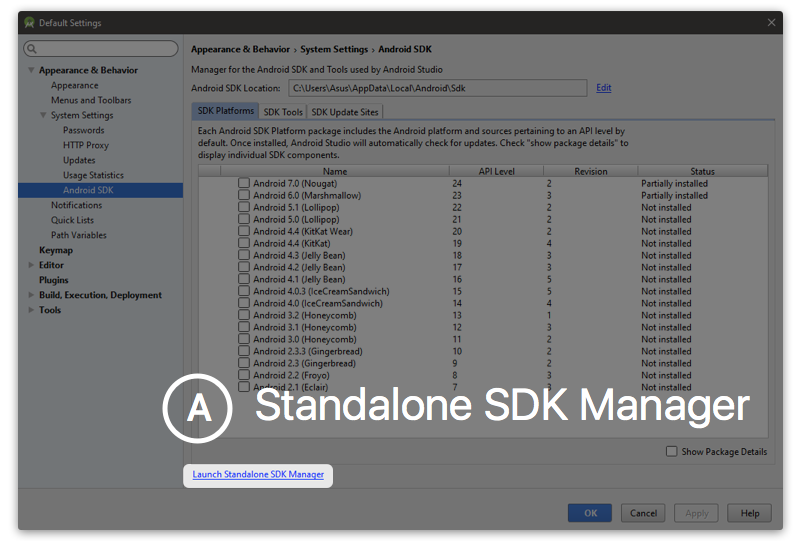


Figura : Abrindo o Standalone SDK Manager

1. Nesta nova tela, instale todas as ferramentas que desejar.

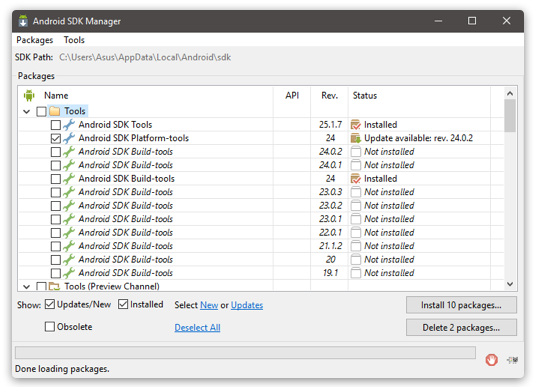


Figura : Tela do Standalone SDK Manager aberta

Agora que você sabe adicionar novos pacotes ao seu SDK, realize uma atualização básica.

1. Neste curso você desenvolverá aplicativos a partir da versão 5.1 (*Lollipop*), então, com o **Standalone SDK Manager** aberto, selecione a API 22, 23 e 24**A** como na imagem seguinte:

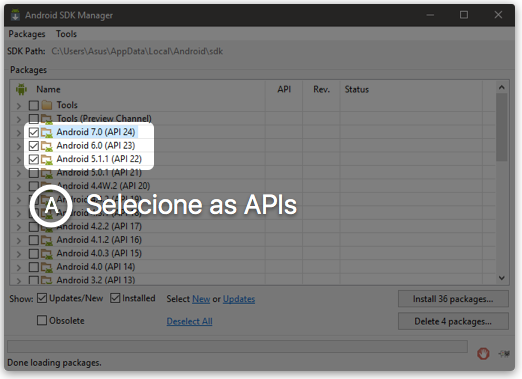


Figura : Selecionando algumas APIs para download

**DICA**: Para facilitar, feche todos os detalhes da lista de pacotes para obter uma visualização como na imagem anterior.

1. Desça ao final da lista e abra o pacote **Extras.** Você vai selecionar alguns recursos desse pacote. Selecione os itens**B**:
   1. Android Support Repository;
   2. Google Play Services;
   3. Google Repository;
   4. Google USB Driver;
   5. Intel x86 Emulator Accelerator (HAXM installer).

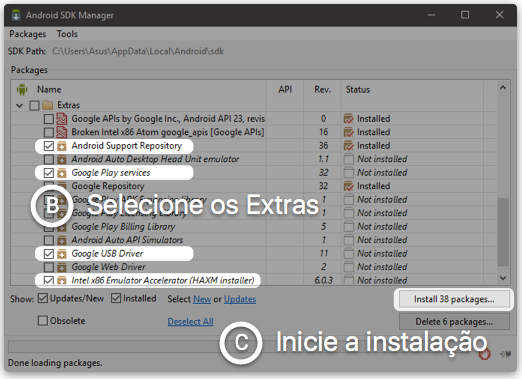


Figura : Selecionando algumas ferramentas extras para download

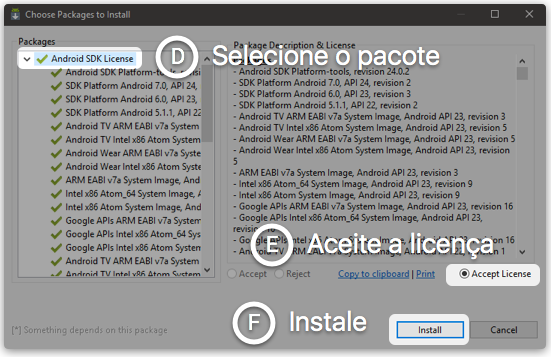
1. Clique no botão **Install**C;
2. Aceite os termos de licença. Para isso selecione o **item de licençaD** e clique em **AcceptLicenceE**. Faça isso para todas as licenças e clique em **InstallF** para confirmar a operação. Este processo é bem demorado, então aproveite para tomar uma água ou café enquanto isso.

Figura : Termos e aceite final para instalação das novas bibliotecas

Agora que você tem essas ferramentas instaladas, é importante lembrar que você deve sempre ficar de olho no SDK Manager. Sempre que houver alguma novidade, melhorias, correções de *bugs* em APIs, o SDK Manager vai disponibilizá-las para você. Então é importante que você sempre se mantenha atualizado e verifique as novidades.

## Escrevendo o primeiro programa - Hello World

Finalmente você vai criar seu primeiro *App*. Primeiro você criará o projeto (lembrando que você utilizará a versão 2.1 do Android Studio para os exemplos deste curso).

### Criando um novo projeto

Existem dois caminhos para criação de um novo projeto:

1. Se você ainda não criou nenhum projeto, apenas a tela inicial do Android Studio aparecerá para você. Então abra o Android Studio e selecione a opção **Start a new Android Studio ProjectA**.

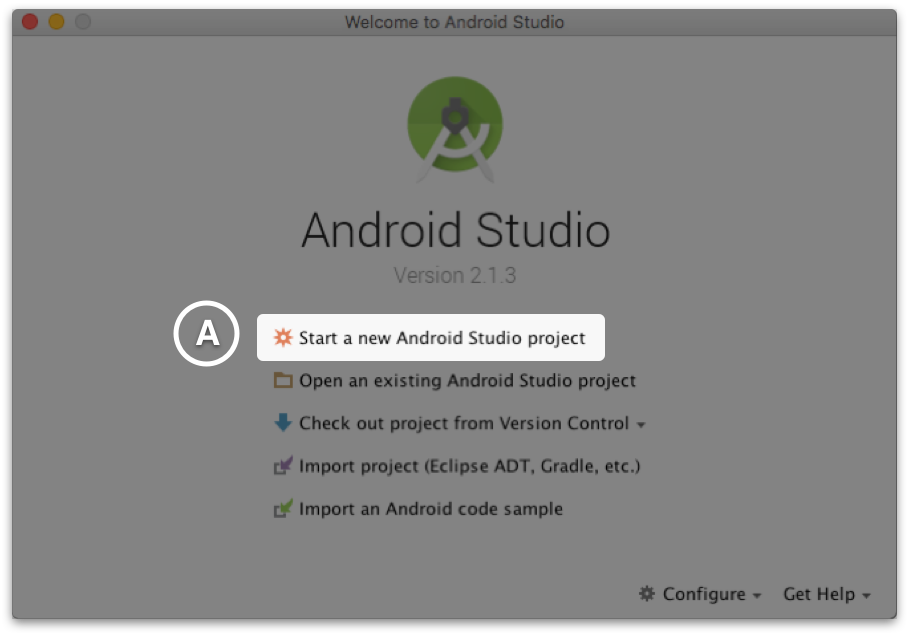


Figura : Tela inicial do Android Studio

1. Se você já criou algum projeto no Android Studio anteriormente e a interface principal do Android Studio já está sendo mostrada, clique em **File > New > New Project.**

A partir de agora as etapas são comuns aos dois caminhos citados anteriormente, então siga os seguintes passos:

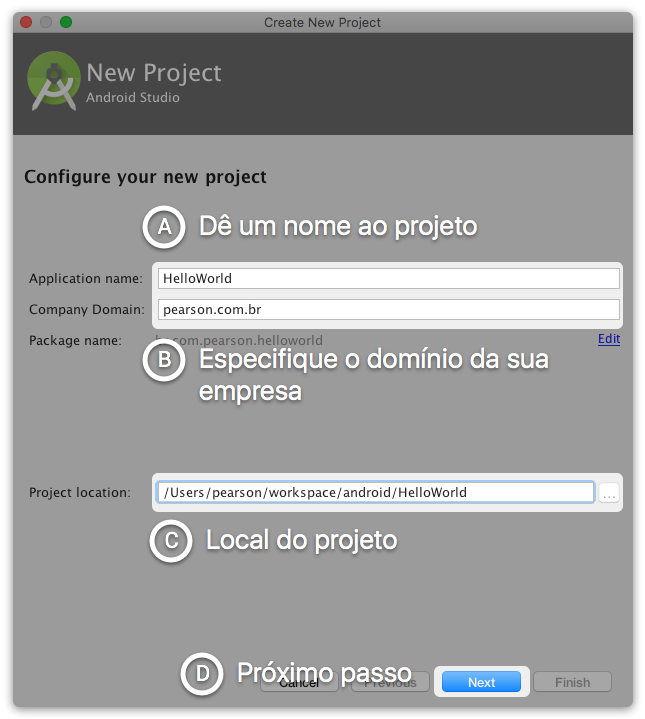


Figura : Primeiro passo para criação do projeto - Nome e Local

1. Na tela que se abriu, escolha um nome para seu projeto. No exemplo foi escolhido o nome, que no caso será **HelloWorldAppA;**
2. Escolha um domínio para sua organização (pode ser fictício). No exemplo foi utilizado **pearson.com.brB**;
3. Escolha o local desejado para salvar o projeto**C** e clique em **NextD**;
4. Na nova tela, deve-se escolher a API mínima e as plataformas que o *app* suportará. Deixe marcado a opção **Phone and TabletE** e selecione **API 19: Android 4.4 (KitKat)** no campo **Minimum SDKF**. Clique em **NextG.**

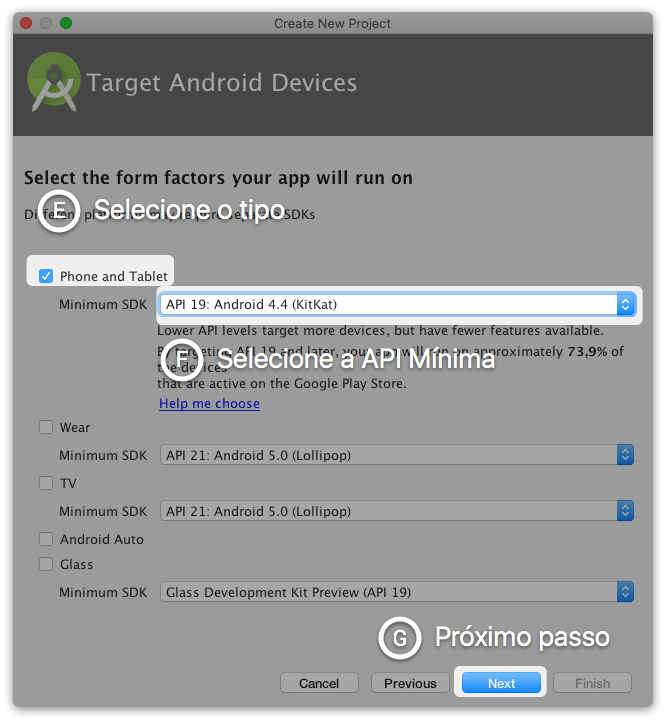


Figura : Segundo passo para criação do projeto - Seleção da API mínima

1. Neste passo deve-se escolher um ponto de partida para seu *app*. Existem diversos pontos de partida que geram telas com diferentes funcionalidades. Como você está aprendendo, inicie pela **Empty ActivityH**. Clique em **Next**I.

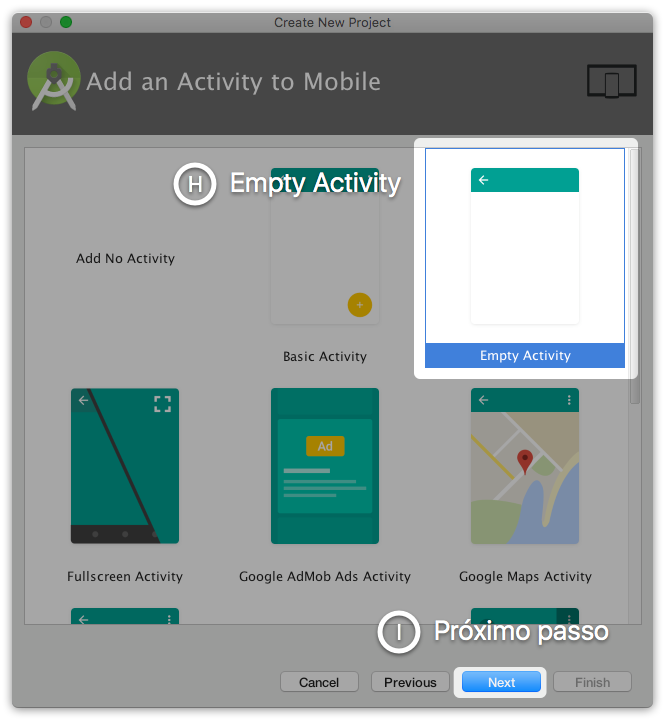


Figura : Terceiro passo para criação de um projeto - Adicionando uma Activity

1. Agora é preciso escolher um nome para sua primeira Activity (cujo conceito você entenderá nas próximas aulas, mas de modo geral, as *Activities* são as telas). No seu caso, esse nome será MainActivity. Deixe como está e clique em **Finish**.

