# Introdução à Programação para dispositivos móveis

Ewerton Mendonça

## Curso Técnico em Informática

Educação a Distância

2016

# EXPEDIENTE


**Professor Autor**
Ewerton Mendonça

***Design* Instrucional**
Deyvid Souza Nascimento
Maria de Fátima Duarte Angeiras
Renata Marques de Otero
Terezinha Mônica Sinício Beltrão

**Revisão de Língua Portuguesa**
Letícia Garcia

**Diagramação**
Izabela Cavalcanti

**Coordenação**
Anderson Elias

**Coordenação Executiva**
George Bento Catunda

**Coordenação Geral**
Paulo Fernando de Vasconcelos Dutra

Conteúdo produzido para os Cursos Técnicos da Secretaria Executiva de Educação Profissional de Pernambuco, em convênio com o Ministério da Educação (Rede e-Tec Brasil).

**Outubro, 2016**

Catalogação na fonte

Bibliotecário Hugo Carlos Cavalcanti, CRB4-2129

M539i

Mendonça, Ewerton.

Introdução à Programação para dispositivos móveis: Curso Técnico em Informática: Educação a distância / Ewerton Mendonça. – Recife: Secretaria Executiva de Educação Profissional de Pernambuco, 2016.

39 p.: il.

Inclui referências bibliográficas.

1. Educação a distância. 2. Engenharia de Software. 3. Dispositivos Móveis. I. Mendonça, Ewerton. II. Título. III. Secretaria Executiva de Educação Profissional de Pernambuco. IV. Rede e-Tec Brasil.

CDU – 004.41(043)

## Sumário

# Introdução

Este caderno é sua base principal de conhecimento para a disciplina de Introdução a Programação de Dispositivos Móveis. Apesar de termos um conteúdo auxiliar através de vídeo, é de suma importância que leia com bastante atenção, até mais de uma vez, o conteúdo deste caderno. Nele, você encontrará exemplos e proposta de exercícios. Você deve repetir os exemplos e fazer todos os exercícios para se preparar para as atividades avaliativas.

Este caderno foi preparado para o tempo que a disciplina ocorrerá, por isto, é limitado, mas possui o conhecimento introdutório sobre o assunto que pode e deve ser ampliado futuramente. Durante o curso o aluno deve procurar informações complementares na internet. Existe muito conteúdo em texto e vídeo que pode ajudar o aluno a entender um conceito ou outro que não ficou claro o suficiente. Já é esperado que o aluno tenha esta atitude.

A linguagem de programação usada para criar aplicações no Android é Java. Chamamos o Java de uma linguagem porque ela é uma forma de comunicação com o computador. Ela possui regras de gramática e ortografia muito rígidas, ou seja, o computador só entende da maneira correta. Um errinho de grafia ou uma troca de lugar e o interpretador não vai entender o que você quer que ele faça. Por isso, tenha muita atenção e cuidado ao escrever os códigos. Se algo der errado, verifique letra a letra, palavra a palavra e linha a linha, para ver se você não escreveu algo de forma incorreta. A maior parte dos erros no começo do aprendizado acontece devido a problemas de digitação.

Programação é uma arte na resolução de problemas e, muitas vezes, vamos ter que usar a criatividade para resolver algo com os comandos que temos, porque nem tudo pode ser copiado de algum lugar. Quando terminar os exemplos e atividades disponibilizados, procure outros exemplos na internet e se dedique para entender como o problema é resolvido. O ser humano aprende por repetição. A cada exemplo visto e atividade realizada você entende melhor um conceito. Repita os exemplos e atividades até se sentir confortável com a ideia apresentada. Assim, logo você criará suas próprias soluções.

Antes de começar a meter a mão na massa, ou no código, e criar suas próprias aplicações, é necessário entender o que é a tecnologia Android, de onde ela veio, quais suas versões e seus conceitos básicos. Assim você terá uma fundação melhor e estará mais preparado para alcançar objetivos maiores.

Divirta-se enquanto programa. Programando você pode fazer qualquer coisa, desde que siga as regras que veremos em breve.

Vamos lá?

# 1.Competência 01 | Conhecer os Conceitos Básicos de um Sistema Operacional Móvel

Nesta primeira competência vamos tentar entender o que aconteceu com o mundo para que praticamente todas as pessoas andem com um computador de cima para baixo sem desgrudar um minuto destas maquininhas. Quais foram as tecnologias que permitiram essa revolução e as empresas que hoje dominam o mercado? O que é o Android, quem criou, quem comprou, suas versões ou, podemos dizer, “sabores”?

Pois é. Eu sei que é muita coisa. Mas a história por trás vai nos inspirar a enfrentar o que virá a seguir. Na segunda competência, vamos montar o ambiente de desenvolvimento para escrever o código, encontrar os erros e testar nosso app (abreviação de *application*).

Vamos, então, entrar no mundo da programação móvel, ou mobile (pronuncia-se “mobaio”).

## 1.1 Do desktop para o mobile

O termo *desktop* é utilizado para referenciar os computadores de mesa. Programas *desktop* são programas que foram projetados para serem independentes da internet e dominaram os anos de 1980. Até hoje utilizamos vários deles. Uma calculadora, por exemplo, não precisa estar conectada na internet para desempenhar seu papel. Então, surgiu a internet e, com ela, a web. Entenda que internet e web são duas coisas diferentes. A internet é o meio onde a web existe. Na internet, além da web, temos vários outros serviços que chamamos protocolos. Mas não precisamos entrar em detalhes sobre isso. O ponto a que quero chegar é o que mudou no comportamento do usuário de computador.

Em um ambiente isolado, o *desktop*, o usuário produzia seu conteúdo e pouco compartilhava. Com seu computador conectado à internet as possibilidades de compartilhamento expandiram. O benefício mais visível foi o *e-commerce,* termo utilizado para designar o comércio realizado através da web. Vieram as tecnologias da web, como as páginas e sites. O usuário, então, mudou o

comportamento. Ele se comunica mais, compara mais, é mais crítico, gasta mais. Existem mais clientes, mas também é mais fácil perdê-los.

Dessa forma, as pessoas entenderam melhor a internet. Vieram as redes sociais, a internet colaborativa, os *creators*, *crowdfunding,* etc. O poder das massas. Juntos somos mais fortes. A chamada web 3.0. A necessidade de permanecer constantemente conectado. Durante este período de transformação, o hardware, as peças físicas dos aparelhos, ganharam novos materiais. A internet, agora, podia ser transmitida por rádio ao invés de cabos, estava mais rápida e os celulares prometiam algo maior e melhor.

**CREATOR** é como é chamado o criador de conteúdo para o YouTube.

**CROWDFUNDING** é uma “vaquinha” na internet. Diversos sites propõem vários projetos onde pessoas contribuem para que aquele projeto saia do papel e se torne um produto real

A empresa Apple, comandada por Steve Jobs, foca seu desenvolvimento em uma linha nova de produtos, voltada justamente para esta convergência que a tecnologia estava tomando e lança o iPhone. Um computador pessoal, móvel e conectado. Surge uma nova revolução.

Figura 1 – Steve Jobs mostra o iPhone para o mundo.
Fonte: https://images.rapgenius.com/68d81b5c84469e8c5bdd00f43f576a69.1000x750x1.jpg
Descrição: Steve Jobs segurando o iPhone.

O problema da Apple, se é que podemos chamar de problema, é sua estratégia de mercado. São produtos excelentes, muito bem-acabados e absurdamente caros. Na época, para se desenvolver aplicativos para iPhones, você precisaria de um iPhone, de um Mac e pagar todas as licenças envolvidas na distribuição de seu aplicativo na Apple Store, a loja de aplicativos da Apple. Realmente não era para qualquer um. Mas, graças à visão de Jobs, temos todo um ecossistema para explorar, e muitos outros seguiram seus passos.Com isso, o usuário modificou seus hábitos e percepções novamente, e estamos descobrindo quem ele é e o que precisa, para podermos oferecer.

## 1.2 Breve história do Android

O iPhone mudou a forma como as pessoas se relacionavam com a internet, mas a política da empresa limitava em diversas formas o acesso a esta tecnologia. Diversas empresas viram que poderiam lucrar neste novo mundo e resolveram colocar suas fichas nesta ideia. Entre elas, a gigante Google. Mas vamos voltar um pouco no tempo para saber onde o Android nasceu.


Figura 2 – Criadores do Android.
Fonte: www.singularissoftwares.com/img/pendiri-android.jpg
Descrição: Foto dos quatro criadores do Android.

Palo Alto fica na Califórnia, Estados Unidos. É lá onde as grandes empresas de tecnologia do mundo estão sediadas. E foi lá que Andy Rubin, Rich Mineer, Nick Sears e Chris White fundaram a Android Inc. para desenvolver dispositivos móveis mais inteligentes que estejam mais cientes das preferências e da localização do seu dono. Inicialmente, a empresa começou a desenvolver um sistema operacional para câmeras digitais, mas viram que o mercado não era grande o suficiente. Foi aí que mudaram o foco para telefones móveis.

A empresa enfrentou problemas financeiros até que a Google a comprou em 2005. Foi a forma da gigante entrar no mercado de dispositivos móveis. A empresa, então, convenceu os fabricantes de celulares e operadoras a adotarem o Android com a promessa de ser um sistema flexível e atualizável. Para garantir uma maior adoção, a Open Handset Alliance foi fundada. Ela é um consórcio de tecnologia móvel que inclui a Google, Motorola, HTC, Sony, Samsung, Sprint Nextel, entre outras, com o objetivo de gerenciar e evoluir a tecnologia móvel através de padrões abertos. O Android foi o primeiro produto da Open Handset Alliance e se tornou o principal concorrente para o iPhone.

O que você leu foi um resumo da história por trás do Android. Saiba mais em https://pt.wikipedia.org/wiki/Android

Não se limite apenas a conhecer um sistema. Procure informações sobre o iPhone para ampliar seus horizontes. https://pt.wikipedia.org/wiki/IPhone

### 1.3 O que é o Android

O Android é um sistema operacional para dispositivos móveis, baseado em uma plataforma de código aberto sob a licença Apache. Mas vamos entender melhor os aspectos que tornaram o Android tão popular e que se encontra em sua definição.


Figura 3 – Marca do Android.
Fonte: http://blogdalu.magazineluiza.com.br/wp-content/uploads/2016/03/Conhe%C3%A7as-algumas-das-novidades-do-Android-N.jpg

Descrição: Marca do Android.

Um sistema operacional é uma camada de software que fica entre o *hardware* e as aplicações que os desenvolvedores constroem. Ele é quem gerencia a memória, tanto temporária quanto a persistente, o processador, os diversos sensores como câmera e acelerômetro, e fornece uma forma segura e mais simples dos desenvolvedores utilizarem estas características em seus aplicativos. O sistema possui em seu núcleo uma versão do Linux bem mais leve, com modificações destinadas aos dispositivos móveis. O Linux é um sistema bem estável e que permite que os fabricantes modifiquem a sua necessidade. Seu código fonte é aberto, ou seja, você pode fazer o

*download* do código do sistema e modificar as suas necessidades sem ter que pagar nada a ninguém. Isto dá a liberdade necessária para adicionar novos recursos e incorporá-los ao sistema para que ele evolua rapidamente.

O Android além de ser utilizado em smartphones, também pode ser utilizado em tablets, TV digital, carros e relógios de pulso (Android Wear).

Como ele é baseado em uma plataforma de código aberto, a linguagem utilizada para desenvolver o Android é o Java, que também é gratuita, deixando os produtos Android mais baratos tanto para o usuário final quanto para os desenvolvedores. Então, com um computador com bom desempenho, você pode baixar tudo o que é necessário e começar a desenvolver. O único ponto que é pago no Android é a disponibilização na Google Play para venda de seu aplicativo, no caso, a Google fica com uma pequena porcentagem da venda.

Toda essa filosofia livre para fabricantes, desenvolvedores e consumidores garantiram mais de 3 bilhões de usuários no mundo.

**CURIOSIDADE**
O sistema possui aproximadamente 12 milhões de linhas de código, sendo 3 milhões em XML; 2.8 milhões em C; 2.1 milhões em Java; e 1.75 milhões em C++. Mas você não precisa aprender tudo isto para desenvolver aplicações para o sistema.


Dê uma olhadinha no site da Open Handset Alliance. www.openhandsetalliance.com/

## 1.4 Arquitetura do Android

A arquitetura do Android é construída em camadas, como se fosse um bolo de camadas. Cada

camada só tem contato com o que está logo abaixo e logo acima. Isto beneficia sua modificação e melhora aspectos de segurança. Por exemplo, isto impede que alguém faça um aplicativo que modifique as chamadas telefônicas do usuário sem que ele saiba.

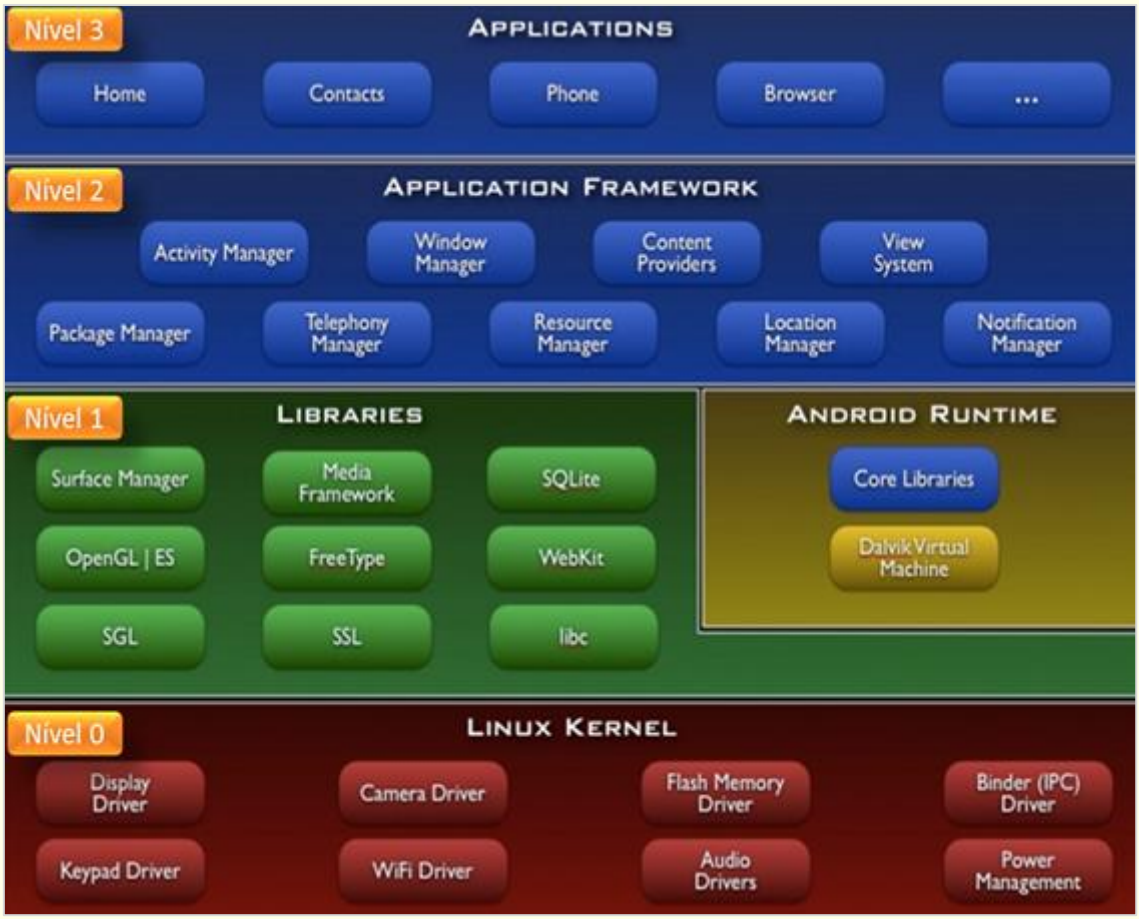


Figura 4 – Arquitetura em camadas do sistema Android.
Fonte: www.tiselvagem.com.br/wp-content/uploads/2011/10/Arquitetura-Android.png
Descrição: A figura mostra a arquitetura do Android dividida em camadas. Embaixo fica o nível 0, acima o nível 1, depois o nível 2 e no topo o nível 3.

O **nível zero**, aquele que está mais próximo do hardware gerencia a memória, configurações de segurança e vários drivers. Drivers são *softwares* que controlam as partes do *hardware*, como a câmera, a tela, o sensor de toque, etc. Quem faz o driver do componente é o fabricante dele. Esta parte do sistema é chamada de núcleo por estar mais profundo no sistema. Para o Android foi utilizado o Sistema Operacional Linux versão 2.6.

No **nível um** temos as bibliotecas, libraries. A biblioteca é um conjunto de código que possibilita

fazer algo para o sistema.

Neste nível, também temos o Android Runtime, parte do sistema responsável por executar as aplicações do aparelho, que, como são escritos em Java, temos uma máquina virtual enxuta chamada Dalvik, a DVM.

É uma característica do Java possuir uma máquina virtual para executar a aplicação. A Dalvik foi criada para aparelhos modestos em desempenho e recursos, como os smartphones.

O **nível dois** é a camada de framework (camada de trabalho para o desenvolvedor). Os desenvolvedores têm acesso total ao framework que possui ferramentas básicas para construção de aplicações mais complexas.

O **nível três** é a camada de interação entre o usuário e o dispositivo móvel, é onde ficam os aplicativos que você vai desenvolver.

A linguagem de programação Java é muito versátil. Ela pode ser utilizada para construir aplicações desktop, web e android. Ela ainda possui diversas bibliotecas gratuitas que agilizam o processo de desenvolvimento.


Saiba mais sobre a linguagem de programação Java antes de continuar. Ela foi criada utilizando o paradigma orientado a objeto. Lembre-se de que conhecimento é poder. https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)

## 1.5 Os diversos sabores do Android

Graças a arquitetura e a utilização de padrões livres, diversas pessoas e empresas contribuem para o aperfeiçoamento do Android. Assim que um conjunto de melhoramentos fica estável, ele é disponibilizado para o público atualizar seus aparelhos. Perceba que, mesmo que uma nova versão fique disponível, ela demora para chegar mais para algumas marcas do que outras. Isto acontece

porque o fabricante modifica para suas necessidades e depois disponibiliza a versão oficial da empresa. Algumas delas nem são disponibilizadas para alguns aparelhos mais antigos. Por isso, é importante saber qual a versão do Android instalado em seu aparelho. Alguns aplicativos só funcionam em versões mais modernas que possuem sensores mais modernos.

Mas, por que eu disse “sabores” em vez de versões do Android?

Figura 5 – O jardim da sede da Android possui estátuas que representam a mascote referente à versão distribuída.
Fonte: www.androidos.in/wp-content/uploads/2014/10/android_statues_google_lawn.jpg
Descrição: Fotografia da entrada da sede da Android exibindo as estátuas das mascotes de todas as versões lançadas.

Cada versão do Android possui um *codiname*, um apelido, e a equipe de desenvolvimento batiza as versões com nomes de doces. Supõe-se que isto aconteça por conta de uma brincadeira interna que nunca veio a público. Outra característica sobre estes *codinames* é a que a primeira letra de cada doce segue a ordem alfabética. Observe a seguir. O nome das duas primeiras versões também nunca veio a público.

### 1.5.1 Android Cupcake

A versão 1.5 do Android foi apelidada de Cupcake. Esta foi a primeira versão a ser disponibilizada comercialmente ao público em smartphones. Ela tinha características voltadas para a captura de imagens, fotografias e vídeo. Além disso, associava essas características com o Picasa, rede social de

fotos da Google, e o YouTube, serviço de vídeos também do Google.

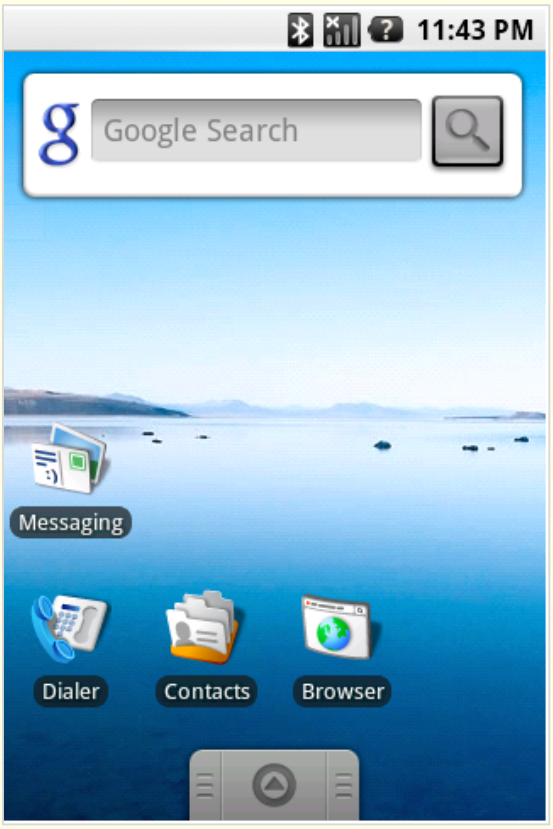


Figura 6 – Interface do Android Cupcake.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial do Android Cupcake.

### 1.5.2 Android Donut

A versão 1.6 ganhou o apelido de donut e vinha com versão aperfeiçoada de busca por voz e galeria de fotos e vídeo melhorada. É claro que cada nova versão vem com melhoramento de desempenho e correção de *bugs* (erros), mas o que estamos destacando aqui são as características mais visíveis para o usuário final.


Figura 7 – Interface do Android Donut
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial do Android Donut.

### 1.5.3 Android Eclair

Eclair é uma bomba de chocolate. Duas versões possuem este *codiname*. Temos, então, a versão Android 2.1, quando o sistema ficou maduro suficiente. Temos uma integração com o Google Maps, já para aproveitar os novos sensores de GPS com geolocalização. Também foi acrescentado suporte ao HTML5 e o Bluetooth 2.1. Os aparelhos começam a ganhar telas maiores.


Figura 8 – Suporte para interfaces com resoluções maiores.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial do Android Eclair em duas resoluções diferentes.

### 1.5.4 Android Froyo

Froyo são conhecidos no Brasil como frozen yogurt. O Froyo corresponde ao Android 2.2. Marcou pelo melhoramento em desempenho e estabilidade. O navegador ganhou um interpretador JavaScript V8 e temos ferramentas de tethering via USB e hotspot.

**TETHERING** é um termo em inglês que corresponde à prática de se utilizar um dispositivo móvel, como um celular, que atua como uma ponte para oferecer acesso de rede a outros equipamentos, remotamente.


Figura 9 – Interface da tela do Android Froyo.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial do Android Froyo.

### 1.5.5 Android Gingerbread

Na versão Android 2.3 houve um melhoramento na interface com suporte a resoluções maiores. Os sensores NFC foram implementados.


Figura 10 – Interface da tela do Android Gingerbread. Possui diferenças sutis de interface.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial do Android Gingerbread.

### 1.5.6 Android Honeycomb

Favo de mel adoçou o desenvolvimento da versão Android 3.0. Houve uma otimização de interface para *tablets*, que foi a febre em 2011. Houve uma integração com mais um serviço Google, o Google Talk, que possibilitou os vídeos chats, também conhecidos como vídeos chamadas.


Figura 11 – Interface da tela do Android Honeycomb em um tablet.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial de um tablet com o Android Honeycomb.

### 1.5.7 Android Ice Cream Sandwich

A versão Android 4.0 foi desenvolvida com o pensamento em tablets e celulares. Uma novidade desta versão foi o desbloqueio do aparelho através de reconhecimento facial.


Figura 12 – Interface da tela do Android Ice Cream Sandwich.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial do Android Ice Cream Sandwich.

### 1.5.8 Android Jelly Bean

A versão Android 4.1 veio com melhoramento de desempenho no sistema de toque e a inclusão do Google Now. A ideia do serviço é ele saber o que você precisa antes de você precisar. São os primeiros passos para um assistente pessoal inteligente.

