



Introdução à Programação para dispositivos móveis

Rebecca Cristina Linhares de Carvalho



Curso Técnico em Desenvolvimento de Sistemas
Educação a Distância
2020



Introdução à Programação para dispositivos móveis

Rebecca Cristina Linhares de Carvalho

Curso Técnico em Desenvolvimento de Sistemas

Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa

Educação a Distância

Recife

Mar. 2020



Licença Pública Creative Commons
Atribuição-NãoComercial-Compartilhual 4.0 Internacional

Professor Autor

Rebecca Cristina Linhares de Carvalho

Coordenação de Curso

José Américo Teixeira de Barros

Coordenação Design Educacional

Deisiane Gomes Bazante

Design Educacional

Ana Cristina do Amaral e Silva Jaeger

Helisangela Maria Andrade Ferreira

Izabela Pereira Cavalcanti

Jailson Miranda

Roberto de Freitas Moraes Sobrinho

Descrição de imagens

Sunnye Rose Carlos Gomes

Catálogo e Normalização

Hugo Cavalcanti (Crb-4 2129)

Diagramação

Roberto de Freitas Moraes Sobrinho

Coordenação Executiva

George Bento Catunda

Renata Marques de Otero

Manoel Vanderley dos Santos Neto

Coordenação Geral

Maria de Araújo Medeiros Souza

Maria de Lourdes Cordeiro Marques

Secretaria Executiva de

Educação Integral e Profissional

Escola Técnica Estadual

Professor Antônio Carlos Gomes da Costa

Gerência de Educação a distância

Março, 2020

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISDB

M539i

Carvalho, Rebecca Cristina Linhares de.

Introdução à Programação para dispositivos móveis: Curso Técnico em Desenvolvimento de Sistemas: Educação a distância / Rebecca Cristina Linhares de Carvalho. – Recife: Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa, 2020.

93 p.: il.

Inclui referências bibliográficas.

Caderno eletrônico produzido em março de 2020 pela Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa.

1. Engenharia de Software. 2. Dispositivos Móveis. I. Título.

CDU – 004.41(043)



Sumário

Introdução	7
1.Competência 01 Conhecer os Conceitos Básicos de um Sistema Operacional Móvel	8
1.1 Uma Breve história do Android	8
1.2 O que é o Android	12
1.2.1 Sistema operacional Linux	13
1.2.2 Código aberto e livre	14
1.3 Arquitetura da plataforma	15
1.4 Os diversos sabores do Android	19
1.5 Google Play	22
2.Competência 02 Configurar o Ambiente de Desenvolvimento e Conhecer Arquivos Básicos de XML	24
2.1 Requisitos para o desenvolvimento	24
2.1.1 Instalação do JDK	25
2.1.2 Instalação e configuração do Android Studio	27
2.1.2.1 Configuração do AVD no Android Studio	34
2.2 Hello World – Criando o nosso primeiro projeto	38
3.Competência 03 Formatar uma Activity para o Desenvolvimento, Utilizando Operadores Matemáticos.....	45
3.1 Conceitos Básicos	45
3.1.1 Iniciando um novo projeto com Android Studio	45
3.1.2 Estrutura de um projeto Android	50
3.1.2.1 Arquivo AndroidManifest.xml	51
3.1.2.2 Classe MainActivity	52
3.1.2.3 Arquivo de Layout activity_main.xml	54
3.1.3 Processo de compilação e execução de um projeto Android	55



3.2 Variáveis	57
3.3 Tipo de dados	58
3.3.1 Tipo de dado primitivo	59
3.3.2 Tipo de dado lógico	59
3.3.3 Tipo de dado numérico Inteiros	60
3.3.4 Tipo de dado numérico de ponto flutuante	61
3.3.5 Tipo de dado de caractere	62
3.3.6 Tipo de dado complexo: String	63
3.4 Separadores	64
3.5 Operadores	65
3.5.1 Operadores Aritméticos	65
3.5.1.1 Incremento e decremento	66
3.5.2 Operadores Relacionais e Lógicos	69
3.5.3 Operadores de atribuição	70
3.5.3.1 Operadores de atribuições abreviadas	70
4.Competência 04 Formatar uma Activity para o Desenvolvimento de Estruturas de Controle Condicional	73
4.1 Estrutura if	73
4.2 Estrutura if-else	74
4.3 Estrutura if-else aninhada	75
4.4 Estrutura <i>Switch</i>	77
5.Competência 05 Formatar uma Activity para o Desenvolvimento de Estruturas de Repetição ...	80
5.1 Estrutura For	80
5.2 Estrutura While	81
5.3 Estrutura Do-While	82
6. Competência 06 Formatar uma Activity para utilização de Listas e Arrays	85



6.1 Arrays	85
6.1.1 Arrays Unidimensionais	86
6.1.2 Arrays Multidimensionais	88
6.1.2.1 Arrays Bidimensionais	88
Conclusão	91
Referências	92
Minicurrículo do Professor	93



Introdução

Olá, desenvolvedor em formação!

Estamos iniciando nossa jornada na disciplina de Introdução à Programação para Dispositivos Móveis (IPDM). O objetivo geral desta disciplina é por meio de uma linguagem simples, descomplicada e com aplicações práticas facilitar seu caminho para uma melhor compreensão das noções iniciais e essenciais do desenvolvimento de aplicativos para a plataforma Android.

Para tal, começaremos conhecendo algumas das principais ferramentas necessárias para o desenvolvimento de aplicações para dispositivos móveis neste ambiente operacional, aprendendo os pormenores de suas instalações e da correta configuração delas. Também aprenderemos a identificar e a trabalhar com termos como: variáveis, tipos de dados, operadores, estruturas de controle condicional e de repetição e arrays.

Espero que o conteúdo discutido lhe sirva como uma primordial ferramenta em sua busca por um amplo universo de conhecimentos, e que você alce um voo muito além das disciplinas deste curso.

Cordialmente

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

Esta primeira competência tem o objetivo de apresentá-los ao sistema operacional Android. Começaremos com uma breve história do Android e seguiremos para a introdução de conceitos básicos do Android e sua arquitetura. Por fim, vamos acompanhar a evolução das “doces” versões do Android.

1.1 Uma Breve história do Android

A história tem início em outubro de 2003, quando foi fundada, na cidade de Palo Alto (Califórnia, estado dos EUA), a empresa *Android Inc.*, por Andy Rubin, Rich Miner, Nick Sears e Chris White. O propósito, segundo Rubin (Figura 1.1), era o desenvolvimento de “dispositivos móveis inteligentes mais cientes da localização e preferências do seu dono”.



Figura 1- Andy Rubin.

Fonte: showmetech.com

Descrição da imagem: a imagem apresenta um homem de braços cruzados em uma sala.

Quase dois anos mais tarde, em 17 de julho de 2005, a empresa *Google LLA* adquiriu a pequena empresa *Android Inc.* (Figura 1.2). E Andy Rubin passou a integrar o corpo de membros da *Google*.



Figura 2 - A empresa Google LLA adquire a pequena empresa Android Inc.

Fonte: super.abril.com

Descrição da imagem: a imagem apresenta na parte superior a logomarca da empresa Google LLA com a palavra Google nas cores azul, vermelho, amarelo, azul, verde e vermelho. Em seguida o sinal de soma, e logo abaixo a logomarca da empresa Android, com uma figura de um robô na cor verde.

Em junho de 2007, o diretor executivo da empresa *Apple Inc.*, *Steve Jobs* (Figura 1-3), apresentou o seu primeiro modelo de *smartphone*, o *iPhone*:

“Um iPod, um telefone e um comunicador móvel de internet. Esses não são três dispositivos separados. E nós o estamos chamando de iPhone. Hoje a Apple vai reinventar o telefone.

”



Figura 3 - Steve Jobs apresenta primeiro iPhone.

Fonte: techtudo.com

Descrição da imagem: a imagem apresenta um homem em pé, e logo atrás três ícones no formato de um quadrado na seguinte sequência: ícone na cor amarela representando o iPod, ícone na cor verde representando o telefone e ícone na cor azul representando a Internet.



Veja o vídeo onde o Steve Jobs apresenta primeiro iPhone

https://www.youtube.com/watch?time_continue=1&v=9ou608QQRq8&feature=emb_logo

Em novembro de 2007, depois que *Apple* anunciou e lançou o *iPhone*, a *Open Handset Alliance (OHA)* foi apresentada oficialmente ao mercado internacional como um grupo formado por gigantes do mercado de telefonia de celulares liderados pela *Google*. O objetivo do grupo era definir uma plataforma única e aberta para dispositivos móveis para deixar os consumidores mais satisfeitos com o produto final. A aliança teve 34 membros fundadores, desde operadores móveis a empresas de *software*, e fabricantes de *hardware*.



Quando este caderno foi escrito, a OHA era formada por 84 integrantes, a lista completa e atualizada pode ser verificada em:

<http://www.openhandsetalliance.com/>

Ainda em 2007, foi tornado público o primeiro resultado dessa união: uma plataforma para dispositivos móveis chamada *Android*.



Em 5 de novembro de 2007, Steve Horowitz, um dos engenheiros de software responsáveis pelo *Android*, fez a primeira apresentação pública do *Android*:

https://www.youtube.com/watch?v=1FJHYqEORDg&feature=emb_logo

Em 22 de outubro de 2008, foi lançado oficialmente o primeiro *Android* comercial do mercado, o *Android Alpha*, rodando em um *HTC Dream* (Figura 1.4). O *T-Mobile G1* ou *HTC Dream* foi primeiro aparelho a ter o sistema operacional móvel *Android* como padrão de fábrica.



Figura 4 - O T-Mobile G1 ou HTC Dream foi primeiro aparelho a ter o sistema operacional móvel Android.

Fonte: super.abril.com

Descrição da imagem: A imagem apresenta o aparelho de celular HTC Dream na cor preta.

O mundo da tecnologia está sempre em evolução, e desde 2008 que o *Android* tem recebido várias atualizações que melhoraram substancialmente o sistema, com a adição de novas funcionalidades e a resolução de erros de versões anteriores. A cada grande atualização muda o nome de código do sistema operacional *Android*, sendo este identificado com nomes de doces em inglês, seguindo uma ordem alfabética (Figura 1.5).

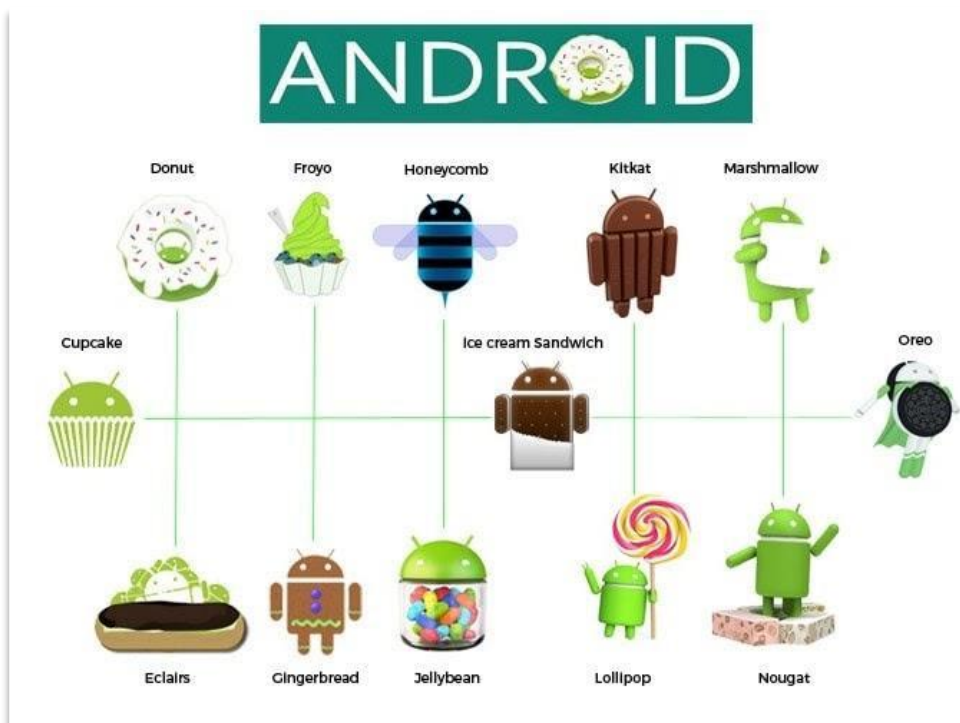


Figura 1.5 - Versões do Android.

Fonte: fmm2019apk.net

Descrição da imagem: a imagem apresenta as logomarcas das últimas treze versões do Android, dispostas na ordem cronológica de tempo, cada uma contendo a seguinte descrição: um cupcake no formato de um robô na cor verde, uma



rosquinha com um robô na cor verde, um doce bomba de chocolate com um robô na cor verde, um iogurte gelado na cor verde, um biscoito de gengibre no formato de um robô, um doce favo de mel no formato de uma abelha robô, um robô na cor verde com balas de jujuba de diversas cores, um sanduíche de sorvete no formato de um robô, um chocolate kitkat no formato de um robô, um robô na cor verde segurando um pirulito, um robô na cor verde segurando um marshmallow, um robô na cor verde sobre o doce torrão e um biscoito oreo no formato de um robô na cor verde.



A história do Android

<https://www.youtube.com/watch?v=5K4pEk19nhs>

Assim, há 14 anos atrás a *Google* anunciou a aquisição da pequena empresa *Android Inc.*, de Palo Alto. Esta compra acabou mudando para sempre o mercado mobile, dando origem ao *Android*, o sistema operacional mais usado do mundo.



Android passa Windows e se torna o sistema operacional mais usado do mundo:

<https://g1.globo.com/tecnologia/noticia/android-passa-windows-e-se-torna-o-sistema-operacional-mais-usado-do-mundo.ghtml>

1.2 O que é o Android

O *Android* (Figura 1.6) é o sistema operacional móvel da *Google* completamente livre e de código aberto, e líder mundial nesse segmento. Na conferência anual da *Google*, *Google I/O 2017*, foi anunciado que já existem mais de dois bilhões de dispositivos Androids ativados no mundo.

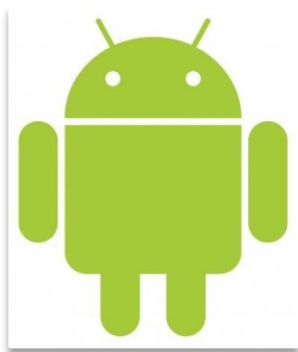


Figura 1.6 - Marca do Android.

Fonte: veja.abril.com.

Descrição da imagem: a imagem apresenta um robô na cor verde.



Atualmente, o *Android* está disponível para diversas plataformas, como *smartphones* e *tablets*, TV, relógios, carros, e é o sistema operacional móvel mais utilizado no mundo.



Veja os principais anúncios da gigante Google na conferência Google I/O 2019:
<https://canaltech.com.br/google-io/google-io-2019-os-principais-anuncios-da-gigante-para-este-ano-138705/>

1.2.1 Sistema operacional Linux

O sistema operacional do *Android* é baseado no *Kernel* do *Linux*, que é responsável por gerenciar memória, processos, *threads*, segurança dos arquivos e das pastas, redes e *drivers*, além da energia.

Cada aplicativo no *Android* dispara um novo processo no sistema operacional. Diversos processos e aplicativos podem ser executados simultaneamente, e o *kernel* do sistema operacional é o responsável por realizar todo o controle de memória. Por exemplo, caso seja necessário, o próprio sistema operacional pode decidir encerrar algum processo para liberar memória e recursos. E quando a situação estiver controlada, talvez o mesmo processo seja reiniciado.



Linux é um Sistema Operacional, assim como o Windows e o Mac OS, que possibilita a execução de programas em um computador e outros dispositivos, e pode ser livremente modificado e distribuído. Mas, em uma definição mais profunda e técnica, Linux é o nome dado apenas ao núcleo do sistema operacional, chamado de Kernel.



Kernel é um conjunto de instruções que controla como será usado o processador, a memória, o disco e dispositivos periféricos. É o software presente em todo sistema operacional que dita como o computador deve funcionar. Saiba mais sobre o Linux e o Kernel:

<https://www.4linux.com.br/o-que-e-linux>

<https://www.vivaolinux.com.br/artigo/Como-explicar-o-que-e-kernel-para-um-leigo/>

1.2.2 Código aberto e livre

Em 21 de outubro de 2008 o *Android* se transformou em *Open Source* (Figura 1.7), com seu código publicado como *AOSP (Android Open Source Project)*. Se tornando a primeira plataforma para aplicativos móveis completamente livre e de código aberto (*open source*), o que representou uma grande vantagem competitiva para a sua evolução, uma vez que diversas empresas e desenvolvedores do mundo podem contribuir para melhorar a plataforma.