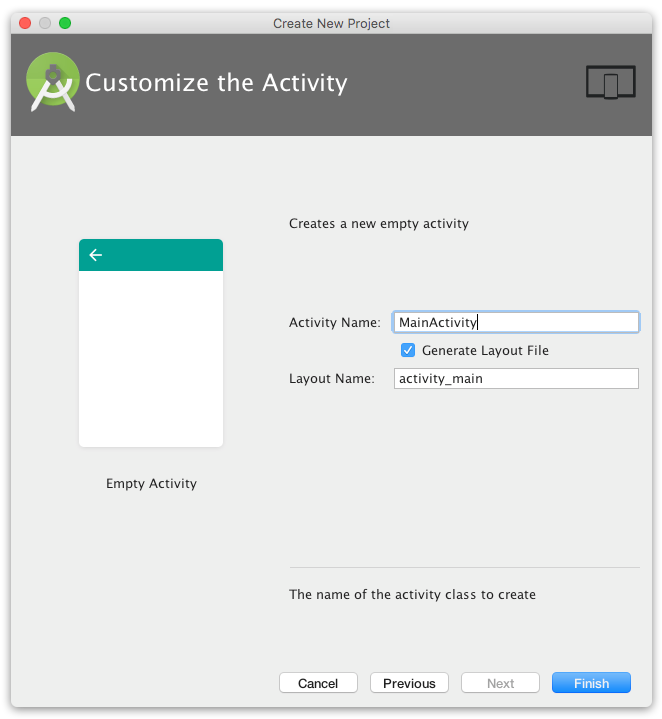


Figura : Nomeando a primeira Activity

Agora você tem seu primeiro projeto criado.

Se você se lembrar do seu primeiro projeto no curso de POO, se lembrará da maldição do Hello World. Trata-se de uma lenda entre os programadores que diz que se o primeiro programa que um programador escrever quando estiver estudando uma nova linguagem não for o HelloWorld, esse programador estará amaldiçoado. Sua maldição é não conseguir aprender a linguagem em questão.

Aparentemente a Google sabe disso e leva a sério essa lenda, por isso, para garantir que você não seja amaldiçoado no Android, todos os novos projetos já vem com o HelloWorld.

Já que o HelloWorld vem pronto, é hora de incrementá-lo.

### Brincando no Android Studio

Será feita uma modificação pequena. Alterar a *label* “HelloWorld!”e traduzi-la para “Olá mundo!”.

Para facilitar os passos, acompanhe a imagem a seguir:

1. Com o Android Studio aberto, abra com um duplo clique o arquivo **activity\_main.xml** (A).

Não se preocupe ainda com a interface do Android Studio, logo ela será apresentada. Esse arquivo se encontra no painel de navegação (à esquerda) no caminho **app > res > layout > activity\_main.xml.** Você pode guiar-se pela imagem abaixo.

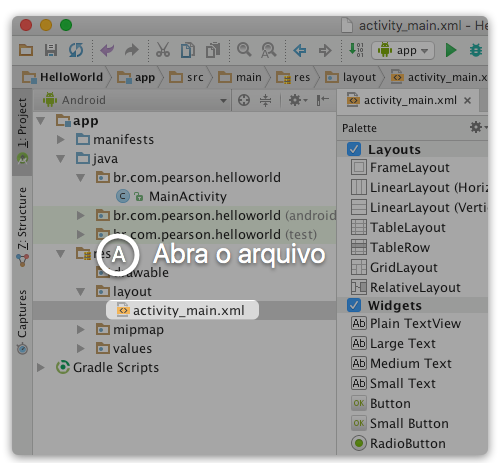


Figura : Localização do arquivo **activity\_main.xml**

1. Agora você estará no modo ***Desing*** da tela. Localize o botão de *zoom*  e aumente-o adequadamente para que a visualização do texto "Hello World!" fique confortável;
2. Agora dê um duplo clique no texto "Hello World!" e uma caixa de edição aparecerá (C). Altere o texto para "Olá mundo!".



Figura : Exibindo caixa de edição de texto

### Executando o aplicativo no celular

O último passo é executar o projeto. Quando se desenvolve aplicativos, é necessário testar em diferentes resoluções de telas e diversos *devices*. Para isso existem emuladores a sua disposição, mas você aprenderá a instalá-los somente na aula seguinte. Para executar esse projeto, será utilizado seu próprio *smartphone* (Android, é claro). Então siga em frente:

1. Conecte seu Android ao seu computador utilizando um cabo USB.

**DICA**: Se você estiver utilizando Windows, alguns modelos de celular podem requerer a instalação de um *Driver* para que o Windows o reconheça. Cada *smartphone* tem um procedimento diferente, então se o seu computador não reconheceu seu Android, instale o *Driver* apropriado (consulte o fabricante do seu *smartphone* em caso de dúvida).

1. Agora você deve habilitar o modo **Programador (*Developer*)** de seu celular. Esse processo varia devido à grande quantidade de dispositivos Android diferentes, então ele será exemplificado utilizando um *device* que possui uma versão “pura” do Android.

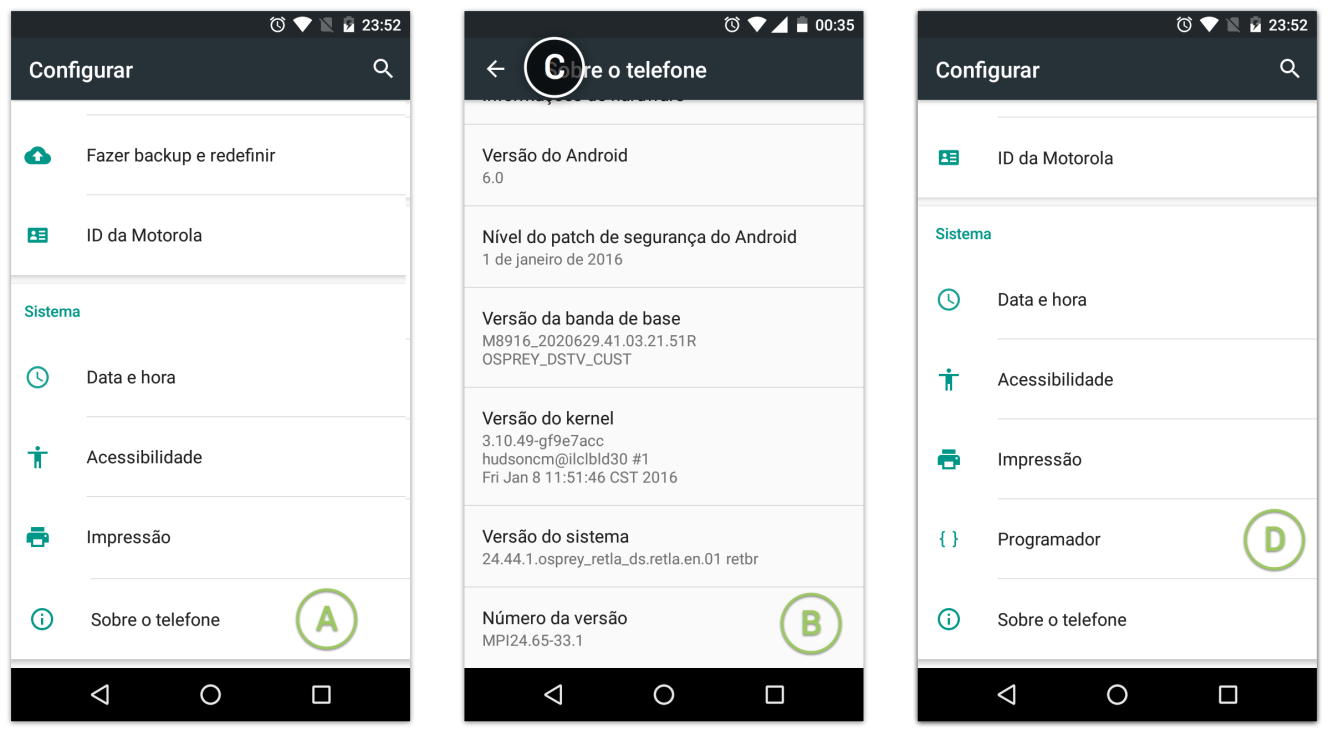


Figura : Sequência (A > B > C > D) para ativação do modo programador

* 1. Em seu *smartphone* vá para **Configurações > Sobre o TelefoneA**;
  2. Na nova tela, localize o campo **Número da versão** e toque nele enfaticamente, até que a mensagem de liberação apareça;
  3. Volte para a tela de **Configurações** e veja que existe uma nova opção: **Programador.**

1. Em seu *smartphone*, vá para **Configurações >ProgramadorC** e ative o modo **ProgramadorD**;
2. Nessa mesma tela, ative a **Depuração USBE** e permita-a quando uma caixa de diálogo aparecer**F**. Clique em **OK**.



Figura : Configurando o modo debug

1. Uma notificação deverá aparecer. Você agora poderá executar o aplicativo no seu *device*;
2. Volte ao projeto HelloWorldApp no Android Studio e clique no botão **Run**;
3. Aguarde seu *device* aparecer na lista, selecione-o e clique em **OK**.

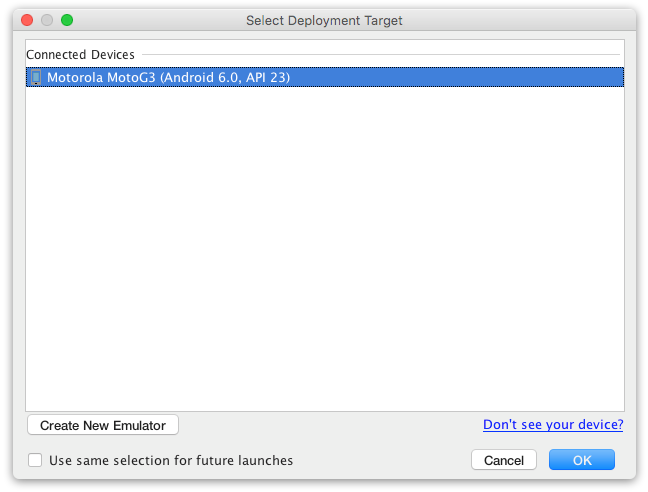


Figura : Tela do ADB mostrando o seu smartphone conectado

Veja! O seu primeiro *App* está pronto! Emocionante, não é?

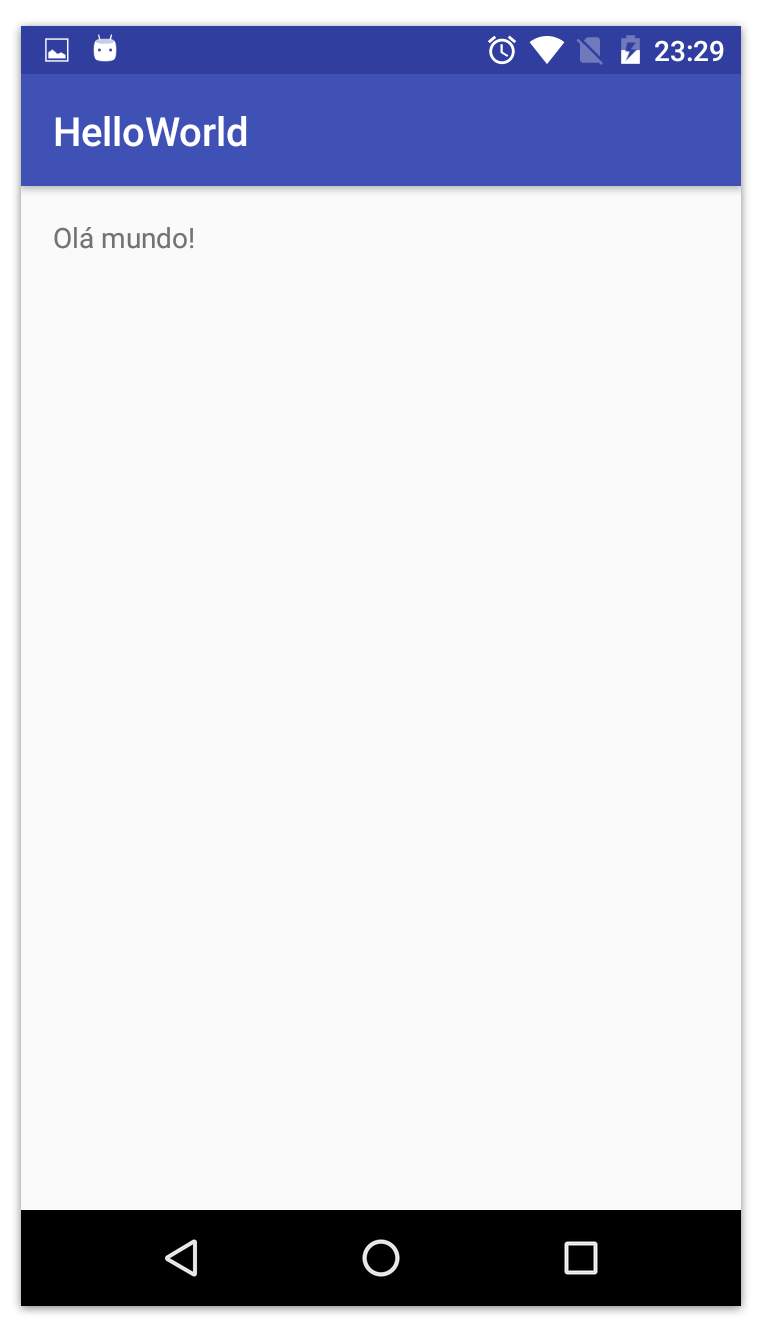


Figura : App HelloWorld sendo executado

# Interface do Android Studio

O Android Studio sem dúvidas é a melhor IDE para desenvolvimento de aplicativos em Android, que agora, em sua versão 2.1, está melhor ainda. Porém nem tudo são flores.