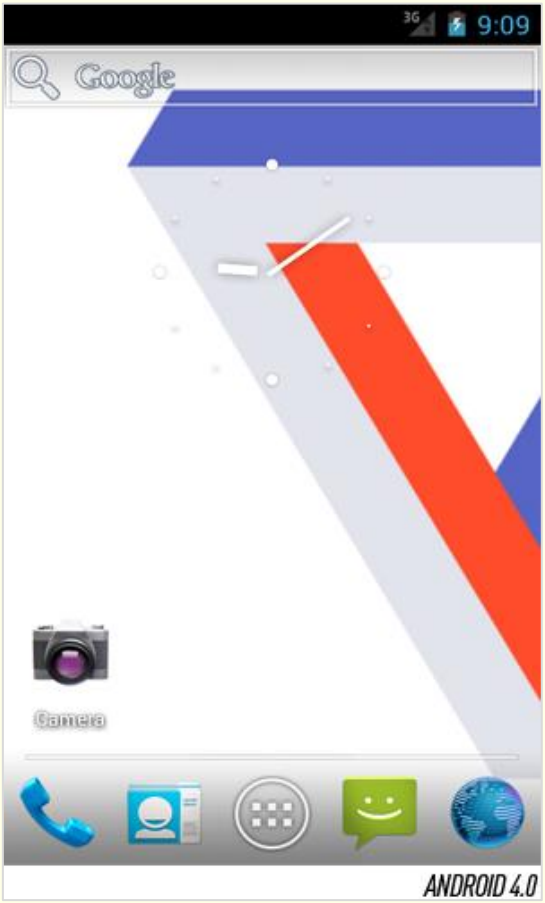


Figura 13 – Interface da tela do Android Jelly Bean.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Captura da tela inicial do Android Jelly Bean.

### 1.5.9 Android KitKat

Lançado em 2013, o Android 4.4 fez uma grandiosa campanha com a Nestlé, dona da marca KitKat de chocolate. Introduziu a barra de notificações transparente e o Google Now na tela inicial. O sistema ficou mais simples, melhorando o desempenho e sendo possível sua instalação em aparelhos mais antigos.


Figura 14 – Interface da tela do Android Kitkat.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Imagem mostra a tela inicial do Android Kitkat.

### 1.5.10 Android Lollipop

Lollipop foi o doce escolhido para o Android 5.1. Foi introduzida uma nova linguagem de design em todo o Android. Project Material baseada inicialmente no Google Now. A implementação de aplicativos multitarefas foi redefinida e o sistema operacional foi preparado para sua expansão nos dispositivos Wear.


Figura 15 – Computação vestível. O relógio inteligente com o Android Lollipop.
Fonte: www.theverge.com/2011/12/7/2585779/android-history
Descrição: Relógio inteligente que exibe o tempo restante para se chegar em casa, baseado no GPS do usuário.

### 1.5.11 Android Marshmallow

Cada nova versão vem com melhorias que estão relacionadas com a época em que foram lançadas, para esta versão foi o ano de 2015. A versão Android 6.0 veio com suporte para leitor de digitais e modo de tela 4k.


Figura 16 – Interface da tela do Android Marshmallow.
Fonte: https://pt.wikipedia.org/wiki/Hist%C3%B3rico_de_vers%C3%B5es_do_Android#Android_Marshmallow
Descrição: Captura da tela inicial do Android Marshmallow.

### 1.5.12 Android Nougat

No ano de 2016 foi lançado o Android 7.0. Foi dada atenção para a segurança nesta versão, o sistema operacional agora possui encriptação nativa. O sistema também ganhou novos emojis. Aqueles desenhos que aparecem em comentários e que representam emoções. Além disso, temos agora um modo de realidade virtual.


Figura 17 – Interface da tela do Android Nougat.
Fonte: http://cdn.bgr.com/2016/08/google-nexus-2016-android-7-0-nougat-launcher-search-3.jpg?quality=98&strip=all&strip=all
Descrição: Captura da tela inicial do Android Nougat.

Assista ao vídeo de lançamento do codiname do Android 7.0.
www.youtube.com/watch?v=8xn9iq3lG_w

### 1.5.13 Dor de barriga

Por que mostramos todas estas versões?

Você pode pensar que basta utilizar a mais atual e pronto. Bem. Isto vai te dar poder e alguns problemas. As versões não são todas compatíveis entre si. Algumas alterações fundamentais foram feitas e muita coisa mudou. Assim, se você for produzir um aplicativo para Android, antes tem que

determinar qual é a mínima versão que ele pode rodar. E, dependendo da biblioteca que utilizar, as versões mais antigas ficarão de fora, e nem todo mundo tem um celular top de linha. Então, seja modesto. Comece com aplicativos para a versão 2.3 e quando tiver maior conhecimento, vá aumentando. O mercado possui diversas versões, das mais antigas às mais modernas. Será ótimo se seus aplicativos puderem rodar na maioria delas. Lembre-se: o que você pode, ou não utilizar, vai depender da versão mínima que escolher.

O aluno do EAD deve ter a característica de pesquisar aquilo que não conhece. Exercite esta capacidade pesquisando todos os termos técnicos e até o nome dos doces que não conhece para um melhor aproveitamento do curso.

Se você tem um smarthphone com Android, procure na internet como achar a versão do sistema operacional que está instalada nele. Apesar das formas entre versões serem bem similares, alguma coisa mudou de uma versão para a outra. Será um bom exercício para pegar a intimidade com o Android.

## 1.6 Google Play

A Google fornece o serviço Google Play. Um mercado no qual os programadores podem ofertar seus aplicativos para usuários do Android. Os usuários utilizam o Google Play para comprar e instalar aplicativos em seus aparelhos.

O Google Play também oferece um serviço de atualização. Se um desenvolvedor fizer o upload de uma nova versão de seu aplicativo, o Google Play informa aos usuários que possuem o aplicativo sobre a atualização.

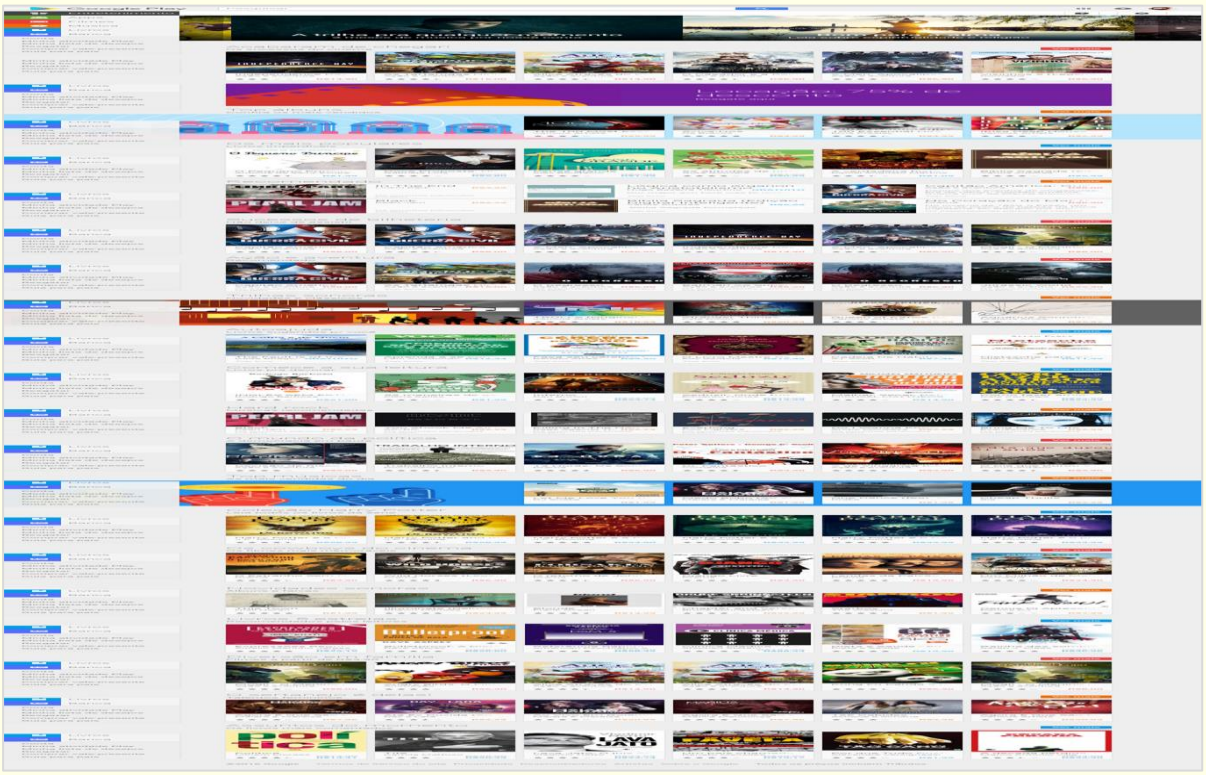

Figura 18 – Página inicial do Google Play.
Fonte: Autor.
Descrição: Página inicial do Google Play, em que aparecem vários aplicativos gratuitos e pagos.

## 1.7 Preparação para a próxima competência

Na próxima competência montaremos o ambiente de desenvolvimento e o ambiente de teste. Os programas necessários para isso são grandes. Um deles é de mais de 1.6 Gb. Sugerimos que desde já baixe os programas para que você tenha tempo de correr atrás de alguma solução, caso dê algum problema.

Baixe o JDK no link abaixo:

www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html?ssSourceSiteId=otnpt

Baixe o Android Studio no link abaixo:

https://developer.android.com/studio/index.html#downloads

Não precisa instalar. Apenas deixe baixado, ou consiga com algum colega que já o tenha feito. Assim não terá problema para a próxima disciplina. Caso precise de mais instruções para baixar os programas, o começo da próxima competência explica passo a passo. Dê uma olhada lá.

## 2.Competência 2 | Configurar o Ambiente de Desenvolvimento e Conhecer Arquivos Básicos de XML

Vimos na competência anterior uma introdução para sabermos onde estamos nos metendo. Aprendemos sobre o Android, sua história e suas diversas versões. Isto vai nos dar uma base para sabermos o poder do Android que precisamos sem deixar muita gente de fora. Agora vamos preparar nosso ambiente de desenvolvimento e teste, e ver como é um projeto básico com o significado de alguns arquivos.

Se você se preocupou em fazer o download do material como explicado na competência anterior, todo os arquivos necessários já estão em seu computador e agora só precisamos instalar e configurar. Como dito anteriormente, os arquivos são muito grandes e você precisa de uma conexão com a internet de boa qualidade para não ter problemas.

### 2.1 Baixando e instalando o JDK

Como a linguagem utilizada na produção de aplicativos é o Java, devemos baixar a biblioteca e ferramentas de desenvolvimento do Java. Todas elas estão armazenadas em um conjunto de arquivos chamado JDK (Java Development Kit, traduzido fica Quite de Desenvolvimento Java). Vá para a página do link abaixo. Irá aparecer a página da Figura 19. Marque a opção circundada de vermelho e clique no link correspondente à versão de seu sistema operacional. Como exemplo, deixei destacada em amarelo a versão para o Windows 64 bits.

www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html?ssSourceSiteId=otnpt

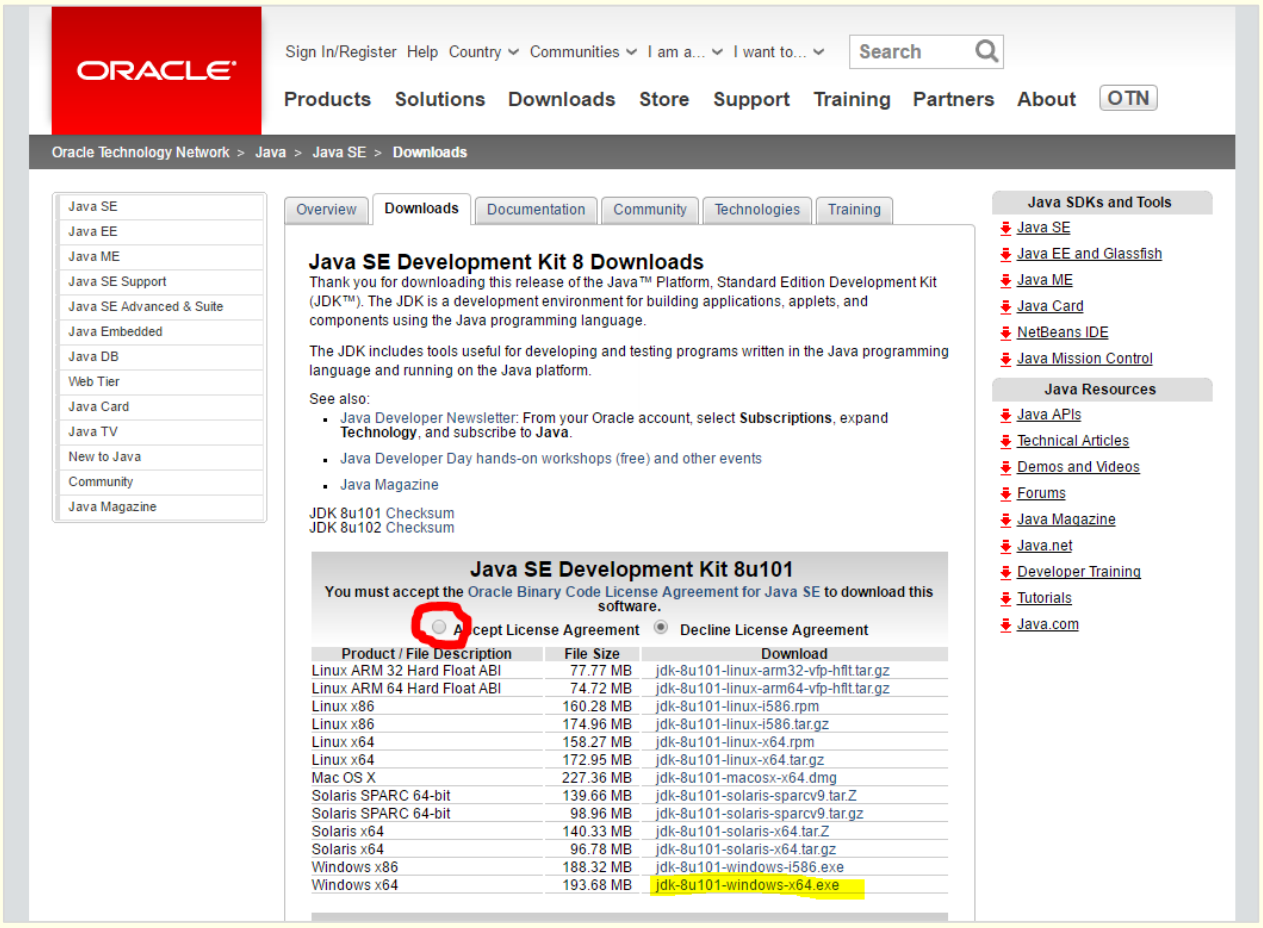


Figura 19 – Página de download do JDK.
Fonte: Autor.
Descrição: Página de download do JDK com uma marca vermelha no lugar para aceitar a licença de uso e um destaque amarelo em um link para o JDK do Windows 64 bits.

No momento da escrita deste caderno, a versão do JDK era a 8u101. Ela possui o tamanho de 194 Mb e, dependendo da qualidade de sua conexão, deve demorar um pouco para baixar.

Seguimos com o passo a passo para o sistema operacional Windows. Para outros sistemas faça uma pesquisa na internet para a instalação do JDK para seu respectivo sistema operacional.

Vai aparecer a janela do programa de instalação. Basta clicar em “Next >” ou “Próximo >” para prosseguir com a instalação.

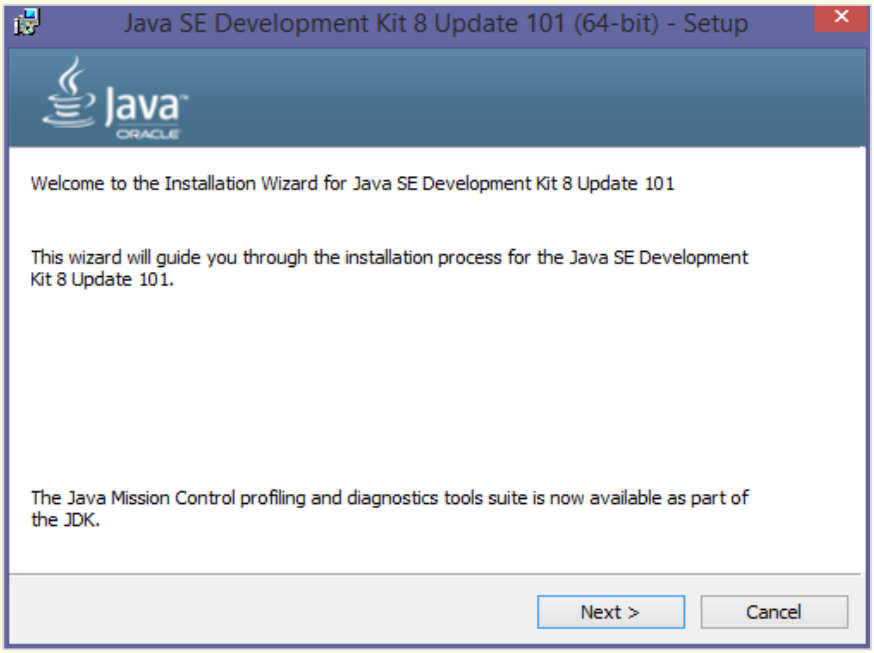


Figura 20 – Janela de instalação do JDK.
Fonte: Autor.
Descrição: Janela de instalação do JDK.

Ao final aparecerá a janela da Figura 21. Basta clicar em “Close” para encerrar a instalação.


Figura 21 – Janela de instalação do JDK ao final do processo.
Fonte: Autor.
Descrição: A imagem mostra a tela final do processo de instalação do JDK com um botão “Close” para fechá-la.

## 2.2 Android Studio

Além do JDK, precisaremos das bibliotecas específicas para a versão do Android para qual produziremos, chamada de Android SDK. Além do Android SDK precisaremos de uma IDE. Uma IDE é um programa que ajuda a programar, que será o Android Studio. Além do SDK e do Android Studio, vamos precisar de uma máquina virtual para fazer de conta que tem um celular com Android no computador, a Android Virtual Device, com a versão específica do Android para poder rodar, que chamamos imagem.

Ufa!

Mas não é tão difícil quanto parece. O Android SDK, o Android Studio e a Android Virtual Device vêm tudo de uma vez quando a gente baixa o Android Studio. Então, se não baixou, terá que baixar agora. A IDE e o SDK juntos têm o tamanho de mais de 1.6 Gb. Pode levar algumas horas para baixar. Pois é! Muito grande. Outro ponto negativo é que você precisará de um computador potente para ter mais conforto durante os testes. Memória é essencial para rodar uma máquina virtual e vamos rodar a IDE e a máquina virtual, juntas.

Outra observação é que o Android Studio será atualizado depois de instalado. Então, você precisa estar conectado à internet para terminar a instalação. O bom é que só fazemos isso uma vez, para começar.

Antes a Google disponibilizava o Eclipse como IDE e um plug-in que deveria ser instalado no Eclipse para que ele pudesse ajudar no desenvolvimento para Android, mas o projeto do plug-in foi descontinuado e substituído pelo Android Studio.

### 2.2.1 Download do Android Studio

Depois de todos esses avisos, vamos começar a preparar o ambiente. Vá para a página do link abaixo. A Figura 22 mostra a página que foi capturada no momento da escrita deste caderno. A seta

vermelha aponta para o botão de download.

https://developer.android.com/studio/index.html#downloads

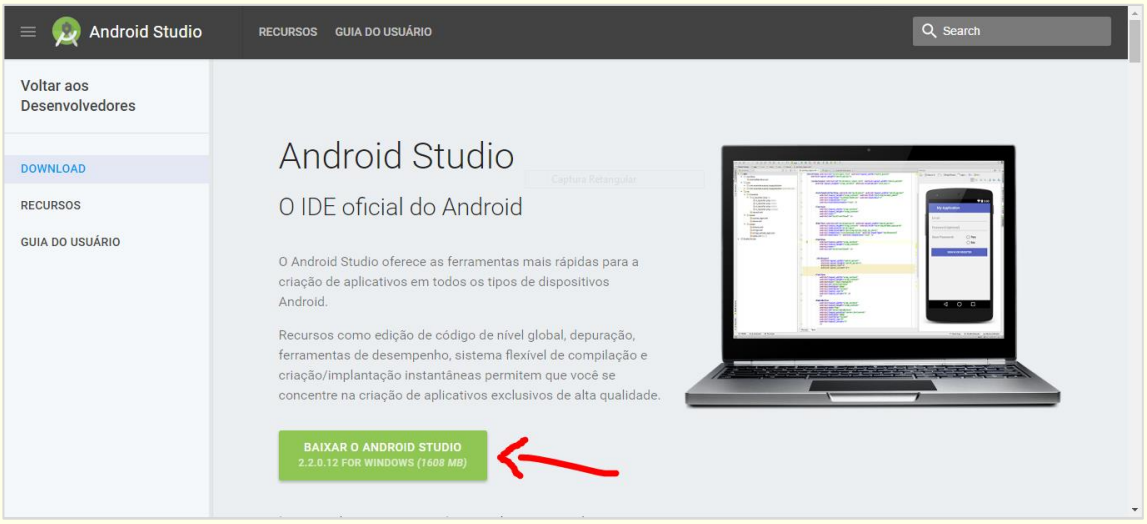


Figura 22 – Página de download do Android Studio.
Fonte: Autor.
Descrição: Página de download do Android Studio com uma seta vermelha apontando para o botão de download.

Leia e concorde com os termos e condições de uso e baixe o instalador. Veja a Figura 23, a seta aponta para a caixa de marcação para concordar com o contrato de uso.

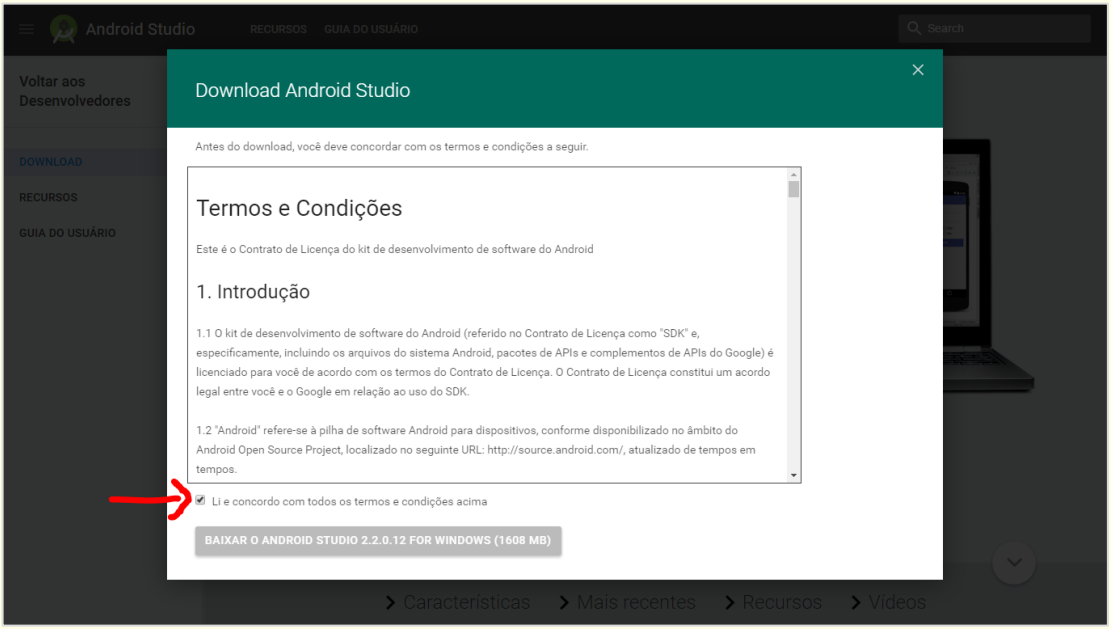


Figura 23 – Página com o contrato de uso do Android Studio.
Fonte: Autor.
Descrição: Página com o contrato de uso do Android Studio. Uma seta vermelha aponta para a caixa de marcação para aceitar o contrato e poder fazer o download.

Enquanto baixa, você pode ler o tutorial para o seu sistema operacional. Na Figura 24, a seta vermelha mais alta aponta para a caixa de seleção do sistema operacional correspondente. Além disso, a página contém um vídeo que mostra a instalação no sistema operacional correspondente, que está apontado pela seta vermelha mais abaixo. O processo é bem simples. Siga os passos do vídeo.