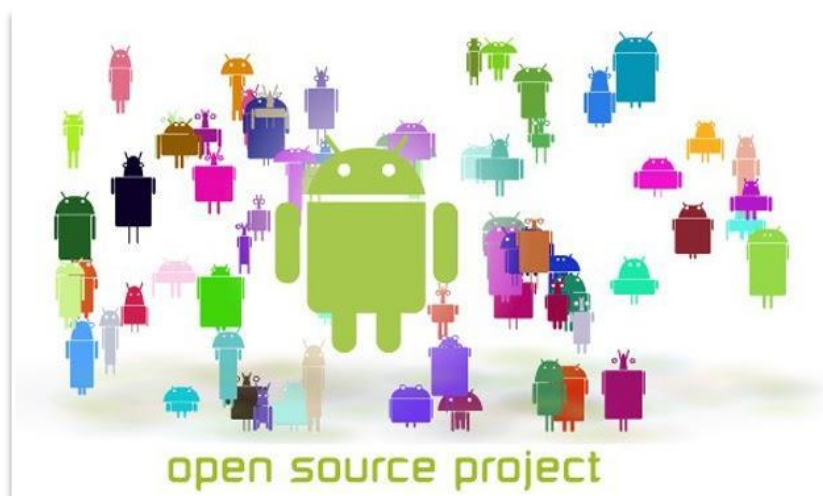


Figura 1.7 - Android Open Source Project.

Fonte: mobilenet.cz.

Descrição da imagem: no centro da imagem há um robô na cor verde e ao seu entorno há diversos outros tipos de robôs, nos mais variados tamanhos e cores.

Para os fabricantes de celulares, isso também é uma grande vantagem, uma vez que é possível utilizar o sistema operacional do *Android* em seus celulares sem ter de pagar por isso.

Toda a segurança do sistema operacional *Android* é baseada na segurança do sistema Linux. Isto significa que, o *Android* é um sistema Linux multiusuário em que cada aplicativo é um usuário diferente, ou seja, para cada aplicação instalada no celular é criado um usuário no sistema operacional para ter acesso a sua estrutura de diretórios. Assim, somente o usuário atribuído àquele aplicativo poderá acessá-lo.



O Android Open Source Project (AOSP) é uma iniciativa que visa disponibilizar versões do Android totalmente baseadas em código-fonte aberto. Saiba mais sobre o AOSP:

<https://source.android.com/>

1.3 Arquitetura da plataforma

Como descrito no site oficial, “O *Android* é uma pilha de *software* com base em *Linux* de código aberto criada para diversos dispositivos e fatores de forma”. Isto significa que, a arquitetura do *Android* é construída em camadas, os quais são *softwares* agrupados em uma pilha. E estas, se dividem em 6 (seis) camadas: Linux kernel (*Kernel* do *Linux*), HAL(hardware abstraction layer)/HIDL(HAL Interface Description Language) Camada de abstração de hardware (HAL) e Idioma da Descrição da Interface HAL (HIDL), *Android Runtime* (tempo de execução do android), Native Libraries (Bibliotecas C/C++ nativas), Android Framework (Estruturas da Java API (*Application Programming Interface*)) e as Apps (aplicativos do sistema) (vide Figura 1.8).

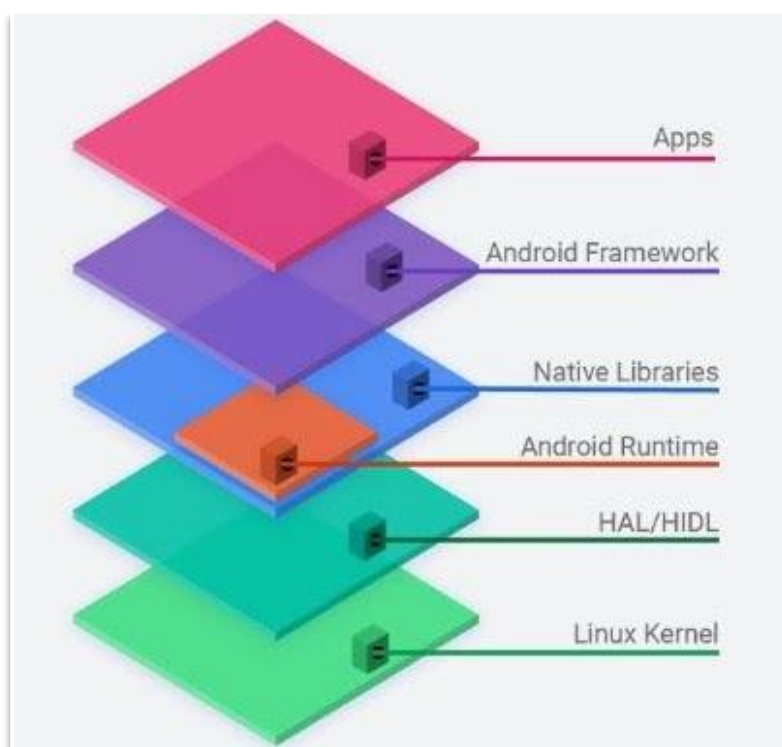


Figura 1.8 - As seis camadas da arquitetura da plataforma Android.
Fonte: source.android.com.



Descrição da imagem: a imagem apresenta figuras geométricas no formato de um quadrado representando camadas, sobrepostas de cima para baixo na seguinte sequência: rosa pink, lilás, laranja, azul, verde e verde claro.

Dentro de cada camada temos alguns componentes da arquitetura Android, que estão ilustrados na imagem abaixo:

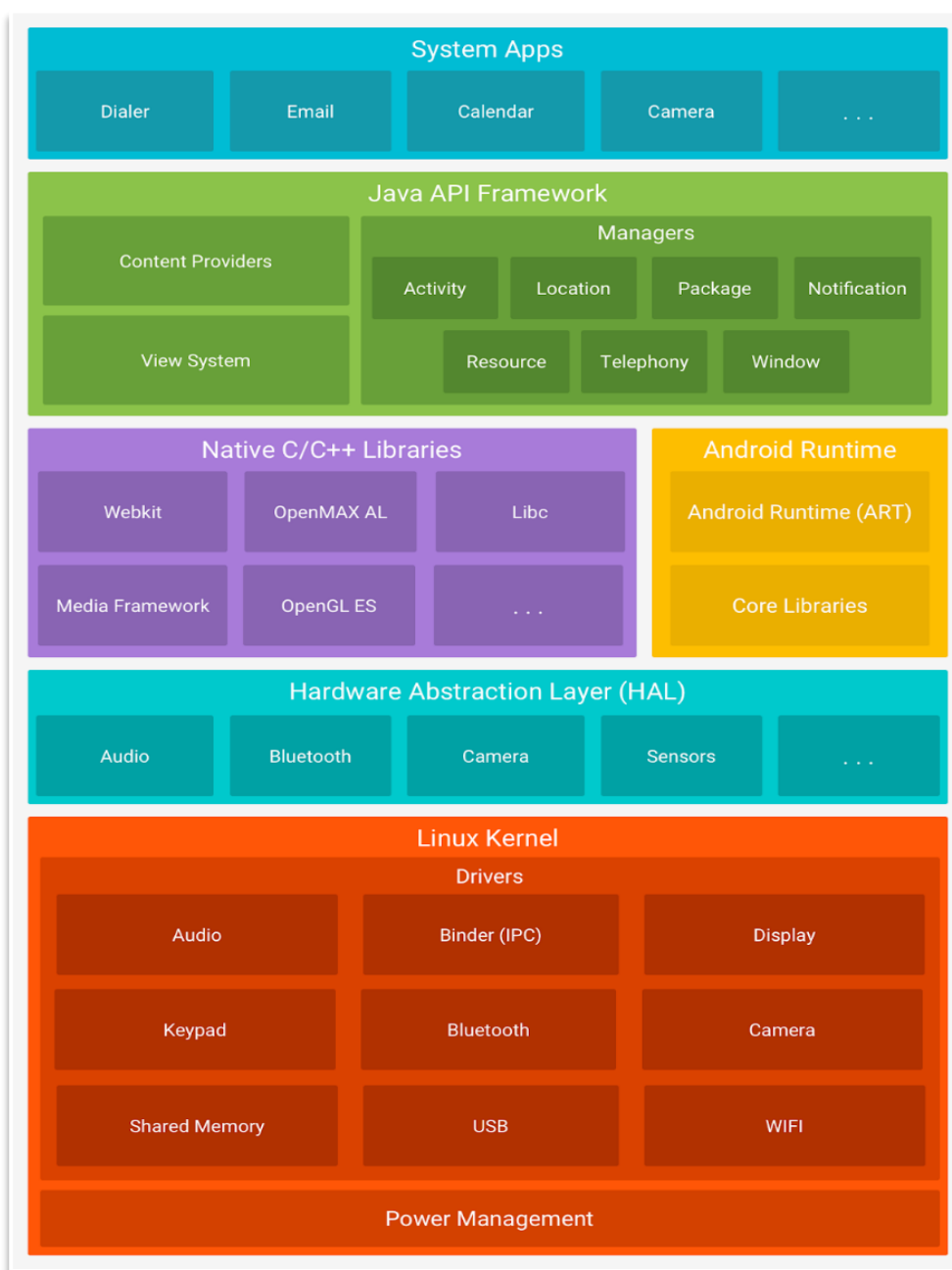


Figura 1.9 - A pilha de software do Android.

Fonte: developer.android.com

Descrição da imagem: a imagem apresenta figuras geométricas no formato de um retângulo representando uma pilha, onde os elementos estão em cima do outro na seguinte sequência (base-topo): laranja, azul claro, lilás e amarelo, verde claro e azul escuro.



- **Kernel do Linux (*Linux Kernel*):** é o nível mais baixo e a base da arquitetura *Android*. Por exemplo: o *Android Runtime* (ART) confia no *kernel* do *Linux* para cobrir funcionalidades como encadeamento e gerenciamento de memória de baixo nível. Usar um *kernel* do *Linux* permite que o *Android* aproveite os recursos de segurança principais e que os fabricantes dos dispositivos desenvolvam *drivers* de *hardware* para um *kernel* conhecido.
- **Camada de abstração de hardware (*Hardware Abstraction Layer* - HAL):** fornece interfaces padrões que expõem as capacidades de *hardware* do dispositivo para a estrutura da Java API de maior nível. A HAL consiste em módulos de biblioteca, que implementam uma interface para um tipo específico de componente de *hardware*, como o módulo de câmera ou *Bluetooth*. Quando um *Framework* API faz uma chamada para acessar o *hardware* do dispositivo, o sistema *Android* carrega o módulo da biblioteca para este componente de *hardware*.
- ***Android Runtime*:** para dispositivos com *Android* versão 5.0 ou mais recente, cada aplicativo executa o próprio processo com uma instância própria do *Android Runtime* (ART). O ART é projetado para executar várias máquinas virtuais em dispositivos de baixa memória executando arquivos DEX, o qual é um formato de bytecode projetado especialmente para *Android*, otimizado para oferecer consumo mínimo de memória.
- **Bibliotecas C/C++ nativas (*Native C/C++ Libraries*):** vários componentes e serviços principais do sistema *Android*, como ART e HAL, são implementados em código nativo que exige bibliotecas nativas programadas em C e C++. A plataforma *Android* fornece as *Java APIs Framework* para expor a funcionalidade de algumas dessas bibliotecas nativas aos aplicativos. Por exemplo, é possível acessar OpenGL ES pela Java OpenGL API da estrutura do *Android* para adicionar a capacidade de desenhar e manipular gráficos 2D e 3D no seu aplicativo. Se estiver desenvolvendo um aplicativo que exige código C ou C++, você pode usar o *Android* NDK para acessar algumas dessas bibliotecas de plataforma nativa diretamente do seu código nativo.
- ***Java API Framework*:** o conjunto completo de recursos do sistema operacional *Android* está disponível pelas APIs programadas na linguagem Java. Essas APIs formam os blocos de programação que você precisa para criar os aplicativos *Android*



simplificando a reutilização de componentes e serviços de sistema modulares e principais, como:

- Um sistema de visualização rico e extensivo útil para programar a interface do usuário um aplicativo, com listas, grades, caixas de texto, botões e até mesmo um navegador da web incorporado;
- Um gerenciador de recursos, fornecendo acesso a recursos sem código como strings localizadas, gráficos e arquivos de layout;
- Um gerenciador de notificação que permite que todos os aplicativos exibam alertas personalizados na barra de status;
- Um gerenciador de atividade que gerencia o ciclo de vida dos aplicativos e fornece uma pilha de navegação inversa;
- Provedores de conteúdo que permitem que aplicativos acessem dados de outros aplicativos, como o aplicativo Contatos, ou compartilhem os próprios dados.

Por fim, os desenvolvedores têm acesso completo às mesmas Framework APIs que os aplicativos do sistema Android usam.

- **Aplicativos do sistema (*System Apps*):** o *Android* vem com um conjunto de aplicativos principais para e-mail, envio de SMS, calendários, navegador de internet, contatos entre outros. Os aplicativos inclusos na plataforma não têm status especial entre os aplicativos que o usuário opta por instalar. Portanto, um aplicativo terceirizado pode se tornar o navegador da Web, o aplicativo de envio de SMS ou até mesmo o teclado padrão do usuário (existem algumas exceções, como o aplicativo Configurações do sistema).

Os aplicativos do sistema funcionam como aplicativos para os usuários e fornecem capacidades principais que os desenvolvedores podem acessar pelos próprios aplicativos. Por exemplo: se o seu aplicativo quiser enviar uma mensagem SMS, não é necessário programar essa funcionalidade — é possível invocar o aplicativo de SMS que já está instalado para enviar uma mensagem ao destinatário que você especificar.

1.4 Os diversos sabores do Android

Em 22 de outubro de 2008, foi lançado oficialmente o primeiro *Android* comercial do mercado, o *Android Alpha*, com o famoso *HTC Dream*, e após isso o *Android* não parou mais de evoluir.

E essa evolução aconteceu de uma forma engraçada e doce, pois cada nova versão do *Android* foi apelidada com o nome de um doce. Quando este caderno foi escrito a versão mais recente chamada extraoficialmente *Android Q*, foi batizada como *Android 10* (Figura 1.10). Isso significa que houve uma quebra de tradição, e desta vez não houve doce.

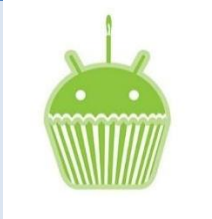



Figura 1.10 - Imagem de divulgação do Android 10.





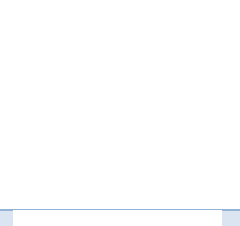
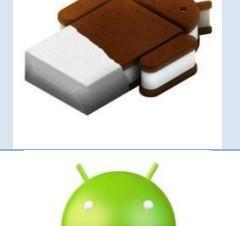
Fonte: techtudo.com.

Descrição da imagem: a imagem apresenta a palavra *Android* na cor branca seguida por uma cabeça de um robô na cor verde.

A Tabela 1.1 mostra os diversos sabores, ou melhor, as diversas versões do *Android*:

Data de Lançamento	Versão	Nome	Características	Figura do Android
Abril.2009	Android 1.5	Cupcake	A principal novidade foi o lançamento do primeiro <i>Android</i> (HTC Magic) com apenas o <i>touch screen</i> e o teclado virtual. Foi no Cupcake que nasceram os <i>widgets</i> .	
Set.2009	Android 1.6	Donut	Com o auxílio das pesquisas de voz, a <i>home</i> do <i>Android</i> ganhou mais funcionalidades, e o usuário pôde começar a pesquisar na agenda de contatos, na galeria de música.	



Out.2009 (atualizado em Jan.2010)	Android 2.0 e 2.1	Eclair	Trouxe uma interface de usuário diferenciada e adicionou os <i>Live Wallpapers</i> (plano de fundo animado na tela inicial). E também lançou o suporte a múltiplas contas do <i>Google</i> e sincronização.	
Mai.2010	Android 2.2	Froyo	Trouxe diversas melhorias de desempenho para o sistema operacional.	
Dez.2010	Android 2.3	Gingerbread	Trouxe novidade na câmera, pois era possível alternar entre a câmera frontal e a traseira. E melhorias na funcionalidade de <i>copy-paste</i> . Foi nesta versão que tivemos um grande ganho com relação ao gerenciamento da bateria.	
Fev.2011	Android 3.0	Honeycomb	Trouxe um sistema operacional totalmente focado nos <i>tablets</i> , com uma experiência de usuário totalmente diferenciada para telas grandes. Nesta versão, o <i>Android</i> deixou de ter botões físicos e os botões de voltar e início (<i>home</i>) passaram a fazer parte da barra de navegação dentro da tela com o touch screen.	
Out.2011	Android 4.0	Ice cream Sandwich	Unificou a plataforma de desenvolvimento entre <i>smartphones</i> e <i>tablets</i> executando o mesmo sistema operacional.	
Jun.2012	Android 4.1	Jelly Bean	Trouxe um ganho significativo com relação ao desempenho do sistema, e todo o sistema operacional ganhou melhorias no suporte às animações, deixando a interface mais sensível ao toque e mais fluida.	