Essa IDE possui tudo que você precisa, coisas que você precisará algumas vezes e outras que quase nunca precisará. Isso acaba atrapalhando os iniciantes na plataforma, pois são muitas informações em uma tela só. Pensando nisso, foi preparado um *tour* básico pela interface do Android Studio. Vamos lá!

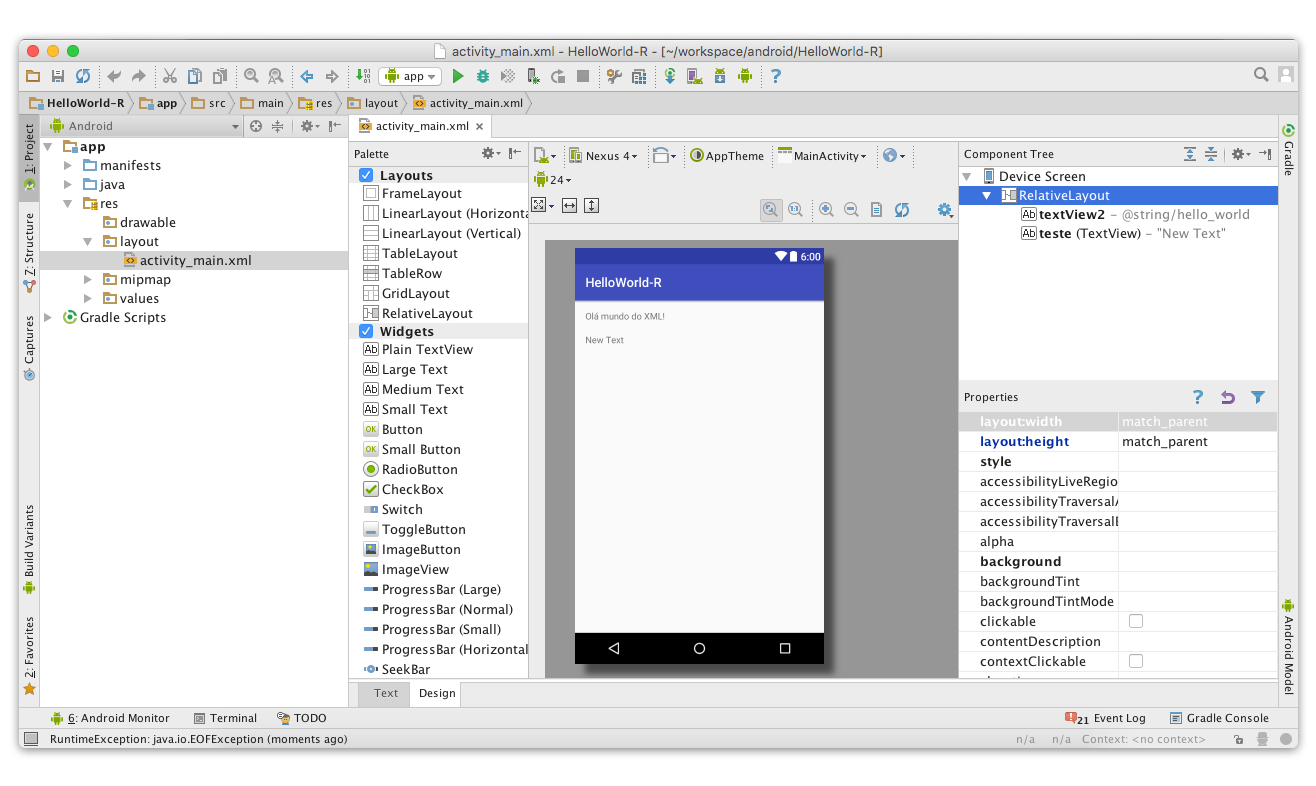


Figura : Visão geral do Android Studio

1. **Geral:** a interface é dividida em seções que serão mencionadas a seguir. Todas essas seções são redimensionáveis e também possuem o botão esconder para que o uso do Android Studio seja confortável em qualquer tamanho de tela;
2. **Estrutura do projeto:** nesta seção estão listados todos os arquivos contidos nos projetos, sejam eles criados pelo Android Studio automaticamente ou por você. Essa estrutura é apresentada em forma de árvore de diretório e será bastante utilizada no decorrer do curso.

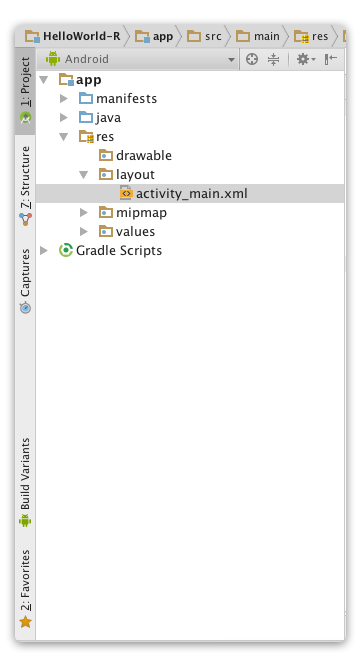


Figura : Estrutura do projeto

1. **Android Monitor:** aqui ficam todas as informações do que acontece internamente quando o *app* está sendo executado. Coleta de erros, *debug* (que será abordado na aula seguinte), *logs* e mensagens são apresentados nesta seção.

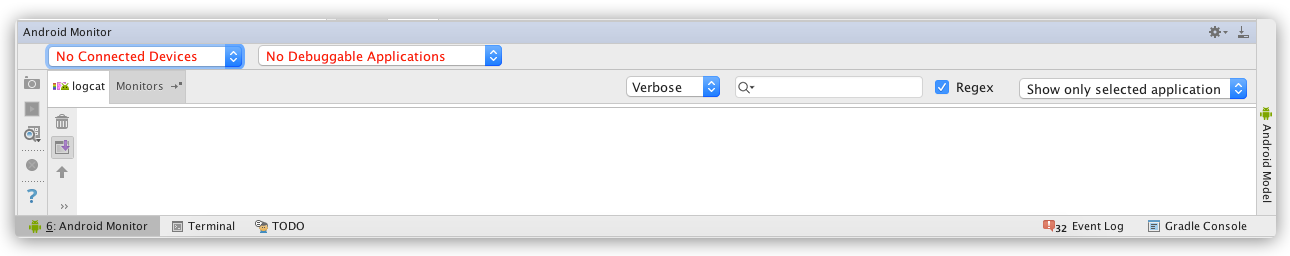


Figura : Android Monitor

1. **Paleta:** quando se tem selecionado um arquivo .xml, haverá a opção de acessar seu “modo *Design*”. O modo *Design* permite que você componha telas apenas clicando e arrastando. Os componentes disponíveis estão na Paleta de componentes, então basta clicar em algum deles e arrastá-lo para a Canvas.

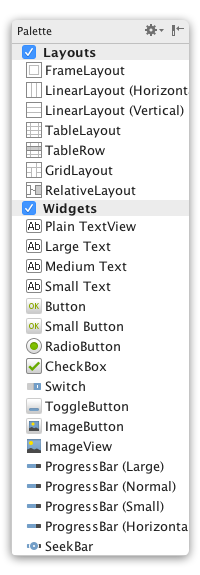


Figura : Paleta

1. **Canvas:** é o painel para sua obra de arte. Quando em modo *Design*, a seção Canvas aparece e você pode posicionar e redimensionar os componentes da paleta colocados aqui. Quando colocar os componentes no Canvas, além de ter uma previsão de como a tela realmente ficará, pode-se ver também a estrutura hierárquica que esses componentes estão assumindo por meio da ***Component Tree*** (árvore de componentes).

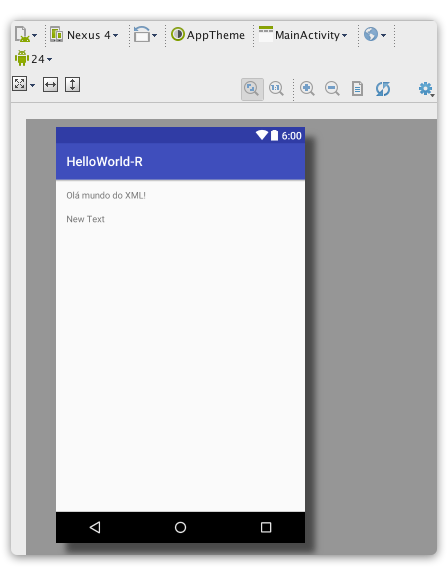


Figura : Canvas

1. **Árvore de componentes:** nesta seção é possível ver a organização hierárquica dos componentes no Canvas;
2. **Propriedades:** quando se clica em um destes componentes no Canvas, pode-se então alterar suas propriedades por meio da seção ***Properties*.** Basta clicar no elemento, localizar a propriedade na seção e alterá-la.

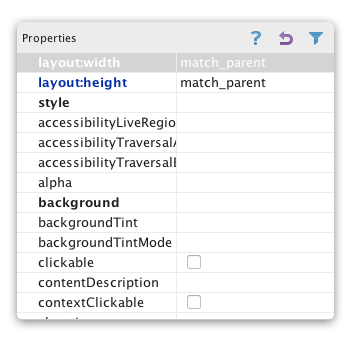


Figura : Propriedades

1. **Modos:** além de ter o modo *Design*, também há o modo texto como alternativa. Na verdade, pode-se pensar que o modo *Design* é tudo de bom, mas quase nunca satisfará suas necessidades de *layout*. A maioria dos problemas de *layout* é resolvida apenas no modo texto. O modo texto é uma versão em código do Canvas e esse código está na linguagem de marcação XML (você verá mais detalhes futuramente).

Tendo em vista as grandes seções do Android Studio (AS), agora você localizará onde estão as ferramentas básicas da IDE:



Figura : Comandos básicos

1. **Executar :** botão utilizado para executar o projeto;
2. **Depurar :** este botão executa o projeto em modo de depuração, no qual você pode definir pontos de parada para assistir o comportamento de variáveis durante essas paradas. Debug (depuração) será abordado na próxima aula;
3. **Preferências :** botão de acesso rápido à tela de preferências;
4. **ADV Manager :** abre a tela que gerencia os Emuladores, que serão vistos melhor nas aulas seguintes;
5. **SDK Manager :** abre a tela do SDK Manager.

Existem muitas outras ferramentas que o Android Studio dispõe, mas vale a pena mencioná-las somente no momento de seus respectivos usos, então, não se preocupe, pois você dominará o Android Studio até o final deste curso.

## Resumo

Finalmente foi dado início a sua jornada. Depois desta longa aula, você pôde escolher com mais sabedoria a API mínima que seu *app* suportará, baseando-se no número de *devices* que ele atingirá. Seu ambiente de desenvolvimento foi preparado instalando o Java e o Android Studio. Seu primeiro *app* foi criado e rodado no seu próprio celular, mas antes você teve que atualizar os SDKs. Por fim, ao final de tudo, foi feito um *tour* pela interface do Android Studio e visitadas as suas ferramentas essenciais.

## Exercícios

## TDP

# **Aula 2**

# Arquivos essenciais

Para que seu *app* funcione corretamente, seja distribuído e você consiga utilizar bibliotecas de terceiros é necessário configurar o seu projeto em diferentes pontos de vista.

Existe uma vasta quantidade de configurações que podem ser feitas em seu *app*, como temas, nome, nome do pacote, internacionalização de textos, permissões, assinatura de pacotes para distribuição, e para isso é preciso conhecer alguns arquivos essenciais.

## Manifesto do aplicativo

Todo aplicativo deve possuir um arquivo **AndroidManifest.xml** (precisamente com esse nome) e ele deve estar no diretório raiz do projeto. O arquivo de manifesto apresenta informações essenciais sobre o aplicativo em relação ao sistema Android. Essas informações se fazem necessárias ao sistema antes que o código do aplicativo seja executado para apresentar o *app* apropriadamente com suas devidas funcionalidades.

Entre outras coisas, o manifesto é útil para:

* nomear o pacote Java para o aplicativo. O nome desse pacote serve como identificador exclusivo para o aplicativo, ou seja, o manifesto faz uma cópia única do seu Java para que o aplicativo faça uso exclusivo dele mesmo. Esse recurso evita possíveis incompatibilidades com o pacote Java, caso o usuário do computador decida atualizá-lo;
* descrever os componentes do aplicativo, como atividades, serviços, receptores de transmissão e os provedores de conteúdo que compõem o aplicativo. Essas declarações permitem ao sistema Android saber quais são os componentes e em que condições eles podem ser iniciados;
* determinar que processos hospedarão componentes de aplicativo;
* declarar as permissões que o aplicativo deve ter para acessar partes protegidas da API e interagir com outros aplicativos;
* declarar também as permissões que outros devem ter para interagir com os componentes do aplicativo;
* listar as classes Instrumentation que fornecem geração de perfil e outras informações durante a execução do aplicativo. Essas declarações estão presentes no manifesto somente enquanto o aplicativo está em desenvolvimento e teste (elas são removidas antes da publicação do aplicativo);
* declarar o nível mínimo da API do Android que o aplicativo exige, como visto na inicialização do projeto. Se for preciso alterar esse recurso no meio do projeto, isso é permitido pelo manifesto;
* listar as bibliotecas às quais o aplicativo deve se vincular.