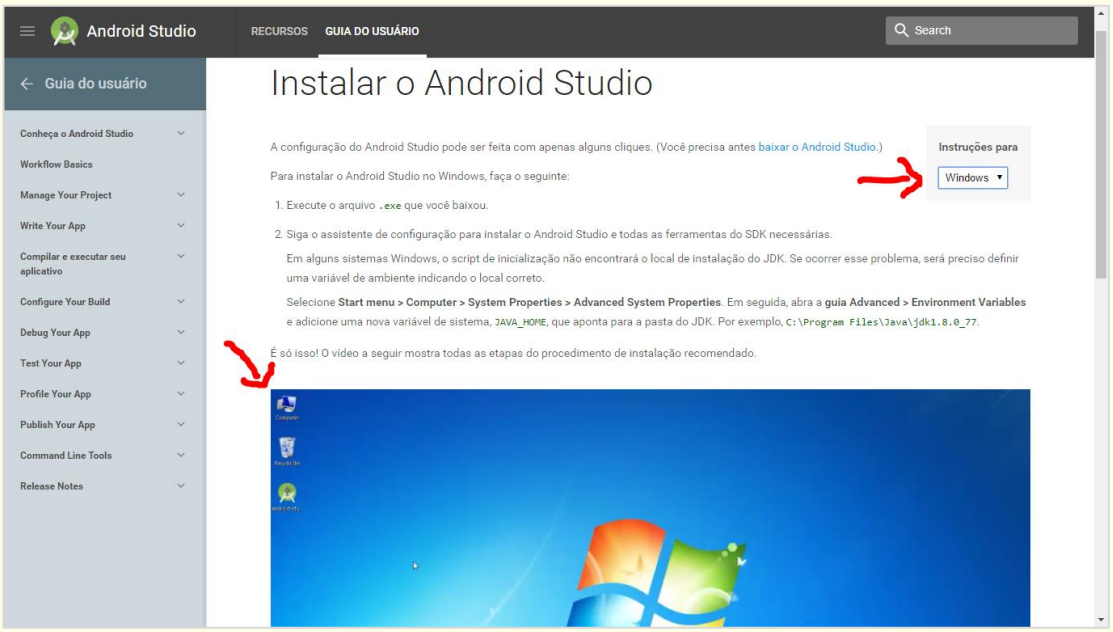


Figura 24 – Página com o tutorial de instalação do Android Studio.
Fonte: Autor.
Descrição: Página com o tutorial de instalação do Android Studio. Uma seta vermelha ao lado direito aponta para a caixa que muda o sistema operacional relativo ao tutorial, com opções de Windows, Mac e Linux. Outra seta à esquerda aponta para o vídeo que mostra a instalação no sistema operacional correspondente. Na imagem temos o Windows.

### 2.2.2 Instalando o Android Studio

Depois de baixar o arquivo, execute-o. A janela de instalação aparecerá. O processo é o padrão de instalação de aplicações Windows. Caso você queira assistir antes, tem um vídeo na página de tutorial que mostra o procedimento. Aceite as definições e configurações padrão.

Quando a instalação terminar aparecerá a janela da Figura 25.

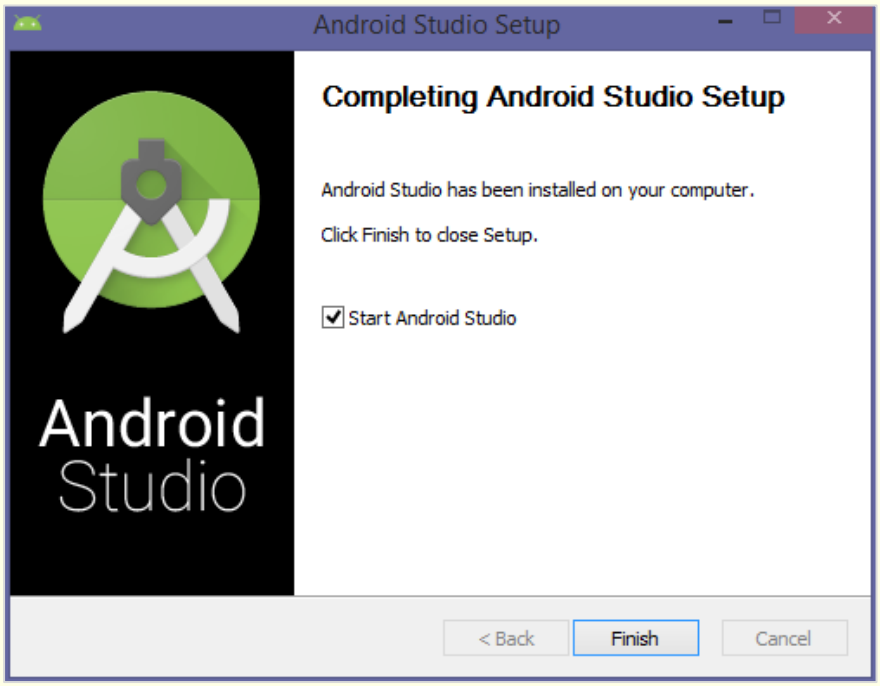


Figura 25 – Janela de encerramento de instalação do Android Studio.
Fonte: Autor.
Descrição: A imagem mostra a tela final da instalação, com a opção de iniciar o Android Studio marcada e um botão “Finish” para finalizar o procedimento.

Ao executar o Android Studio pela primeira vez, aparecerá uma janela perguntando se você deseja importar alguma configuração prévia. Deixe como está e pressione “Ok”.

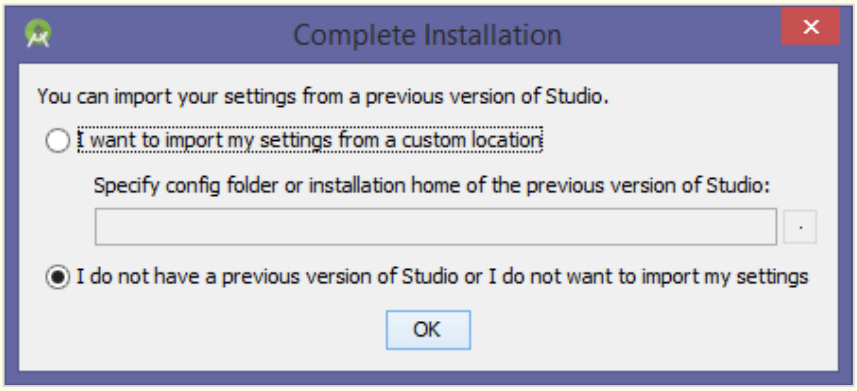


Figura 26 – Janela de importação de configuração.
Fonte: Autor.
Descrição: A imagem mostra a janela de importação de configuração com a opção de não importar uma configuração prévia marcada e um botão “OK”.

A splash de inicialização aparecerá.


Figura 27 – Splash do Android Studio.
Fonte: Autor.
Descrição: esta imagem é mostrada sempre que o Android Studio é aberto enquanto prepara os arquivos necessários para funcionar corretamente. Mostra apenas a logo do mesmo e uma barra de progresso indicando o andamento da preparação dos arquivos necessários.

Agora o Android Studio fará o download das atualizações.

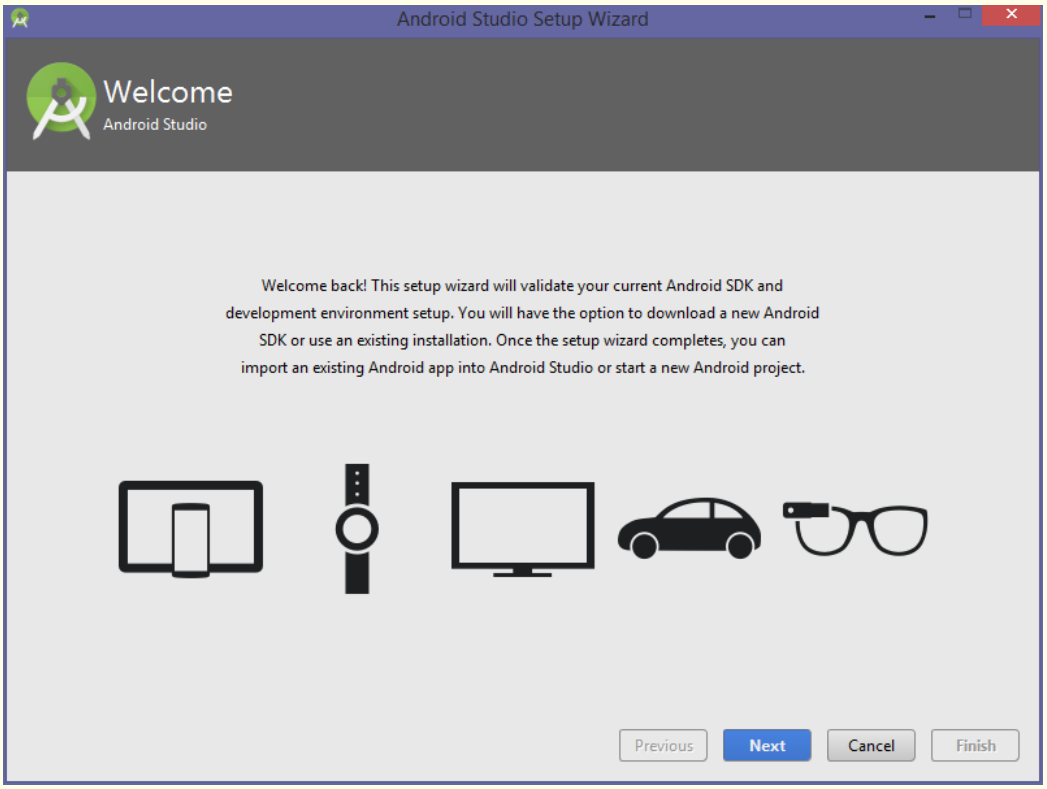


Figura 28 – Tela do assistente de configuração do Android Studio.
Fonte: Autor.
Descrição: A imagem mostra uma tela de boas vindas do Android Studio dizendo que validará o Android SDK padrão e as configurações iniciais padrão. No futuro você poderá mudar a versão do Android SDK, fazendo o dowload dela ou utilizar alguma que já esteja disponível. A tela exibe símbolos referentes as tecnologias ambiente para o Android: smartfones e tablets, relógios, TVs, carros e Google Glass.

O download deve demorar um tempo maior ou menor, dependendo da velocidade de conexão. Aguarde terminar.

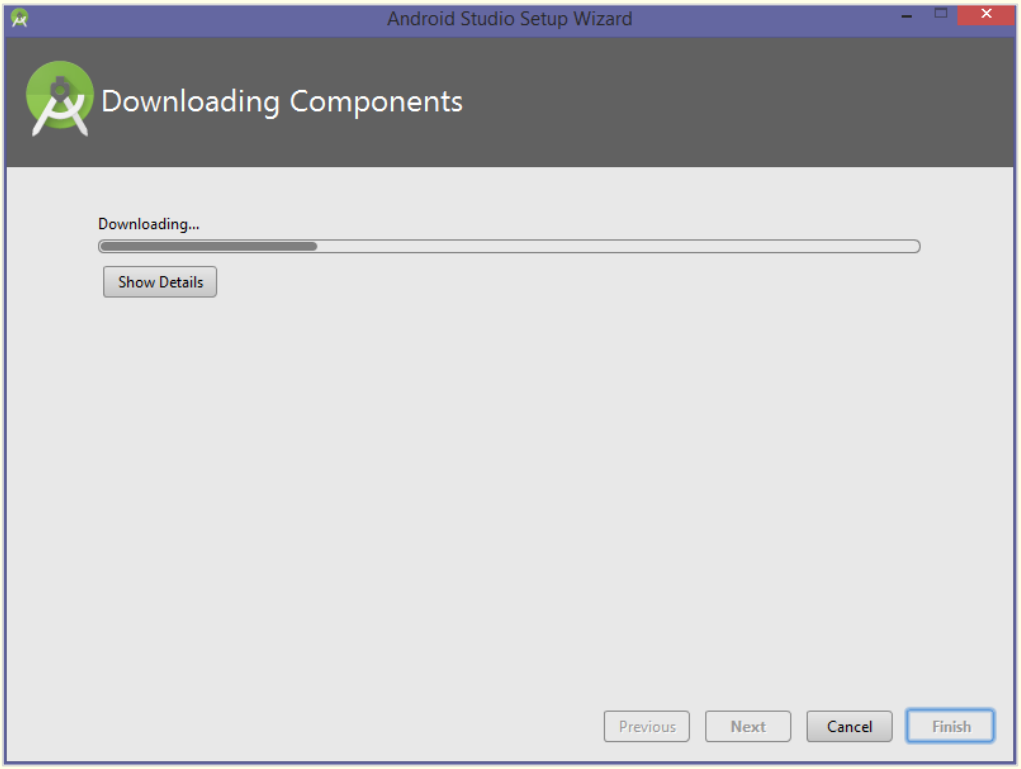


Figura 29 – Tela de download da atualização.
Fonte: Autor.
Descrição: na tela de download da atualização é mostrada uma barra de progresso indicando o andamento do download das atualizações necessárias, há também um motão com o texto “show details” no qual é possível clicar para ver os detalhes do download, além disso na parte inferior direta da tela há quatro botões, porém apenas o botão “cancel” está ativo, ou seja, é possível cancelar o download. Caso seja cancelado o Android Studio tentará realizar a atualização na próxima vez que for iniciado. Se a atualização não for feita pode ser que o programa não funcione corretamente.

### 2.2.3 Iniciando um novo projeto

Quando tudo estiver atualizado, a janela de inicialização vai aparecer com as opções básicas.

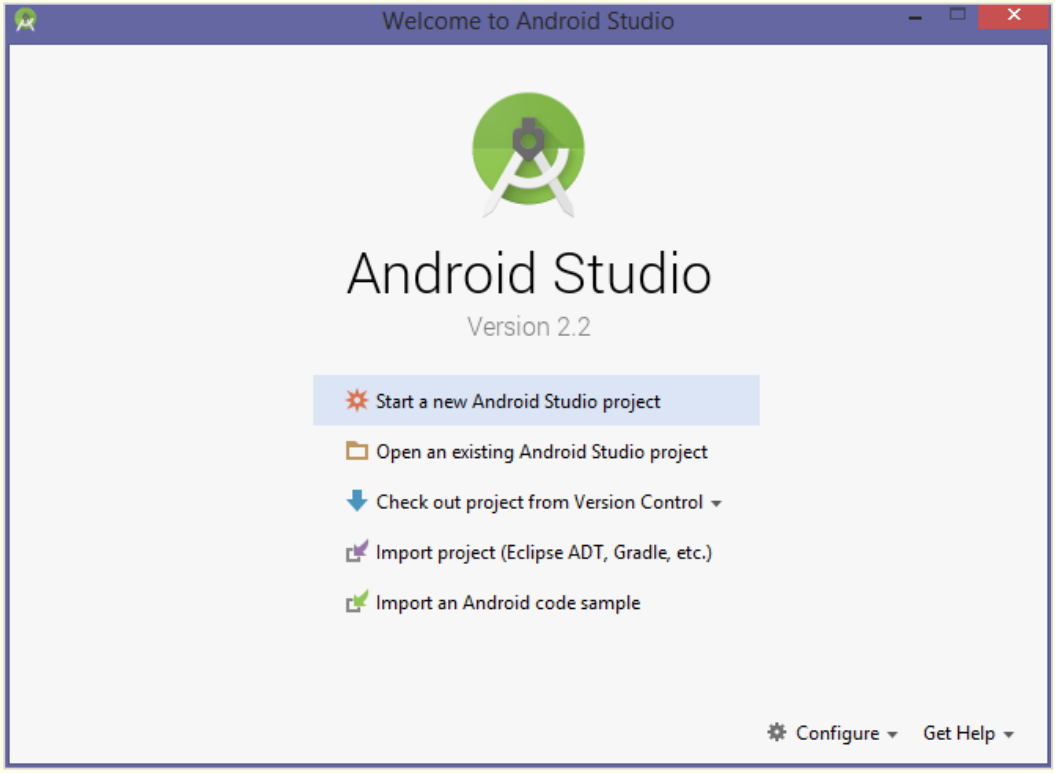


Figura 30 – Janela de inicialização do Android Studio.
Fonte: Autor.
Descrição: A imagem mostra a janela inicial do Android Studio com as opções de iniciar um novo projeto, abrir um projeto existente, fazer um download de um servidor de versão, importar um projeto de outros aplicativos e importar um código de exemplo.

De cima para baixo as opções são:

- Iniciar um novo projeto Android Studio;
- Abrir um projeto do Android Studio;
- Baixar um projeto de Controle de Versão;
- Importar um projeto;
- Importar um código de exemplo do Android.

Clique na primeira opção, “Iniciar um novo projeto Android Studio”. Aparecerá uma janela solicitando dados de configuração do projeto.

| CAMPO | VALOR | DESCRIÇÃO |
|---|---|---|
| Application name | Alo Mundo | O nome da aplicação |
| Company domain | edu.pe.gov.br | O domínio da empresa dona da aplicação. |
| Project location | Deixe como está | Mostra o local onde serão salvos os arquivos do projeto. |
| Package name | Deixe como está | É formado pelo domínio e o nome da aplicação. |

Tabela 1 – Campos, descrição dos campos e valores propostos.
Fonte: Autor.
Descrição: Campos, descrição dos campos e valores propostos.

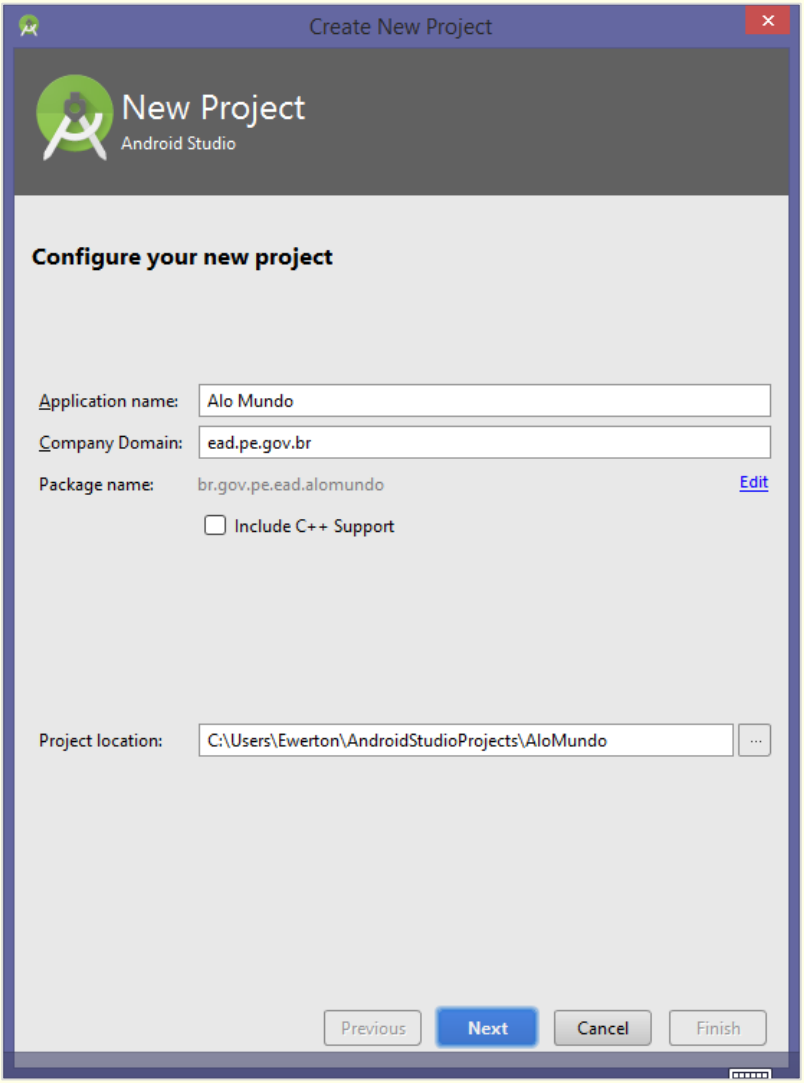


Figura 31 – Janela de Configuração de Novo Projeto no Android Studio.
Fonte: Autor.
Descrição: A imagem mostra a tela de configuração de um novo projeto no Android Studio, com os campos de nome da aplicação, domínio da companhia e o caminho da localização dos arquivos do projeto.

A próxima tela configura o dispositivo e a versão mínima do projeto. Como é um projeto de exemplo, deixe como está.

O Android Studio transferirá automaticamente o SDK do Android, caso ainda tenha sido feito. Basta pressionar “Next”.

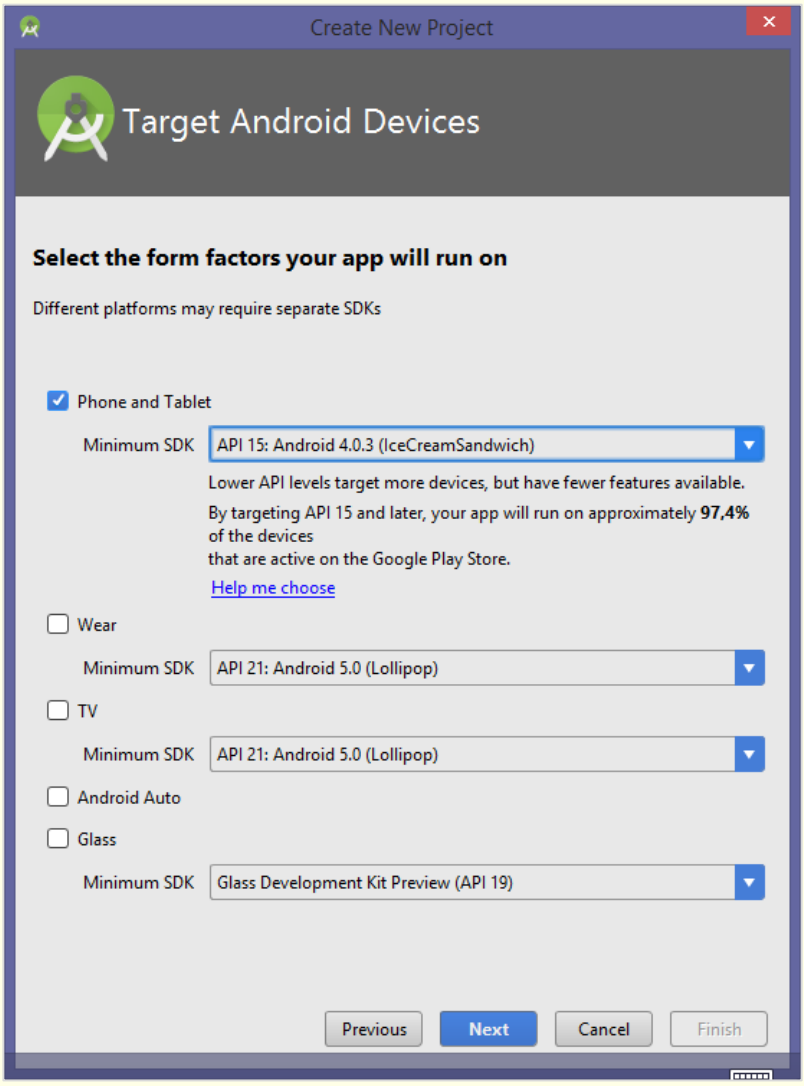


Figura 32 – Janela de compatibilidade da versão mínima e outras plataformas para a futura aplicação.
Fonte: Autor.
Descrição: A imagem mostra as opções de compatibilidade mínima com as versões do sistema operacional Android, além de aplicações para o Wear (tecnologia vestível), TV, Android Auto (Carros) e Glass.

O Android Studio vem como alguns projetos iniciais pré-configurados. Vamos selecionar o “Empty

Activity”.