Out.2013	Android 4.4	KikKat	Otimizou o Android ao máximo. E também trouxe aperfeiçoamento no <i>Bluetooth</i> .	
Nov.2014	Android 5.0	Lollipop	Foi a maior <i>release</i> focada na interface de usuário, usabilidade, animações e experiência do usuário.	
Agost.2015	Android 6.0	Marshmallow	Trouxe um novo recurso, o novo modelo de permissão em tempo de execução, chamado <i>Runtime Permissions</i> . O <i>Google Now</i> também recebeu atualizações.	
Google I/O 2016	Android 7.0	Nougat	Trouxe o suporte ao sistema de várias janelas (<i>multi-window</i>), uma nova central de notificações mais ricas, respostas diretas nas notificações e atalhos para executar ações nos aplicativos pela tela <i>Home</i> .	
Google I/O 2017	Android 8.0	Oreo	Foi a versão que mais trouxe novidades para o usuário final. Dentre elas, tivemos uma inicialização mais rápida do sistema operacional.	
Google I/O 2018	Android 9.0	Pie	Dentre as principais novidades tem o suporte ao <i>Wi-Fi RTT</i> para obter melhor localização do usuário, mesmo em ambientes internos.	
Set. 2019	Android 10	Android 10	O Android 10 foi construído em torno de três temas importantes. Primeiro, ele está liderando a inovação móvel com <i>machine</i>	



			<i>learning</i> avançado e compatibilidade com novos dispositivos, como <i>smartphones</i> habilitados para 5G e dobráveis. Segundo o Android 10 tem um foco central em privacidade e segurança, com quase 50 recursos que oferecem aos usuários maior proteção, transparência e controle. Por fim, o Android 10 expande os controles de bem-estar digital dos usuários, para que indivíduos e famílias possam encontrar um melhor equilíbrio com a tecnologia.	
--	--	--	---	--

Tabela 1.1 - As diversas versões do Android

Fonte: in.c.mi.com., tecmundo.com., developer.android.com.

Descrição da imagem: um cupcake no formato de um robô na cor verde, uma rosquinha com um robô na cor verde, um doce bomba de chocolate com um robô na cor verde, um iogurte gelado na cor verde, um biscoito de gengibre no formato de um robô, um doce favo de mel no formato de uma abelha robô, um robô na cor verde com balas de jujuba de diversas cores, um sanduíche de sorvete no formato de um robô, um chocolate kitkat no formato de um robô, um robô na cor verde segurando um pirulito, um robô na cor verde segurando um marshmallow, um robô na cor verde sobre o doce torrone, um biscoito oreo no formato de um robô na cor verde e um robô na cor verde segurando um pedaço de torta.

1.5 Google Play

A *Google Play* é a loja de aplicativos, livros, filmes e músicas do *Android*, e é por meio dela que usuários de todo o mundo podem fazer o download de conteúdo para o seu aparelho.

Foi criada para auxiliar na distribuição dos aplicativos, além da divulgação de sua nova plataforma. Inicialmente se chamava *Android Market*. O objetivo do site foi oferecer aos desenvolvedores um lugar comum para disponibilizar seus aplicativos.

E aí, estudante, foi tranquilo o caminho até aqui? Você já conhecia algo sobre *Android*? O legal é que até aqui aprendemos sobre:

- A história do *Android*.
- Os conceitos básicos do *Android*.
- Arquitetura da plataforma.
- A evolução das “doces” versões do *Android*.



Será que captamos as lições corretamente? Será que precisamos revisar?



Pronto para inserir mais informações em seu banco de dados? Acesse o AVA e assista a nossa primeira videoaula competência 1.



Agora, acesse o AVA e responda as questões da **competência 1**.



Ficou com alguma dúvida na competência 1? Acesse o Fórum - “**Competência 1**” para saná-las e discutir com seus colegas sobre os assuntos estudados.

Agora, o nosso próximo passo é mergulhar na competência 2, que trata do ambiente de desenvolvimento de aplicações *Android* e dos seus componentes. E então, vamos lá?



2.Competência 02 | Configurar o Ambiente de Desenvolvimento e Conhecer Arquivos Básicos de XML

Vamos iniciar a segunda competência? Nela, iremos apresentar os componentes necessários para o ambiente de desenvolvimento de aplicações Android: o *Java Development Kit* (JDK) e o *Android Studio*. Além disso, iremos instalar e configurar estes componentes. E por fim, vamos criar o nosso primeiro projeto no Android Studio o *Hello World*.

2.1 Requisitos para o desenvolvimento

Para começarmos a estudar a tão esperada “Programação para dispositivos móveis” é necessário configurar o ambiente de desenvolvimento de aplicações para o Android no seu computador ou no computador onde aplicação vai ser criada. Para isso o computador deve possuir os seguintes componentes: o *Java Development Kit* (JDK) e o Android Studio.

Caso não tenha estes componentes no computador, você deverá instalá-los. Se esse for o seu caso, fique tranquilo nas próximas seções serão explicados tanto o processo de instalação como o processo de configuração do ambiente de desenvolvimento de aplicações para o Android.

Antes de realizar a instalação dos componentes é necessário verificar se o computador possui a configurações mínimas para executá-los. Os requisitos são:

- Sistema operacional Microsoft Windows 7/8/10 (32 bits ou 64 bits)
- No mínimo de 3 GB de RAM, porém o recomendado é de 8 GB de RAM (mais 1 GB para o emulador do Android).
- No mínimo 2 GB de espaço disponível em disco, porém o recomendado é 4 GB (500 MB para IDE mais 1.5 GB para Android SDK e imagem do sistema emulador).
- Resolução mínima de tela de 1280 x 800.



Você usa outro Sistema Operacional?
Você utiliza outro SO (Sistema Operacional) como Mac OS e Linux e quer saber quais são os requisitos mínimos? Segue o link com as configurações mínimas para executar os componentes:

<https://www.javaworld.com/article/3095406/android-studio-for-beginners-part-1-installation-and-setup.html>



2.1.1 Instalação do JDK

Java Development Kit (JDK) significa Kit de desenvolvimento Java, ou seja, é um pacote de ferramentas que permitem o desenvolvimento de uma aplicação baseado em Java. É composto por compilador e bibliotecas.

O Java é fornecido em dois pacotes: o pacote JDK e o pacote *Java Runtime Environment* (JRE). JRE é o pacote de *runtime* (pacote de ferramenta para executar o código Java) e faz parte do JDK.

A versão de JDK que iremos instalar será Java SE 11.0.5, a qual está disponível no site da Oracle para download no link abaixo:



Download do JDK

<https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html>

Assim, para fazer o Download do JDK, acesse o link e selecione a versão Java SE 11.0.5 de acordo com o sistema operacional do seu computador (Figura 2.1).



Figura 2.1 - Seleção do pacote JDK.

Fonte: oracle.com

Descrição da imagem: a imagem apresenta textos que informam a versão do JDK e as diversas opções do software para download.

Finalizado o download, execute o JDK e siga as instruções para completar o processo de instalação (Figuras 2.2, 2.3 e 2.4).



Figura 2.2 - Instalação do pacote JDK.

Fonte: Java SE 11.0.5

Descrição da imagem: a imagem apresenta uma janela com texto descritivo para instalação do pacote JDK que será confirmado mediante o click do mouse sobre um dos botões: Next, se deseja continuar a instalação e Cancel, se deseja cancelar a instalação.

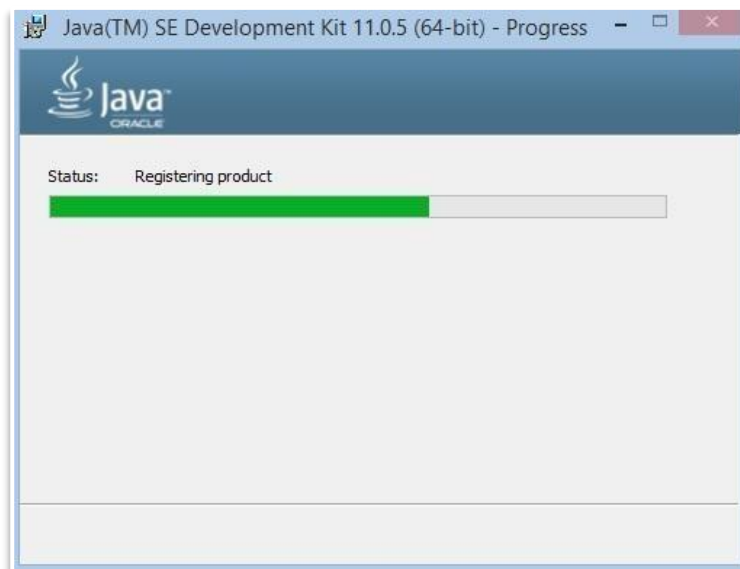


Figura 2.3 - Processo de instalação do pacote JDK.

Fonte: Java SE 11.0.5

Descrição da imagem: a imagem apresenta uma janela com texto descritivo do progresso da instalação do pacote JDK.



Figura 2.4 - Finalizado o processo de instalação do pacote JDK.

Fonte: Java SE 11.0.5

Descrição da imagem: a imagem apresenta uma janela com texto descritivo da conclusão da instalação do pacote JDK. A janela será fechada mediante o click do mouse sobre o botão Close.



O que é Java?

Java é uma linguagem de programação e plataforma computacional lançada pela primeira vez pela Sun Microsystems em 1995.

Saiba mais: https://www.java.com/pt_BR/download/faq/whatis_java.xml

2.1.2 Instalação e configuração do Android Studio

O Android Studio é o IDE (*Integrated Development Environment*) ou ambiente de desenvolvimento integrado padrão da Google cuja função é o desenvolvimento exclusivo de aplicações Android. Atualmente na versão 3.5.2, o Android Studio traz diversas ferramentas para auxiliar no processo de desenvolvimento, testes, análise e depuração de aplicações para Android. Além de ser gratuito. Por este motivo, utilizaremos o Android Studio para criarmos os projetos que desenvolveremos no decorrer deste caderno.



O que é SDK?

O Android SDK (Software Development Kit) ou Kit de Desenvolvimento de Software para Android é um pacote completo de ferramentas utilizadas pelo Android Studio e pelos desenvolvedores Android, incluindo componentes como o SDK Tools, Build Tools e o Platform Tools.

Saiba mais: <https://www.androidpro.com.br/blog/android-studio/android-sdk/>

A versão do Android Studio que iremos instalar será 3.5.2, a qual está disponível no site oficial do Android Studio para download, segue o link abaixo:



Downloads do Android Studio para os sistemas operacionais Windows, Mac, Linux e Chrome OS:

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

Assim, para fazer o Download do Android Studio, acesse o link e clique no botão **baixar Android Studio** (Figura 2-5) e aceite os termos e condições e clique no botão **fazer o download de Android Studio para Windows** para iniciar o download.



Figura 2.5 - Download do Android Studio.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta a página de acesso para fazer o Download do Android Studio, com texto descritivo do Android Studio em que na parte inferior tem-se um botão com fundo verde-escuro com o texto baixar android studio.

Finalizado o download, execute o Android Studio e siga as instruções para completar o processo de instalação (Figuras 2.6, 3.7, 2.8, 2.9, 2.10 e 2.11).



Figura 2.6 - Instalação do Android Studio.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta uma janela com texto descritivo para instalação do Android Studio que será confirmado mediante o click do mouse sobre um dos botões: Back, se deseja voltar, Next, se deseja continuar a instalação e Cancel, se deseja cancelar a instalação.



Figura 2.7 - Instalação do Android Studio.

Fonte: Android Studio 3.5.2



Descrição da imagem: a imagem apresenta uma janela com texto descritivo para instalação dos componentes do Android Studio que será confirmado mediante o click do mouse sobre um dos botões: Back, se deseja voltar, Next, se deseja continuar a instalação e Cancel, se deseja cancelar a instalação.

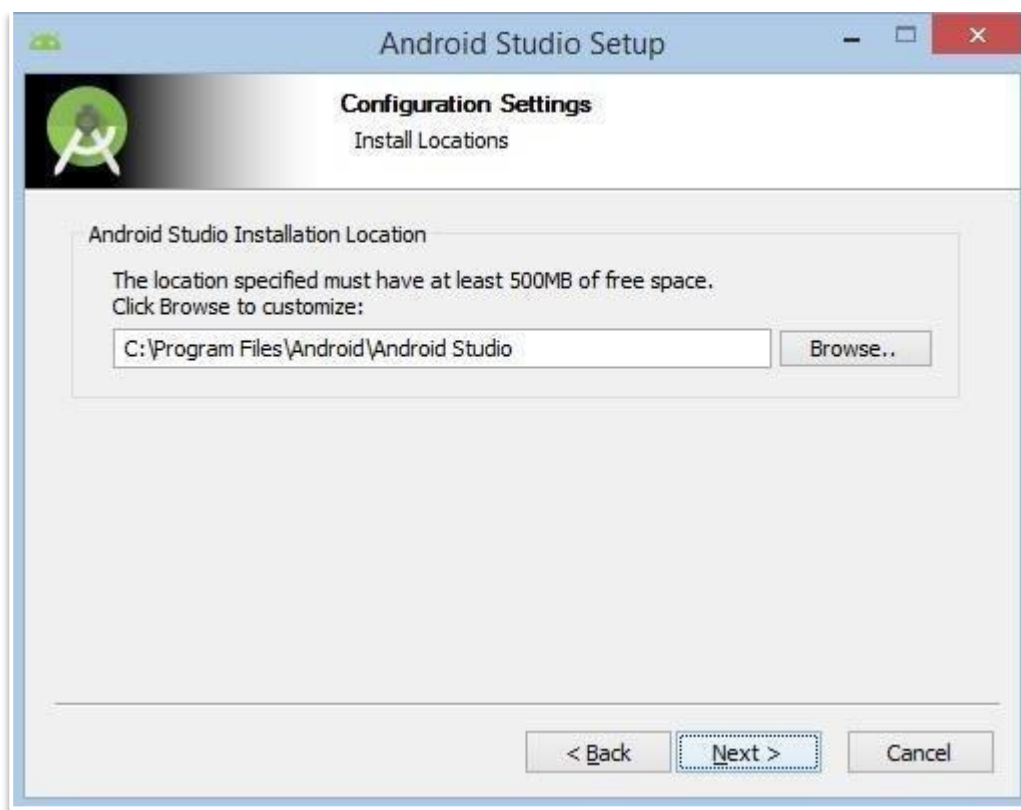


Figura 2.8 - Instalação do Android Studio.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta uma janela com texto descritivo para a instalação do Android Studio. E logo abaixo uma caixa de texto com o texto descritivo do local de instalação, em que no lado direito tem-se um botão com fundo cinza com o texto Browser. A instalação do Android Studio será confirmado mediante o click do mouse sobre um dos botões: Back, se deseja voltar, Next, se deseja continuar a instalação e Cancel, se deseja cancelar a instalação.

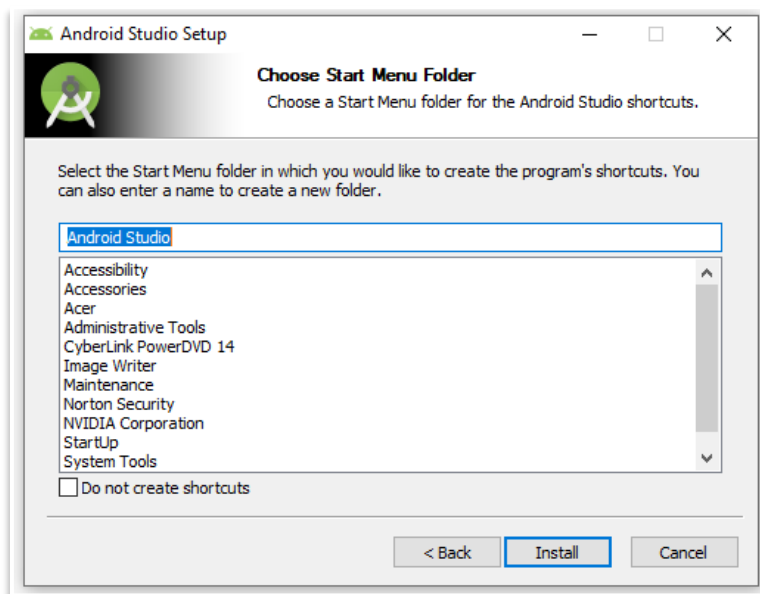


Figura 2.9 - Instalação do Android Studio

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta uma janela com texto descritivo para instalação do Android Studio que será confirmado mediante o click do mouse sobre um dos botões: Back, se deseja voltar, Next, se deseja continuar a instalação e Cancel, se deseja cancelar a instalação.