### Estrutura do arquivo de manifesto

A listagem abaixo ilustra a estrutura geral do arquivo de manifesto e cada elemento que ele pode conter. Na documentação do Android (<https://developer.android.com/guide/index.html>) você pode conferir os detalhes de cada elemento listado abaixo.

<?xml version="1.0" encoding="utf-8"?>  
<manifest>  
 <uses-permission/>  
 <permission/>  
 <permission-tree/>  
 <permission-group/>  
 <instrumentation/>  
 <uses-sdk/>  
 <uses-configuration/>  
 <uses-feature/>  
 <supports-screens/>  
 <compatible-screens/>  
 <supports-gl-texture/>  
  
 <application>  
 <activity>  
 <intent-filter>  
 <action/>  
 <category/>  
 <data/>  
 </intent-filter>  
 <meta-data/>  
 </activity>  
  
 <activity-alias>  
 <intent-filter> ... </intent-filter>  
 <meta-data/>  
 </activity-alias>  
  
 <service>  
 <intent-filter> ... </intent-filter>  
 <meta-data/>  
 </service>  
  
 <receiver>  
 <intent-filter> ... </intent-filter>  
 <meta-data/>  
 </receiver>  
  
 <provider>  
 <grant-uri-permission/>  
 <meta-data/>  
 <path-permission/>  
 </provider>  
 <uses-library/>  
 </application>  
</manifest>

<https://developer.android.com/guide/topics/manifest/manifest-element.html?hl=pt-br>

Veja que a listagem anterior se trata de um arquivo do tipo **XML** (*e****X****tensible* ***M****arckup* ***L****anguage*).<https://developer.android.com/guide/topics/manifest/uses-sdk-element.html?hl=pt-br>

#### O que é XML e por que é utilizado?

XML é uma linguagem de marcação. Antigamente muito utilizada para fazer *websites*, juntamente com a HTML (*Hiperlink Text Markup Language*), outra linguagem de marcação, eram utilizadas para construir a estrutura dos *sites*. Veja exemplo de um código XML:

1. <?xmlversion="1.0" encoding="ISO-8859-1"?>  
2. <receitanome="bolo">  
3. <ingredientenome="fermento"quantidade="7"unidade="gramas"/>  
4. <ingredientenome="farinha"quantidade="3"unidade="xícaras"/>  
5. ...  
6. </receita>

Um XML é composto de **elementos** e **atributos.** Os elementos possuem duas categorias: elementos de bloco e elementos simples. Os elementos de bloco devem possuir uma ***tag*** de abertura e uma de fechamento, enquanto os elementos simples possuem apenas uma *tag* que se fecha automaticamente. Cada elemento pode possuir diversos atributos, e exclusivamente os elementos de blocos podem possuir outros elementos dentro do bloco. Veja o exemplo:

1. Na linha 1 da listagem acima há apenas um cabeçalho que os arquivos XML devem ter;
2. Na linha 2, pode-se ver a *tag* de abertura do elemento de bloco **receita** (<receitanome="bolo">)**.** Na linha 6 você encontra a *tag* de fechamento (</receita>);
3. Mais abaixo, na linha 3, estão os elementos simples **ingrediente.** Esses elementos compõem o elemento receita e como não tem nenhum outro elemento dentro do ingrediente, ele possui um fechamento automático (<... />);
4. Repare que em todos os elementos existem atributos. Esses atributos definem a característica dos elementos. Por exemplo, ao ler o que este código está querendo dizer, tem-se: “A **receita** de **bolo** possui dois ingredientes: o **fermento** e a **farinha.** A quantidade de fermento deve ser de **7 gramas** e a de farinha de **3 xícaras”.**

Além do Android Manifest, todas as telas de um *app* Android são escritas na linguagem XML, então, no decorrer do curso, você aprenderá com mais profundidade essa linguagem de marcação.

Agora que você entendeu um pouco do XML, é hora de ver algumas convenções para o arquivo manifesto.

### Convenções

Abaixo algumas convenções e regras que se aplicam geralmente a todos os elementos e atributos no manifesto:

**Obs.:** Podem aparecer alguns termos que você não conhece ainda, mas no decorrer do curso eles ficarão claros. Esta aula é para que você a tome como referência na hora de criar seu aplicativo. Quando houver dúvidas sobre o manifesto, basta voltar e revisar esta aula.

**Elementos:**

* Somente os elementos <manifest> e <application>são obrigatórios. Os demais elementos podem ocorrer diversas vezes ou nunca.
* Elementos de mesmo nível geralmente não são ordenados e podem ser combinados entre si em qualquer sequência. O elemento <activity-alias> é uma exceção a essa regra: ele deve seguir o <activity> para o qual é o “apelido”. Isto quer dizer que o elemento <activity-alias>, que a grosso modo define um “apelido” a uma Activity deve estar imediatamente após a Activity no qual ele irá apelidar. O aprofundamento destes termos mais específicos você pode encontrar na documentação oficial do Android, mas não se preocupe, não iremos utilizá-los no nosso curso.

**Atributos:**

* Formalmente, todos os atributos são opcionais. No entanto, alguns deles devem ser especificados para um elemento e assim cumprir com sua finalidade. Use a documentação como guia. Atributos opcionais geralmente possuem um valor padrão intrínseco.
* Exceto por alguns atributos do elemento raiz, todos os nomes de atributo têm um prefixo android:. Exemplo: android:alwaysRetainTaskState. Como esse prefixo é regra, geralmente ao se consultar a documentação oficial, ele é omitido, então não se assuste. O Android Studio tem uma maneira especial de tratá-los também, portanto você não precisa escrevê-lo, pois a IDE reconhecerá o atributo e aplicará o prefixo para você.

**Vários valores:**

* Se houver a necessidade de especificar mais de um valor para um atributo de um mesmo elemento, basta repeti-lo em vez de listar vários valores dentro de um único elemento. Por exemplo, um filtro de intenção pode listar algumas ações:

<intent-filter ... >  
<actionandroid:name="android.intent.action.EDIT"/>  
<actionandroid:name="android.intent.action.INSERT"/>  
<actionandroid:name="android.intent.action.DELETE"/>  
 ...  
</intent-filter>

**Valores de recurso:**

* Alguns atributos têm valores que podem ser exibidos aos usuários, por exemplo, uma *string* ou um ícone de uma atividade. Os valores desses atributos devem ser localizados e, portanto, definidos com base em um recurso ou tema. Os valores de recurso são expressos no formato a seguir:

@[<i>pacote</i>:]<i>tipo</i>:<i>nome</i>

O nome do **pacote** pode ser omitido se o recurso estiver no mesmo pacote que o aplicativo. O item **tipo** é um tipo de recurso (como uma "*string*" ou "*drawable*" (desenhável)) e o item **nome** é o nome que identifica o recurso específico. Por exemplo:

<activity android:icon="@drawable/smallPic" ... >

* Os valores de um tema são expressos de forma semelhante, mas com um '?' em vez de '@':

?[<i>pacote</i>:]<i>tipo</i>:<i>nome</i>

**Valores de *string*:**

* Quando o valor de um atributo é uma *string*, deve-se usar duas barras invertidas ('\\') para caracteres de escape, por exemplo, '\\n' para uma nova linha ou '\\uxxxx' para um caractere Unicode.

### Características do arquivo

Até agora falou-se sobre a estrutura do arquivo manifesto. Você leu termos como “*intent*” e “ícones” e agora chegou a hora de explicá-los. As seções a seguir descrevem como alguns recursos do Android refletem-se no arquivo de manifesto.

#### Intenções

Os componentes fundamentais de um aplicativo (suas **atividades, serviços e receptores de transmissão**) são ativados por intenções. Intenções são pacotes de informações (objetos da classe Intent) que descrevem uma ação desejada, inclusive os dados usados em ações, a categoria do componente que deve executar a ação e outras instruções pertinentes. O Android localiza um componente adequado para responder à intenção, inicia uma nova instância do componente, se necessário, e passa-o ao objeto da intenção.

O Intent é um objeto de mensagem que pode ser usado para solicitar uma ação de outro componente do aplicativo. A definição pode parecer complicada, mas um exemplo simples é um botão em uma tela que abre outra tela. Nesse exemplo, o botão utiliza um Intent para solicitar que a outra tela se abra. Embora as intenções facilitem a comunicação entre componentes de diversos modos, há três casos de uso fundamentais:

* **Para iniciar uma atividade:** a Activity representa uma única tela em um aplicativo e será usada nas aulas seguintes. É possível iniciar uma nova instância de uma Activity passando uma Intent, a startActivity(). A Intent descreve a atividade a iniciar e carrega todos os dados necessários. Se você deseja receber um resultado da atividade quando ela finalizar, chame o método startActivityForResult(), isso ajudará a saber se a atividade designada à intenção foi bem sucedida ou não. Nesse caso, sua atividade receberá o resultado como um outro objeto Intent separado no retorno de chamada de onActivityResult() da atividade. Nas aulas seguintes, você aprenderá o ciclo de vida de uma Activity;
* **Para iniciar um serviço:** o Service é um componente que realiza operações em segundo plano, sem o *app* estar necessariamente aberto. Essas operações podem ser, por exemplo, baixar um arquivo em PDF, monitoramento do sensor de movimentos para obter os passos do usuário, rotinas de notificações e comunicação com dispositivos via *Bluetooth*®;
* **Para fornecer uma transmissão:** transmissão é uma mensagem que qualquer aplicativo pode receber. O sistema fornece diversas transmissões para eventos do sistema, como quando o sistema inicializa ou o dispositivo inicia o carregamento.

#### Permissões

As permissões são restrições que limitam o acesso a parte do código ou aos dados de um dispositivo. A limitação é imposta para proteger dados essenciais que podem sofrer mau uso e distorções ou prejudicar a experiência do usuário.

Cada permissão é identificada por uma etiqueta exclusiva. Geralmente a etiqueta indica a ação que foi restringida. A seguir há alguns exemplos de permissões definidas pelo Android:

android.permission.CALL\_EMERGENCY\_NUMBERS

android.permission.READ\_OWNER\_DATA

android.permission.SET\_WALLPAPER

android.permission.DEVICE\_POWER

Um recurso pode ser protegido por, no máximo, uma permissão.

Se um aplicativo precisar de acesso a um recurso protegido por uma permissão, ele deve declarar que precisa da permissão com um elemento <uses-permission> no manifesto. Assim, quando o aplicativo é instalado no dispositivo, o instalador determina se concederá ou não a permissão solicitada, marcando as autoridades que assinaram os certificados do aplicativo e, em alguns casos, perguntando ao usuário. Se a permissão for concedida, o aplicativo será capaz de usar os recursos protegidos. Caso contrário, sua tentativa de acessar esses recursos simplesmente falhará sem nenhuma notificação ao usuário.

Um aplicativo também pode proteger seus componentes (atividades, serviços, receptores de transmissão e provedores de conteúdo) com permissões. Ele pode empregar qualquer uma das permissões definidas pelo Android ou declaradas por outros aplicativos, ou, então, pode definir as suas próprias. As novas permissões são declaradas com o elemento <permission>. Uma atividade pode ser protegida da seguinte forma:

<manifest ... >  
<permissionandroid:name="com.example.project.DEBIT\_ACCT" ... />  
<uses-permissionandroid:name="com.example.project.DEBIT\_ACCT"/>  
 ...  
<application ...>  
<activityandroid:name="com.example.project.FreneticActivity"  
android:permission="com.example.project.DEBIT\_ACCT"  
 ... >  
 ...  
</activity>  
</application>  
</manifest>

Observe que, nesse exemplo, a permissão DEBIT\_ACCT, além de declarada com o elemento <permission>, tem seu uso solicitado com o elemento <uses-permission>. Ela deve ser solicitada para que outros componentes do aplicativo iniciem a atividade protegida, mesmo que a proteção seja imposta pelo próprio aplicativo.

Se, no mesmo exemplo, o atributo permission fosse definido como uma permissão declarada em outro lugar (como android.permission.CALL\_EMERGENCY\_NUMBERS), não seria necessário declará-la novamente com um elemento <permission>. No entanto, ainda seria necessário solicitar seu uso com <uses-permission>.

O elemento <permission-tree> declara um espaço de nome de um grupo de permissões que será definido no códigoe <permission-group> define uma etiqueta de um conjunto de permissões (ambos (<permission-tree> e <permission-group>) são declarados no manifesto com elementos <permission> e as declaradas em outro lugar). Ele afeta somente a forma com que as permissões estão agrupadas quando apresentadas ao usuário. O elemento <permission-group> não especifica que permissões pertencem ao grupo; ele só dá um nome ao grupo. Para incluir uma permissão no grupo, designa-se o nome do grupo ao atributo permissionGroup do elemento <permission>.

#### Bibliotecas

Todo aplicativo está vinculado à biblioteca Android padrão, que contém os pacotes básicos para programar aplicativos com classes comuns, tais como Activity, Service, Intent, View, Button, Application, ContentProvider etc.