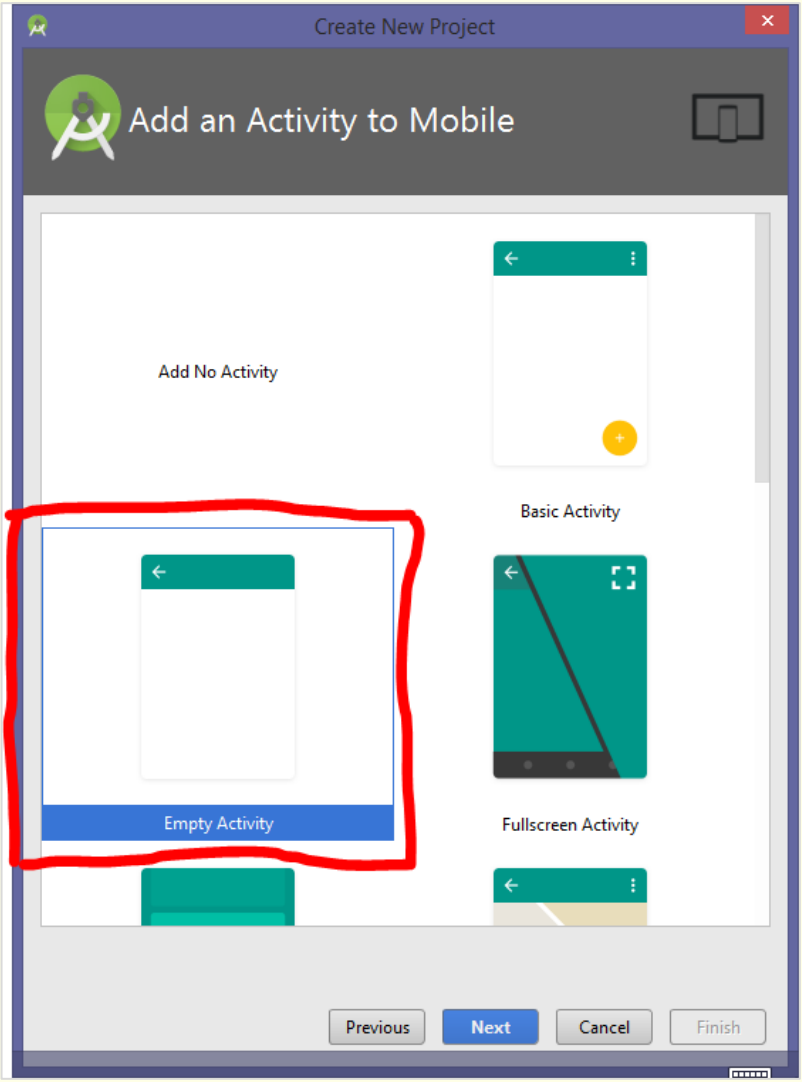


Figura 33 – Janela de templates para a Activity inicial. A imagem marca a opção que deve ser selecionada para nosso exemplo.
Fonte: Autor.
Descrição: Janela de templates para a Activity inicial. Template é um modelo inicial.

Desmarque a caixa “Backwards Compatibility (AppCompat)”.

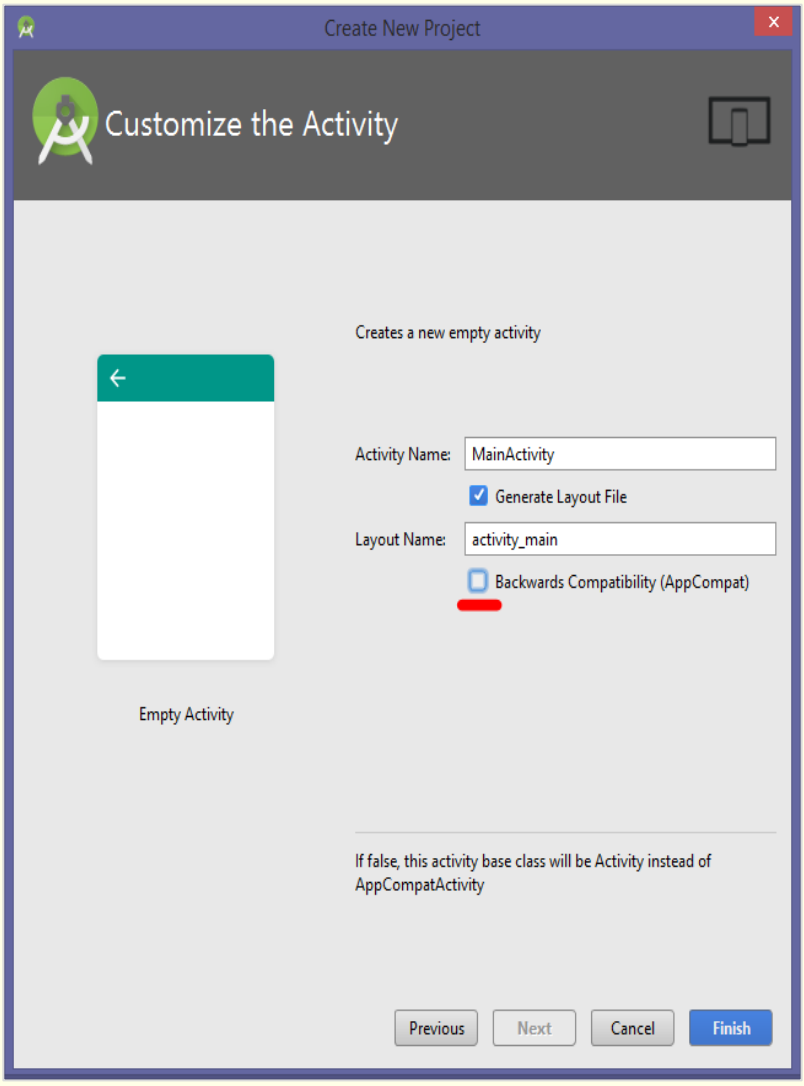


Figura 34 – Janela de personalização da Activity inicial. Desmarque a caixa de marcação destacada em vermelho.
Fonte: Autor.
Descrição: A imagem mostra a tela de personalização da Activity inicial com campos para colocar o nome da classe, uma caixa marcada para gerar um leiaute para esta activity, o nome do arquivo de layout e uma caixa, que deve ser desmarcada, de compatibilidade com o AppCompat, para compactação.

Com isso, finalizamos as configurações no nosso primeiro projeto “Alo Mundo”. Pressione “Finish” e o Android Studio criará todas as pastas e arquivos necessários para você começar.

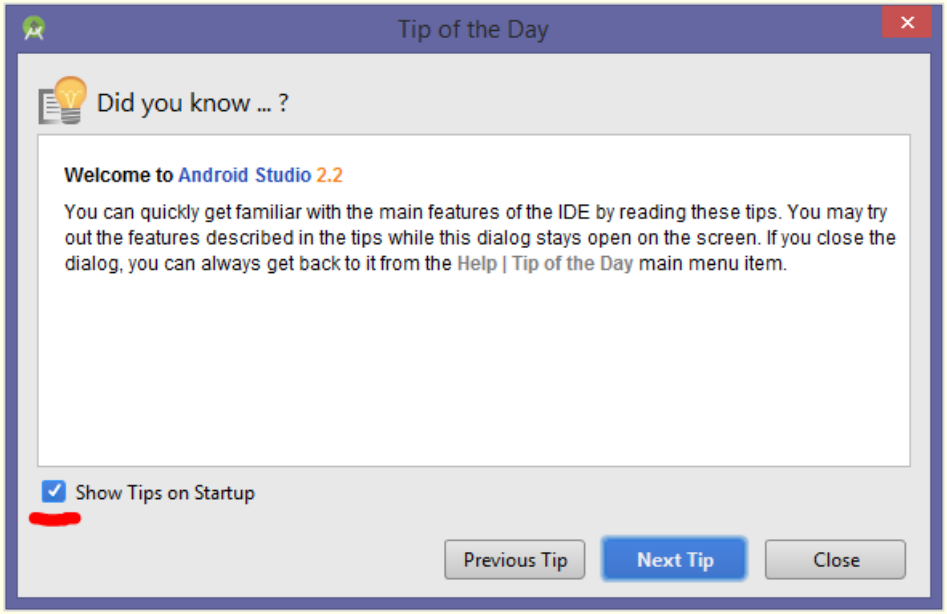


Figura 35 – Janela de dicas.
Fonte: Autor.
Descrição: Janela de dicas. Para não exibir mais, selecione a caixa de marcação destacada em vermelho.

A janela de dicas aparecerá. Desmarque a caixa “Show Tips on Startup” para que ela não apareça mais e clique em “Close” para fechar.

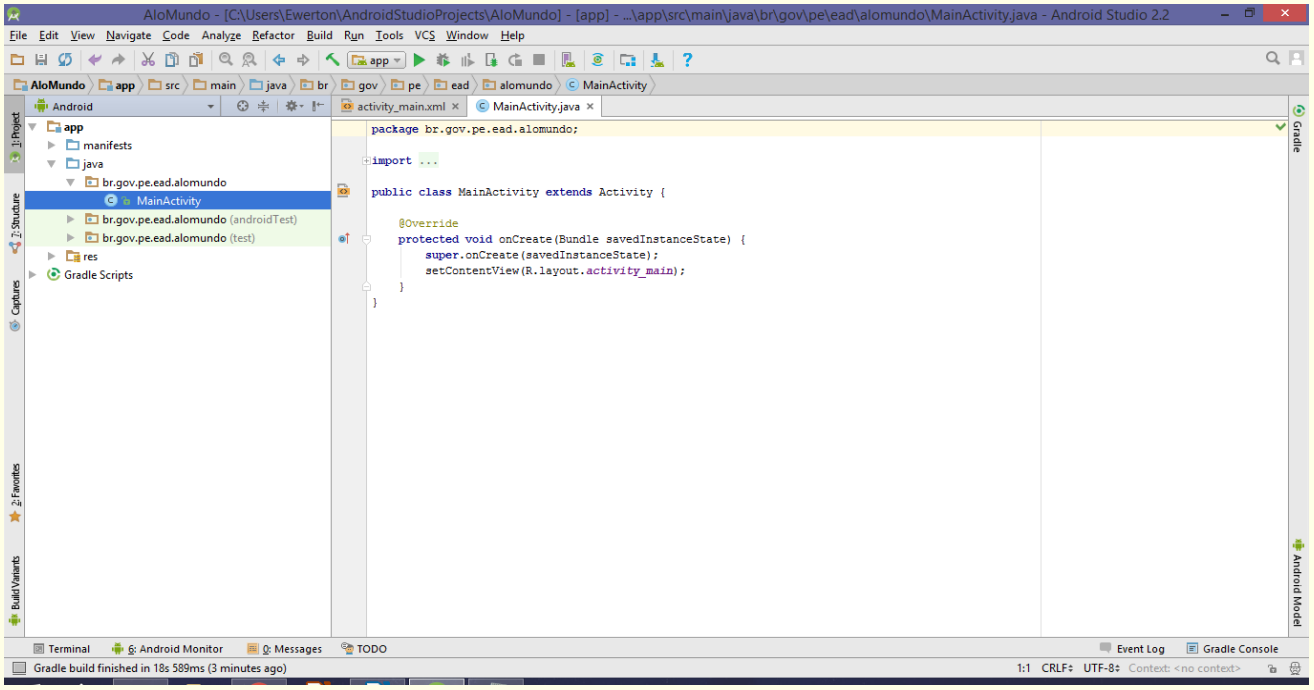


Figura 36 – Área de trabalho do Android Studio.
Fonte: Autor.
Descrição: A imagem mostra a área de trabalho do Android Studio com o arquivo MainActivity.java aberto e o código básico já escrito.

A Figura 36 mostra o Android Studio com o projeto “Alo Mundo” aberto.

### 2.2.4 Conhecendo a estrutura do projeto

Como disse, uma IDE é um programa que ajuda a programar. Ele ajuda de várias maneiras, uma dessas maneiras é construindo para você as pastas e arquivos iniciais para a programação de um projeto em Android. São várias as pastas e arquivos construídos, mas não utilizaremos todos. Os arquivos são textos simples, sem formatação, então, podem ser editados até no Bloco de Notas do Windows, mas a IDE nos dará mais ajuda em outros aspectos. Vamos, então, conhecer a estrutura básica de um projeto Android.

Observe na Figura 37. Nela temos uma visualização da estrutura lógica do projeto e não da estrutura real. O Android Studio nos mostra uma visão simplificada daquilo que vai nos interessar mais. Temos, então, uma pasta com o nome ‘app’ e nela três outras: ‘manifests’, ‘java’ e ‘res’.

Na pasta ‘res’ fica os recursos da aplicação, que são imagens em diversas resoluções da aplicação; leiautes alternativos para visualizações horizontais/verticais e leiautes para outros dispositivos; e arquivos com valores padrão e traduções para outras línguas. Hoje em dia ficou mais complicado desenvolver porque temos que pensar que a aplicação pode ser vista em diversos tamanhos de tela, em diversas resoluções e em diversas línguas, e o desenvolvedor tem que se preocupar com tudo isso.

A pasta ‘java’ é onde ficará todo o código da aplicação. Esse código é dividido em vários arquivos, cada um com uma classe. Cada classe tem uma responsabilidade. Assim, fica mais fácil de achar o que se procura com cada coisa em seu lugar. Observe na Figura 37 que temos um arquivo ‘MainActivity’ na pasta ‘br.gov.pe.ead.alomundo’. **De agora em diante, chamaremos os arquivos de código Java de Classe e as pastas de pacote.** A classe ‘MainActivity’ é a primeira classe a ser lida em nosso projeto, o início de tudo.

Na pasta ‘manifests’ ficam arquivos de texto escritos em uma linguagem chamada XML, que organiza a informação. Este arquivo, ele não é uma classe, possui informações essenciais sobre a

aplicação como, por exemplo, a versão mínima do sistema operacional com que a aplicação pode ser executada e o nome do pacote que serve de identificador único para diferenciar de outras aplicações.

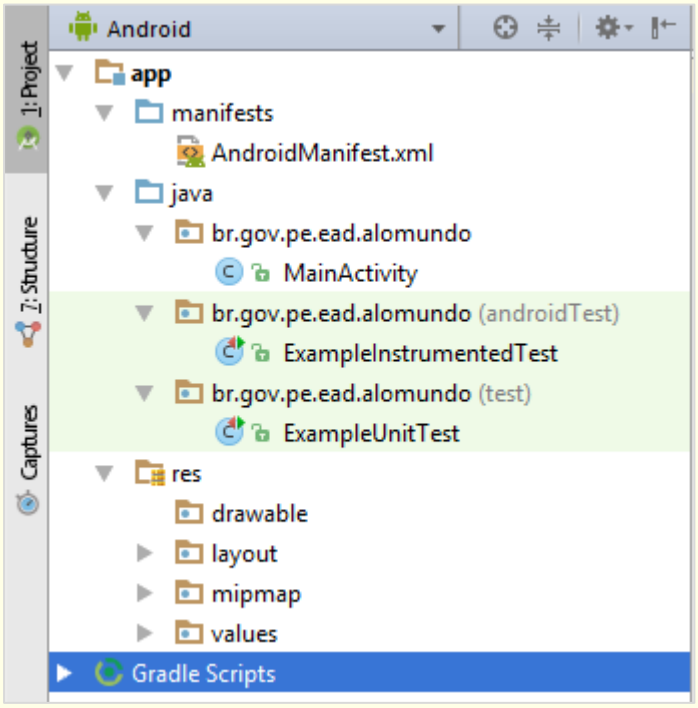


Figura 37 – Estrutura de um projeto no Android Studio.
Fonte: Autor.
Descrição: Estrutura de um projeto no Android Studio, exibindo as pastas básicas do projeto: manifests, java e res.

### 2.2.5 Criando uma máquina virtual

Nesse contexto, uma máquina virtual é um smartphone. Ele facilita o desenvolvimento evitando que você faça o código, empacote como uma aplicação finalizada e carregue para um smartphone real. Muito trabalho, já que testamos diversas vezes. Uma máquina virtual ainda ajuda na procura de erros, já que a IDE também auxilia a monitorar o que se passa durante a execução da aplicação.

Agora que temos um projeto, que está vazio, mas é um projeto que pode ser rodado, vamos criar uma máquina virtual para rodá-lo.

Abra o Gerenciador de AVD indo no menu Tools > Android > AVD Manager. Quando a janela do AVD Manager abrir, pressione 'Create virtual device...'.

AVD é o acrônimo de Android Virtual Device, que, em uma tradução livre, significa Dispositivo Virtual Android.

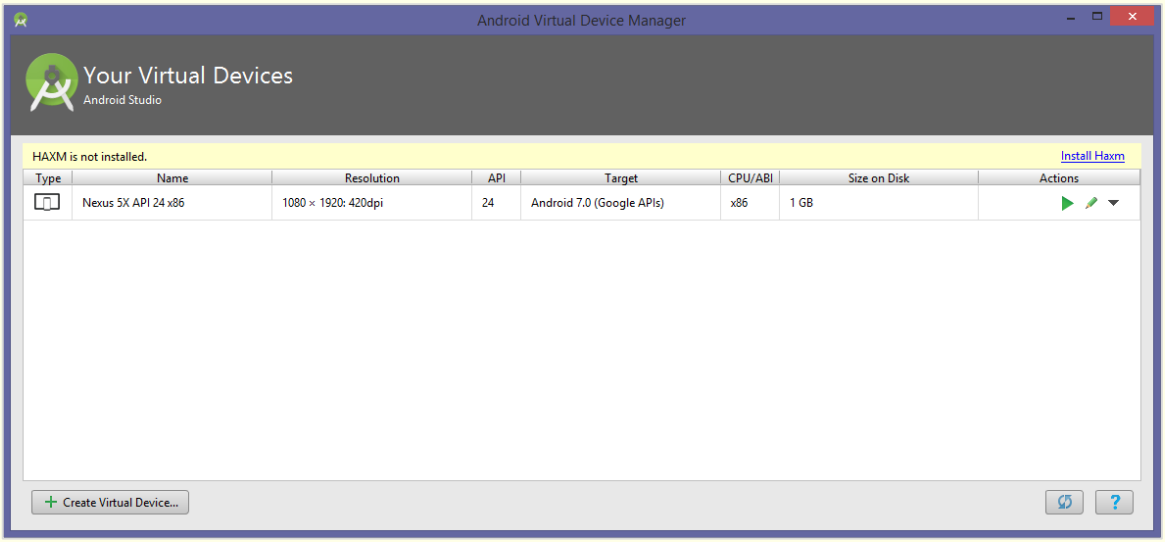


Figura 38 – Gerenciador de dispositivos virtuais, o AVD Manager.
Fonte: Autor.
Descrição: Janela do AVD Manager, o gerenciador de dispositivos virtuais do Android Studio.

Observe na Figura 39 que nesta tela escolhemos o *hardware*. Mais à esquerda temos a categoria do dispositivo. Ao centro, a lista de modelos prontos para alguns dispositivos e à direita a visualização de algumas informações como o tamanho do dispositivo selecionado. Os modelos disponíveis são dos smartphones da Google, mas alguns fabricantes disponibilizam modelos de suas linhas na internet. Vamos selecionar o 'Nexus One' por ter uma resolução menor e, com isso, gastar menos recursos do computador.

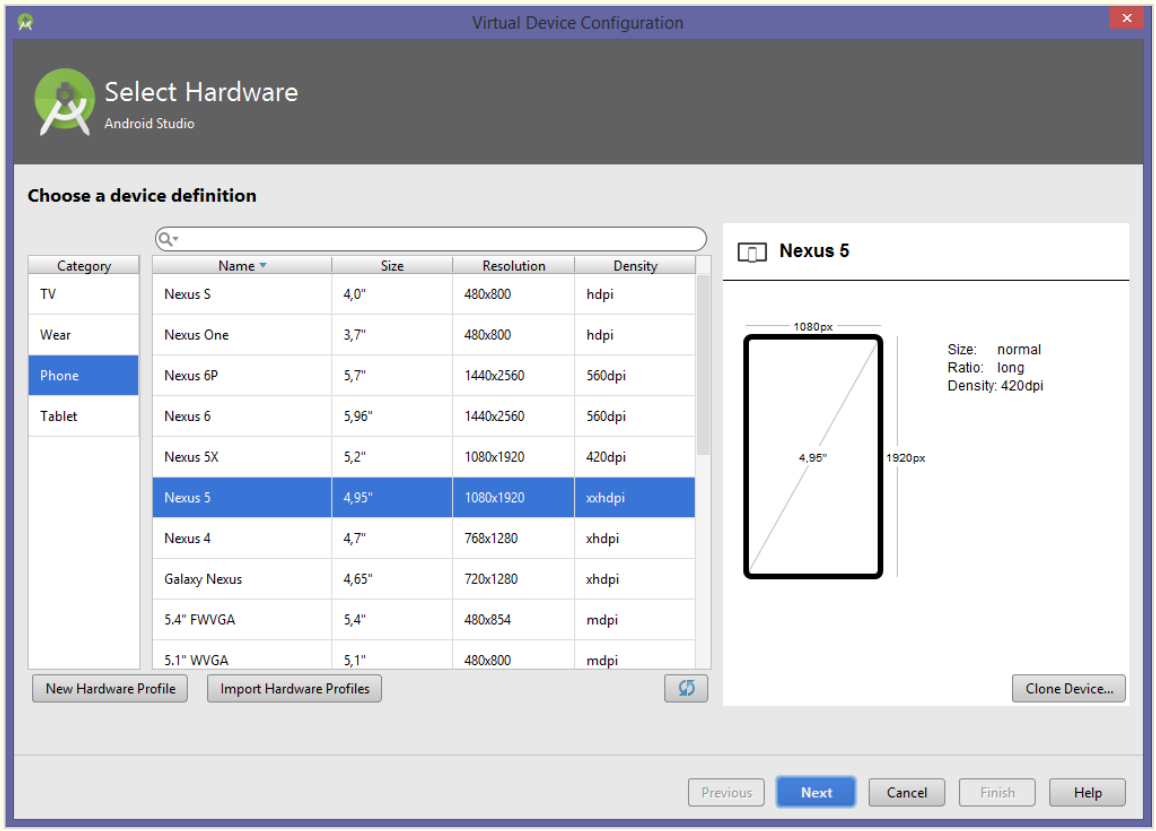


Figura 39 – Janela de opções para imagens de dispositivos.
Fonte: Autor.
Descrição: Nesta janela você encontra opções de tipo de dispositivo à esquerda da imagem, imagens de dispositivos ao centro e uma representação do dispositivo à direita.

Agora que temos um aparelho virtual, vamos colocar uma versão do sistema operacional Android. Como existem muitas, o Android Studio disponibiliza a versão mais recente. Caso você queira uma versão mais antiga, deve baixar. Como sempre, pode demorar muito, dependendo da qualidade de sua conexão com a internet. Vamos utilizar a versão mais recente, a Nougat.

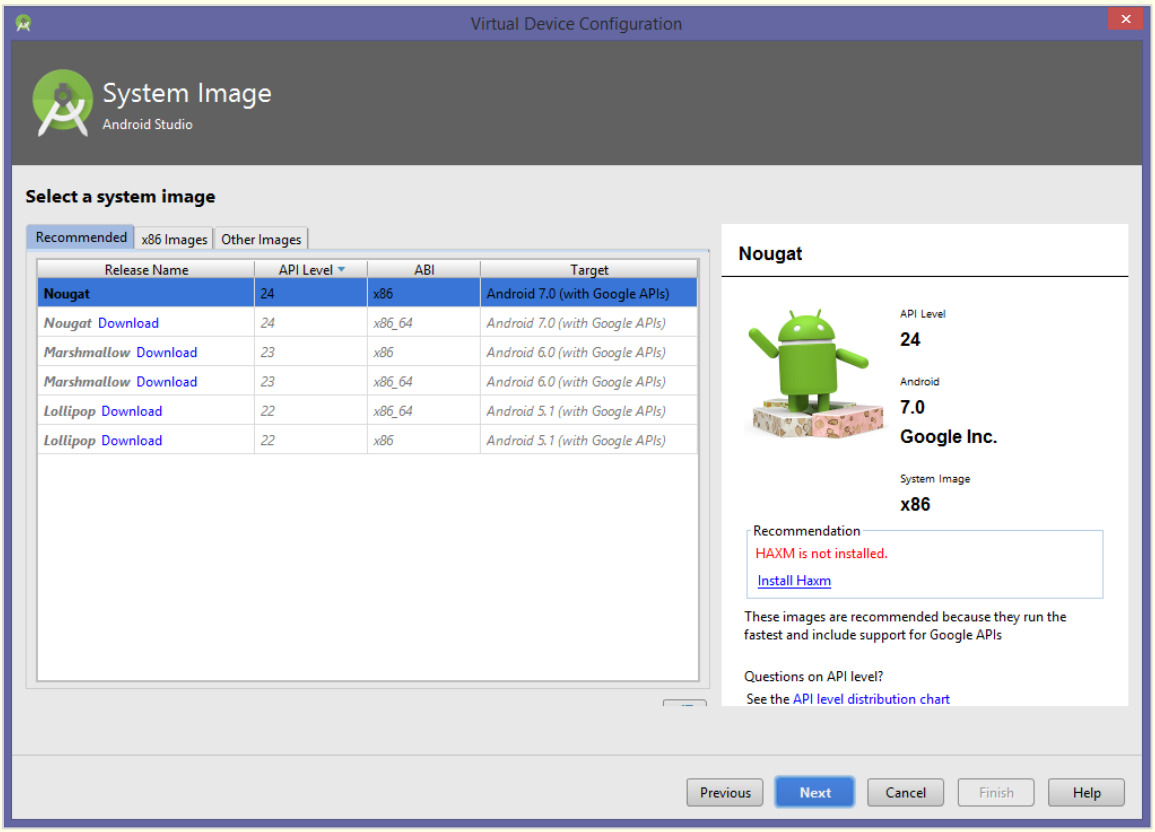


Figura 40 – Janela de seleção para a versão do sistema operacional Android.
Fonte: Autor.
Descrição: Janela de seleção para a versão do sistema operacional Android, que executará na máquina virtual.