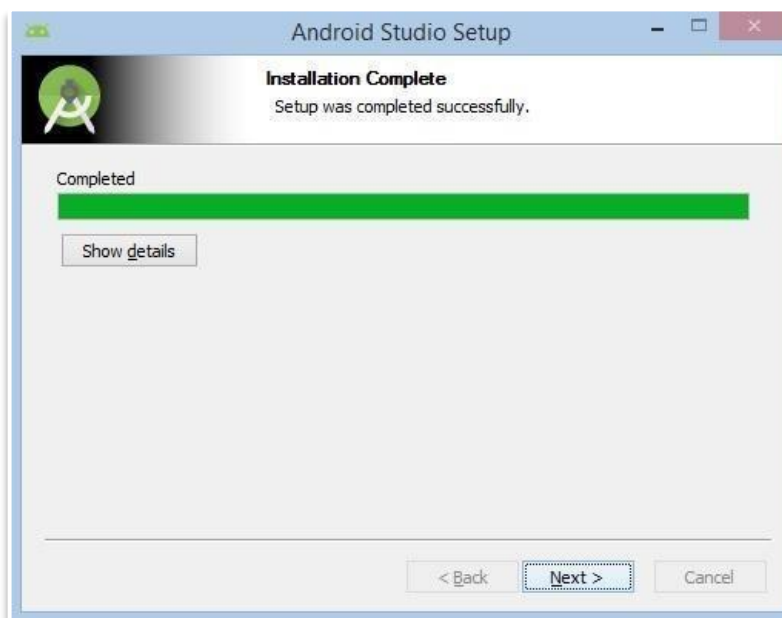


Figura 2.10 - Instalação do Android Studio

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta uma janela com texto descritivo do progresso de instalação do Android Studio. A instalação do Android Studio será confirmado mediante o click do mouse sobre um dos botões: Back, se deseja voltar, Next, se deseja continuar a instalação e Cancel, se deseja cancelar a instalação.



Figura 2.11 - Instalação do Android Studio

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta uma janela com texto descritivo da instalação completa do Android Studio. O Android Studio será executado mediante o click do mouse sobre um dos botões: Back, se deseja voltar, Finish, se deseja finalizar a instalação e Cancel, se deseja cancelar a instalação.



Você usa outro SO?

Você utiliza outro SO como Mac OS e Linux e quer instalar o Android Studio Segue o link com o guia de instalação para estes SO:

<https://developer.android.com/studio/install#mac>

<https://developer.android.com/studio/install#linux>

Após o processo de instalação, o Android Studio irá executar pela primeira vez e então surgirá a janela de carregamento do editor (Figura 2.12).



Figura 2.12 - Tela de carregamento do Android Studio 3.5.1

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza com figuras geométricas em diversos tamanhos, no centro



a palavra Android na cor verde seguida pela palavra Studio na cor cinza-claro.

Surgirá a tela intitulada *Android Studio Setup Wizard*, siga as instruções para o Android Studio realizar o *download* de alguns componentes padrão do SDK. Para finalizar o processo, clique no botão *finish*.

Em seguida, irá aparecer a tela intitulada *Welcome to Android Studio*, onde existem várias opções, selecione a opção *Start a new Android Studio project*. Esta opção irá conduzi-lo para o processo de configuração de um novo projeto Android.

Finalizado o processo de configuração de um novo projeto Android, é exibida a janela principal do Android Studio (Figura 2.13).

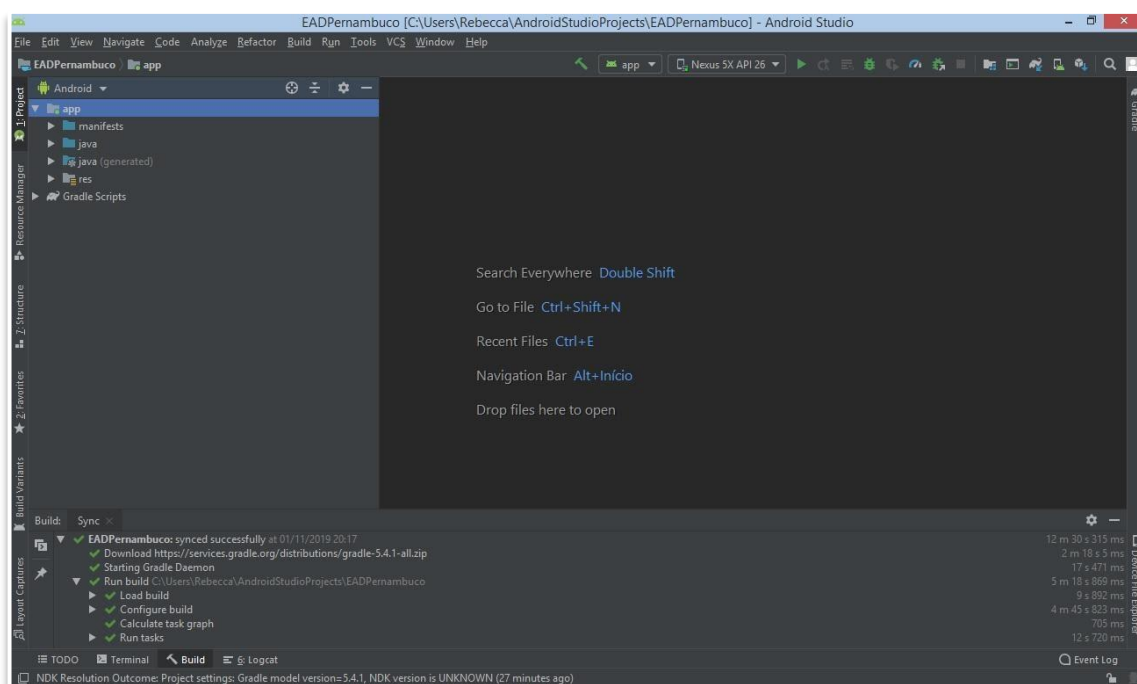


Figura 2.13 - janela principal do Android Studio

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual, a barra de ferramentas e a barra de navegação. Logo em seguida tem-se a barra de janela de ferramentas e a janela do editor. E na parte inferior tem-se outra janela de ferramenta e a barra de estado.

Agora, iremos conhecer cada área destacada na janela principal do Android Studio (Figura 2.13):

- 1) **Menu textual:** onde os comandos e controles são exibidos de forma textual.
- 2) **Barra de ferramentas:** é a barra de ícones com comandos. Permite executar diversas ações, incluindo a execução de aplicações e iniciar ferramentas do Android.



- 3) **Barra de navegação:** é a barra que permite a navegação pelo projeto e a abertura de janelas para edição. Além de oferecer uma visualização mais compacta da estrutura visível na janela Project.
- 4) **Janela das ferramentas ou Janela Project:** é a janela que dá o acesso a tarefas específicas, como gestão de projetos, controle de versão e etc.
- 5) **Barra de janela de ferramentas:** é a barra que contém botões que permitem expandir ou recolher a janela de cada ferramenta.
- 6) **Janela do editor:** é a janela que permite ao programador criar e modificar o código.
- 7) **Outra janela de ferramentas:** neste caso, o LogCat. A janela Logcat no Android Studio exibe mensagens do sistema, como quando ocorre uma coleta de lixo, e mensagens que você adicionou ao seu app com a classe Log. Ela exibe mensagens em tempo real e mantém um histórico para que você possa ver mensagens mais antigas.
- 8) **Barra de estado:** é a barra que mostra o estado do projeto e do próprio IDE. Além de mostrar advertências e mensagens.




O que é LogCat?

Saiba mais: <https://developer.android.com/studio/debug/am-logcat.html?hl=pt-br>

2.1.2.1 Configuração do AVD no Android Studio

O *Android Virtual Device* (AVD) é um emulador do *Android Studio* onde podemos testar as aplicações desenvolvidas. Então, é no AVD que configuramos e definimos as características de um *smartphone* ou *tablet Android*, *Wear OS*, *Android TV* ou um dispositivo *Android Automotive OS* que você queira simular no Android Emulator. Já o *AVD Manager* é uma interface que pode ser iniciada no Android Studio para ajudar a criar e gerenciar AVDs.

Para abrir o *AVD Manager*, siga um dos procedimentos a seguir:

Selecione o ícone **AVD Manager**  na barra de ferramentas, na janela principal do Android Studio:

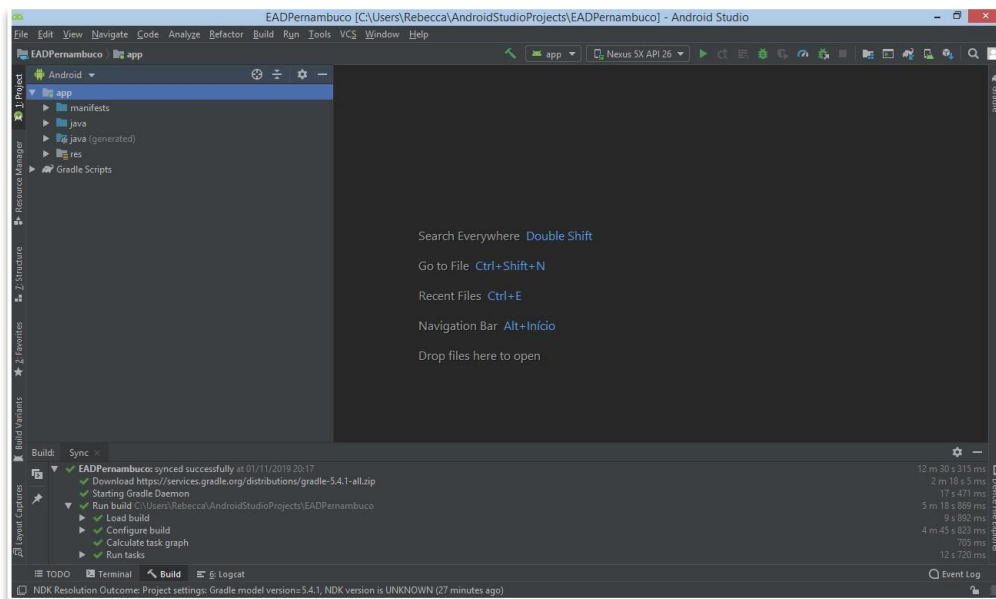


Figura 2.14 - Ícone AVD Manager na janela principal do Android Studio.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual, a barra de ferramentas e a barra de navegação. Na barra de ferramentas, tem-se o ícone com um celular na cor branco e cinza-escuro com uma cabeça de um robô na cor verde.

Logo em seguida tem-se a barra de janela de ferramentas e a janela do editor. E na parte inferior tem-se outra janela de ferramenta e a barra de estado.

Surgirá a tela intitulada *Android Studio Device Manager* (Figura 2.15), clique no botão **+Create Virtual Device** na parte inferior da caixa de diálogo.

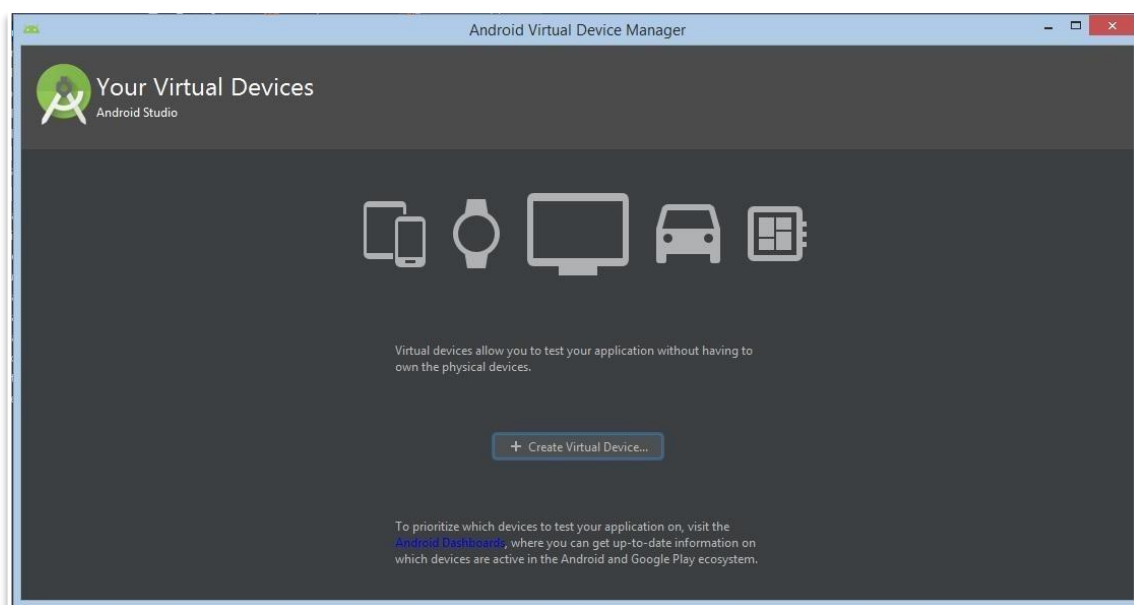


Figura 2.15 - Configuração do Android Virtual Device



Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro, com as figuras de um celular na cor cinza-claro, um relógio na cor cinza-claro, uma tv na cor cinza-claro e um carro na cor cinza-claro, com texto descritivo do Android Virtual Device. E logo abaixo, na parte inferior tem-se um botão com fundo cinza-claro com o texto create virtual device.

A janela *Select Hardware* será exibida.

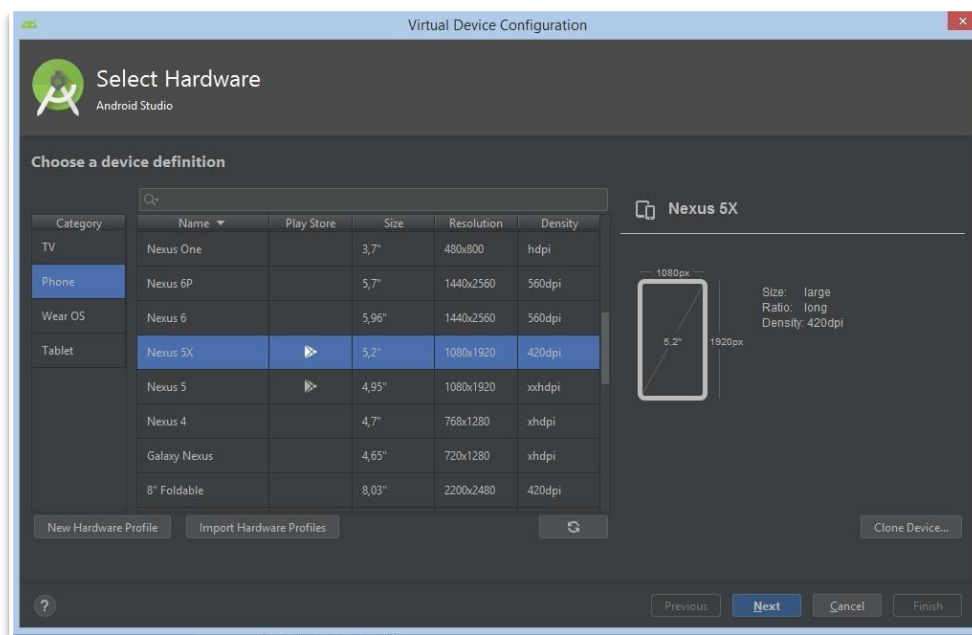


Figura 2.16 – Configuração do Hardware

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro, com texto descritivo para a configuração do hardware. A configuração do hardware será confirmado mediante o click do mouse sobre um dos botões: Next, se deseja continuar a configuração e Cancel, se deseja cancelar a configuração.

O AVD Manager já contém alguns perfis de hardware pré-carregados, nós iremos utilizar o perfil *Nexus 5K*. Para definir o perfil de hardware selecione primeiro a categoria **Phone**, após selecione o perfil **Nexus 5K** e para finalizar clique no botão **Next** (Figura 2.16).

A janela *System Image* será exibida.

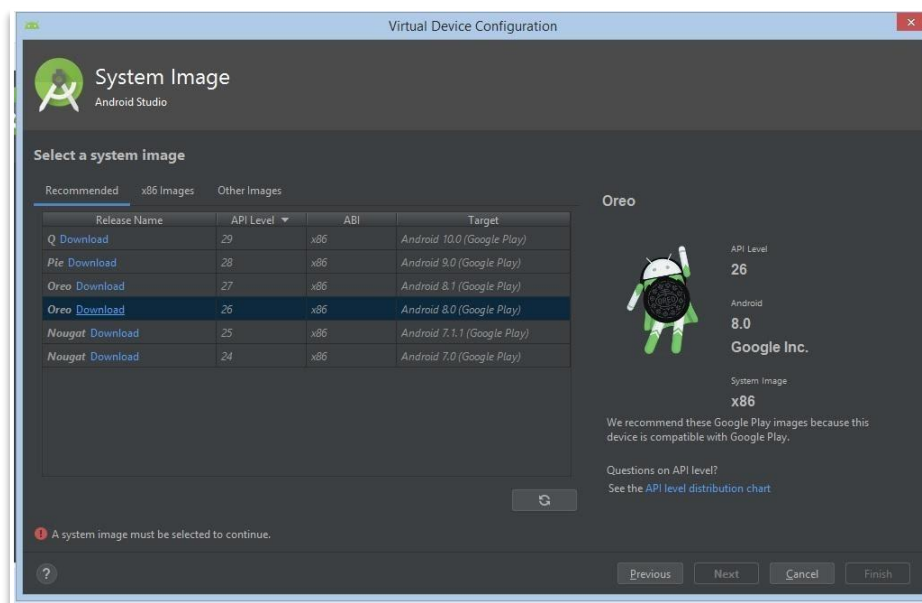


Figura 2.17 – Configuração do Android Oreo 8.0.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro, com texto descritivo para a seleção da versão Android. A configuração do Android Oreo 8.0 será confirmada mediante o click do mouse sobre um dos botões: Next, se deseja continuar a configuração e Cancel, se deseja cancelar a configuração.