No entanto, às vezes são necessárias algumas bibliotecas externas. Essas bibliotecas devem receber solicitação explícita para serem vinculadas ao *app*. O manifesto deve conter um elemento <uses-library> separado para nomear cada uma das bibliotecas e relacioná-las ao projeto.

## A classe R

Um padrão de projeto aplicado ao Android é a exteriorização dos recursos do aplicativo. Entende-se como recurso (*resources*) as imagens, *strings*, cores, telas em XML e quase tudo que não seja o código Java. Essa exteriorização é aplicada para que seja possível manter os recursos independentemente do código das classes.

Deve-se também fornecer recursos alternativos para configurações específicas do dispositivo, agrupando-os em diretórios de recursos especialmente nomeados. Em tempo de execução, o Android usa o recurso adequado com base na configuração atual. Por exemplo, você pode querer fornecer um *layout* diferente dependendo do tamanho da tela ou *strings* diferentes dependendo do idioma configurado no dispositivo.

Ao exteriorizar os recursos do aplicativo, é possível acessá-los usando IDs de recurso que são gerados na classe R do projeto.

Quando o aplicativo é compilado, a **aapt** (*Android Asset Packaging Tool*) gera a classe R, que contém os IDs de cada recurso no diretório res/. Para cada tipo de recurso, há uma subclasse R (por exemplo, R.drawable é uma subclasse para todos os recursos desenháveis, como imagens) e, para cada recurso daquele tipo, há um número inteiro estático (por exemplo, R.drawable.icon). Esse número inteiro é o ID do recurso que pode ser usado para recuperá-lo.

Apesar da classe R ser o local onde os IDs de recursos são especificados, não deve nunca ser necessário verificá-la para descobrir um ID de recurso. Ele é sempre composto de:

* **tipo de recurso**: cada recurso é agrupado em um "tipo", como string, drawable e layout.
* **nome do recurso**: é o nome do arquivo, excluindo a extensão, ou o valor no atributo android:name do XML, se o recurso for um valor simples (como uma *string*).

Há duas formas de acessar um recurso:

* **No código**: usando um número inteiro estático de uma subclasse de sua classe R, como:

R.string.hello

string é o tipo de recurso e hello é o nome do recurso. Há muitas APIs do Android que podem acessar os seus recursos quando você fornece um ID de recurso nesse formato.

* **No XML**: usando uma sintaxe XML especial que também corresponde ao ID de recurso definido em sua classe R, como:

@string/hello

string é o tipo de recurso e hello é o nome do recurso. Você pode usar essa sintaxe em um recurso XML em qualquer lugar no qual um valor é esperado e que seja fornecido em um recurso.

Agora você criará um novo Hello World.

### Hello World 2.0

Para ilustrar o funcionamento da classe R, em XML e no código, você vai criar um recurso para o texto "Hello World!".

Abra o Android Studio e crie um novo projeto. Dessa vez o nome dele deverá ser **HelloWorld-R**. Relembre o passo a passo para criação de projetos no tópico 1.4.1 desta unidade. Ao criar, você terá esta tela:

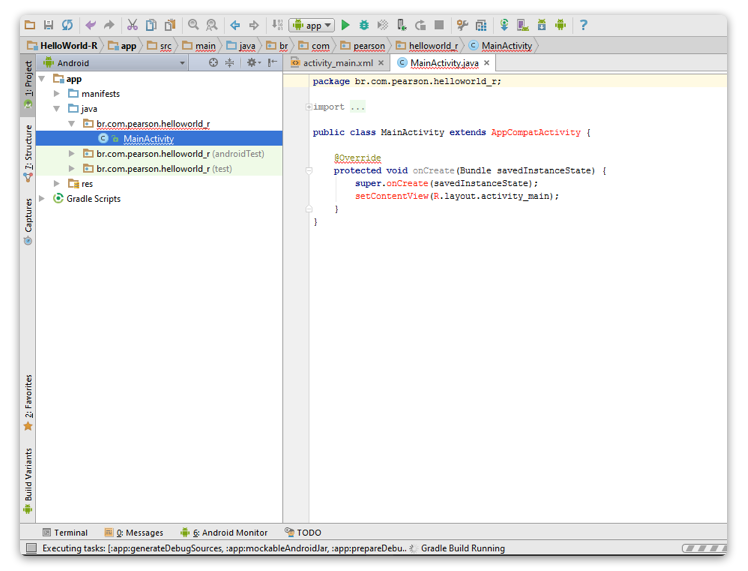


Figura : Novo projeto sendo construído após ser criado

Agora você criará um recurso para o texto "Hello World!". Siga os passos:

1. Com o projeto aberto, localize o arquivo **strings.xml** na seção **Estrutura do Projeto**. O arquivo está contido no diretório **res > values.** Veja a imagem a seguir:

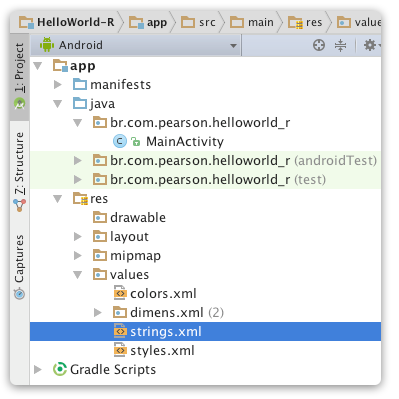


Figura : Localização do arquivo **strings.xml**

1. Quando você abrir o arquivo **strings.xml**, terá um arquivo como este:

<resources>  
 <string name="app\_name">HelloWorld-R</string>  
</resources>

Veja que ele é um *resource* e é do tipo XML. Adicione mais um elemento <string> dentro do elemento <resources>:

<resources>  
<string name="app\_name">HelloWorld-R</string>

<string name="hello\_world">Olá mundo do XML!</string>  
</resources>

1. Agora que você tem uma nova *string* adicionada aos recursos, você vai referenciá-la no TextView da sua **activity\_main.xml.** Na seção de **Estrutura do Projeto**, localize o arquivo **activity\_main.xml** em **res > layout.**

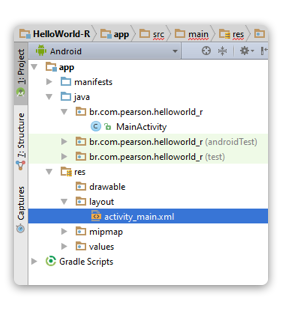


Figura : Localização do arquivo **activity\_main.xml**

1. Com a activity\_main.xml aberta, entre no **modo TextA**:

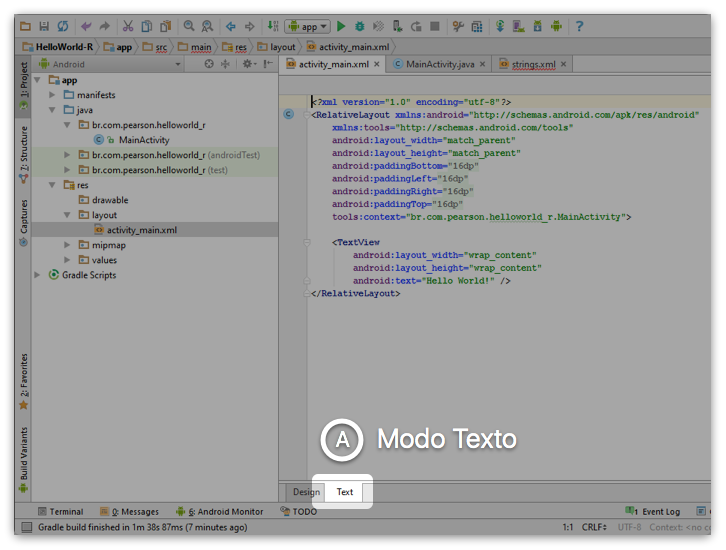


Figura : Entrando no modo texto do arquivo XML

1. Altere a propriedade text para corresponder com o código abaixo:

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:tools="http://schemas.android.com/tools"  
android:layout\_width="match\_parent"  
android:layout\_height="match\_parent"  
android:paddingBottom="@dimen/activity\_vertical\_margin"  
android:paddingLeft="@dimen/activity\_horizontal\_margin"  
android:paddingRight="@dimen/activity\_horizontal\_margin"  
android:paddingTop="@dimen/activity\_vertical\_margin"  
tools:context="br.com.pearson.helloworld\_r.MainActivity">  
  
<TextView  
android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content"  
android:text="@string/hello\_world"/>  
  
</RelativeLayout>

Agora você pode executar o código no seu celular e ver o resultado.

O que foi feito no exemplo anterior foi adicionar uma *string* em um arquivo de recurso e referenciá-la na sua *activity,* lembrando que *activity* é uma tela. Quando o *resource* é referenciado no TextView, ele exibe o valor referente ao recurso.

Essa exteriorização de *strings* é válida principalmente quando você pretende internacionalizar seu *app* com outros idiomas. A estratégia seria criar um arquivo **strings.xml** para cada idioma em que seria dado suporte no *app*. O compilador cuidaria do resto: escolher qual arquivo de *strings* usar para cada região.

Agora você verá como alterar um texto que está na XML da tela via código por meio da classe R.

1. Utilizando o mesmo projeto, abra o arquivo **activity\_main.xml** em **modo *Design*** e adicione um novo TextView. Para isso, procure por um PlainTextView na **PaletaA** e arraste-o para o **CanvasB.**

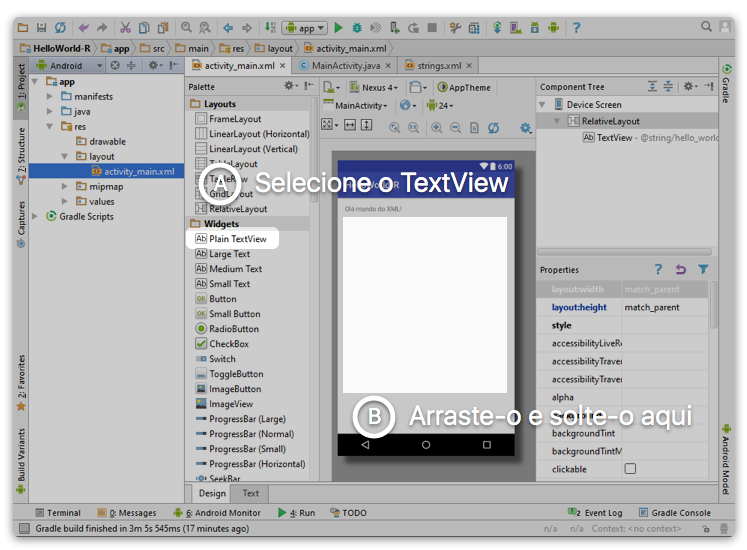


Figura : Inserindo um TextView no Canvas

1. Selecione o TextView**C** e, na seção de Propriedades, altere o atributo ID**D** do novo TextView. Coloque o valor “teste”.

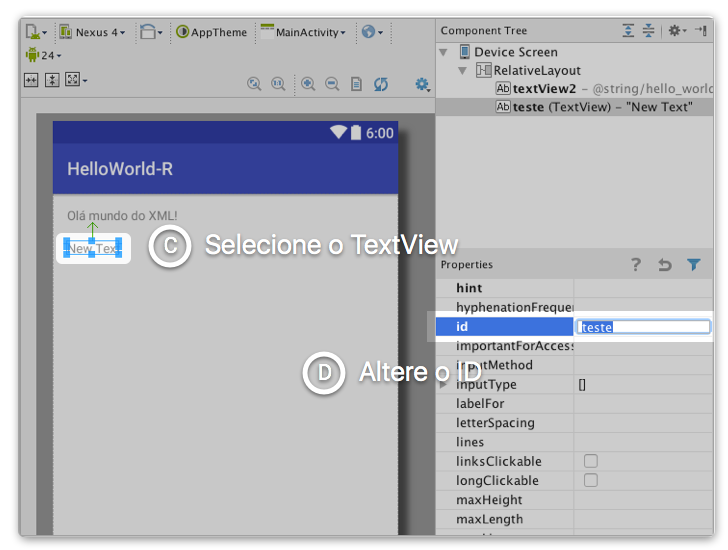


Figura : Alterando a propriedade ID pelo painel de propriedades

1. Feito isso, você deve encontrar o arquivo **MainActivity.java**. Você vai referenciar seu novo TextView neste arquivo.

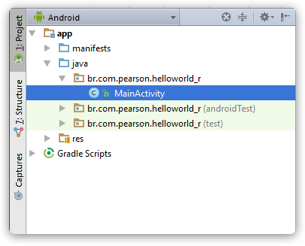


Figura : Localização do arquivo **MainActivity.java**

1. Com o arquivo **MainActivity.java** aberto, localize o método onCreate() e insira a referenciação do novo TextView ao final do método, correspondendo à listagem a seguir:

@Override  
protectedvoidonCreate(BundlesavedInstanceState) {  
super.onCreate(savedInstanceState);  
setContentView(R.layout.activity\_main);  
  
// Obtém o TextView com id "teste" atribuído  
TextViewtextView = (TextView)findViewById(R.id.teste);  
  
// Altera o texto do TextView obtido  
textView.setText("Olá mundo do código!");  
}

**OBS.**: Lembrando que agora você está programando em Java, veja que você colocou a chamada de dois métodos ao final do método onCreate(), que foi deixado fora de foco para destacar as mudanças. Primeiramente você obteve o TextView usando o ID que foi definido. Sempre que for definido um ID, ele automaticamente fica salvo na classe R, então é por ela que você resgata seus IDs. O segundo método, setText(), altera o texto contido no TextView obtido anteriormente.