Por fim, é exibida uma tela com o resumo das escolhas e campos para solicitar o nome e A orientação inicial do dispositivo (vertical ou horizontal). Vamos deixar como está. Pressione 'Finish' para concluir a criação do dispositivo virtual.

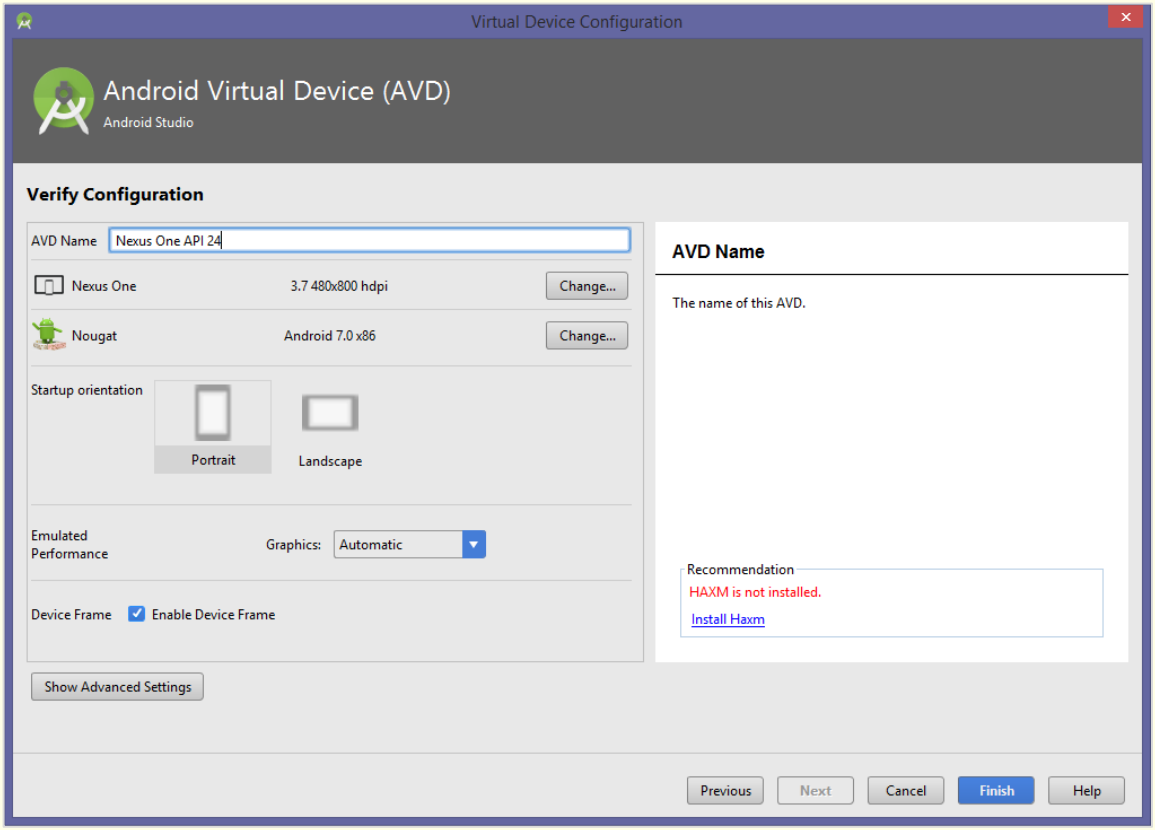


Figura 41 – Tela de resumo das opções selecionadas.
Fonte: Autor.
Descrição: A imagem mostra a tela de resumo das opções selecionadas para a geração do AVD, com um campo para nomeação da configuração.

Observe na Figura 42 que temos duas máquinas virtuais no AVD Manager. Uma já veio pronta para uso no Android Studio e a outra nós criamos agora.

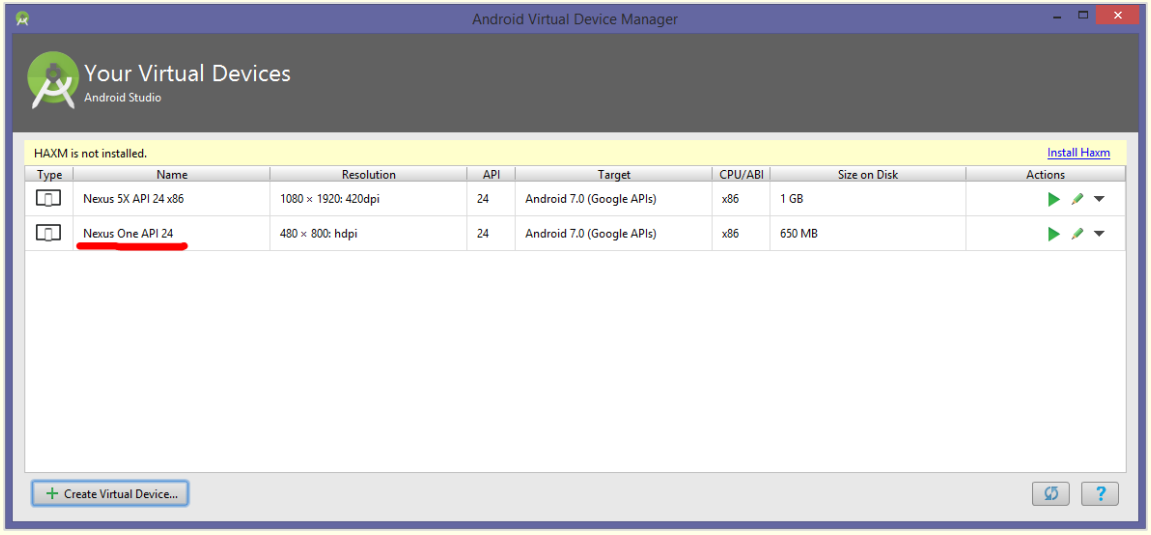


Figura 42 – AVD Manager exibindo duas máquinas virtuais. Em destaque, a que criamos.
Fonte: Autor.
Descrição: AVD Manager exibindo duas máquinas virtuais. Em destaque, a que criamos.

Para executar uma máquina virtual basta clicar no símbolo de ‘play’ em verde. Observe na Figura 42 que temos dois símbolos de ‘play’, uma para cada máquina virtual.

### 2.2.6 Executando uma aplicação na máquina virtual

Vamos testar tudo. No Android Studio clique no botão com o símbolo de ‘play’ em verde. Observe que ele está circulado em vermelho na Figura 43. Irá aparecer a janela para selecionar em qual das máquinas virtuais será executada a aplicação. No exemplo está selecionado o Nexus One, veja a seta vermelha na Figura 43. Depois, pressione ‘OK’.

Pressione ‘Next’ para qualquer janela que aparecer. São janelas sobre a memória utilizada para o dispositivo e download de algum componente que esteja faltando e que seja necessário.

Aguarde por um tempo. A inicialização de máquinas virtuais demora um pouco. Se tudo der certo, aparecerá a máquina virtual com a aplicação ativa, figura 46. **Leia a observação adiante.**

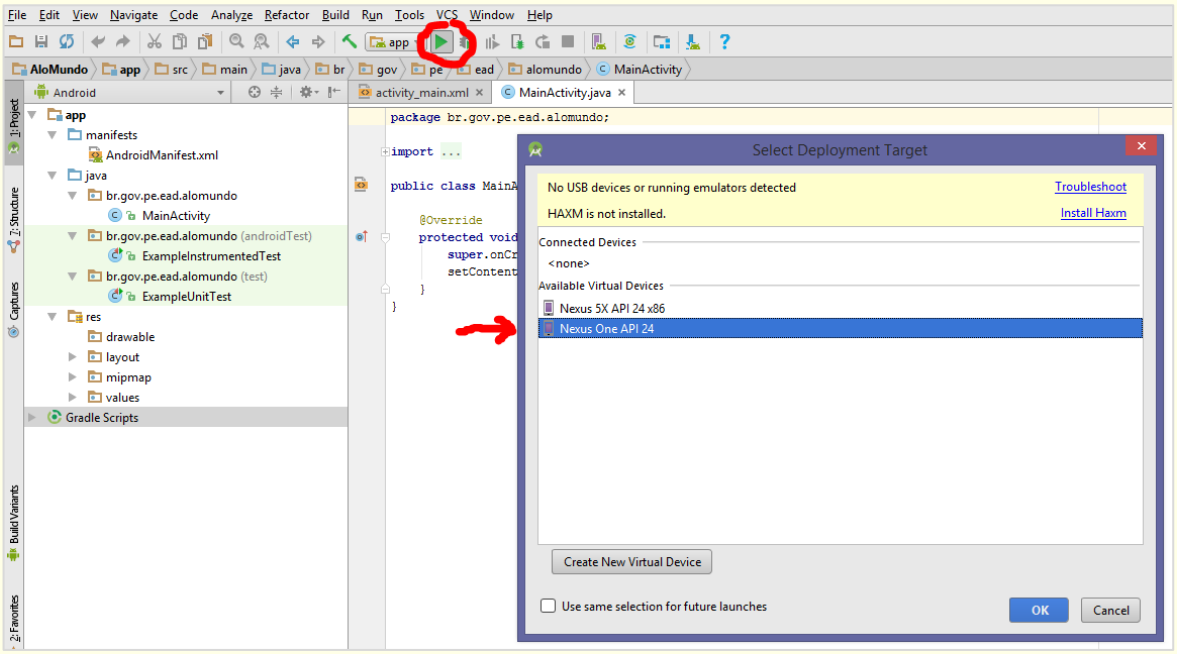


Figura 43 – Execução de teste do aplicativo em uma máquina virtual.
Fonte: Autor.
Descrição: Destaque circular em vermelho no símbolo de 'play' no Android Studio e seta na seleção da máquina virtual recém-criada.

Observação!

Em alguns computadores modernos, quando se executa a máquina virtual, aparece uma mensagem como na Figura 44, e a máquina virtual não é executada. Isto acontece porque a virtualização é desabilitada no computador. A solução é entrar no menu de BIOS do computador e habilitar a opção de virtualização. No meu notebook foi necessário, observem a Figura 45 com uma foto do menu da minha BIOS e a modificação que fiz, 'enabled'.

O link abaixo tem uma explicação em vídeo do processo.

www.youtube.com/watch?v=af6di_zDzAs

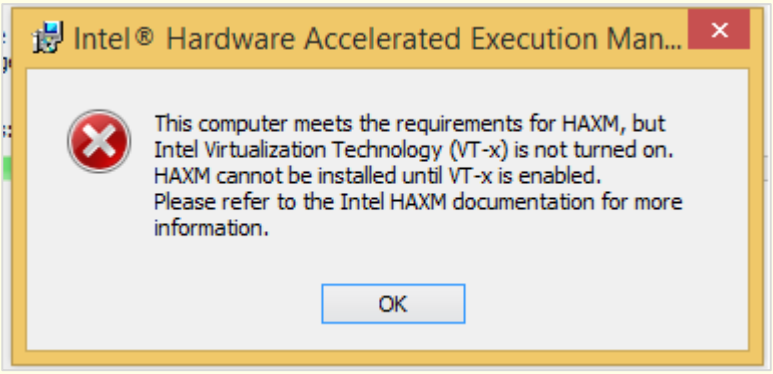


Figura 44 – Mensagem solicitando a habilitação da tecnologia de virtualização da Intel.
Fonte: Autor.
Descrição: Mensagem solicitando a habilitação da tecnologia de virtualização da Intel.

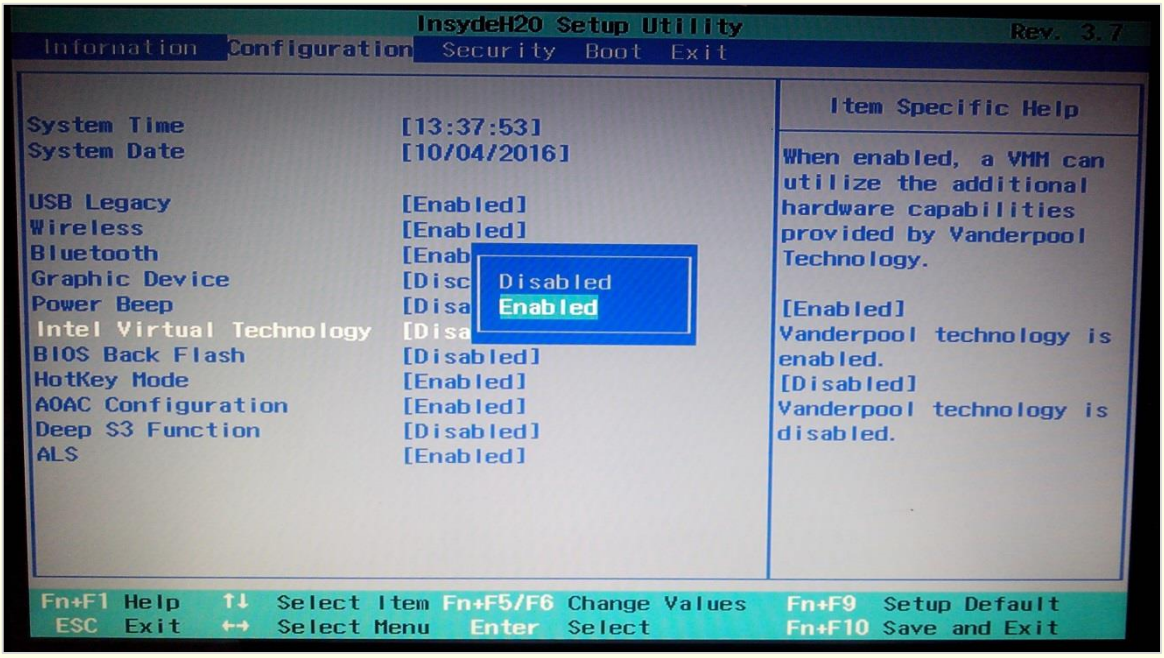


Figura 45 – Menu da BIOS com a opção ‘Intel Virtual Tecnhnology’ selecionada e a opção ‘Enabled’ destacada.
Fonte: Autor.
Descrição: Fotografia do menu de BIOS com a opção de virtualização selecionada e a opção para habilitá-la destacada.

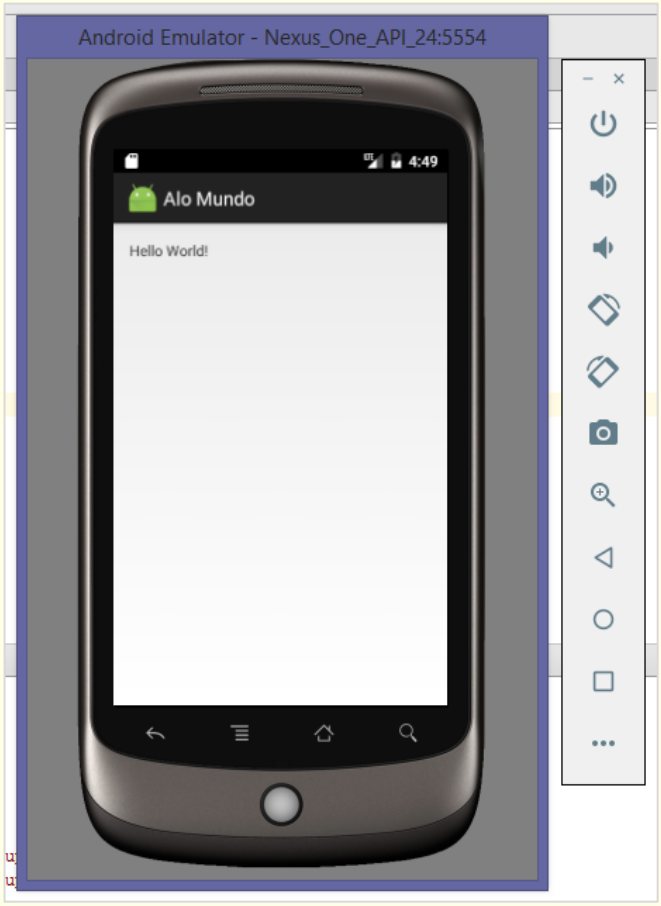


Figura 46 – Janela com a máquina virtual e a aplicação que mostra a mensagem 'Hello World!'.
Fonte: Autor.
Descrição: Janela com a máquina virtual rodando com a aplicação. A aplicação exibe apenas uma mensagem com o texto 'Hello World! ".

Você pode fazer vários projetos de aplicações diferentes e utilizar a mesma máquina virtual. Não precisa fazer uma máquina virtual por aplicação.

### 2.2.7 AndroidManifest.xml

O arquivo AndroidManifest.xml é um arquivo de texto escrito em XML, onde ficam guardadas todas as configurações necessárias para executar a aplicação. Observe na Figura 47 um exemplo deste arquivo para a aplicação 'Alo Mundo' que fizemos anteriormente. Neste caso, o Android Studio fez para a gente.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.gov.pe.ead.alomundo">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Alo Mundo"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Figura 47 – AndroidManifest.xml
Fonte: Autor.
Descrição: Conteúdo do arquivo AndroidManifest.xml da aplicação 'Alo Mundo'.

O pacote principal do projeto deve ser declarado na tag <manifest> utilizando-se o atributo <package>.

Uma aplicação Android é um conjunto de activities e todas elas devem ser declaradas no AndroidManifest.xml, caso contrário não será possível utilizá-las. A declaração é feita na tag <activity>.

Um projeto pode conter nenhuma ou várias activities. Uma delas deve ser o ponto de partida da aplicação e é declarada na tag com as ações android.intent.cation.MAIN e a categoria andooid.intent.category.LAUNCHER dentro da tag.

Para utilizar funcionalidades dos aparelhos, como GPS, SMS, ligações, etc. você precisa declarar esta intenção no AndoridManifest.xml. Por exemplo:

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

Esta linha pede permissão para acessar o sdCard.

Você pode aprender Android sem conhecer sobre XML, mas conhecendo fica mais fácil. Este curso não aborda XML, mas existem diversos cursos gratuitos sobre o assunto. Uma pesquisa rápida mostrará diversas alternativas. Amplie seus conhecimentos e aprenda XML.

O importante é você saber o que é o AndroidManifest.xml e para que serve. Nesta disciplina não haverá necessidade de alterarmos manualmente este arquivo. Na maior parte das vezes o Android Studio escreverá o que precisa por nós.

### 2.2.8 Telas em XML

Você pode desenvolver as telas da aplicação escrevendo código em Java ou em XML. O Android Studio tem uma ferramenta de desenho que ajuda a desenhar o leiaute das telas, escrevendo o código XML por você. Mas, quando estivermos programando, será necessário buscar os elementos na tela para manipulação.

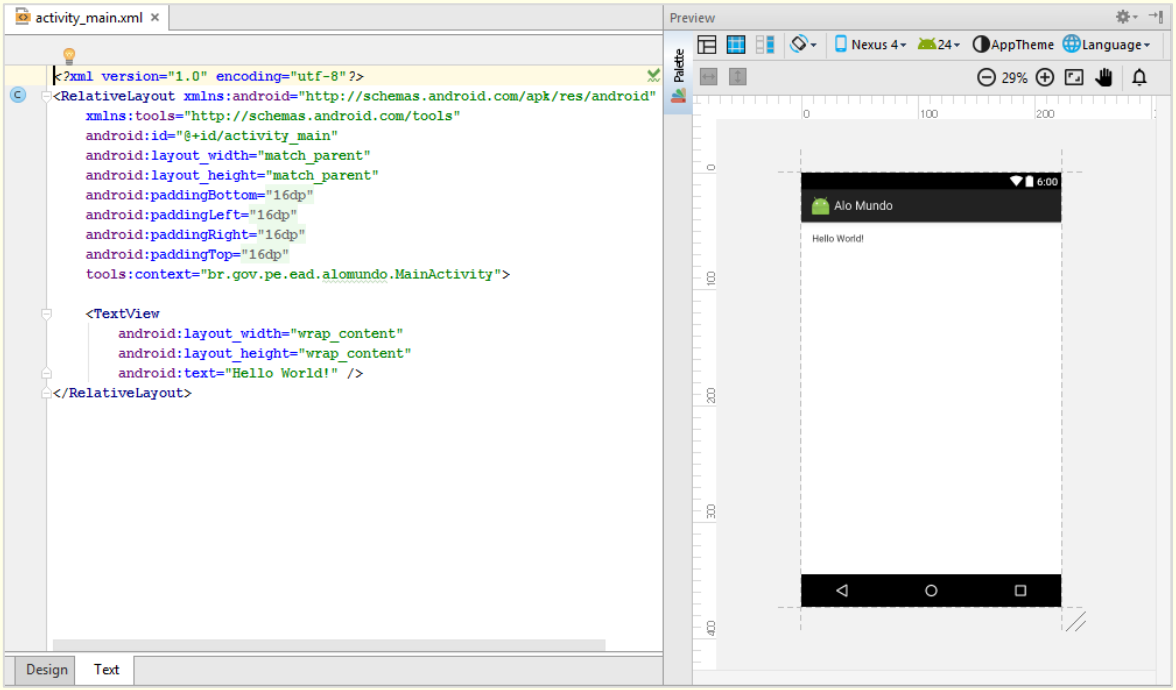


Figura 48 – Código XML da tela da aplicação 'Alo Mundo'.
Fonte: Autor.
Descrição: Código XML à esquerda e visualização da tela à direita, equivalente ao código.

### 2.2.9 Texto, cores, valores e dimensões

Temos um arquivo XML para cada um destes itens. Isto significa que, em uma aplicação bem escrita, os valores correspondentes a cores, texto, valores e dimensões são colocados nestes arquivos. Fica mais fácil de entender o porquê através do texto exibido na aplicação. Todo o texto da aplicação é colocado em um arquivo. Assim, quando formos traduzir nossa aplicação para o inglês, por exemplo, não teremos de catar no código o texto para alterar. Basta escrever outro arquivo de texto na língua que desejarmos.

Estes arquivos ficam em app > res > values.

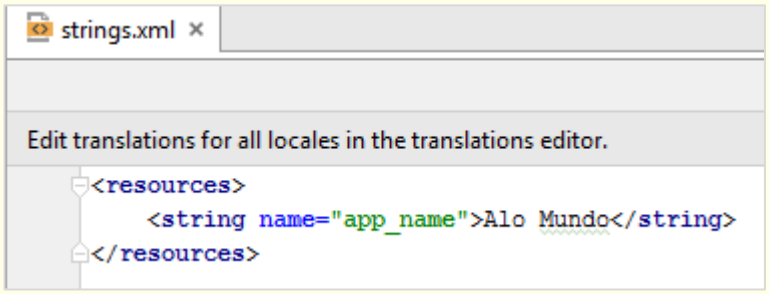


Figura 49 – Arquivo strings.xml.
Fonte: Autor.
Descrição: Arquivo XML para guardar todo o texto de nossa aplicação.