Selecione a imagem de sistema com o *Release Name* **Oreo** e o *Target* **Android 8.0** (Google Play), e clique em **Next** (Figura 2.17).

A janela *Verify Configuration* será exibida.

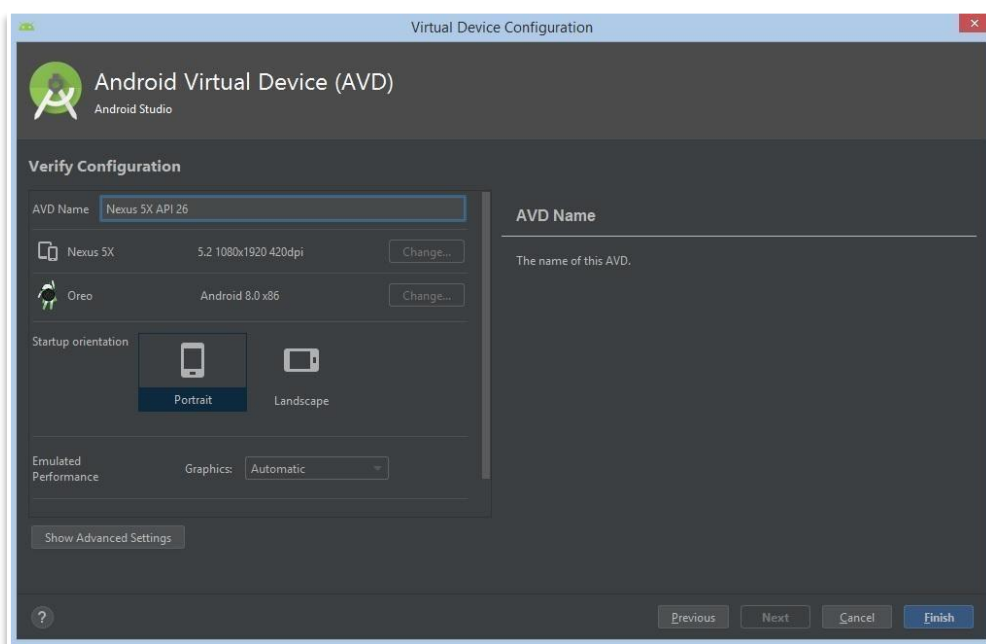


Figura 2.18 – Verificando a configuração do AVD.

Fonte: Android Studio 3.5.2



Descrição da imagem: a imagem apresenta um fundo cinza-escuro, com texto descritivo para a configuração do AVD. A configuração do AVD será confirmada mediante o click do mouse sobre um dos botões: Next, se deseja continuar a configuração e Cancel, se deseja cancelar a configuração.

Altere as propriedades na *Verify Configuration* conforme as propriedades exibidas na Figura 2.18 e clique em **Finish**.



Saiba mais sobre AVD

Ficou curioso? Quer saber mais sobre o AVD e sua configuração? Então acesse:

<https://developer.android.com/studio/run/managing-avds.html?hl=pt-br>

2.2 Hello World – Criando o nosso primeiro projeto

Vamos criar o nosso primeiro projeto Android o “Hello World”? Abra o Android Studio, clique em **File** e selecione a opção **New → New Project** (Figura 2.19).

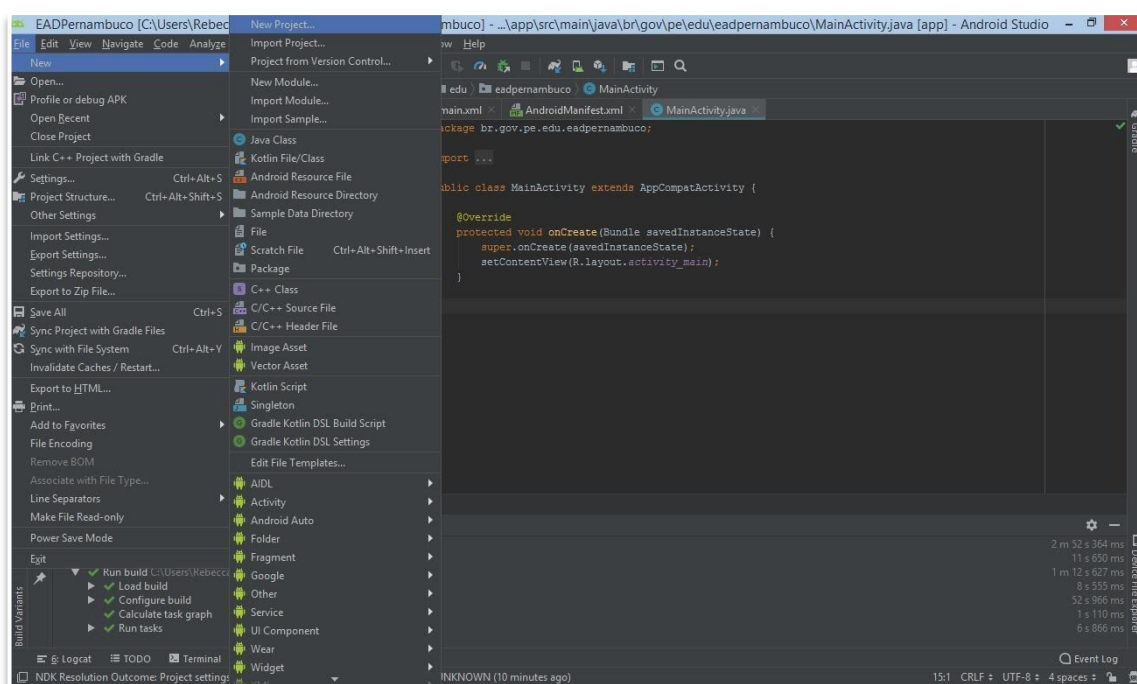


Figura 2.19 – Iniciando um Novo projeto.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual. Um Novo projeto é iniciado mediante o click do mouse sobre File → New → New Project, no menu textual.



Surgirá a janela *Create New Project*, onde será iniciado o processo de configuração de um novo projeto. Para escolher o seu projeto selecione primeiro a opção **Phone and Tablet**, após selecione **Empty Activity** e clique no botão **Next** (Figura 2.20).

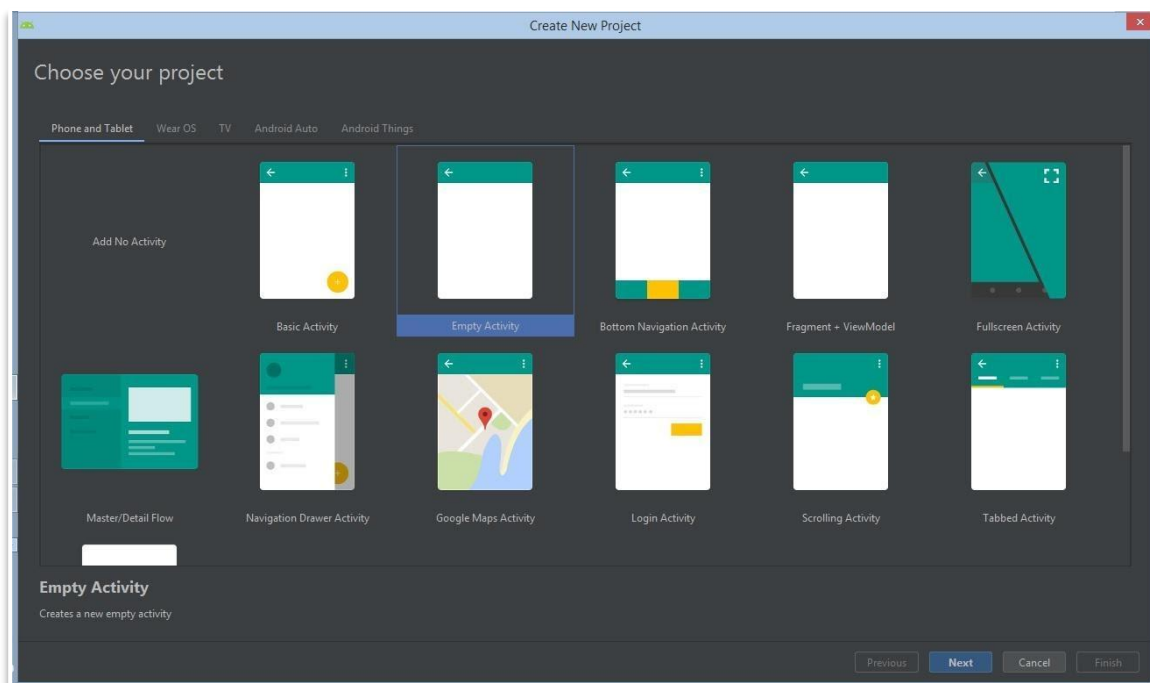


Figura 2.20 – Escolhendo seu Novo projeto.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro, com texto descritivo para escolher seu projeto. Além de dez retângulos em formato paralelo representando os projetos a serem escolhidos. A escolha do projeto será confirmada mediante o click do mouse sobre um dos botões: Next, se deseja continuar a configuração e Cancel, se deseja cancelar a configuração.

A janela *Configure your project* será exibida.

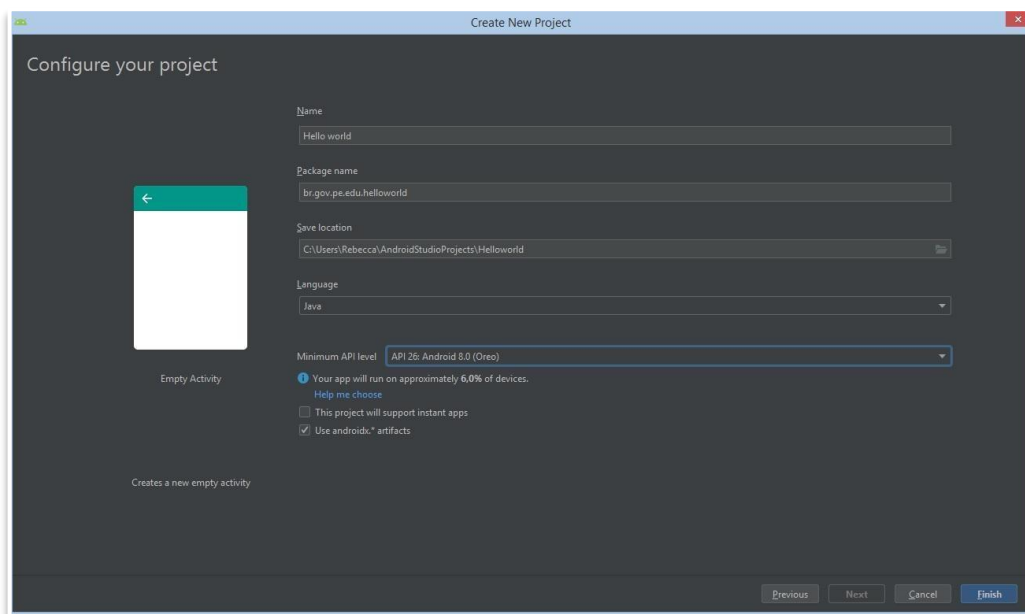


Figura 2.21 – Configurando o seu Novo projeto.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro, com texto descritivo para configurar seu projeto. Em seguida a palavra Name, e logo abaixo uma caixa de texto. A palavra language, e logo abaixo uma caixa de seleção. A palavra Minimum API level, e ao lado uma caixa de seleção. A configuração do novo projeto será confirmada mediante o click do mouse sobre um dos botões: Cancel, se deseja cancelar a configuração, e Finish, quando estiver tudo pronto para criar o projeto.

Nesta janela (Figura 2.21) você irá definir algumas configurações para criar um novo projeto. Vamos a configuração:

1. Em **Name** digite Hello World;
2. Em **Language** selecione Java;
3. Em **Minimum API level** selecione API 26: Android 8.0 (Oreo).

Quando estiver tudo pronto para criar o projeto, clique em **Finish**.

Depois que o projeto for criado, iremos visualizá-lo igual à da Figura 2.22.

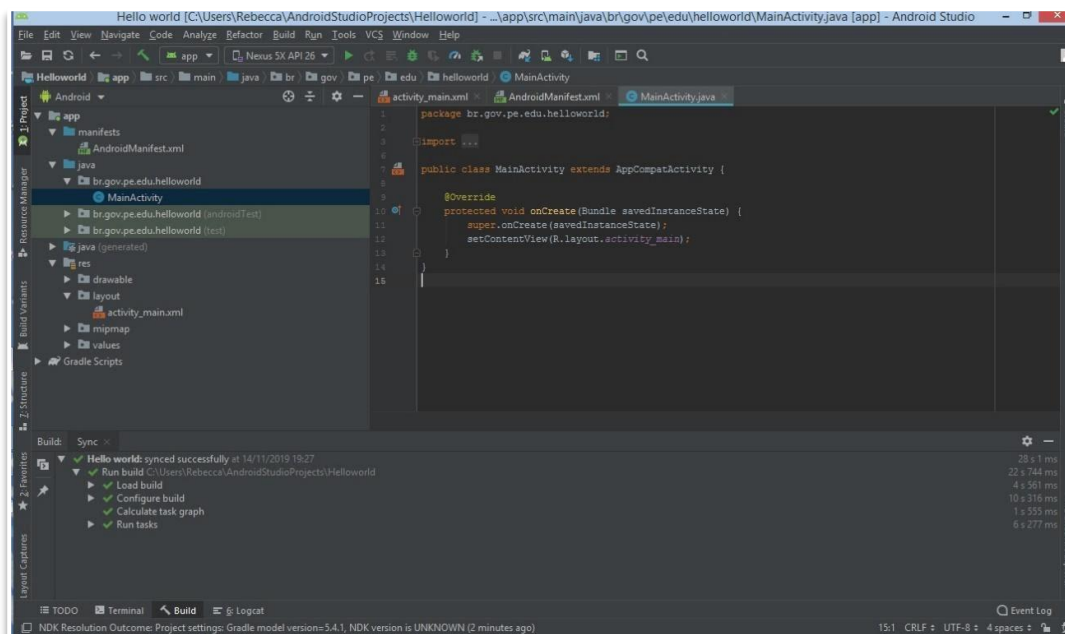


Figura 2.22 – Novo projeto criado.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual, a barra de ferramentas e a barra de navegação. Logo em seguida tem-se a barra de janela de ferramentas e a janela do editor. E na parte inferior tem-se outra janela de ferramenta e a barra de estado.

Para compilar e executar o novo projeto *Hello World*, selecione **Run** → **Run** na barra de menus ou clique em **Run** na barra de ferramentas (Figura 2.23). Uma terceira opção para compilar e executar o novo projeto é selecionar **AVD Manager** na barra de ferramentas. Será exibida a tela *Available Virtual Devices* e no campo *Actions* selecione **Run** (Figura 2.24).

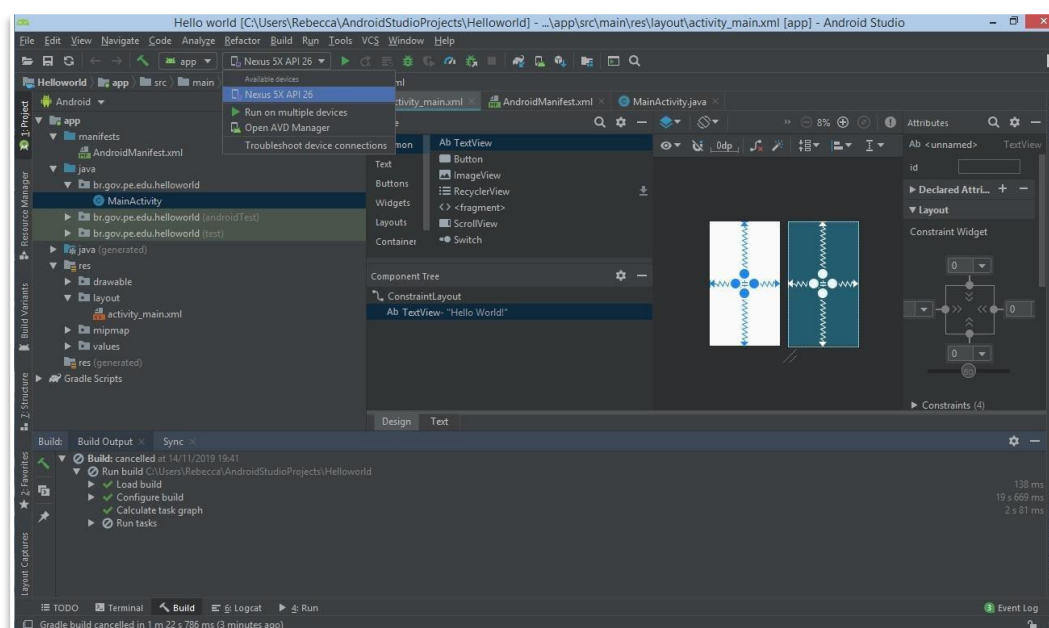




Figura 2.23 – compilar e executar o novo projeto *Hello World*.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual. O processo de compilar e executar o novo projeto é iniciado mediante o click do mouse sobre Run → Run, no menu textual.