Execute a aplicação novamente e veja o resultado.

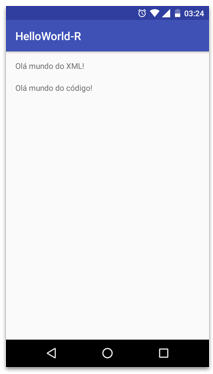


Figura : Resultado do projeto HelloWorld-R

Nesta segunda parte prática, você fez algo que utilizará muito: relacionar as *views* (código XML) aos *controllers* (código Java). No curso de POO, você aprendeu a arquitetura Model-View-Controller, e é exatamente dela que se está tratando agora. Na unidade 3, você poderá revisar este padrão, agora aplicando-o ao mundo do Android.

## O Gradle

O **Gradle,** resumidamente, é um sistema de compilação, ou mais comumente chamado de *build system.*

### O que é um *build system*?

Antes do Android Studio, você usava a IDE Eclipse para criar os *apps* em Android, e, nesta época, precisava saber como compilar seu projeto.

Você pode fazer isso via linha de comando, mas precisaria aprender o funcionamento de cada ferramenta contida no SDK. O Eclipse então lhe salvou de todos estes recursos em baixo nível –baixo, porém importante ‒ concedendo seu próprio sistema de compilação.

Os que já programaram na plataforma Eclipse, já se perguntaram porque a pasta **res** está no mesmo diretório que a pasta **src**?

É aí que o sistema de compilação entra em cena. Ele pega automaticamente todos os arquivos (.java e .xml), aplica o tratamento apropriado e agrupa tudo em um arquivo compactado, o amado APK.

Para quem não está habituado, o APK é o pacote com a aplicação pronta para ser instalada. É uma espécie de instalador, similar ao **.exe** do Windows, o **.dmg** do macOS e o **.ipa** do iOS.

O *build system* usa algumas convenções, por exemplo, um diretório específico para os códigos-fonte (pasta **src**) e outro específico para os recursos (pasta **res**).

Agora, tendo em vista que o *build system* automatiza todas essas tarefas (e muitas mais), pode-se dizer que ele é um *script*, e você pode escrever seu próprio *build system* usando algum *shell script* em Linux e MacOS ou *Batch* usando Windows.

### O que é o Gradle

O Gradle é o *build system* do AS, que reúne o melhor do *build system* usado no Eclipse com algumas funcionalidades extras. Ele possui melhorias baseadas em suas próprias deficiências. É baseado na JMV (*Java Virtual Machine*), que significa que você pode escrever seus próprios *scripts* de automatização de *builds* usando Java, e isso é o que o Android Studio faz.

O Gradle também fornece um mecanismo de gerência de dependências simplificado, totalmente com base em arquivos com formato Groovy e JSON, e controle e acesso direto da central de dependências do MVN. Ele é concorrente do Maven (utilizado em aplicações Java SE) e é similar ao Cocoa Pods e ao Swift Package Manager do iOS, ou o NPM do NodeJS e PIP do Python. Com o gerenciador de dependências você pode aplicar soluções de terceiros a seus aplicativos.

## Resumo

Nesta aula você conheceu alguns arquivos essenciais de uma aplicação Android. O arquivo manifesto, que é o coração do aplicativo, é responsável por fazer a comunicação do *app* com o sistema. Ele é um arquivo de configuração escrito em XML usado para configurar seu aplicativo de maneira geral. O arquivo R é o responsável por interligar os ***R****esources* com os *controllers*, ou seja, o código em Java. Ele armazena os IDs dos componentes criados nos *layouts* e guarda também um identificador para *strings*, imagens, cores e etc. Por fim, descobriu que o Gradle é um facilitador na geração das suas *builds* e que ele será usado futuramente no seu curso como gerenciador de dependências.

## Exercícios

## TDP

# 

# **Aula 3**

# Ferramentas de desenvolvimento

## Emuladores

O Android Emulator simula um dispositivo e o exibe no seu computador de desenvolvimento. Com ele, você pode criar protótipos, desenvolver e testar aplicativos do Android sem usar um dispositivo físico. O emulador é compatível com celulares e *tablets* Android, com o Android Wear e com dispositivos Android TV. Ele vem com tipos de dispositivo predefinidos para que você comece rapidamente e, além disso, também possa criar suas próprias definições de dispositivo e aparências para o emulador.

O Android Emulator é uma ferramenta rápida, eficaz e repleta de recursos. Ele pode transferir informações com mais rapidez do que em um dispositivo físico (*device*) conectado, o que agiliza o processo de desenvolvimento. O recurso de vários núcleos permite que o emulador aproveite os processadores de vários núcleos do seu computador de desenvolvimento para melhorar ainda mais o desempenho do emulador.

Como dito na primeira aula, está na casa do bilhão a quantidade de dispositivos que rodam Android, inviabilizando a garantia de que seus *apps* funcionarão na maioria deles. Para isso, foram utilizados os emuladores, pois pode-se simular uma amostra significativa de tipos de *devices* para garantir um mínimo de qualidade aos *apps*.

### Instalando um emulador

Você pode iniciar um aplicativo no emulador executando seu projeto ou arrastar um arquivo de APK para o emulador para instalá-lo. Assim como ocorre com um dispositivo físico, depois que você instala um aplicativo em um emulador, ele permanece lá até ser desinstalado ou substituído. Se necessário, você pode testar como vários aplicativos, como o seu próprio ou os aplicativos do sistema, interagem entre si.

A instalação de emuladores é muito simples. Veja:

1. Desconecte seu dispositivo físico do computador, abra qualquer projeto do Android Studio e clique no botão **ADV Manager** **.** Se você não tem nenhum emulador instalado, a seguinte tela se abrirá:

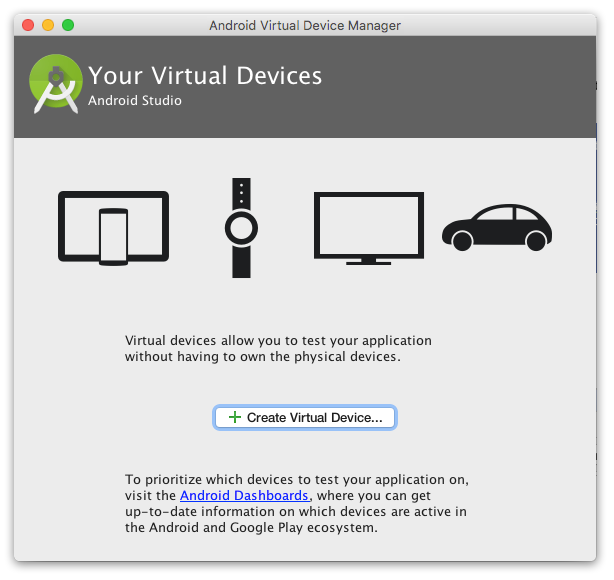


Figura : Tela de criação do primeiro emulador

1. Na nova tela, clique em **Create Virtual Device**;
2. A tela de configurações do *device* virtual aparecerá. Aqui você pode escolher diversos tipos de *devices* predefinidos. Além disso, é possível importar um virtual *device*. Por agora, escolha o **Nexus 5x.** Selecione o *device* e clique em **Next.**

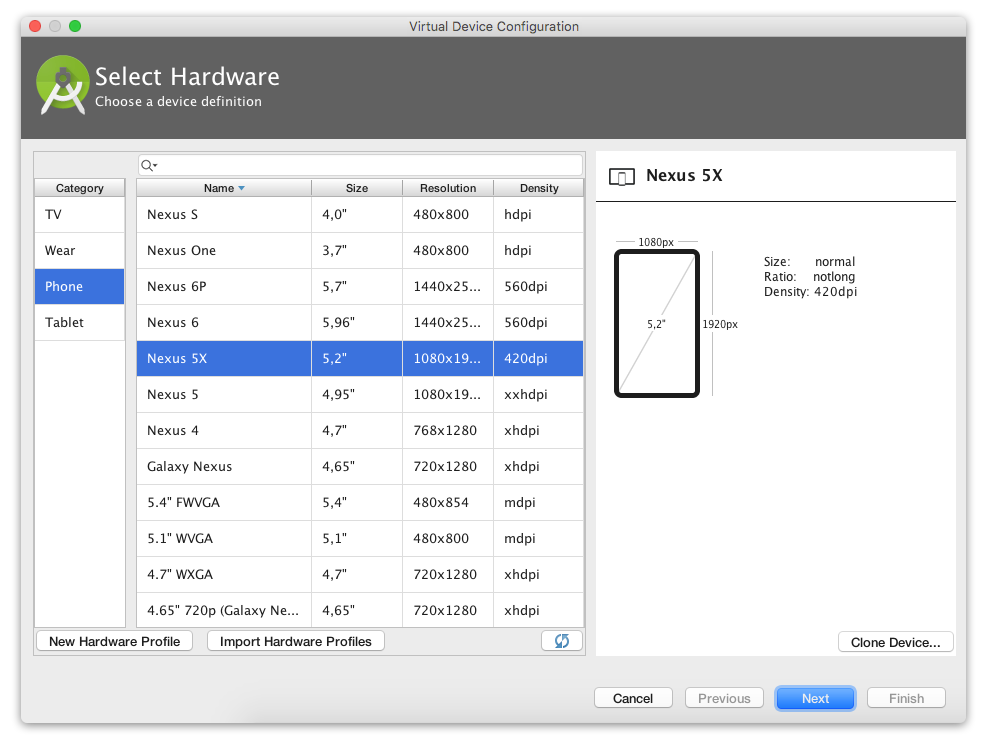


Figura : Escolhendo um device específico

1. Agora você deve escolher a imagem do sistema, ou seja, qual versão do Android aquele emulador instalará. Você pode escolher qualquer uma. Nesse caso, foi escolhida a ***Marshmallow***,que já vem com o Android Studio. Selecione a versão e clique em **Next.**

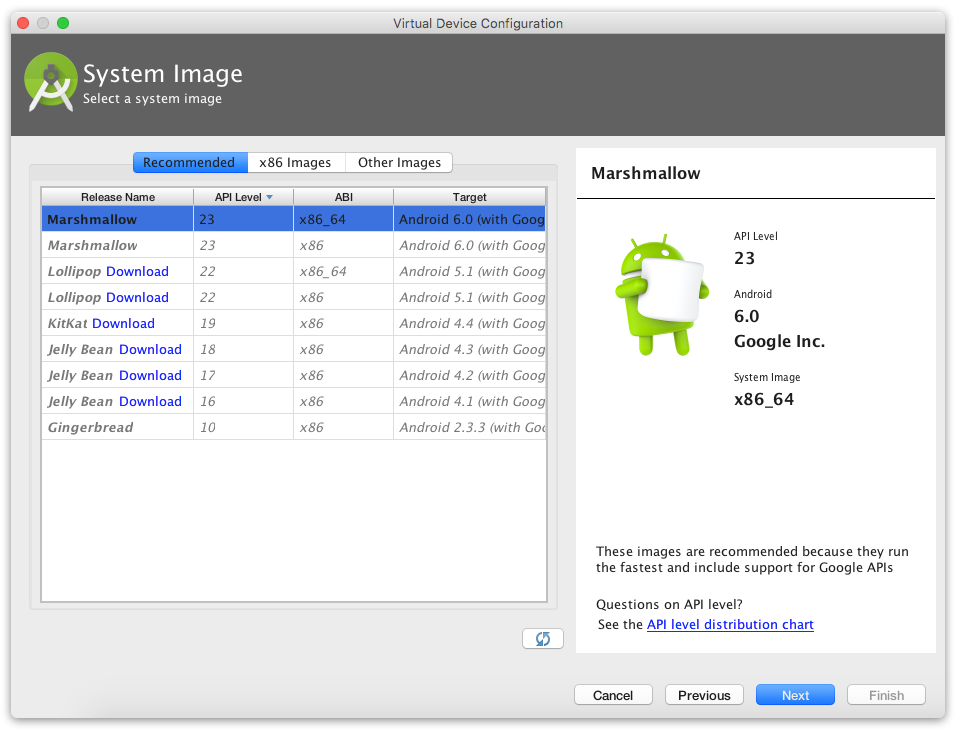


Figura : Escolhendo a versão do Android para o emulador

**OBS.:** Se não houver imagem disponível já baixada, faça *download* de alguma e prossiga.

1. Nessa tela você pode fazer algumas configurações no novo emulador. Explore-a. Nesse exemplo nada foi mudado. Quando terminar, clique em **Finish;**
2. Para testar clique em **Run** . Espere o ADV carregar seu emulador e depois selecione-o. Clique em **OK** para executar.