## 2.3 Exercício

Agora que você já criou seu primeiro projeto, o 'Alo Mundo!', crie um projeto novo para a próxima competência. O exercício é importante para fixar o conteúdo exposto. Crie o projeto 'EAD Pernambuco'.

Observe que o Android Studio trabalha com um projeto de cada vez. Isto é uma crítica de muitos desenvolvedores. Apesar do Android Studio ser a ferramenta oficial da Google para desenvolvimento Android, ela ainda está em evolução e se acredita que seus desenvolvedores escutarão as críticas dos usuários e modificarão a ferramenta para acessar vários projetos ao mesmo tempo, assim como o Eclipse.

## 3.Competência 03 | Formatar uma Activity para o Desenvolvimento, Utilizando Operadores Matemáticos

Na competência anterior vimos como instalar um ambiente de desenvolvimento e teste para aplicações Android utilizando o Android Studio, a IDE oficial da Google. Também criamos o primeiro projeto e aprendemos como testar os projetos que desenvolveremos.

Agora vamos aprender alguns conceitos básicos de programação em Java para realizarmos algo mais interessante. Vamos utilizar o projeto do Exercício da competência anterior. Observe que ele tem dois arquivos principais. A MainActivity.java, que é o código que será executado quando a aplicação for executada; e activity_main.xml, que é a tela de visualização correspondente àquele código. Então, temos uma associação de uma tela escrita em XML e um código escrito em Java.

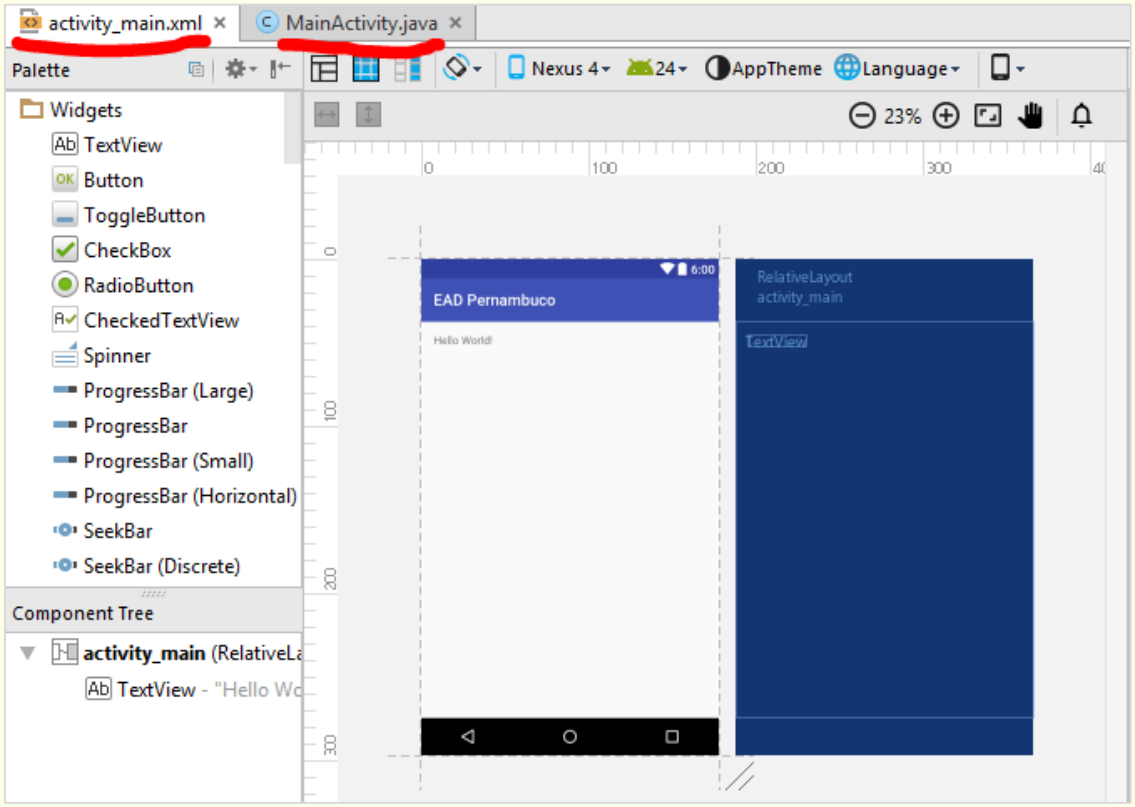


Figura 50 – Exibição da tela desenhada pela activity_main.xml.
Fonte: Autor.
Descrição: Exibição da tela desenhada pela activity_main.xml. Sublinhado em vermelho o nome dos arquivos activity_main.xml e MainActivity.java.

### 3.1 Variáveis

Toda aplicação utiliza dados. Dados são fragmentos de informação, como, por exemplo, ‘42’, ‘#aprendendo’, ‘Estudar muito amanhã :)’, ‘35.50’, etc. Estes valores podem significar idades, nomes, tweets, endereços, vídeos, qualquer coisa. Programas transformam estas informações em um formato que você reconhece como Facebook, Twiiter, Instagram, etc.

Entenda que um programa faz algo útil com os dados. O Facebook, por exemplo, tem um algoritmo (algoritmo são os passos para a resolução de um problema, como um programa) chamado de Algoritmo de Afinidade. Ele utiliza todas as postagens, curtidas, visualizações, cliques, qualquer coisa que se faça quando se está logado no Facebook e dá uma pontuação para seus gostos, seus interesses, ou seja, suas afinidades. Com isso, ele pode prever se você vai gostar de algo antes de ver, com uma probabilidade grande de acerto. É como prever o futuro, em um certo aspecto.

Para que possamos transformar dados em algo útil, como o Algoritmo de Afinidade, temos que primeiro guardar os dados para utilizarmos. Em programação, a estrutura que guarda um valor para ser utilizado depois é a **variável**.

### 3.2 Tipo de dados primitivos

Para utilizarmos uma variável em Java precisamos de um nome e de um tipo. Um **tipo** limita o que cabe na variável. É assim, para não ter desperdício de memória. Desse modo, a quantidade de memória para guardar um número é menor do que para guardar um texto.

Um tipo também define as operações possíveis para aquele valor. Por exemplo, para números podemos somar, subtrair, entre outras operações, mas, para texto, podemos concatenar, que é colar um pedaço de texto em outro, mas não podemos somar ou subtrair.

Em Java, para definir qualquer variável se coloca primeiro o nome do tipo e depois o nome da variável, que chamamos de identificador. Você também pode ir além e colocar o valor inicial. Para utilizar, basta chamar o identificador, que o computador substituirá ele pelo valor. Veja os exemplos

em cada descrição de tipo a seguir.

O Java consegue distinguir letras maiúsculas e minúsculas. Dessa forma, os nomes de variável ‘valor’, ‘VALOR’, ‘Valor’ e ‘vAlOr’ são todas variáveis diferentes. Também não se pode começar com números. Veja mais algumas regras de nomenclatura.

1. Não pode ser uma palavra reservada (palavra-chave);

2. Não pode ser **true** nem **false** - literais que representam os tipos lógicos (booleanos);

3. Não pode ser **null** - literal que representa o tipo nulo;

4. Não pode conter espaços em brancos ou outros caracteres de formatação;

5. Deve ser a combinação de uma ou mais letras e dígitos UNICODE-16. Por exemplo, no alfabeto latino, teríamos:

   a) Letras de A Z;

   b) Letras de a a z;

   c) Sublinha _;

   d) Cifrão $;

   e) Dígitos de 0 a 9.

### 3.2.1 Tipo de dado lógico

São também chamados de dados **booleanos**. Representam estados binários, tais como: verdadeiro/falso, certo/errado, ligado/desligado, aberto/fechado, sim/não, etc. Ele é o menor tipo e se armazena seu valor em um bit. Observe o exemplo da Figura 51. Não dá para ver o resultado, porque o computador cria a variável e não faz nada com ela, mas o objetivo era apenas criar uma variável booleana e aplicar um valor.

```
public class MainActivity extends AppCompatActivity {

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

        boolean gravidez; // Declaração de variável booleana
        gravidez = true;  // Utilização da variável

        boolean gemeos = false; // Declaração e inicialização
    }
}
```

Figura 51 – Declaração e utilização de variáveis booleanas.
Fonte: Autor.
Descrição: A imagem mostra o código para a declaração de uma variável booleana nomeada 'gravidez', depois a utilização desta variável com o valor verdadeiro (true) e outra linha com a declaração e utilização juntas.

### 3.2.2 Tipo de dado numérico inteiro

Existem vários tipos de dados numéricos: byte, short, int e long. Veremos apenas os mais utilizados. O tipo **int** define valores inteiros. Vai de -2.147.483.648 até 2.147.483.647. O tipo **short** é um inteiro que guarda uma quantidade menor de números e serve para economia de memória. O tipo **long** é o dobro do int e guarda valores bem maiores, mas também consome o dobro de espaço. O tipo byte trabalha com bytes e não abordaremos neste caderno.

Podemos representar números negativos colocando o '-' antes do número.

```
public class MainActivity extends AppCompatActivity {

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

        int idade; // Declaração de variável
        idade = 20;  // Utilização da variável

        int temperatura = -2; // Declaração e inicialização
    }
}
```

Figura 52 – Declaração e utilização de variáveis representando números inteiros.
Fonte: Autor.
Descrição: A imagem mostra o código para a declaração de uma variável inteira nomeada 'idade', depois a utilização desta variável com o valor 20 e outra linha com a declaração e utilização juntas.

### 3.2.3 Tipo de dado numérico real

Números reais são números fracionários. Em computação, números fracionários utilizam ponto ao invés da vírgula, como aprendemos na escola. Em Java temos os tipos **float** e **double** para armazenar números reais. Por padrão um número com ponto é do tipo double, mas se quisermos utilizar menos memória, podemos utilizar o float ao invés do double.

```
public class MainActivity extends AppCompatActivity {

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

        double salario; // Declaração de variável
        salario = 945.8;  // Utilização da variável

        double imposto = -94.58; // Declaração e inicialização
    }
}
```

Figura 53 – Declaração e utilização de variáveis representando números fracionários.
Fonte: Autor.
Descrição: A imagem mostra o código para a declaração de uma variável fracionária nomeada 'salario', depois a utilização desta variável com o valor 945,8 e outra linha com a declaração e utilização juntas.

### 3.2.4 Tipo de dado de caractere

Este tipo guarda um número que representa um caractere. Existe uma tabela que liga o valor ao caractere chamada UTF-16. Podemos armazenar apenas um caractere neste tipo de variável, sua denominação é **char**. Caracteres em Java são descritos entre aspas simples. Veja o exemplo da Figura 54. No exemplo, '\u0041' é o código de um caractere. Qual será?

```
public class MainActivity extends AppCompatActivity {

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

        char letra; // Declaração de variável
        letra = 'a';  // Utilização da variável

        char caractere = '\u0041'; // Declaração e inicialização
    }
}
```

Figura 54 – Declaração e utilização de variáveis do tipo caracteres.
Fonte: Autor.
Descrição: A imagem mostra o código para a declaração de uma variável caractere nomeada 'letra', depois a utilização desta variável com o caractere 'a' e outra linha com a declaração e utilização juntas, utilizando a nomenclatura unicode.


Conheça os caracteres da tabela UTF-16 neste link:
www.fileformat.info/info/charset/UTF-16/list.htm

### 3.2.5 Tipo de dado complexo: String

Uma sequência de caracteres não dá para armazenar em um tipo char, então como vamos armazenar um texto?

Em Java uma sequência de caracteres, como seu nome, ou o texto de uma redação, é armazenado

em um tipo complexo chamado String. Sua variável ganha o nome de referência e colocamos as sequências entre aspas duplas. Ela funciona como uma variável primitiva, mas não é. Tem muito mais poder. Observe o exemplo da Figura 55.

```
public class MainActivity extends AppCompatActivity {

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

        String nome; // Declaração de variável
        nome = "Fulano de Tal";  // Utilização da variável

        String texto = "Texto..."; // Declaração e inicialização
    }
}
```

Figura 55 – Declaração de uma referência a partir do tipo String, ou seja, da classe String.
Fonte: Autor.
Descrição: A imagem mostra o código para a declaração de uma referência String nomeada 'nome', depois a utilização desta variável com o valor 'Fulano de Tal' e outra linha com a declaração e utilização juntas.

Tipos complexos em Java são chamados de **classe** e a variável de **referência**. Classes, guardam variáveis, chamadas de **atributos**, e funções, que são chamadas de **métodos**. Atributos é o que a classe sabe, métodos é o que a classe sabe fazer. Classes definem como **objetos** são criados. Objetos são as unidades básicas de computação. Os objetos é que realizam o trabalho.

Aprender os conceitos de classes e objetos são importantes para desenvolver melhor em Java. Amplie seu conhecimento www.youtube.com/watch?v=Am6fdgMGY0k

## 3.3 Operadores

Operadores são símbolos que representam atribuições, cálculo e ordem de dados. É o que utilizamos para transformar dados. Quando somamos 2 + 2, o + é um operador de cálculo, e o 2 + 2

é uma **expressão**. Expressões podem ser bem grandes e podem conter valores e variáveis, que guardam valores. Podemos ter uma expressão x + 2 - y. Observe que expressões são construídas com operadores.

Temos três tipos de operadores:

- Unários: que só precisa de um operando para realizar a operação;
- Binários: que precisa de dois operandos para realizar a operação. A maioria deles;
- Ternário: que precisa de três operandos para realizar a operação.

Como disse, as expressões podem ser bem grandes e o computador não calcula tudo de uma vez. Ele faz uma operação de cada vez e algumas são realizadas primeiro que outras. A esta ordem damos o nome de precedência de operadores. A Tabela 2 mostra os operadores mais utilizados e a ordem de solução é de cima para baixo.

| TIPO DO OPERADOR | LISTA DE OPERADORES |
|---|---|
| Sufixais | expr++ expr- - |
| Prefixais | ++expr - -expr ! |
| Multiplicativos | * / % |
| Aditivos | + - |
| Comparativos | < > <= >= instanceof |
| Igualdade | == != |
| Lógico E | && |
| Lógico OU | \|\| |
| Atribuição | = += -= *= /= %= &= ^= \|= <<= >>= >>>= |

Tabela 2 – Classificação de tipos de operadores baseado na ordem de precedência.
Fonte: Autor.
Descrição: Tabela que classifica o tipo do operador com as operações relacionadas. A precedência é dada de cima para baixo na tabela.

### 3.3.1 Operadores Sufixais e Prefixais

Operadores sufixais e prefixais são um atalho para deixar uma expressão menor e diminuir as linhas de programação. Tanto os sufixais quanto os prefixais são utilizados em tipos de dados que podem utilizar operadores matemáticos. Eles adicionam 1 ao valor da variável ou subtraem 1 deste valor. Assim, ++x é igual à expressão x = x + 1 e --x é igual à expressão x = x – 1.

A diferença entre eles está na posição do sinal. Nos operadores sufixais a expressão vem primeiro e depois o sinal. Por exemplo, x++ ou x--. Ele diz que primeiro utilizamos o valor da variável e depois somamos ou subtraímos.

Exemplo: Vamos dizer que o x é igual a 10. E temos a expressão (1 - x++) + (1 + - -x). Coloquei os parênteses para ficar fácil de você entender. Quando a expressão vem primeiro (sufixal), utiliza-se o valor da variável e depois se opera, somando ou subtraindo 1. Quando o operador vem primeiro (prefixal), opera-se e depois se utiliza o valor na expressão. Veja a Tabela 3 para entender o passo a passo de resolução.

| ORDEM | EXPRESSÃO (1 – X++) + (1 + --X) | EXPLICAÇÃO | VALOR FINAL DE X |
|---|---|---|---|
| 1 | (1 – 10++) + (1 + --x) | Ao encontrar a expressão x++, primeiro usa o valor da variável x, depois soma 1 a ela. Assim, (1 – x) é realizado, resultando em 9 e depois a x é somado o valor 1. | x = 11 |
| 2 | (9) + (1 + --11) | Com o valor de x agora sendo 11, temos a operação --x. Neste caso, como o operador vem antes, primeiro subtraímos 1 do valor de x, voltando a ser 10. Depois realizamos a operação (1 + x), resultando em 11. | x = 10 |
| 3 | (9) + (11) | Por fim, fazemos a soma (9 + 11), resultando em 20. O valor de x termina com 10. | x = 10 |

Tabela 3 – Exemplo de execução de operações pelo computador.
Fonte: Autor.
Descrição: Tabela com a execução passo a passo de cada uma das operações na ordem de execução pelo computador.

Exercício: Observe que todo o processo muda se trocarmos a posição dos sinais. Com x valendo 10, faça a expressão (1 - ++x) + (1 + - -x). Tente fazer primeiro e depois veja a resposta na Tabela 4.

| ORDEM | EXPRESSÃO (1 – ++X) + (1 + X--) | EXPLICAÇÃO | VALOR FINAL DE X |
|---|---|---|---|
| 1 | (1 – ++11) + (1 + x--) | Ao encontrar a expressão ++x, primeiro somamos 1 a variável, x passa a valer 11. Agora fazemos a operação (1 – 11), resultando em 10. | x = 11 |
| 2 | (10) + (1 + 11--) | Agora, com x valendo 11, primeiro fazendo a expressão (1 + x), resultando em 12. Depois fazemos o x--, subtraindo 1 de x, resultando em 10. | x = 10 |
| 3 | (10) + (12) | Por fim, fazemos a soma (10 + 12), resultando em 22. O valor de x termina com 10. | x = 10 |

Tabela 4 – Exemplo de execução de operações pelo computador.
Fonte: Autor.
Descrição: Tabela com a execução passo a passo de cada uma das operações na ordem de execução pelo computador.

Temos ainda a operação prefixal ‘not’ que é representado por uma exclamação antes de uma variável booleana. O ‘not’ é ‘não’ em inglês, ai fica ‘não verdadeiro’ igual a falso e ‘não falso’ igual a verdadeiro. Caso o valor da variável booleana seja verdadeiro, ele troca para falso e vice-versa. Por exemplo, !x.

### 3.3.2 Operadores matemáticos

Os operadores matemáticos são os de soma, subtração, multiplicação, divisão e o resto. As quatro primeiras são óbvias, o resto é o resto de uma divisão. Por exemplo 21 % 2 tem como resultado 1, porque 21 dividido por 2 dá 10 e sobra 1. É ótimo para se descobrir se um número é ímpar ou par.

Pela tabela de precedência, os operadores multiplicativos são realizados primeiro que os operadores aditivos. Assim, em uma expressão 2 + 3 * 2 – 3, primeiro é realizado o 3 * 2, que dá 6, depois 2 + 6, que dá 8, e por último 8 – 3, resultando 5.

### 3.3.3 Operadores comparativos

Em uma expressão, depois de ter realizado as operações matemáticas, o computador passa para as operações comparativas. As operações comparativas fazem testes que resultam em operações booleanas (verdadeiro/falto). As operações comparativas são igual (==), diferente (!=), menor que (<), maior que (>), menor que ou igual (<=) e maior que ou igual (>=). Por exemplo, 3 + 2 == 2 + 3? Verdade, porque 3 + 2 é 5, que é igual a 2 + 3 que é 5.

Atente que o teste de igualdade é realizado com ==, dois símbolos de igualdade juntos. O de diferente é != que significa 'não igual', outro modo de dizer diferente.

### 3.3.4 Operadores lógicos

Depois de verificar os operadores comparativos chega a vez dos operadores lógicos, que fazem operações com valores booleanos e resultam, também, em um valor booleano. Os operadores são && para o 'e' lógico e || para o 'ou' lógico. Observe a Tabela 5 para os resultados das comparações.

| a | b | a && b | a \|\| b | !a |
|---|---|---|---|---|
| true | true | true | true | false |
| true | false | false | true | false |
| false | true | false | true | true |
| false | false | false | false | true |

Tabela 5 – Tabela verdade
Fonte: Autor.
Descrição: Tabela com as operações de "e", "ou" e "não" lógicos, relacionando dois valores 'a' e 'b' e seu resultado em uma das operações.

### 3.3.5 Operadores de atribuição

Igual é um operador de atribuição, por isto que o de comparação são dois símbolos de igualdade juntos. A atribuição relaciona um valor com uma variável. Então, quando queremos relacionar o valor 18 com a variável 'idade', fazendo x = 18.

Os outros operadores de atribuição também são atalhos para resumir comandos. Assim, quando queremos somar um valor ao que já existe em x, fazemos: x += valor; se queremos multiplicar o valor que tem em x com outro colocar: x *= valor. E assim por diante. Observe que associamos uma operação matemática com o símbolo de atribuição.

```
public class MainActivity extends AppCompatActivity {

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

        int numero = 3; //numero recebe o valor 3
        numero += 7;    //numero recebe 3+7. Ou seja, 10.
        numero -= 32;   //numero recebe o seu valor menos 32. Ou seja, -22.
        numero %= -3;   //numero recebe o resto da divisão entre seu valor e -3. Ou seja, -1.
        numero *= 6;    //numero recebe o seu valor vezes 6. Ou seja, -6.
        numero /= 2;    //numero recebe o seu valor dividido por 2. Ou seja, -3.

    }
}
```

Figura 56 – Exemplo de operações de atribuição.
Fonte: Autor.
Descrição: Sequências de operações de atribuição e seus resultados, passo a passo.
Exercício: Qual é o resultado para as operações a seguir?

```
int resposta = 8;

resposta *= 2;

resposta -= 10;

resposta += 5;

resposta %= 4;

resposta /= 2;
```

Resposta: 1

### 3.3.6 Operador de concatenação

Concatenação é a junção de uma string com outra. Assim, se concatenarmos “Fulano” com “de Tal” teremos “Fulanode Tal”. Observe que a junção não cria espaços, por isto que os nomes ficaram juntos. Só podemos concatenar Strings, não funcionando com o tipo char.

Para concatenar, também utilizamos o símbolo de soma. Pelo fato de ‘+’ terem dois sentidos, o de soma ser para números, e concatenação, ser para strings, dizemos que é um operador sobrecarregado.

```
public class MainActivity extends AppCompatActivity {

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

        String nome = "Fulano";
        String sobrenome = "de Tal";

        String texto = "Nome: " + nome + " " + sobrenome;
    }
}
```

Figura 57 – Exemplo de concatenação.
Fonte: Autor.
Descrição: Exemplo de concatenação com variáveis String.

## 3.4 Separadores

Os separadores são sinais que separam, ou sejam, indicam/modificam a ordem das operações que podem ou não ser diferentes da comum. A Tabela 6 contém os separadores em Java e quando eles são utilizados.

| SEPARADOR | DESCRIÇÃO |
|---|---|
| Ponto e vírgula ‘;’ | Separa comandos. |
| Parênteses ‘( )’ | Separam expressões alterando a precedência. Por exemplo 2 * 3 + 4, temos primeiro 2 * 3 e depois o resultado + 4. Se colocarmos 2 * (3 + 4), teremos primeiro o 3 + 4 e depois multiplicamos o resultado com 2. Se tivermos parênteses dentro de parênteses, primeiro será realizada a operação dos parênteses mais internos para os mais externos. |
| Colchetes ‘[ ]’ | Colchetes servem para identificar o índice de vetores. Veremos este assunto mais à frente. |
| Chaves ‘{ }’ | Separa blocos de programação. |

Tabela 6 – Descrição dos separadores em Java.
Fonte: Autor.
Descrição: Tabela com a descrição de cada separador em Java.

Exercício: Faça, linha a linha para cada tipo de operação, seguindo a ordem de precedência de Java, para a expressão: (2 + 3) * 5 > 3 * 5 & 3 * 4 < 4 + 9. Não olhe direto a resposta, procure tentar fazer antes e compare o resultado.

Resposta:

| Passo | Regra | Operação |
|---|---|---|
| 1 | () | (2 + 3) * 5 > 3 * 5 & 3 * 4 < 4 + 9 |
| 2 | * / % | 5 * 5 > 3 * 5 & 3 * 4 < 4 + 9 |
| 3 | + - | 25 > 15 & 12 < 4 + 9 |
| 4 | < <= > >= | 25 > 15 & 12 < 13 |
| 5 | && | true & true |
| 6 | \|\| | true |

## 3.5 Visualizando o resultado dos operadores

O primeiro projeto em Android baseado em um modelo possui uma caixa de texto com 'Hello World!' escrita. Vamos modificar este valor assim que o Android terminar de fazer o cálculo.