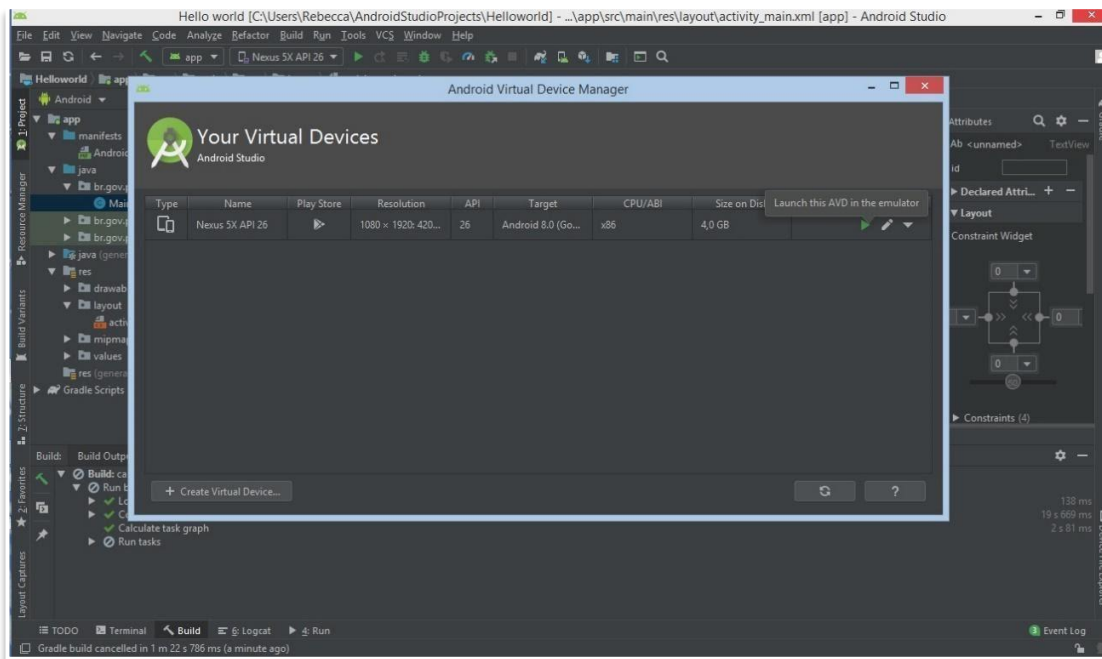


Figura 2.24 – compilar e executar o novo projeto *Hello World*.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual. O processo de compilar e executar o novo projeto é iniciado mediante o click do mouse sobre AVD Manager na barra de ferramentas. Será exibida a tela Available Virtual Devices e no campo Actions selecione Run.

O resultado deve ficar igual ao da Figura 2.25.

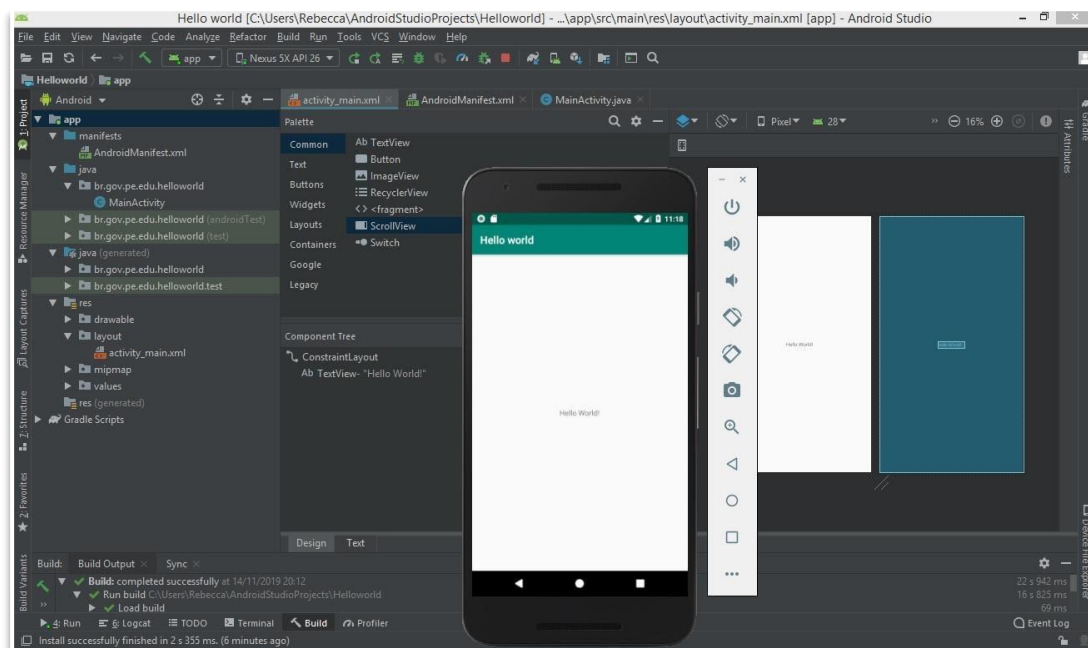


Figura 2.25 – Hello World.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte central tem-se um celular com a palavra *Hello World*.

E aí, estudante, foi legal o caminho até aqui? Você já conhecia algo sobre o ambiente de desenvolvimento de aplicações Android? E sobre os seus componentes? O legal é que até aqui aprendemos sobre:

- Os componentes necessários para o ambiente de desenvolvimento de aplicações Android.
- Instalar o JDK.
- Instalar e configurar do Android Studio.
- Configurar do AVD no Android Studio.
- Criar um projeto no Android Studio.



Pronto para inserir mais informações em seu banco de dados? Acesse o AVA e assista a nossa primeira videoaula **competência 2**.



Agora, acesse o AVA e responda as questões da competência 2.



Ficou com alguma dúvida na competência 2? **Acesse o Fórum - “Competência 2”** para saná-las e discutir com seus colegas sobre os assuntos estudados.

Agora, o nosso próximo passo é mergulhar na competência 3, que trata das variáveis, dos tipos de dados e dos operadores. Esses elementos definem os limites de uma linguagem e determinam o tipo de tarefas às quais ela pode ser aplicada.

Vamos juntos?



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

Vamos iniciar a terceira competência? Nela, iremos apresentar as variáveis. Também conheceremos com detalhes os tipos de dados e estudaremos os operadores.

Porém, antes disso, iremos aprender a criar um projeto Android e conheceremos sua estrutura. Em seguida, falaremos do processo de compilação e execução de um aplicativo.

3.1 Conceitos Básicos

3.1.1 Iniciando um novo projeto com Android Studio

O *Android Studio* é um ambiente de desenvolvimento que facilita a criação de aplicativos *Android* em vários formatos, como celulares, *tablets*, TVs e dispositivos *Wear* (vestíveis). Neste caderno, vamos aprender a criar aplicativos para *Android* com a linguagem Java, focado no desenvolvimento *mobile*.

Vamos criar o nosso projeto *Android*. Execute o *Android Studio*, no menu principal clique em **File** e selecione a opção **New → New Project** (Figura 3.1).

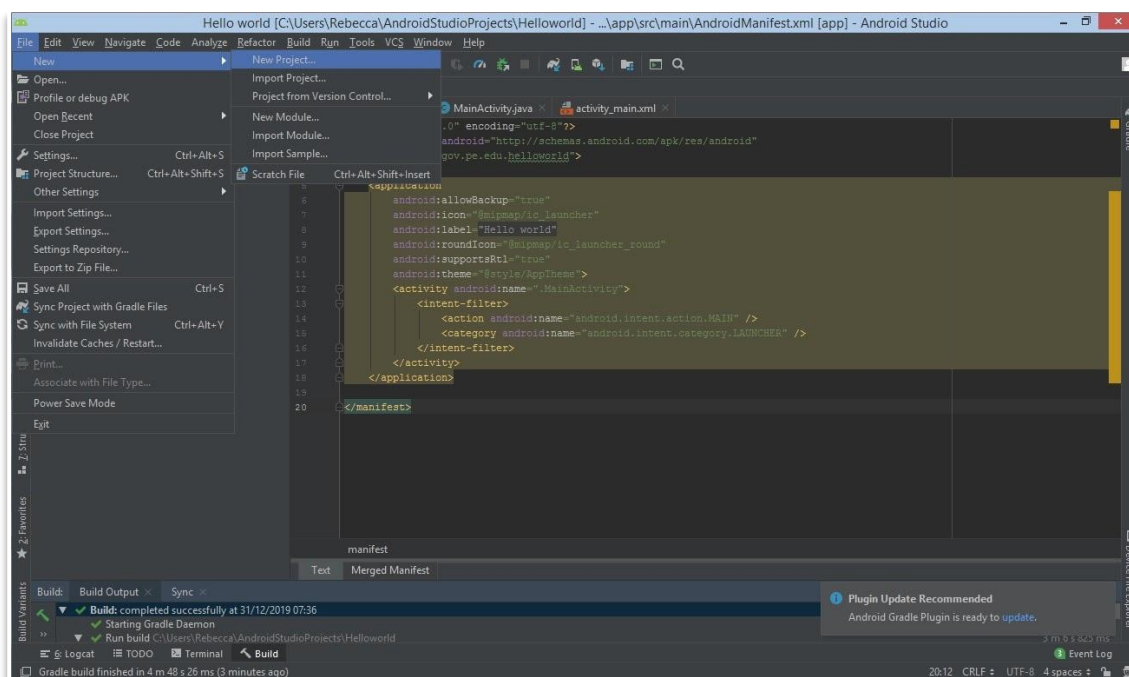


Figura 3.1 - Criando um novo projeto no Android Studio.

Fonte: Android Studio 3.5.2



Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual. Um Novo projeto é iniciado mediante o click do mouse sobre File → New → New Project, no menu textual.

Surgirá a janela *Create New Project*, onde será iniciado o processo de configuração de um novo projeto. Nesta janela é possível adicionar uma *Activity* ao projeto partindo de um template. Para escolher o seu *template* selecione primeiro **Phone and Tablet**, após selecione **Empty Activity** e clique no botão **next** (Figura 3.2).

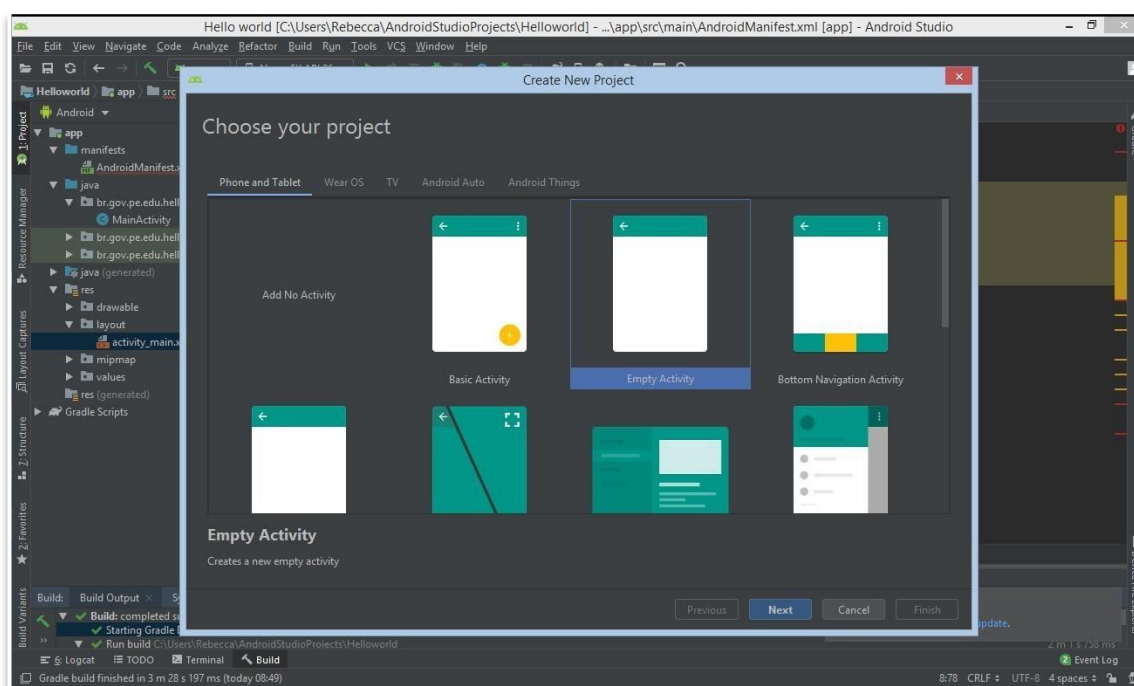


Figura 3.2 - Tela do assistente de criação de um novo projeto.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro, com texto descritivo para escolher seu projeto. Além de dez retângulos em formato paralelo representando os projetos a serem escolhidos. A escolha do projeto será confirmada mediante o click do mouse sobre um dos botões: Next, se deseja continuar a configuração e Cancel, se deseja cancelar a configuração.



O que é um activity?

Uma Activity é responsável por criar uma janela na qual é exibida a interface gráfica do aplicativo.

<https://www.androidpro.com.br/blog/desenvolvimento-android/activity-intro/>

A janela Configure your project será exibida.

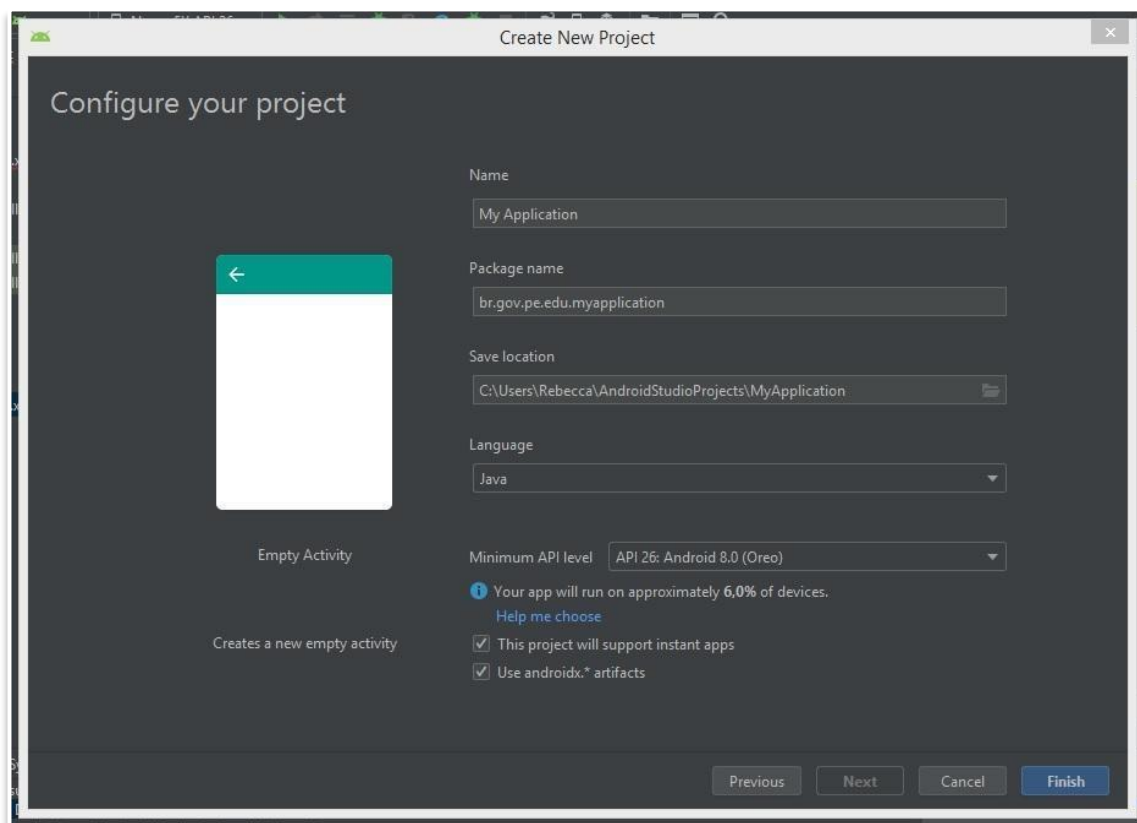


Figura 3.3 - Definindo as configurações iniciais do projeto.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro, do lado esquerdo temos um retângulo representando o projeto escolhido. E do lado temos texto descritivo para configurar seu projeto: a palavra Name, e logo abaixo uma caixa de texto. A palavra language, e logo abaixo uma caixa de seleção de texto. A palavra Minimum API level, e ao lado uma caixa de seleção de texto. A configuração do novo projeto será confirmada mediante o click do mouse sobre um dos botões: Cancel, se deseja cancelar a configuração, e Finish, quando estiver tudo pronto para criar o projeto.

Nesta janela (Figura 3.3) você irá definir algumas configurações para criar um projeto. Vejamos a configuração:

1. Em **Name** digite o nome do seu projeto que aparecerá no aparelho.
2. Em **Package Name** digite o nome do pacote, que define onde ficarão as classes do projeto. Por padrão, esse nome de pacote também se torna o ID do aplicativo, ou seja, não pode haver duas aplicações com o mesmo nome de pacote instalados no aparelho.
3. Em **Save location**, especifique em que local do computador o seu projeto será salvo.