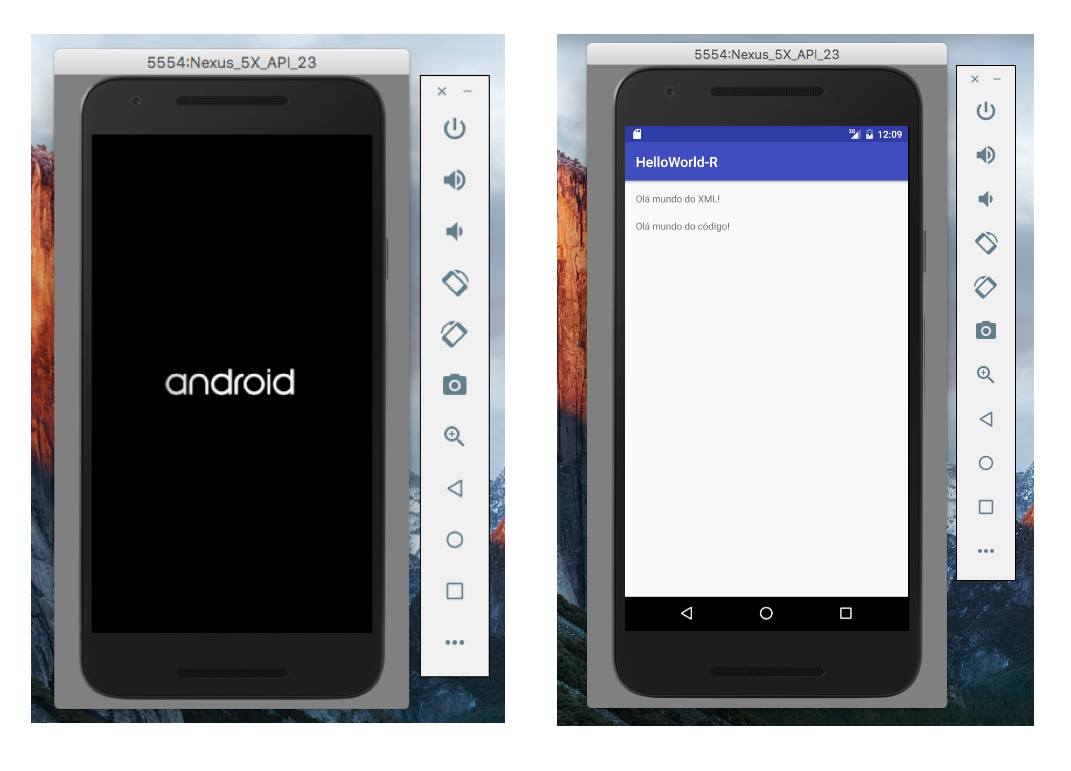


Figura : Aplicativo rodando no emulador criado

Agora você tem um emulador instalado. Sempre que for preciso testar o *app* em um dispositivo muito específico, provavelmente você conseguirá emulá-lo, então essa ferramenta vai lhe ajudar.

### Genymotion

Infelizmente, o Android Emulator nativo do Android Studio não tem um desempenho tão bom em computadores menos potentes, já que ele é uma máquina virtual otimizada para depuração e recursos específicos de desenvolvimento em Android.

Se o seu Android Emulator está lento e com problemas de travamento constantes, existem outros simuladores alternativos externos ao Android Studio.

Quando se utilizava a IDE Eclipse, existia um emulador bem mais pesado do que o que existe hoje no Android Studio, e poucos desenvolvedores o usavam devido a esse baixo desempenho. Como alternativa, surgiram os emuladores da Genymotion.

A Genymotion não só disponibiliza um emulador de Android, mas otimiza o seu desempenho utilizando tecnologia de virtualização de máquinas, ou seja, o Genymotion cria uma máquina virtual em seu computador que roda o Android com muito mais vigor do que se estivesse em um emulador.

O Android Emulator atende bem em muitos casos, mas não é raro presenciar pessoas com queda de desempenho e que optaram por usar o Genymotion para solucionar esse problema.

Tendo isso em vista, instale o Genymotion como emulador alternativo ao padrão.

1. Para instalar o Genymotion, você primeiro deve se cadastrar no site <https://www.genymotion.com/account/create/>;
2. Após ter feito o cadastro, faça o *download* do instalador do Genymotion para seu sistema operacional acessando o link <https://www.genymotion.com/download/>;
3. Com o instalador baixado, faça a instalação padrão, de acordo com seu sistema operacional;
4. Abra o Genymotion, clique no botão **Add** e faça o *login* com o usuário e senha que você criou nos passos anteriores.

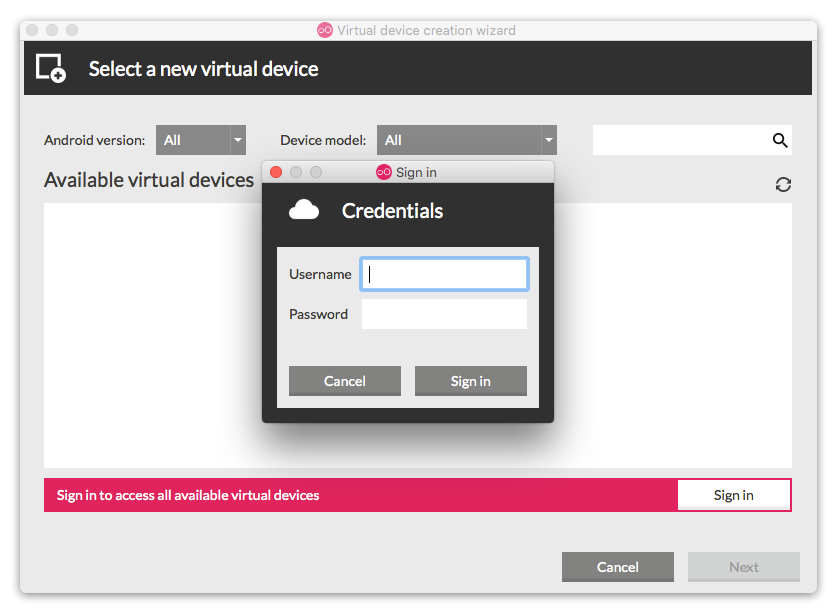


Figura : Autenticação no Genymotion

1. Feito isso, começa a fase da configuração da máquina, parecida com o Android Emulator. Neste exemplo, foi escolhida a mesma configuração para o exemplo do Android Emulator - Nexus 5X com API 23. Veja a imagem a seguir:

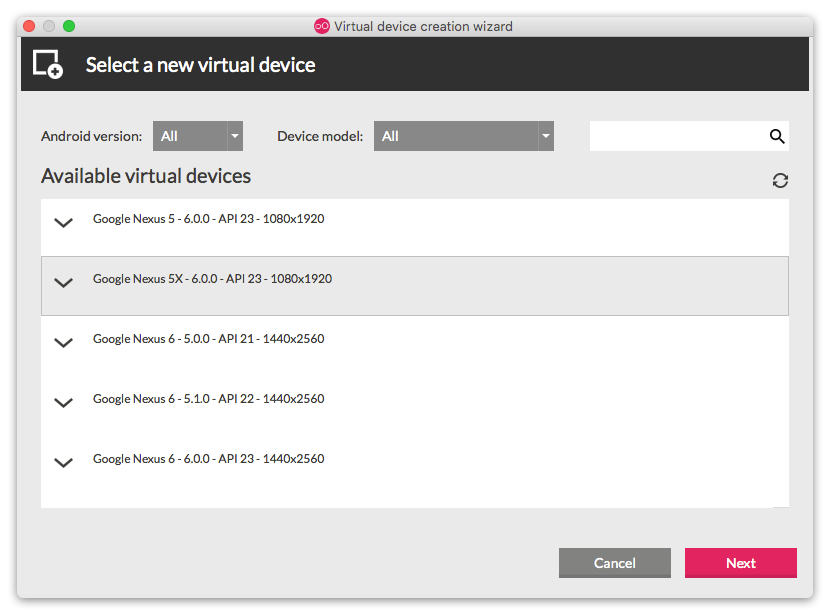


Figura : Escolhendo um tipo de device para o emulador

Para concluir o passo, clique em **Next**.

1. Agora é preciso escolher um nome para sua nova máquina. Como ela será usada no Android Studio, é bom diferenciá-la colocando o seguinte nome: **Google Nexus 5X - 6.0.0 - API 23 - Genymotion**.

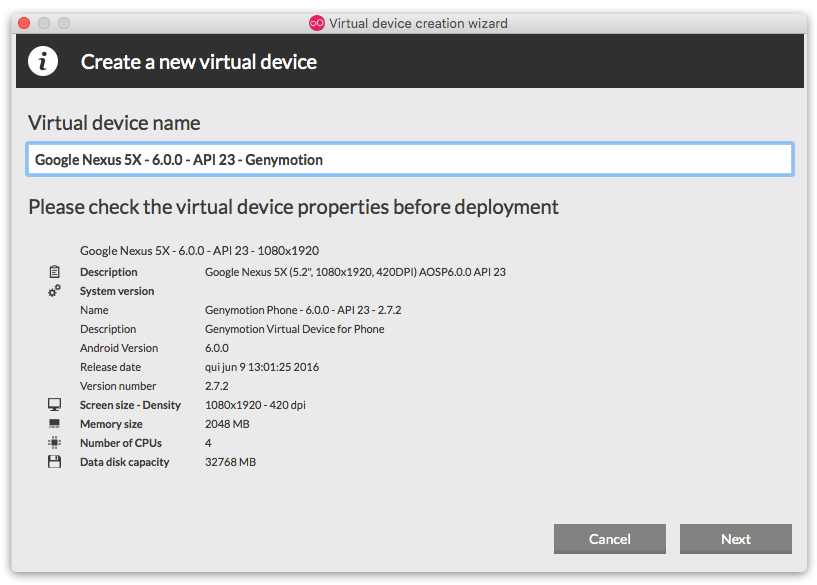


Figura : Nomeando e finalizando a criação de um novo emulador

1. Clique em **Next** e conclua a criação do emulador clicando em **Finish**;
2. Para testar o emulador, selecione a máquina criada e clique em **Start .**

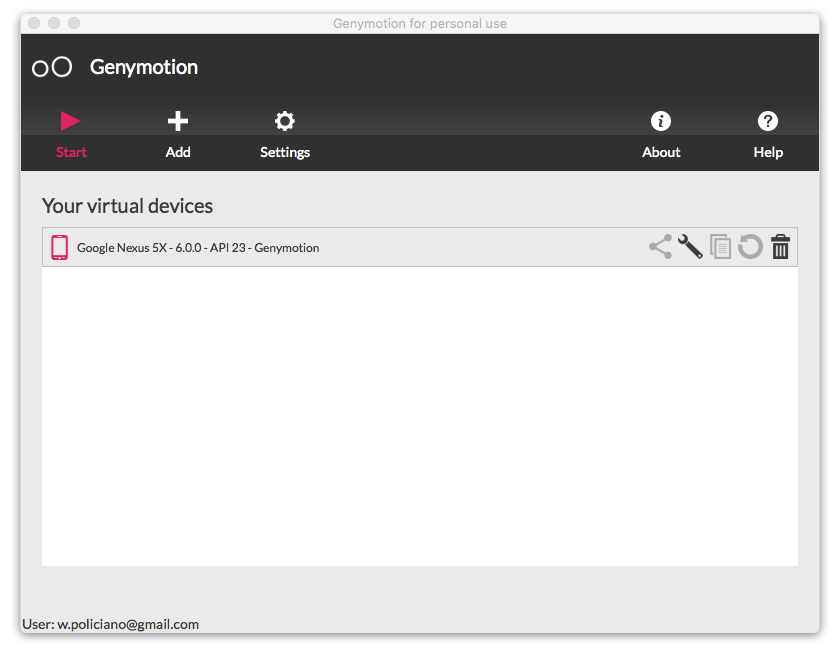


Figura : Abrindo o emulador criado

Feito! Você tem sua máquina virtual do **Genymotion** criada. Veja que ela é um pouco mais simples que a do Android Emulator e por isso é mais leve.

Para usá-la no Android Studio, siga os seguintes passos:

1. Certifique-se de que todos os passos anteriores foram feitos e abra qualquer projeto do Android Studio;
2. Clique no botão **Run**  e veja que o emulador criado pelo **Genymotion** está listado:

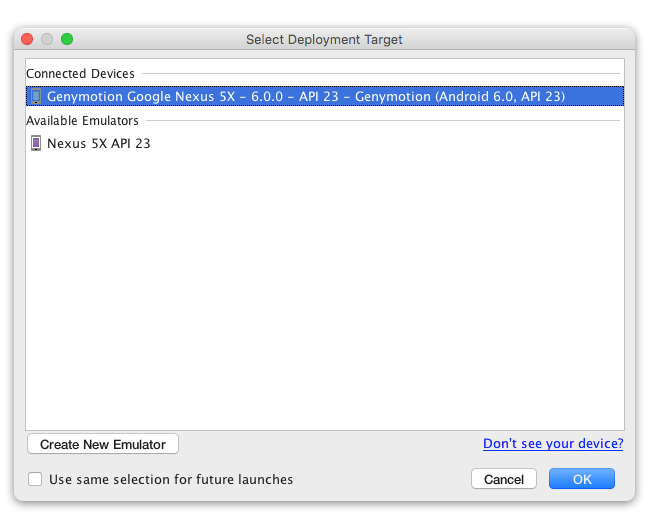


Figura : Selecionando o emulador do Genymotion no ADB do Android Studio

1. Selecione-o, clique em **OK** e aguarde. O emulador do **Genymotion** abrirá com a sua aplicação instalada:

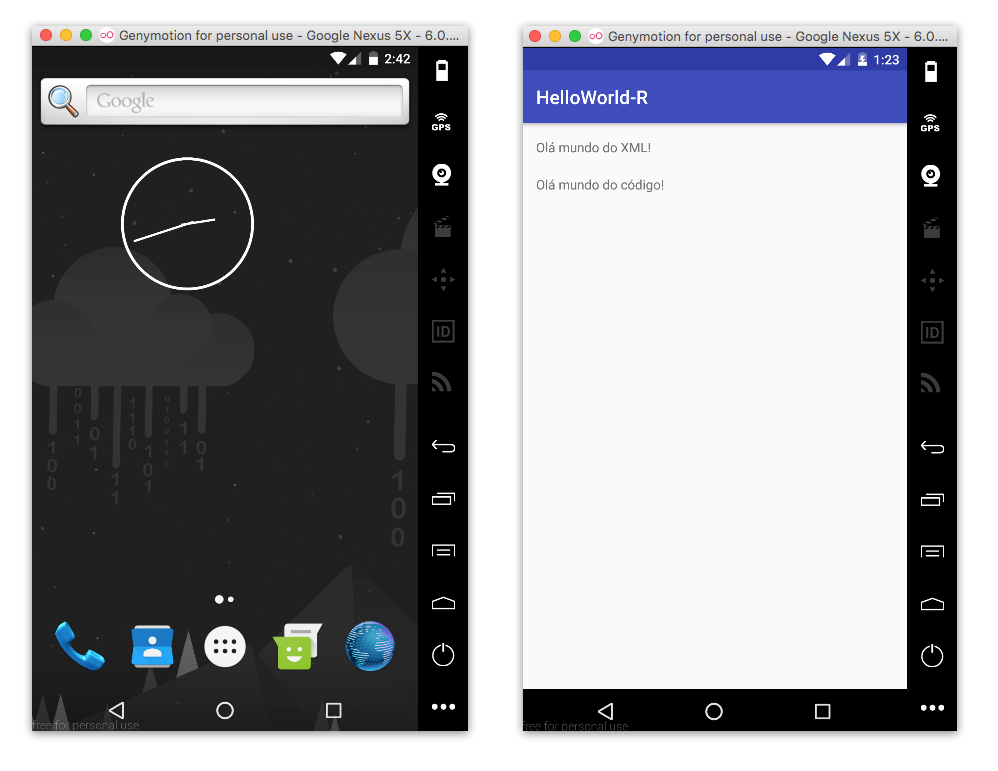


Figura : Aplicativo rodando no Genymotion

## Depuração do código

Algumas vezes, pode-se deixar passar alguns erros no aplicativo, e isso é muito normal. Algum valor sendo mostrado errado, um botão mudando de lugar misteriosamente ou simplesmente um “*crash*” (fechamento inesperado do *app*) podem acontecer e você não faz a mínima ideia de como esses erros estão ocorrendo. É chegada, então, a hora de investigar, ou melhor, depurar o código.

O Android Studio tem incluso um depurador (ou popularmente chamado, *debugger*) que permite que se depure os *apps* em execução, tanto no emulador quanto no dispositivo físico. Com esse *debbuger* você pode:

* Selecionar um *device* para “debugar” seu aplicativo nele;
* Definir ***breakpoints*** (pontos de interrupção) no seu código;
* Examinar variáveis e avaliar expressões em tempo de execução;
* Capturar *screenshoots* e vídeos do seu *app*.

Para iniciar a depuração, clique no botão .

Por enquanto, nada muito novo acontecerá.

### Usando sistema de *logs*

A primeira técnica para *debug* é criar mensagens no código que aparecerão no console do Android Studio.

O sistema de *logs* mostra mensagens enquanto você depura o *app*. Essas mensagens mostram informações relevantes sobre seu *app*. Você tem duas opções de *logs*: os ***Logs* de sistema** e os ***Logs* escritos por você mesmo.**

### Escrevendo suas próprias mensagens de *log*

Para escrever mensagens de *log* em seu próprio código, você deve utilizar a classe Log. Mensagens de *log* podem ajudar a entender o fluxo de execução do seu *app* coletando saídas do sistema enquanto você o usa. Essas mensagens podem revelar as falhas do aplicativo enquanto ele é executado.

Exemplificando abra o seu projeto **HelloWorld-R.**

1. Abra o arquivo **MainActivity.java**. Relembre como foi deixada a sua classe:

packagebr.com.pearson.helloworld\_r;  
import ...  
  
public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected voidonCreate(Bundle savedInstanceState) {  
super.onCreate(savedInstanceState);  
setContentView(R.layout.activity\_main);  
  
*// Obtém o TextView com id "teste" atribuído*  
TextViewtextView = (TextView)findViewById(R.id.teste);  
  
*// Altera o texto do TextViewobtido*  
textView.setText("Olá mundo do código!");  
 }  
}

1. Imagine que você queira acompanhar o comportamento do texto do seu textView. Então verifique qual era o texto contido nele antes de utilizar o método setText(). Insira o *log* como na listagem a seguir:

package br.com.pearson.helloworld\_r;  
import ...  
  
public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
super.onCreate(savedInstanceState);  
setContentView(R.layout.activity\_main);  
  
*// Obtém o TextView com id "teste" atribuído*  
TextViewtextView = (TextView)findViewById(R.id.teste);  
  
*// Mostra o estado anterior do texto do textView*  
Log.e("MainActivity", (String) textView.getText().toString());  
  
*// Altera o texto do TextViewobtido*  
textView.setText("Olámundo do código!");  
 }  
}

1. Execute o *app* e veja o resultado do Android Monitor. Lembre-se de que este textView está definido no arquivo de *layout* **activity\_main.xml** com o texto "New Text".

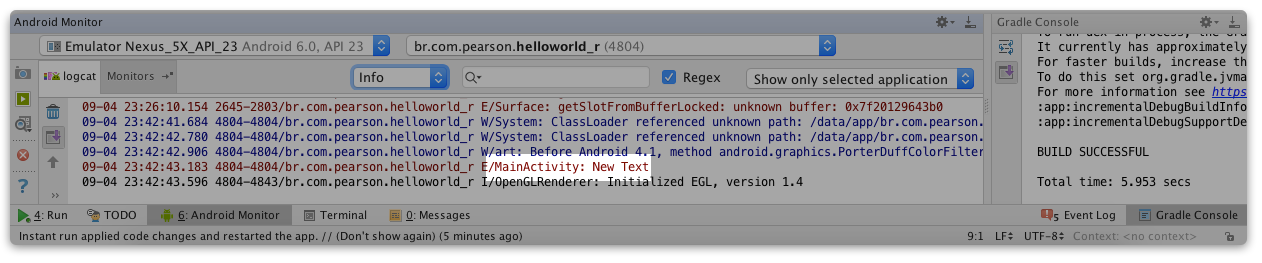


Figura : Log inserido no código sendo mostrado pelo Android Monitor

1. Agora você vai inserir um segundo *log* para verificar o valor do textView após alterar seu valor.

package br.com.pearson.helloworld\_r;  
import ...  
  
public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
super.onCreate(savedInstanceState);  
setContentView(R.layout.activity\_main);  
  
*// Obtém o TextView com id "teste" atribuído*  
TextViewtextView = (TextView)findViewById(R.id.teste);  
  
*// Mostra o estado anterior do texto do textView*  
Log.e("MainActivity", (String) textView.getText().toString());  
  
*// Altera o texto do TextViewobtido*  
textView.setText("Olámundo do código!");

*// Mostra o novo estado do texto do textView*  
Log.e("MainActivity", (String) textView.getText().toString());

}  
}

1. Execute o código e veja o resultado. Observe que, segundo o *log*, o texto que era "New Text" passou a ser "Olá Mundo do código!".

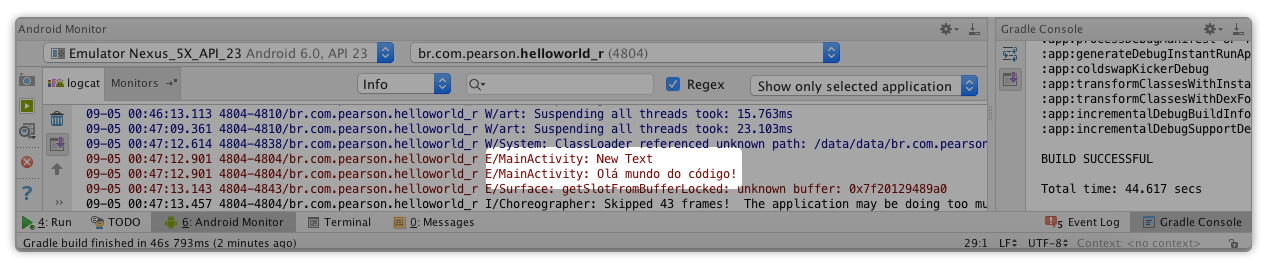


Figura : Outros logs inseridos no código sendo mostrados pelo Android Monitor

Há muito mais coisas que podem ser feitas com a classe Log, então, para saber mais, sempre consulte a documentação oficial do Android - <https://developer.android.com/reference/android/util/Log.html>.

### Trabalhando com *breakpoints*

Talvez seja um pouco complicado trabalhar com os *logs*. Em período de execução, geralmente não se usa essa classe. Os *logs* são realmente úteis após o lançamento do *app* (é chamado de **app em ambiente de produção**), pois pode coletar informações de uso, de erros e estatísticas de usuários reais do *app*. Enquanto o *app* está em **ambiente de produção,** são utilizados os chamados ***breakpoints*** (pontos de interrupção no código).

O Android Studio suporta muitos tipos de *breakpoints* que disparam diferentes ações de depuração. O tipo mais comum é o *breakpoint* em linha de código, que pausa a execução do seu *app* quando aquela linha em específico será executada. Enquanto pausado, você pode examinar os valores das variáveis, avaliar expressões ou então continuar a execução linha por linha para descobrir o que está causando erros em tempo de execução.

Quando o código em execução passa pelo *breakpoint*, o Android Studio pausa a execução do seu *app*. Com o *app* pausado, você pode usar as ferramentas na seção **Debugger** para definitivamente realizar algumas ações:

* Para examinar a árvore de objetos a fim de encontrar uma variável, expanda o item **Variables;**
* Se a seção **Variables** não estiver visível, clique em **RestoreVariablesView** **;**
* Para avaliar uma expressão no ponto de interrupção atual, clique em **EvaluateExpression** **;**
* Para avançar para a próxima linha de código, clique em **StepOver** **;**
* Para avançar para a primeira linha dentro de um método que está sendo chamado, clique em **StepInto****;**
* Para avançar para a próxima linha fora do método atual, clique em **StepOut** ;
* Para retornar à execução normal do *app*, clique em **Resume Program**.

### Visualizando variáveis e avaliando expressões

Para testar os *breakpoints*, você utilizará o mesmo projeto:

1. Abra o projeto **HelloWorld-R** e entre na classe **MainActivity.java;**
2. Para inserir um *breakpoint*, basta clicar na margem esquerda, correspondendo à linha em que se deseja inserir o ponto de interrupção. Insira dois *breakpoints*, sendo um para cada *log* inserido anteriormente:

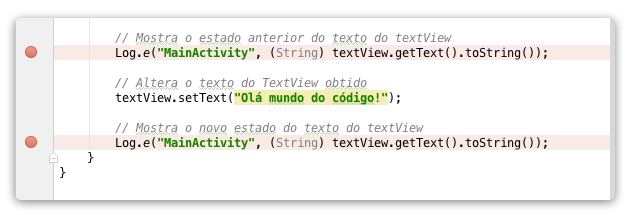


Figura : Breakpoints inseridos no código

1. Agora inicie o modo depuração clicando em ;
2. Espere o *app* iniciar e perceba que ele foi interrompido. Veja como estão as variáveis;
3. Clique em textView e procure o atributo mText. Veja que ainda tem o valor "New Text".

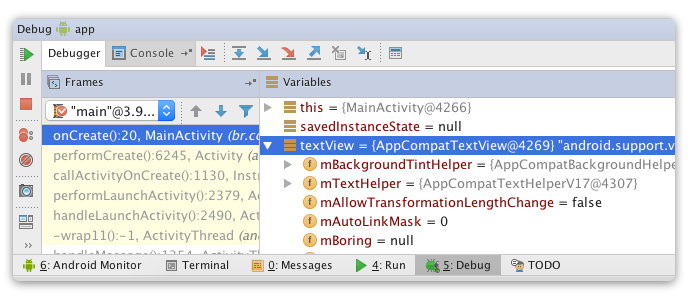


Figura : Caminho para localizar o atributo mText - Parte 1

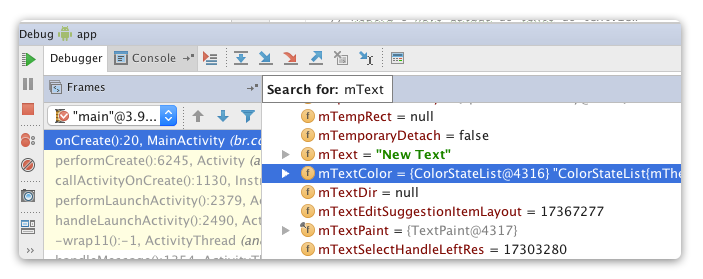


Figura : Caminho para localizar o atributo mText - Parte 2

1. Agora clique em  para pular para o próximo *breakpoint* e veja novamente o valor de mText. Perceba que ele mudou para o novo valor "Olá Mundo do código!".

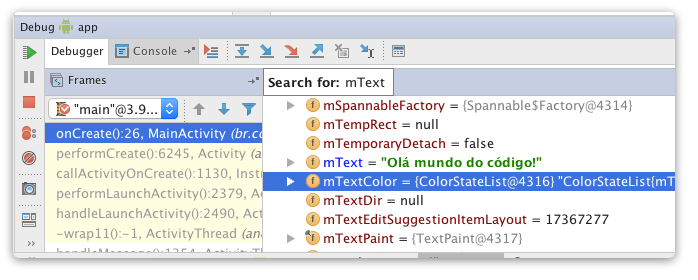


Figura : Valor do mText sendo exibido

Você aprendeu o básico de *Debug*. É claro que é possível utilizar recursos muito mais avançados do que esses. Se você deseja aprender mais, consulte a documentação oficial do Android.