No arquivo **activity_main.xml** temos uma tela pronta, que é a tela do modelo. Nela tem um campo de texto, clique nele e, ao lado, aparecerão as propriedades daquele componente. Vamos colocar um nome, como se fosse um nome de variável. Coloque no campo ID a palavra 'campo'. Observe na Figura 58.

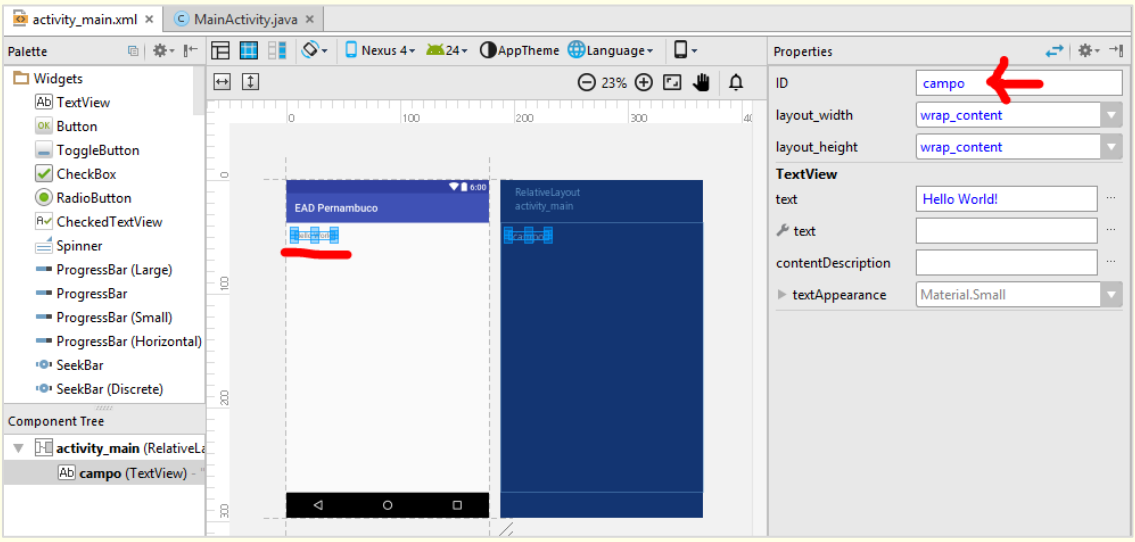


Figura 58 – Identificando um elemento na interface XML.
Fonte: Autor.
Descrição: Na interface de telas XML, no painel de propriedade, o campo ID identifica o elemento. Para o painel de propriedade relativo ao elemento aparecer, o elemento deve estar selecionado.

Agora vá para o arquivo **MainActivity.java** e acrescente o código da Figura 59. No começo das linhas você reconhecerá que é um dos exercícios propostos, as duas últimas linhas explicaremos logo mais.

```
public class MainActivity extends AppCompatActivity {

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

        int resposta = 8;
        resposta *= 2;
        resposta -= 10;
        resposta += 5;
        resposta %= 4;
        resposta /= 2;

        TextView c = (TextView) findViewById(R.id.campo);
        c.setText("Resposta: " + resposta);
    }
}
```

Figura 59 – Exibindo a resposta da expressão em um elemento.
Fonte: Autor.
Descrição: As duas últimas linhas do código recuperam o elemento na tela e configuram o texto de exibição através do método setText().

| | |
|---|---|
| TextView c | Cria uma variável com o identificador ‘c’ que armazena o tipo complexo de uma caixa de texto ‘TextView’. Neste ponto ainda não temos a TextView do leiaute. |
| (TextView) | Quando fazemos uma busca por um componente de leiaute, ele vem em outro formato. Este código faz um casting, que é uma mudança de formato para aquele que queremos. No caso, o TextView. |
| findViewById() | É um método que procura pra gente no leiaute o elemento que queremos. Será aquele de nome ‘campo’. |
| R.id.campo | Para que ele ache temos que dizer o que queremos. A classe R.id guarda todos os identificadores de elementos, entre outras coisas. Então, pedimos aquele elemento que tem o id igual a ‘campo’. |
| c.setText(“Resposta: ” + resposta); | Uma vez que guardamos o objeto TextView na referência ‘c’ podemos utilizá-la. Com o método setText() modificamos o texto dele e colocamos a resposta do nosso exercício. |

Tabela 7 – Tabela com a explicação das linhas de código utilizadas na Figura 59.
Fonte: Autor.
Descrição: Tabela com a explicação das linhas de código utilizadas na Figura 59.

A Figura 60 mostra o resultado na máquina virtual.

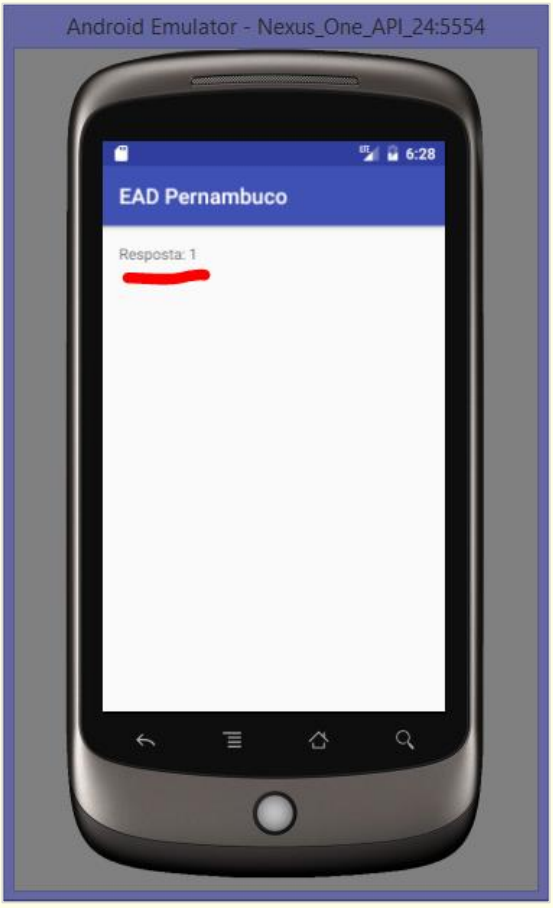


Figura 60 – Exibição do resultado.
Fonte: Autor.
Descrição: Exibição do resultado na máquina virtual, também conhecido como emulador.

## 3.6 Exercícios

Faça os outros exemplos utilizando esta forma de visualização de dados.

Lembre-se de que o método setText() só aceita String. Assim, você precisa concatenar com algum texto os números. Você pode concatenar com uma String vazia “”, também.

# 4.Competência 04 | Formatar uma Activity para o Desenvolvimento de Estruturas de Controle Condicional

Na competência anterior foi o básico do básico. Não poderíamos trabalhar com dados sem saber formatar estes dados, mas é essencial que saibamos deles para podermos trabalhar. Lembre-se! Qualquer aplicação transformará dados em algo útil. Que dados você utilizará e para fazer o quê?

Nesta competência começaremos a trabalhar com as estruturas de controle de fluxo condicional. É quando prevemos um ou mais caminhos que a programação possa ter. Por exemplo, caso aquele dado seja maior que 5 faça isto, senão, faça aquilo. É basicamente isto. Utilizaremos muito os operadores booleanos e suas operações. Se ficou alguma dúvida sobre este assunto, volte e estude novamente.

## 4.1 Estrutura if

A estrutura if tem o formato abaixo.

```
if (expressão) {
      código
}
```

Ele utiliza a palavra-chave ‘if’ que significa ‘se’ em inglês. Temos uma expressão em que o resultado tem que ser verdadeiro ou falso para fazer sentido. Se a expressão resultar em true, o código dentro do bloco é executado. No bloco podemos ter de nenhuma a várias linhas de código. Veja o exemplo na Figura 61.

```
public class MainActivity extends AppCompatActivity {

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

        TextView c = (TextView) findViewById(R.id.campo);

        int idade = 17;
        if (idade >= 16) {
            c.setText("Pode votar");
        }

    }
}
```

Figura 61 – Exemplo da estrutura if.
Fonte: Autor.
Descrição: Foi criada uma variável idade com o valor 17 e se verifica se idade é maior ou igual a 16. Se for, exibe um texto.

No exemplo da Figura 61, se o valor de idade for maior ou igual a 16, será escrito em 'campo' o texto 'Pode votar'. Mas se for menor de 16 não aparecerá nada. Vamos contornar isto logo.

### 4.2 Estrutura if … else

Else em inglês significa 'senão'. No caso, se a expressão resultar em false, o bloco de código depois do else será executado.

```
if (expressão) {
      código
} else {
      código
}
```

Observe o exemplo anterior com esta estrutura.

```
public class MainActivity extends AppCompatActivity {

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

        TextView c = (TextView) findViewById(R.id.campo);

        int idade = 17;
        if (idade >= 16) {
            c.setText("Pode votar");
        } else {
            c.setText("Você não tem idade");
        }

    }
}
```

Figura 62 – Exemplo estrutura if...else.
Fonte: Autor.
Descrição: Mesmo exemplo da Figura 62, acrescentado o 'else'. Caso o teste seja falso, é exibida outra mensagem.

No exemplo da Figura 62, se o valor de idade for menor que 16, aparecerá a mensagem 'Você não tem idade' para votar.

### 4.3 Estrutura if … else aninhadas

Podemos colocar instruções if...else umas dentro das outras para fazermos vários dependentes.

```
if (expressão) {
      if (expressão) {
            código
      } else {
            código
      }
} else {
      if (expressão) {
            código
      } else {
            código
      }
}
```

Observe o exemplo da Figura 63.

```
public class MainActivity extends AppCompatActivity {

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

        TextView c = (TextView) findViewById(R.id.campo);

        String cor = "azul";
        if (cor == "vermelho") {
            c.setTextColor(Color.RED);
        } else {
            if (cor == "azul") {
                c.setTextColor(Color.BLUE);
            } else {
                if (cor == "amarelo") {
                    c.setTextColor(Color.YELLOW);
                } else {
                    c.setText("Não conheço esta cor");
                    c.setTextColor(Color.BLACK);
                }
            }
        }
    }
}
```

Figura 63 – Exemplo de estruturas aninhadas.
Fonte: Autor.
Descrição: Vários ifs aninhados.

No exemplo da Figura 63, primeiro temos uma String com o valor ‘azul’, então verificamos se ela tem o valor ‘vermelho’, como não tem, o bloco else dele é executado. Neste bloco temos uma nova verificação, de que se a cor é ‘azul’, como é o texto do TextView é pintado de azul. Caso não fosse, entrava no bloco else desta estrutura. Nela tem outra verificação, se cor vale ‘amarelo’, senão for ele escreveria ‘Não conheço esta cor’ e pintaria o texto de preto.

### 4.4 A estrutura switch

Observe que ficou bem confusa estas instruções (estruturas) if...else aninhadas. Às vezes, podemos utilizar a estrutura switch no lugar, porque ela funciona de modo semelhante. Veja a estrutura switch. Ela é composta da palavra-chave switch e recebe um valor. Este valor é comparado com o valor1. Caso seja igual, o processamento das linhas segue a partir daqui. Caso o valor seja igual a valor2, então, o processamento seguirá deste ponto em diante. A palavra-chave break serve para interromper o processamento do bloco de código. O processamento do bloco segue a partir de default se todos as comparações forem falsas.

```
switch (valor) {
      case valor1:
            código
            break;
      case valor2:
            código
            break;
      default:
            código
}
```

Veja na Figura 64, o exemplo anterior utilizando a estrutura switch. Bem melhor de ser lido.

```
public class MainActivity extends AppCompatActivity {

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

        TextView c = (TextView) findViewById(R.id.campo);

        String cor = "azul";
        c.setText(cor);
        switch (cor) {
            case "vermelho":
                c.setTextColor(Color.RED);
                break;
            case "azul":
                c.setTextColor(Color.BLUE);
                break;
            case "amarelo":
                c.setTextColor(Color.YELLOW);
                break;
            default:
                c.setTextColor(Color.BLACK);
                c.setText("Não conheço esta cor");
        }
    }
}
```

Figura 64 – Exemplo estrutura switch.
Fonte: Autor.
Descrição: Foi criada uma variável de texto e no switch é testada a correspondência. O fluxo de execução segue a partir do ponto onde é detectada a correspondência.

A estrutura switch possui apenas um bloco e marca o lugar onde o processamento deve começar. Ela chega a ser bem diferente do if...else. Utilizamos o break para interromper o fluxo de execução do bloco, mas ele é opcional, assim como o default, podemos desejar seguir com o fluxo. Veja o exemplo da Figura 65. Neste exemplo a partir de um valor os cálculos são realizados. Utilizamos no exemplo o cálculo de fatorial.

Caso não lembre como se realiza o cálculo de fatorial, assista a este vídeo antes de procurar entender o algoritmo. www.youtube.com/watch?v=KR4xUGGgYZk

```
public class MainActivity extends AppCompatActivity {

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

        TextView c = (TextView) findViewById(R.id.campo);

        int valor = 3;
        switch (valor) {
            case 4:
                valor *= 3;
            case 3:
                valor *= 2;
            case 2:
                valor *= 1;
        }

        c.setText("Resposta: " + valor);
    }
}
```

Figura 65 – Exemplo da estrutura switch sem os breaks.
Fonte: Autor.
Descrição: É criada uma variável inteira e, dependendo da correspondência de valor, é realizada uma sequência de operações.

## 4.5 Exercício

Agora vamos fazer um exercício para entender melhor.

Observe o código da Figura 66. As duas últimas linhas do código pegam o calendário do sistema. A partir dele podemos pegar a data e a hora. A última linha coloca na TextView a hora a partir de um número inteiro. Observe na máquina virtual que a hora é 8:23 da noite, o equivalente é 20:23. Calendar.HOUR_OF_DAY retorna o valor de 20.

```java
package br.gov.pe.ead.eadpernambuco;

import ...

public class MainActivity extends AppCompatActivity {

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

        TextView t = (TextView) findViewById(R.id.campo);

        Calendar hora = Calendar.getInstance();
        t.setText("Hora: " + hora.get(Calendar.HOUR_OF_DAY) + "hs");
    }
}
```

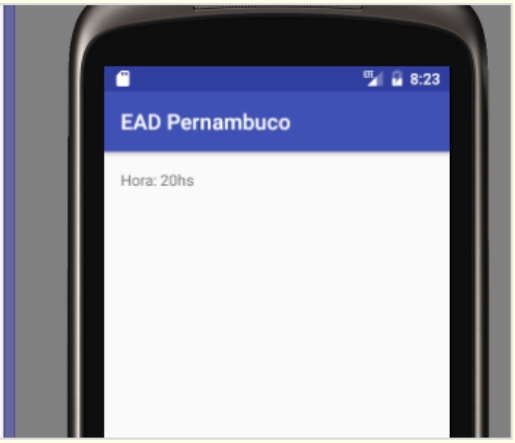


Figura 66 – Exibição de horas passadas no dia.
Fonte: Autor.
Descrição: Código que utiliza um objeto do tipo Calendar para recuperar a quantidade de horas passadas no dia.

Com base neste código, se esta hora for até 12 escreva ‘Bom dia’, se for entre 12 e 18 escreva ‘Boa tarde’ se passar de 18 escreva ‘Boa noite’.

Resposta:

```java
package br.gov.pe.ead.eadpernambuco;

import ...

public class MainActivity extends AppCompatActivity {

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

        TextView t = (TextView) findViewById(R.id.campo);

        Calendar hora = Calendar.getInstance();
        int h = hora.get(Calendar.HOUR_OF_DAY);
        if (h <= 12) {
            t.setText("Bom dia.");
        } else {
            if (h <= 18) {
                t.setText("Boa tarde.");
            } else {
                t.setText("Boa noite.");
            }
        }
    }
}
```

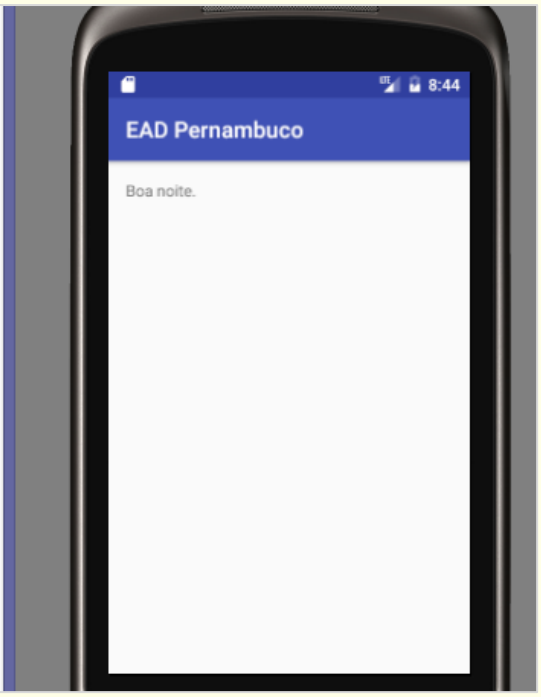


Figura 67 – Exemplo de utilização de if...else aninhados.
Fonte: Autor.
Descrição: Utilização da recuperação de horas passadas do dia para mostrar mensagem de ‘bom dia’, ‘boa tarde’ ou ‘boa noite.

## 5. Competência 05 | Formatar uma Activity para o Desenvolvimento de Estruturas de Repetição

Na competência passada conseguimos fazer algum tratamento em alguns dados, como a hora do sistema. Estruturas de decisão são a base para o processamento de informação. É onde a magia acontece. No entanto, as vezes temos que repetir várias vezes um comando ou até um bloco de código. Você pode até pensar, “Tudo bem! Basta eu copiar e colar”. Mas isso é muito problemático, principalmente na hora de alterar. Tem também o fato de que algumas vezes não sabemos quantas vezes é para repetir, até o sistema estar rodando. Precisaremos de uma estrutura que possa repetir um ou mais comandos para a gente, as estruturas ou laços de repetição.

São três as estruturas: for, while e do...while. A diferença do for para as estruturas while está em saber a quantidade de repetições. Quando se sabe, é mais apropriado utilizar o for, quando não se sabe, utilizamos uma das estruturas while. Já a diferença entre estas estruturas está no teste inicial. Vamos então conhecê-las melhor.

### 5.1 Estrutura for

A estrutura for precisa determinar um início, um teste de parada e o passo de incremento para se alcançar o teste. É mais indicada quando se sabe quantas vezes se deve repetir o bloco de comandos.

```
for (inicialização; teste de parada; passo de incremento) {
     código
}
```

Quando o comando é lido, a variável de inicialização é definida e um teste é feito para saber se o bloco deve ser executado. Se for verdadeiro o teste, o bloco é executado. Ao final da execução é realizado um passo de incremento, um novo teste é feito e, se for verdadeiro, é executado o bloco. Novamente ocorre um passo de incremento e o teste se repete. Acontece assim, sucessivas vezes, até o teste retornar falso e a repetição ser encerrada. Observe o exemplo da Figura 68.

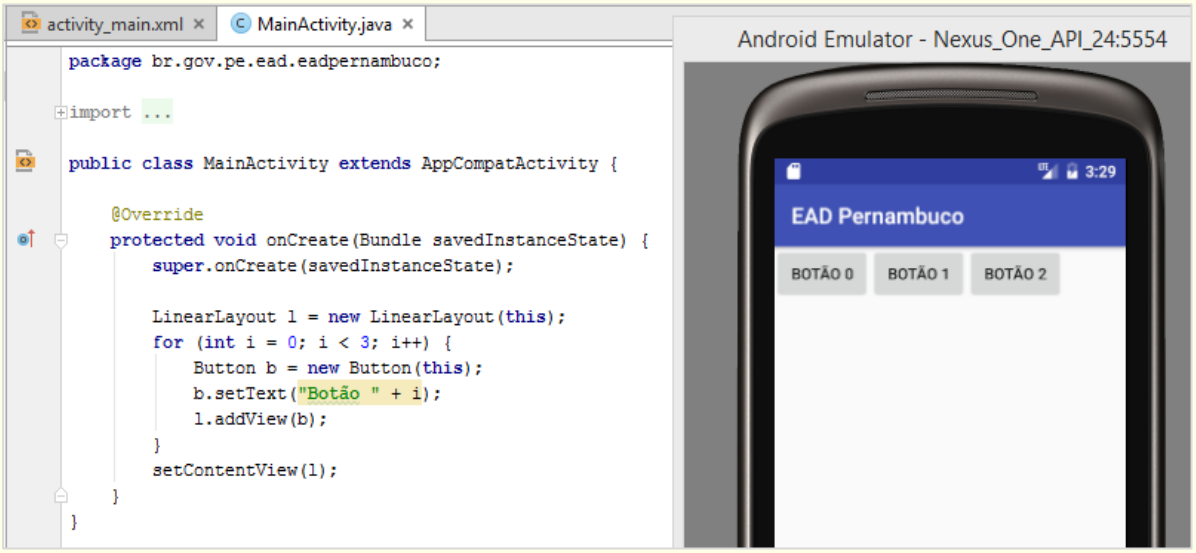


Figura 68 – Exemplo de estrutura for.
Fonte: Autor.
Descrição: Criamos um leiaute e utilizamos uma estrutura for para colocar uma quantidade de botões nela.

No exemplo da Figura 67, criamos um LinearLayout. É uma parte da tela para colocar elementos.

Depois, utilizamos uma estrutura for para criar três botões e colocá-los no layout 'l'. O laço é iniciado com 0 (i = 0), depois é testado se i < 3 (0 < 3), então é criado um botão e colocado na tela. O laço se reinicia. O valor de i é acrescentado 1 (i++) e testado novamente (1 < 3). Enquanto for verdadeiro o teste (i < 3), o código será repetido. Note que o teste é verdadeiro para 0, 1, 2. O 3 não é verdadeiro, porque 3 não é menor que 3. Observe que utilizamos a variável 'i' normalmente.

Ao final, configuramos este leiaute como o principal para esta activity. A outra em XML será ignorada. Esta é uma forma de construir uma tela sem o XML. Não vamos focar nisso. O foco aqui é o laço de repetição.

Exercício: Na área de colocar os passos podemos colocar qualquer expressão que altere a variável de controle. Faça o seguinte. Mude o laço para 'for (int i = 0; i < 10; i += 2)' e observe o número nos botões. A resposta está na Figura 69.

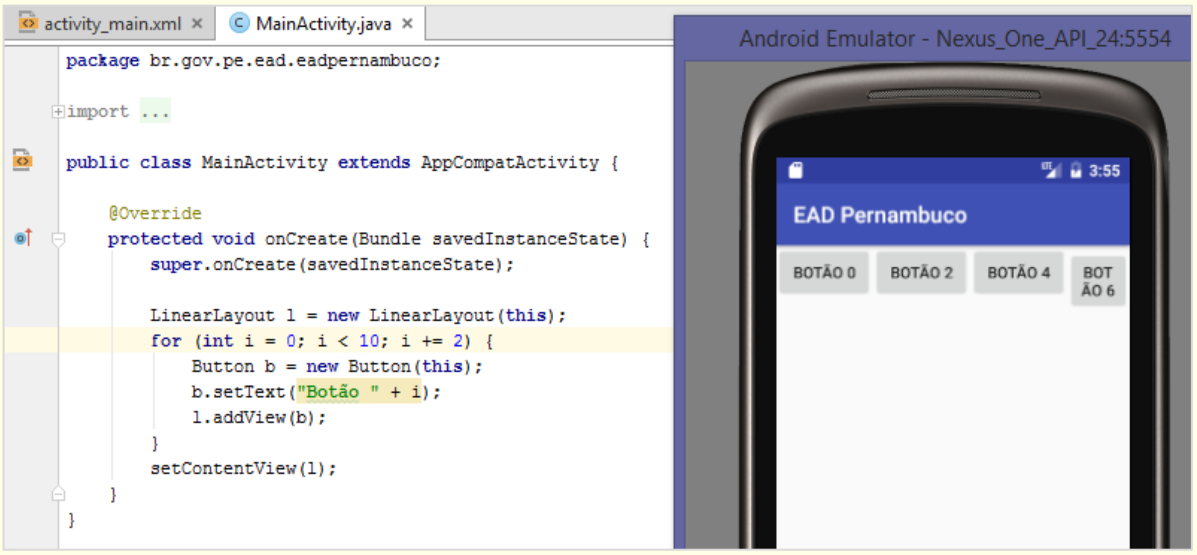


Figura 69 – Exemplo de laço for com contagem diferente.
Fonte: Autor.
Descrição: O mesmo exemplo da Figura 67, com contagem de 2 em 2.

Exercício: Também podemos fazer um laço que conte de trás para frente. Altere o laço para 'for (int i = 10; i > 0; i -= 3)'. Observe a ordem dos números dos botões. A resposta está na Figura 70.

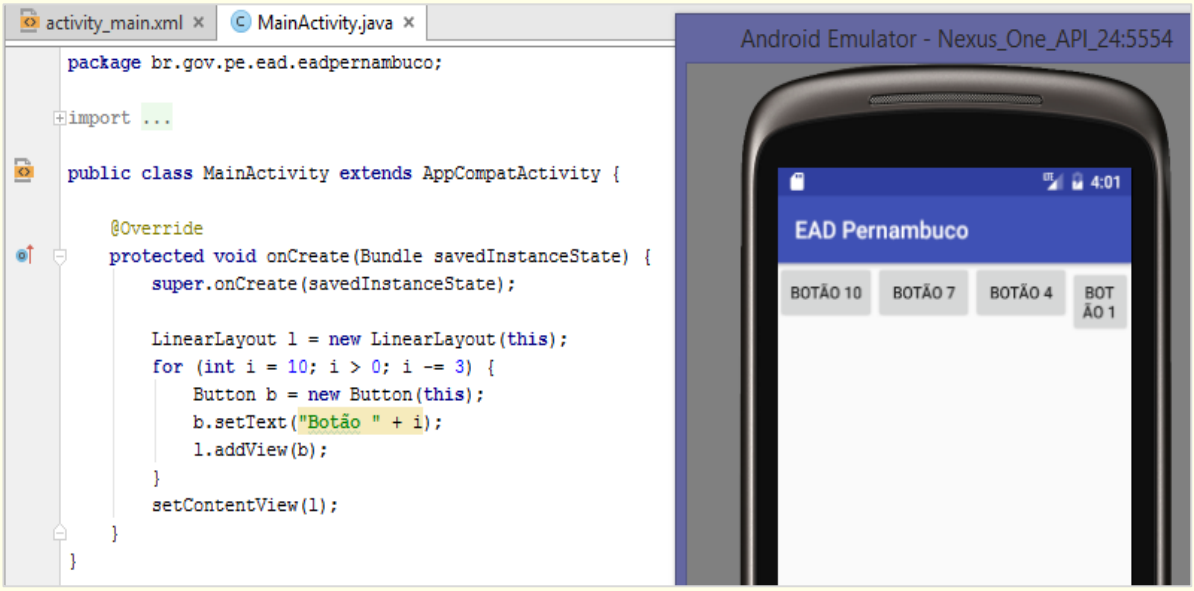


Figura 70 – Exemplo de laço for com contagem decrescente.
Fonte: Autor.
Descrição: Mesmo exemplo da Figura 67 com contagem decrescente e passo 3.

## 5.2 Estrutura while

A estrutura ou laço while é utilizada quando não sabemos a quantidade de repetição. Sua estrutura

é mostrada abaixo.

```
while (teste) {
      código
}
```

Observe que temos apenas o teste. É claro que podemos transformar isso para se parecer mais com um laço for, mas por que em vez de transformar não utilizamos um laço for?

No exemplo da Figura 71, criamos um objeto Random, que sorteia números. Pedimos que ele sorteie um número de 0 até 4 usando o método nextInt(5). Até aí não temos a menor ideia do número que ele sorteará. Pode ser 0, 1, 2, 3 e 4. O while executará o código se o número sorteado for diferente de 0. Então, pode ser que ele sorteie logo de primeira o 0 e não mostre nada. Se for diferente de 0, ele cria um botão com o número e sorteia outro número e faz o mesmo teste. A repetição ocorrerá até sortear um 0. No exemplo da Figura 71, o primeiro sorteio foi 2, o segundo foi 2 e o terceiro foi 0. A quantidade de botões deve mudar a cada teste da aplicação.

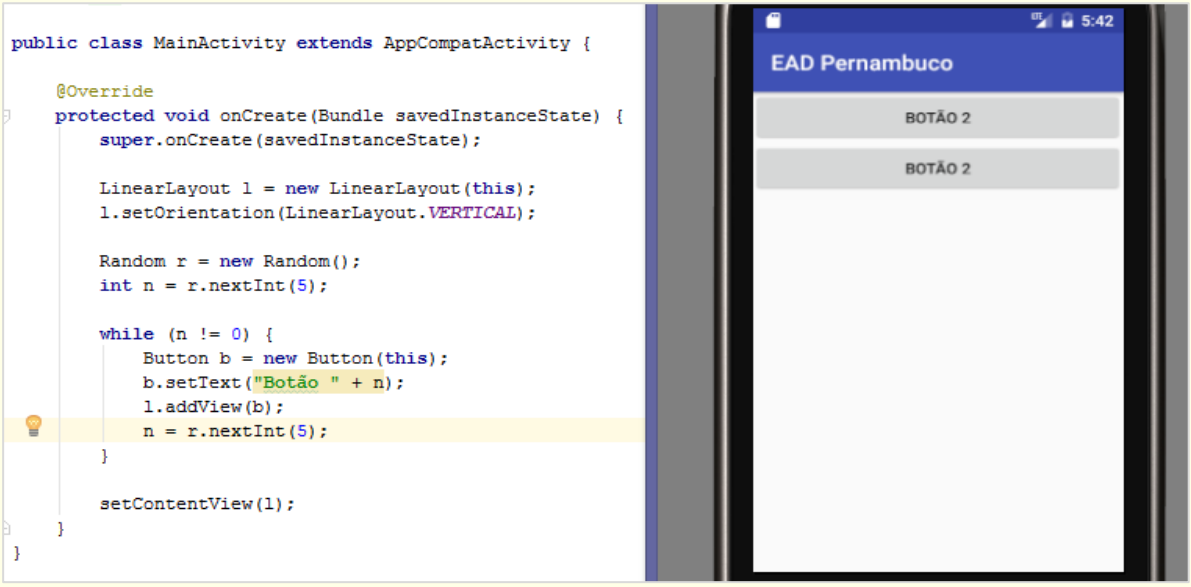


Figura 71 – Exemplo de estrutura while.
Fonte: Autor.
Descrição: Neste exemplo é sorteado um número de 0 até 4, se o número for diferente de 0 é colocado na tela e se sorteia outro. A repetição acontece até ser sorteado um número 0. No exemplo foram sorteados 2, 2 e 0.

### 5.3 Estrutura do...while

Observe que no exemplo da Figura 71 tivemos que sortear um número antes do teste e depois dentro do teste. Mas não teríamos necessidade de repetir código, se o teste do while fosse feito no final. O do...while é uma estrutura while onde o teste é no final. Observe sua estrutura.

```
do {
      código
} while (teste);
```

Assim, o código é executado ao menos uma vez e depois é feito um teste para as próximas. Veja o exemplo do código anterior utilizando do...while na Figura 72.

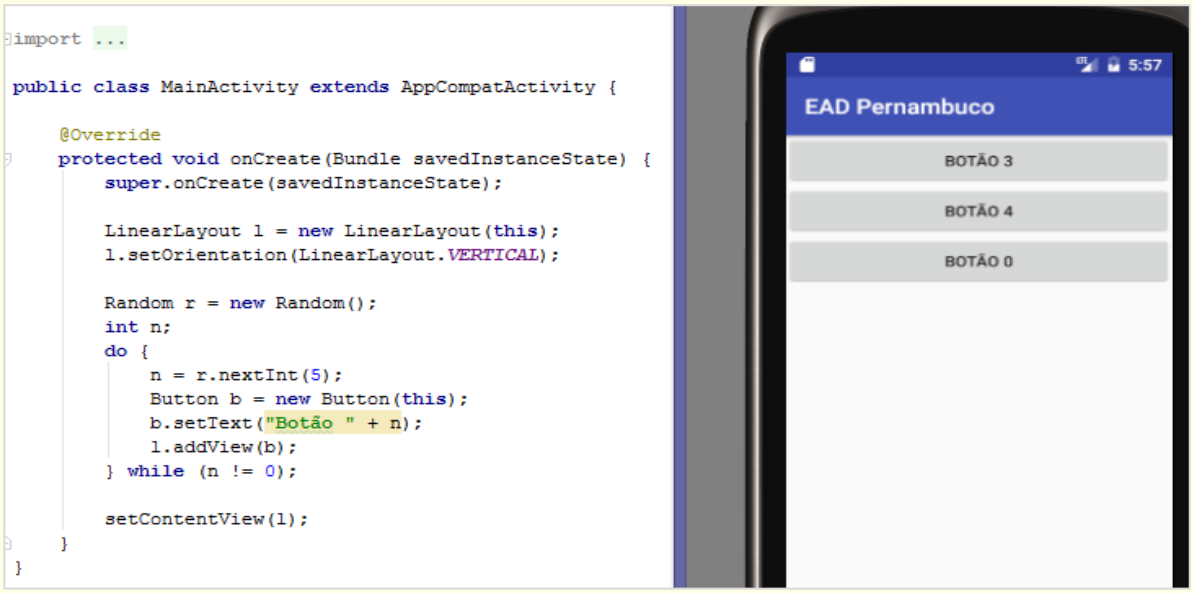


Figura 72 – Exemplo da estrutura do...while.
Fonte: Autor.
Descrição: Mesmo código do exemplo da Figura 70, só que utilizando a estrutura do...while. Os números sorteados foram 3, 4 e 0.

Observe que, como o teste ocorre depois de colocar o botão, foi sorteado um 0, seu botão foi construído e depois encerrado o laço. Observe, também, que só precisamos colocar uma linha para sortear.

### 5.4 Estrutura foreach

A estrutura foreach, “para cada” em inglês, é uma estrutura relativamente nova em Java. Ela utiliza uma lista de variáveis conhecida como array, que veremos em nossa próxima competência. Por isso, deixaremos para mostrar essa estrutura em um momento futuro.

## 6. Competência 06 | Formatar uma Activity para a Utilização de Listas e Arrays

Até agora vimos os componentes básicos. Existem bem mais conhecimento necessário para se fazer tudo que se pode, mas o básico se completa com este assunto: arrays. Então, vamos completar nossos estudos com esse assunto. Mas, lembre-se: existe muito mais conteúdo sobre programação e desenvolvimento Java. Não pare por aqui.

### 6.1 Vetores

Um vetor é uma estrutura de dados formada por um conjunto de elementos de mesmo tipo. Utilizamos vetores para simplificar o trabalho com muitas variáveis iguais. Os vetores são chamados arrays em programação.

Imagine que você queira guardar o nome de dez produtos em variáveis. Você teria que declarar nome1, nome2, nome3, nome4, etc. E se fossem 1000 nomes? Daria muito mais trabalho. Os vetores resolvem este problema.

O vetor pode ter uma ou mais dimensões. Uma dimensão determina o posicionamento do valor. Esta posição é chamada de índice. Assim, de acordo com a Figura 73, temos um vetor de 10 posições. O posicionamento dos vetores sempre começa por 0 em Java.

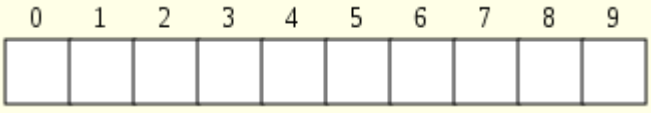


Figura 73 – Representação visual de um vetor (array).
Fonte: https://pt.wikibooks.org/wiki/Java/Vetores
Descrição: Representação visual de um vetor (array) com 10 espaços, numerados de 0 até 9.

Quando os vetores possuem mais dimensões, chamamos de matrizes. Matrizes bidimensionais possuem duas dimensões. Pense em seu monitor, a imagem é formada por pequenos pontos. Esses pontos formam uma matriz da largura com a altura. Outro exemplo, para entender as matrizes, é o jogo de batalha naval. Nele, a localização do mar é dada por uma linha e coluna. A Figura 74 mostra uma referência para uma matriz. Em cada quadradinho podemos colocar um valor, mas todos os

quadrados devem ter o mesmo tipo, ou seja, se a matriz for de String, só pode ter String; se for de int, só pode ter números inteiros.

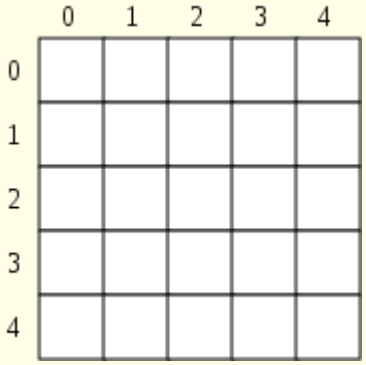


Figura 74 – Representação visual de uma matriz.
Fonte: https://pt.wikibooks.org/wiki/Java/Vetores
Descrição: Representação visual de uma matriz de 5x5 (cinco por cinco) espaços, totalizando 25 espaços.

Matrizes de três dimensões são muito utilizadas em programas e jogos 3D. Nelas podemos ter a altura a largura e profundidade dos elementos. É um endereço formado por três valores. A Figura 75 mostra uma representação de uma matriz tridimensional.

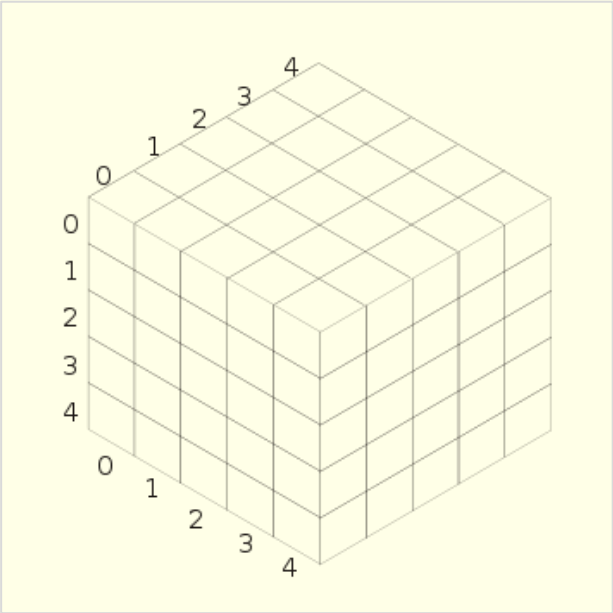


Figura 75 – Representação visual de uma matriz tridimensional.
Fonte: https://pt.wikibooks.org/wiki/Java/Vetores
Descrição: Representação visual de uma matriz de 5x5x5 espaços, totalizando 125 espaços.

Podemos ter mais dimensões de acordo com a necessidade, mas até três dimensões são mais comuns.

Então, como criamos estas variáveis que podem guardar vários valores nela?

```
tipo[] identificador = new tipo[quantidade];

tipo[][] identificador = new tipo[quantidade][quantidade];

tipo[][][] identificador = new tipo[quantidade][quantidade][quantidade];
```

Para declarar um vetor, ou matriz, colocamos o tipo, um par de colchetes para cada dimensão e um identificador. Para iniciar um vetor, ou matriz, atribuímos com a palavra-chave 'new' o tipo e a quantidade de itens de cada dimensão. Observe no exemplo da Figura 76 a criação de três vetores de uma, duas e três dimensões de tipos diferentes.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        int[] numeros = new int[5];

        char[][] caracteres = new char[10][20];

        String[][][] strings = new String[10][100][1000];
    }
}
```

Figura 76 – Exemplo de declaração e inicialização de arrays.
Fonte: Autor.
Descrição: Declaração e inicialização de arrays, unidimensional, bidimensional e tridimensional.

A utilização de vetores é bem simples. Basta você colocar o identificador do vetor e nos colchetes colocar o endereço. Como em Java, o primeiro índice começa sempre de 0, o exemplo da Figura 77 coloca o valor 10 na posição (3) do vetor unidimensional; o caractere 'd' na coluna 3 da linha 15 da matriz bidimensional; e a string "dez" na coluna 3 da linha 15, de profundidade 545 da matriz tridimensional.

Então, para utilizar em uma expressão basta fazer o mesmo. O computador vai substituir o endereço do vetor por seu valor, se existir. Como se fosse uma variável comum.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        int[] numeros = new int[5];
        char[][] caracteres = new char[10][20];
        String[][][] strings = new String[10][100][1000];

        numeros[3] = 10;
        caracteres[3][15] = 'd';
        strings[3][15][545] = "dez";


    }
}
```

Figura 77 – Atribuindo valores em arrays.
Fonte: Autor.
Descrição: Exemplo de atribuição de valores em um array unidimensional, bidimensional e tridimensional.

## 6.2 Utilizando laços for aninhados para preencher um array

É bem comum utilizar laços for para preencher arrays. Quando se tem uma dimensão, utilizamos um laço for; quando temos duas, aninhamos os laços para fornecer a posição da linha e da coluna. Se tiver mais dimensões, vamos aninhando mais.

O exemplo da Figura 78 mostra o preenchimento com o número da linha, e da linha somado com o da coluna, dos arrays. Preste atenção à quantidade de itens de cada dimensão.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        int[] a = new int[10];

        for (int i = 0; i < 10; i++) {
            a[i] = i;
        }

        int[][] b = new int[20][50];

        for (int i = 0; i < 20; i++) {
            for (int j = 0; j < 50; j++) {
                b[i][j] = i + j;
            }
        }
    }
}
```

Figura 78 – Utilização de laços para preenchimento de arrays.
Fonte: Autor.
Descrição: Exemplo de utilização de um laço para preenchimento de um array unidimensional e laços aninhados para preenchimento de array bidimensional.

## 6.3 Estrutura de repetição foreach

Como dissemos a estrutura foreach foi construída para facilitar a leitura de variáveis array. Ela passa por cada item do array colocando cada valor em uma variável para utilização, de uma forma mais simples do que pela estrutura for. Ela também utiliza palavra-chave for, mas de um jeito diferente.

```
for (tipo variável : array) {
      código
}
```

Em 'tipo variável' colocamos uma variável que utilizaremos no código. Em array colocamos o array. Observe o exemplo na Figura 79 que soma todos os valores do array 'a'.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        int[] a = new int[10];

        for (int i = 0; i < 10; i++) {
            a[i] = i;
        }

        int total = 0;
        for (int v : a ) {
            total += v;
        }
    }
}
```

Figura 79 – Exemplo da estrutura foreach para iteração em array.
Fonte: Autor.
Descrição: Exemplo da estrutura foreach para iteração em array.

## 6.4 Exercício

Agora vamos fazer um exercício que une tudo que aprendemos.

1. Faça um array de 5x5 de caracteres.

2. Preencha o valor de acordo com a seguinte regra: se o valor da linha e da coluna forem iguais, coloque o caractere ‘x’, senão, coloco o caractere ‘o’. Por exemplo, a primeira linha e a primeira coluna é [0] e [0], então são iguais, coloca-se o valor ‘x’. Para preencher, utilize laços de repetição.

3. Agora faça um leiaute igual ao da Figura 80, com cinco botões e no texto de cada um dos botões tem a lista de caracteres da linha equivalente. Assim, a primeira linha terá os valores x – o – o – o – o – . Utilize laços foreach para isto.

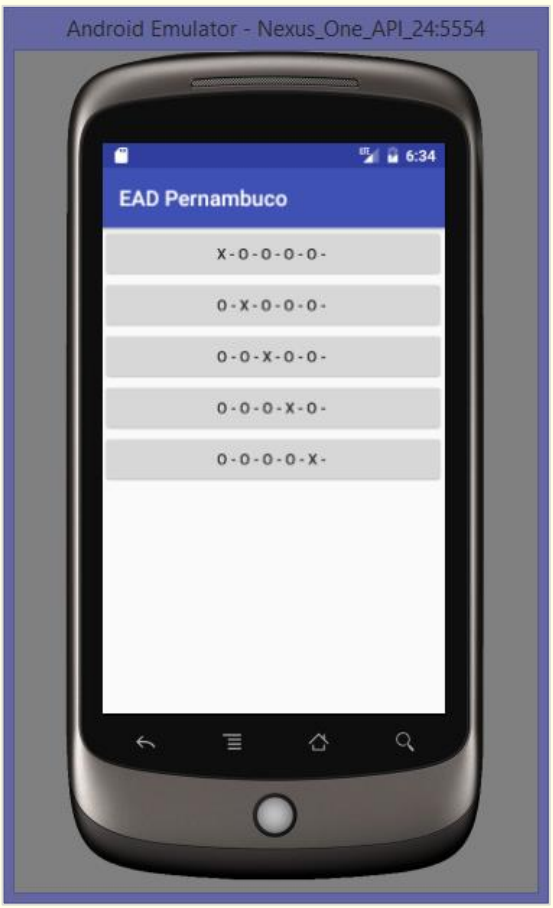


Figura 80 – Exibição da resposta no exercício no emulador.
Fonte: Autor.
Descrição: Exibição da resposta no exercício no emulador, em que há cinco fileiras.

Resposta:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        char[][] array = new char[5][5];
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (i == j)
                    array[i][j] = 'x';
                else
                    array[i][j] = 'o';
            }
        }
        LinearLayout l = new LinearLayout(this);
        l.setOrientation(LinearLayout.VERTICAL);
        for (char[] v : array) {
            Button b = new Button(this);
            String s = "";
            for (char c : v) {
                s += c + " - ";
            }
            b.setText(s);
            l.addView(b);
        }
        setContentView(l);
    }
}
```

Figura 81 – Resposta do exercício.
Fonte: Autor.
Descrição: Resposta do exercício, onde o resultado é exibido na Figura 79

## Conclusão

Chegamos ao final da nossa disciplina. Você viu uma introdução à tecnologia Android e como montar um ambiente de desenvolvimento e teste, fazendo a primeira aplicação de teste, o 'Alo Mundo'. Depois, nos embrenhamos pelos meandros da linguagem Java com alguns toques do paradigma orientado a objeto e programação Android. Passamos por variáveis, estruturas de controle e decisão e finalizamos com arrays.

Deu para perceber que é muita coisa e bem complexa. Mas não é algo de outro mundo. É até bem parecido com o nosso. Nada que algumas horas de estudo não resolvam. Então, não é hora de parar. O que vimos foi uma brevíssima introdução. A linguagem Java e o desenvolvimento de aplicativos ainda tem muito mais para ser visto. Anime-se! Se dedique com afinco que você consegue.

Caso algum conceito não tenha ficado bem entendido, volte e leia. Procure mais material na internet. Você encontrará em texto e em vídeo. Às vezes, o que falta, é uma explicação diferente.

A qualidade do profissional de desenvolvimento está na quantidade de horas de trabalho. Quanto mais experiência ele tem, melhor ele é. Estima-se que um profissional, na maioria das profissões, precisa de 10.000 horas de experiência para se considerar profissional. Então, não perca tempo. Comece a desenvolver-se o mais rápido possível e não se esqueça de se divertir no processo.

Grande abraço e boa sorte.

## Referências


BURTON, Michael; FELKER, Donn. **Desenvolvimento de Aplicativos Android para Leigos**. Rio de Janeiro: Alta Books Editora, 2014.

GLAUBER, Nelson. **Dominando o Android**: do básico ao avançado. 2.ed. atual e ampl. São Paulo: Novatec Editora, 2015.

LECHETA, Ricardo R. **Google Android**: Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 3.ed. São Paulo: Novatec Editora, 2013.

MONTEIRO, João Bosco. Google **Android**: Crie aplicações para celulares e tablets [recurso eletrônico]. [*s.l.*]: Editora Casa do Código, 2013.

SIERRA, Kathy,BATES, Bert. **Use a cabeça**: Java. 2.ed. Rio de Janeiro: Alta Books, 2009.

## Minicurrículo do Professor

**Ewerton Mendonça** é formado em Sistemas de Informação pela UPE e Design pela UFPE, com mestrado em Ciência da Computação pela UFPE. Atualmente é professor na Faculdade de Ciências e Letras de Caruaru e Devry Unifavip. Possui experiência na área de desenvolvimento WEB e design gráfico desde 1998.