4. Em **Language**, devemos selecionar a linguagem de programação que o *Android Studio* irá usar ao criar um código. Todos os exemplos deste caderno serão desenvolvidos com a linguagem Java, por isso, no campo **Language**, selecione Java.
5. Em **Minimum API level** devemos selecionar a versão mínima do *Android* que o aparelho deve possuir para executar a aplicação. Todos os exemplos apresentados neste caderno devem funcionar no *Android Oreo*, por isso, no campo **Minimum API level**, selecione *API 26: Android 8.0 (Oreo)*.
6. Se o tipo de projeto selecionado for compatível com experiências instantâneas por meio do **Google Play Instant** e você quiser ativá-las para a sua aplicação, marque a caixa ao lado de **This project will support instant apps**. O **Google Play Instant** não será utilizado nos exemplos apresentados neste caderno, desta forma desmarque a caixa ao lado de **This project will support instant apps**.
7. Se você quiser que o projeto use bibliotecas **AndroidX** por padrão, que são substituições aprimoradas de *Android Support Libraries*, marque a caixa ao lado de **Use AndroidX artifacts**. As bibliotecas **AndroidX** não serão utilizadas nos exemplos apresentados neste caderno, desta forma desmarque a caixa ao lado de **Use Androidx.* artifacts**.

Quando estiver tudo pronto para criar o projeto, clique em **Finish**.

Depois que o projeto for criado, iremos visualizá-lo igual à da Figura 3.4.

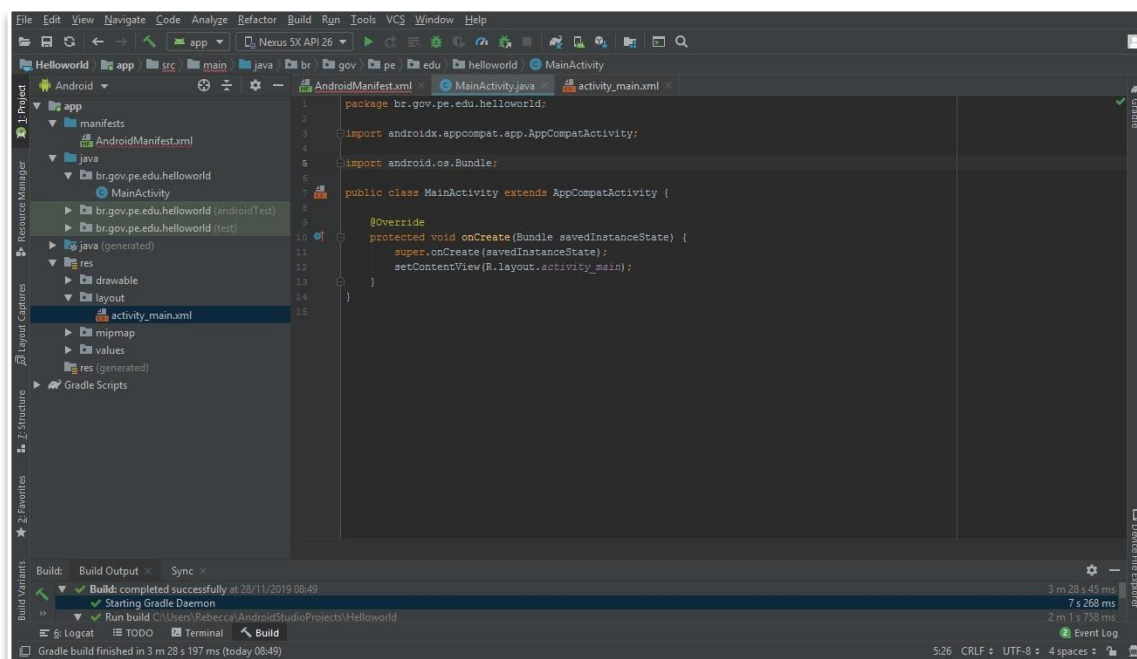


Figura 3.4 – Novo projeto criado.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual, a barra de ferramentas e a barra de navegação. Logo em seguida tem-se a barra de janela de ferramentas e a janela do editor. E na parte inferior tem-se outra janela de ferramenta e a barra de estado.

O que é Google Play Instant?

O Google Play Instant é um serviço que permite que aplicativos sejam iniciados em dispositivos com Android 5.0 sem serem instalados, ou seja, sem realizar o download. Desta forma, você irá experimentar a aplicação Android antes de instalar.

<https://developer.android.com/topic/google-play-instant/overview?hl=pt-br>
<https://www.techtudo.com.br/noticias/2019/08/o-que-e-google-instant-apps-no-android-saiba-como-funciona-o-servico.ghtml>

O que é AndroidX?

O AndroidX é o projeto de código aberto que a equipe Android usa para desenvolver, testar, incluir, controlar a versão e liberar bibliotecas no Jetpack.

<https://developer.android.com/jetpack/androidx/?hl=pt-br>



O que é Android Jetpack?

O Android Jetpack é uma coleção de bibliotecas para facilitar o desenvolvimento de aplicativos Android. Esta coleção de bibliotecas ajudam o desenvolvedor a seguir as práticas mais recomendadas e simplifica tarefas complexas.

<https://developer.android.com/jetpack?hl=pt-br>

3.1.2 Estrutura de um projeto Android

Depois de aprender a criar um projeto, vamos conhecer a estrutura de diretórios e arquivos gerados. Ao iniciar o projeto criado, no menu lateral esquerdo será exibida a estrutura do projeto conforme você pode visualizar na Figura 3.5:

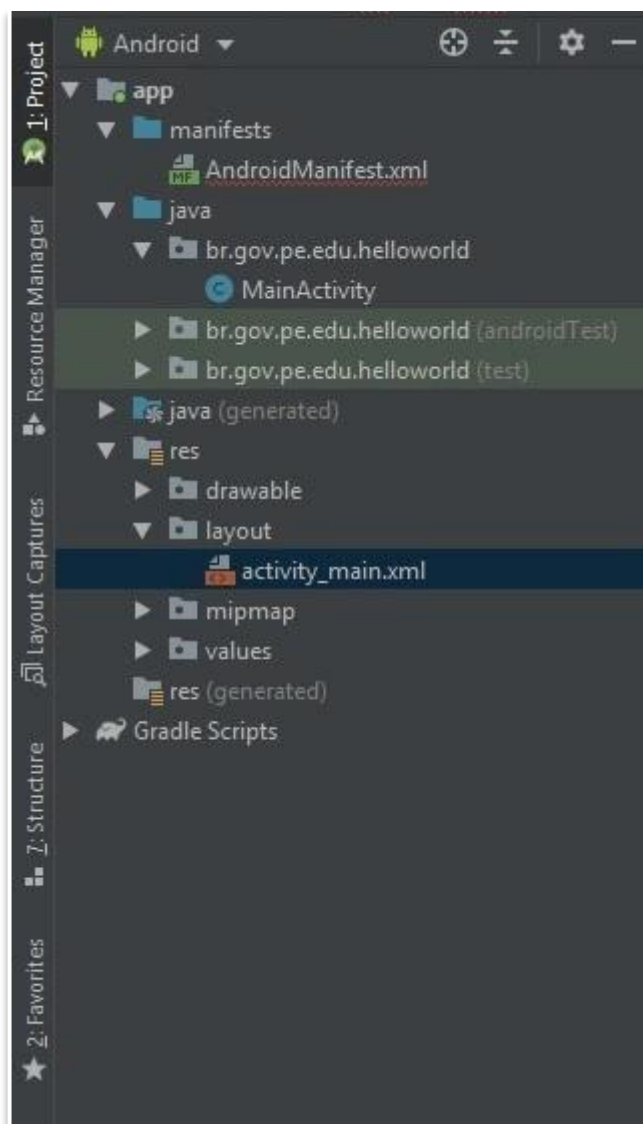




Figura 3.5 – Estrutura de um projeto Android

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que tem-se a janela de ferramentas, com texto descritivo na cor branca. E do lado esquerdo temos a barra de janela de ferramentas contendo botões.

Na parte superior, ao selecionar a opção *Project*, é exibida outra forma de visualização do projeto mais próxima que existe no sistema de arquivos. Alguns arquivos desta estrutura serão detalhados neste caderno, são eles: o *AndroidManifest.xml*, a Classe *MainActivity* e o arquivo de *Layout activity_main.xml*. Vejamos com detalhes cada arquivo desse nas seções a seguir.

3.1.2.1 Arquivo AndroidManifest.xml

O arquivo *AndroidManifest.xml* é a base de um aplicativo e contém boa parte das informações que foram declaradas no assistente de criação do projeto (Figura 3.6).

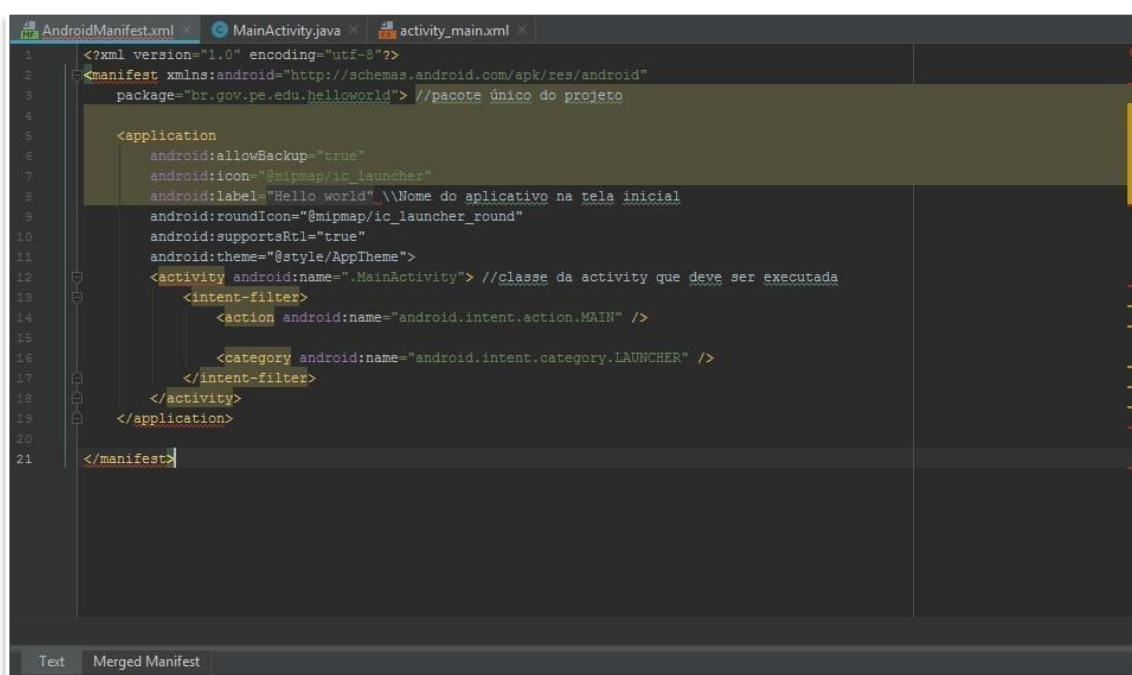


Figura 3.6 - Arquivo AndroidManifest.xml

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se três botões: *AndroidManifest.xml*, *MainActivity.java* e *activity_main.xml*. Logo abaixo tem-se a janela do editor com o código do arquivo *AndroidManifest.xml*.

Vale ressaltar que quando criamos um novo projeto, é gerada a classe *MainActivity*, a qual é configurada automaticamente como a *activity* inicial do projeto no *AndroidManifest.xml*.



Continuando, perceba que no arquivo da Figura 3.6 temos a *tag* `<manifest>` em que é declarado o pacote principal do projeto, utilizando a *tag* `<package>`. O `<package>` é o pacote identificador do projeto e deve ser único.

```
package="com.example.helloworld ">
//pacote único do projeto
```

Por conseguinte, na *tag* `<application>` são adicionadas algumas configurações do aplicativo. A propriedade `android:label="@string/app_name"` informa o nome do aplicativo na tela inicial.

```
android:label="@string/app_name"
// Nome do aplicativo na tela inicial
```



Todas as *activities* das aplicações que serão exibidas para o usuário devem estar declaradas no **AndroidManifest.xml** com a *tag* `<activity>`.

```
<activity android:name=".MainActivity">
// Classe da activity que deve ser executada
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
// A ação MAIN indica que esta activity pode ser executada como a inicial
do app
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

3.1.2.2 Classe MainActivity

A classe *MainActivity* foi configurada pelo *wizard* do Android Studio como a *activity* inicial, portanto, pode ser executada pelo usuário ao clicar no ícone do aplicativo na tela Home do Android.



```
1 package br.gov.pe.edu.helloworld;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13    }
14 }
15
```

Figura 3.7 - Classe MainActivity.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que tem-se a janela do editor com o código do arquivo MainActivity.

Não se esqueça que a classe MainActivity ou qualquer outra *activity* deve ser filha da classe `android.app.Activity` e para isso utilizamos a palavra reservada *extends* conforme código abaixo:

```
public class MainActivity extends AppCompatActivity {
    .....
}
```



Nota: Se no wizard de criação de projetos for selecionado para adicionar a biblioteca de compatibilidade, o Android Studio vai criar a classe MainActivity como filha de AppCompatActivity. A classe AppCompatActivity é uma subclasse de Activity.



Além disso, a classe `android.app.Activity` representa uma tela do aplicativo, sendo responsável por controlar os estados e os eventos da tela. Portanto, para cada tela do aplicativo você criará uma classe-filha de `android.app.Activity`.

3.1.2.3 Arquivo de Layout `activity_main.xml`

No *Android* é possível criar o *layout* da tela em arquivos XML ou utilizar a API via código. Porém, o mais recomendado é criar o *layout* em XML, para separar a lógica de negócios da camada de apresentação. Uma *view* pode ser um simples componente gráfico como botão, *check box*, imagem, entre outros, ou uma *view* complexa, atuando como um gerenciador de *layout*, que pode conter várias *views*-filhas.

A seguir podemos visualizar o código do arquivo `activity_main.xml` (Figura 3.8):

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent" //Largura da tela
6     android:layout_height="match_parent" //Altura da tela
7     tools:context=".MainActivity">
8
9     <TextView //componente (view) que mostra um texto
10         android:id="@+id/Hello World"
11         android:layout_width="wrap_content" //Largura da view
12         android:layout_height="wrap_content" //Altura da view
13         android:text="Hello World!" //Texto que vai aparecer na tela
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintLeft_toLeftOf="parent"
16         app:layout_constraintRight_toRightOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

Figura 3.8 - Arquivo `activity_main.xml`.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que tem-se a janela do editor com o código do arquivo `activity_main.xml`.

No código existem as *tags* `<ConstraintLayout>` e `<TextView>` que correspondem às classes *ConstraintLayout* e *TextView*, respectivamente. A classe *ConstraintLayout* é filha da classe *ViewGroup*. E as classes *ViewGroup* e *TextView* são filhas da classe `android.view.view` do Android. A classe *ConstraintLayout* é um gerenciador de *layout* da plataforma *Android*. Esta classe é similar a classe *LinearLayout*, ou seja, possui as melhores características do *LinearLayout*. Porém, permite a



criação de *layouts* grandes e complexos. Neste exemplo a *tag* <TextView> está contida dentro da *tag* <ConstraintLayout> justamente para compor a interface da tela.



Saiba mais sobre a classe LinearLayout

<https://developer.android.com/guide/topics/ui/layout/linear?hl=pt-BR>



Saiba mais sobre a classe ConstraintLayout

<https://developer.android.com/reference/android/support/constraint/ConstraintLayout>

<https://medium.com/collabcode/implementando-telas-no-android-com-constraint-layout-13a90e44622f>

<https://imasters.com.br/android/como-usar-o-constraint-layout-no-android>



Saiba mais sobre a classe TextView

<https://developer.android.com/reference/android/widget/TextView.html?hl=pt-BR>



Saiba mais sobre a classe ViewGroup

<https://developer.android.com/reference/android/view/ViewGroup>

Saiba mais sobre a classe View

<https://developer.android.com/reference/android/view/View.html>

Saiba Como usar Android Views

<https://www.androidpro.com.br/blog/desenvolvimento-android/android-views-intro/>

3.1.3 Processo de compilação e execução de um projeto Android

Para compilar e executar seu projeto no *Android Studio*, selecione **Run** → **Run** na barra de menus ou clique em **Run** na barra de ferramentas (Figura 3.9). Uma terceira opção para compilar e



executar o novo projeto é selecionar **AVD Manager** na barra de ferramentas. Será exibida a tela *Available Virtual Devices* e no campo *Actions* selecione **Run** (Figura 3.10).

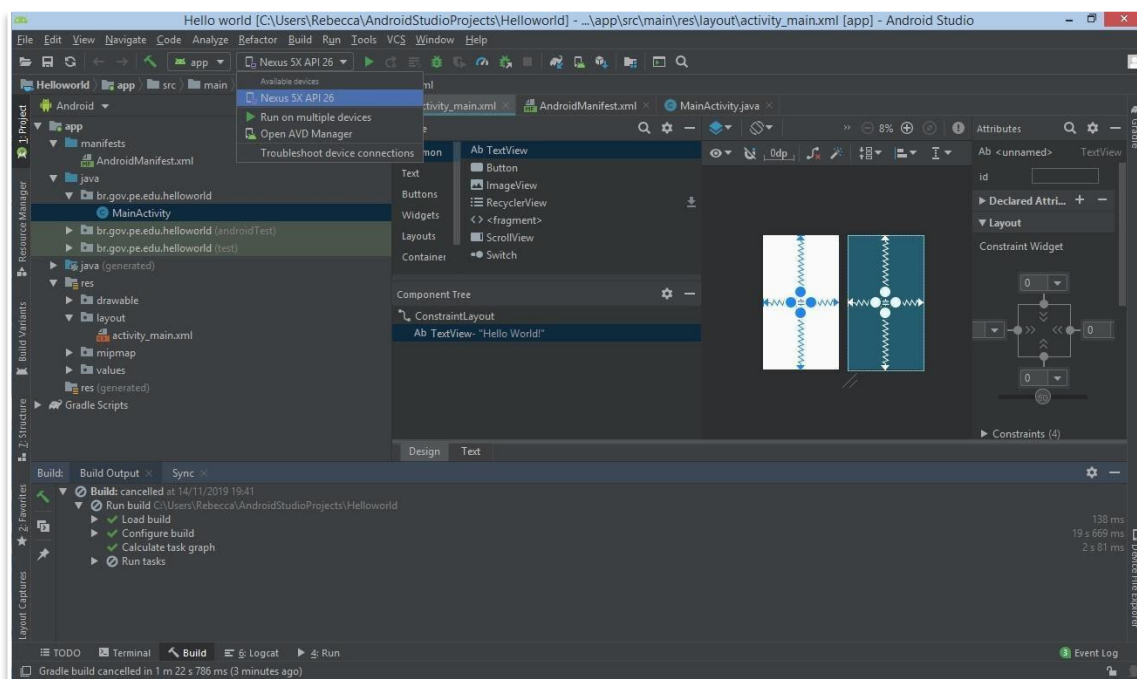


Figura 3.9 – compilando e executando um projeto no Android Studio.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual. O processo de compilar e executar o novo projeto é iniciado mediante o click do mouse sobre **Run** → **Run**, no menu textual.

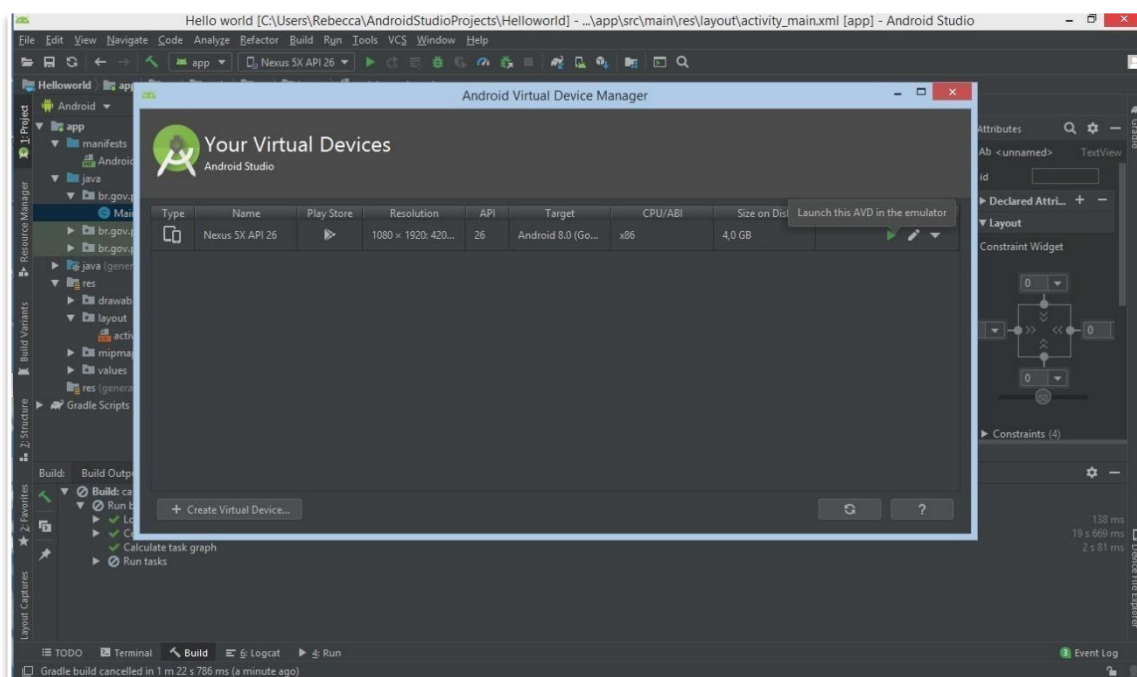


Figura 3.10 – compilando e executando um projeto no **AVD Manager** do Android Studio.



Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte superior tem-se o menu textual. O processo de compilar e executar o novo projeto é iniciado mediante o click do mouse sobre AVD Manager na barra de ferramentas. Será exibida a tela **Available Virtual Devices** e no campo **Actions** selecione **Run**.

O resultado deve ficar igual ao da Figura 3.11:

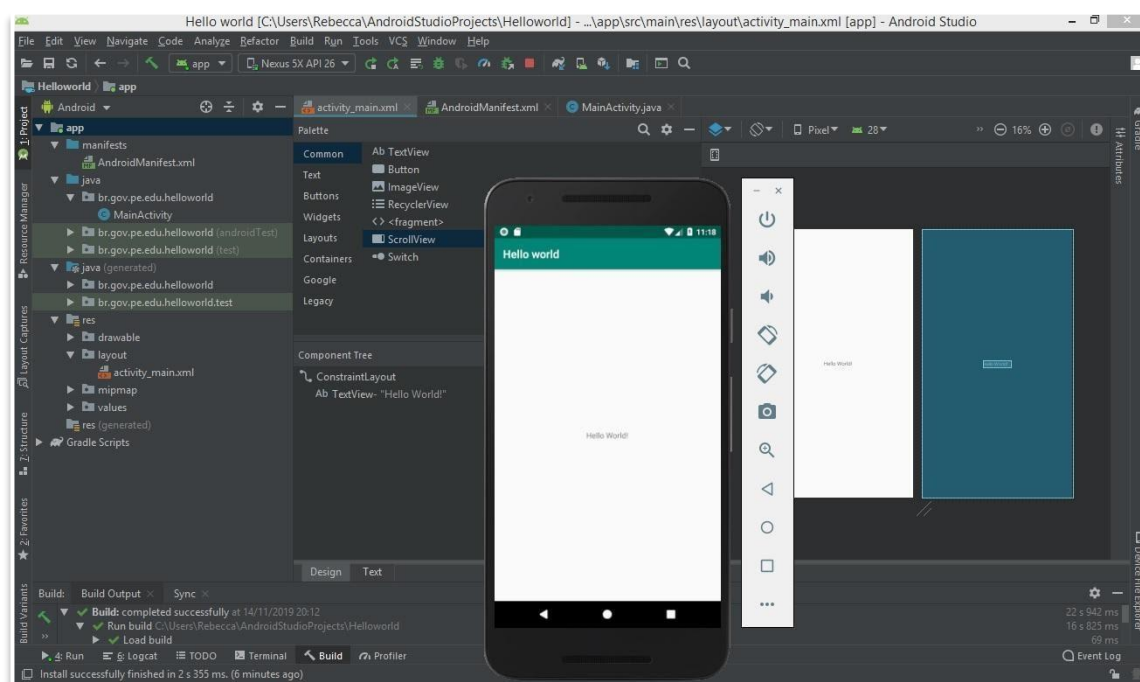


Figura 3.11 – Resultado.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo cinza-escuro em que na parte central tem-se um celular com a palavra Hello World.



Monitorar o processo de compilação

<https://developer.android.com/studio/run#gradle-console>

3.2 Variáveis

As aplicações podem ser vistas como funções que possuem dados de entrada com objetivo de produzir um determinado resultado. Logo, todas as aplicações utilizam algum tipo de



dados. Mas, para que possamos transformar o dado em algo útil, temos que primeiro guardá-los. A estrutura responsável por guardar um valor para ser utilizado depois é a variável.

Talvez nenhuma outra estrutura seja tão importante para uma linguagem de programação quanto a atribuição de um valor a uma variável, a qual é um local nomeado na memória ao qual pode ser atribuído um valor. Tal valor de uma variável é alterável e não fixo.

Por exemplo, vamos criar duas variáveis chamadas `variavel01` e `variavel02` (Figura 3.12):

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        int variavel01; //esta instrução declara uma variável  
        int variavel02; //esta instrução declara uma variável  
  
        variavel01 = 7; //esta instrução atribui 7 a variavel01  
        variavel02 = variavel01 + 1; //esta instrução atribui (7 + 1) a  
        variavel02  
    }  
}
```

Figura 3.12 – Declaração de variáveis.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando as duas variáveis: `variavel01` e `variavel02`

Nota: Em Java todas as variáveis devem ser declaradas antes de serem usadas.



Saiba mais sobre variáveis

<https://www.embarcados.com.br/variaveis-e-constantas/>

Saiba mais sobre variáveis na linguagem de programação Java

<https://www.devmedia.com.br/java-variaveis-e-constantas/38311>

3.3 Tipo de dados

Os tipos de dados são especialmente importantes em Java porque essa é uma linguagem fortemente tipada, ou seja, todas as operações têm a compatibilidade de seus tipos verificada pelo compilador. Para que seja possível fazer a verificação cuidadosa dos tipos, todas as variáveis,



expressões e valores têm um tipo. Não há o conceito de uma variável sem tipo. Vamos entender a seguir alguns tipos de dados da linguagem de programação Java.

3.3.1 Tipo de dado primitivo

Na linguagem Java, existem oito tipos de dados primitivos, também chamados de elementares ou simples, mostrados na Tabela 3.1. O termo primitivo é usado para indicar que esses tipos são valores binários simples.

Tipo	Significado
Boolean	Representa os valores verdadeiro/falso.
Byte	Inteiro de 8 bits.
Char	Caractere.
Double	Ponto flutuante de precisão dupla.
Int	Ponto flutuante de precisão simples.
Long	Inteiro longo.
Short	Inteiro curto.

Tabela 3.1 - Tipo de dados primitivos de Java

Fonte: o autor



Nota: Todos os outros tipos de dados de Java são construídos a partir dos tipos primitivos.

3.3.2 Tipo de dado lógico

São também chamados de dados booleanos. O tipo boolean representa os valores verdadeiro/falso. Java define os valores verdadeiro e falso usando as palavras reservadas **true** e **false**. Desta forma, uma variável ou expressão de tipo boolean terá um desses valores.

Aqui está um exemplo que demonstra o tipo boolean (Figura 3.13):

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        boolean t; //esta instrução declara uma variável do tipo boolean  
        t = true; //esta instrução atribui true a variável t  
    }  
}
```



```
}  
}  
}
```

Figura 3.13 – Declaração da variável boolean.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando variáveis do tipo boolean.

3.3.3 Tipo de dado numérico Inteiros

Java define quatro tipos de inteiros mostrados na Tabela 3.2 abaixo:

Tipo	Tamanho em bits	Intervalo
Byte	8	-128 a 127
Short	16	-32.768 a 32.767
Int	32	-2.147.483.648 a 2.147.483.647
Long	64	-9.223.372.036.854.775.808 a -9.223.372.036.854.775.807

Tabela 3.2 - Tipo de dados inteiros de Java

Fonte: o autor

Os tipos inteiros Byte, Short, int e Long possuem valores de sinal positivo e negativo. Java não suporta inteiros sem sinal, ou seja, somente sinais positivos. O tipo inteiro mais usado é o **Int**. Quando for preciso usar um inteiro com um intervalo maior do que o de **Int** use **Long**. Por exemplo, aqui está um programa que calcula soma de dois valores inteiros (Figura 3.14):

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        int variavel01; // esta instrução declara uma variável  
        int variavel02; // esta instrução declara uma variável  
        int Soma; // esta instrução declara uma variável  
  
        variavel01 = 7; // esta instrução atribuiu 7 a variavel01;  
        variavel02 = variavel01 + 1; // esta instrução atribuiu 7 + 1 a  
variavel02;  
        Soma = variavel01 + variavel02; // esta instrução atribuiu (7 + 8) a  
Soma;  
  
        TextView c = (TextView) findViewById(R.id.campo);
```



```
c.setText("O resultado da soma é: " + Soma);  
  
}
```

Figura 3.14 – Declaração das variáveis int.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando variáveis do tipo int.

A seguir temos a saída gerada por esse programa:

O resultado da soma é: 15

3.3.4 Tipo de dado numérico de ponto flutuante

O tipo de dado ponto flutuante ou número real são representados por números fracionários. Há dois tipos de ponto flutuante, **float** e **double**, como maneira de armazenar números reais. Por padrão um número com ponto flutuante é do tipo **double**, mas se quisermos utilizar menos memória, podemos utilizar o **float** ao invés do **double**. O tipo **float** tem 32 bits e o tipo **double** tem 64 bits. Dos dois o **double** é o mais usado. Por exemplo, aqui está um programa que calcula a média de dois valores de ponto flutuante (Figura 3.15):

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        double variavel01;  
        double variavel02;  
        double Soma;  
        double Media;  
  
        variavel01 = 7.5;  
        variavel02 = 9.8;  
        Soma = variavel01 + variavel02;  
        Media = Soma/2;  
  
        TextView c = (TextView) findViewById(R.id.campo);  
  
        c.setText("O resultado da soma é " + Media);  
    }  
}
```



```
}  
}  
}
```

Figura 3.15 – Declaração das variáveis double.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando variáveis do tipo double.

Ao executar o programa tem-se esta saída gerada por esse programa:

o resultado da média é: 8.6

3.3.5 Tipo de dado de caractere

Em Java, este tipo de dado guarda um número que representa um caractere. Java usa Unicode. O Unicode define um conjunto de caracteres que pode representar todos os caracteres encontrados em todos os idiomas. Char é o tipo de 16 bits sem sinal com um intervalo que vai de 0 a 65.536, podemos armazenar apenas um caractere neste tipo de variável. Aqui está um exemplo que demonstra o tipo char (Figura 3.16):

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        char x;  
        x = 'X';  
  
        TextView c = (TextView) findViewById(R.id.campo);  
  
        c.setText("A variável x contém: " + x);  
  
    }  
}
```

Figura 3.16 – Declaração da variável char.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando variáveis do tipo char.

Vejamos agora a saída gerada do nosso programa ao ser compilado e executado:

A variável x contém: X



Saiba mais sobre tipos de dados

<https://www.embarcados.com.br/tipos-de-dados/>

Saiba mais sobre tipos de dados primitivos em Java

<http://www.bosontreinamentos.com.br/java/tipos-de-dados-primitivos-em-java/>

3.3.6 Tipo de dado complexo: String

Um dos tipos de dados Java mais importantes é o tipo **String**. É esse tipo que define e dá suporte a uma cadeia de caracteres. Em Java, **Strings** são objetos. Por exemplo, na instrução:

```
System.out.println("Em Java, Strings são objetos");
```

Na instrução anterior, a descrição com a string "Em Java, **Strings** são objetos" é convertido automaticamente em um objeto da **String**. É bom ressaltar, no entanto, que a classe **String** é muito grande e aqui só a examinaremos superficialmente.

Você pode construir um **String** como construiria qualquer outro tipo de objeto: usando o comando *new* e chamando o construtor de String. Por exemplo:

```
String game = new String ("Death Stranding");
```

Essa linha cria um objeto **String** chamado **game** que contém o string de caracteres "Death Stranding". Você também pode construir um String a partir de outro. Por exemplo:

```
String jogo = new String ("Death Stranding");  
String kojimaProductions = new String (jogo);
```

Após essa sequência ser executada, objeto **String** chamado **kojimaProductions** também irá conter o string de caracteres "Death Stranding".

Outra maneira de construir um String é:

```
String melhorJogo = new String ("Sekiro: Shadows Die Twice" é eleito o Game do  
Ano no Game Awards 2019");
```



Nesse caso, **melhorJogo** é inicializada com a sequência de caracteres "Sekiro: Shadows Die Twice" é eleito o Game do Ano no Game Awards 2019".

Você também pode concatenar strings, ou seja, realizar a junção de uma string com outra. Por exemplo:

```
String jogo = new String ("Death Stranding");  
String estudio= new String ("Kojima Productions");  
  
String concatenacao = jogo + " é um jogo eletrônico de ação desenvolvido pelo  
estúdio japonês " + estudio;
```

Após essa sequência ser executada, objeto **String** chamado **concatenacao** irá conter o string de caracteres "Death Stranding é um jogo eletrônico de ação desenvolvido pelo estúdio japonês Kojima Productions". Na operação concatenação, utilizamos o símbolo de soma (+) para juntar strings.



Saiba mais sobre o tipos de dado complexo String em Java
<https://www.devmedia.com.br/trabalhando-com-string-string-em-java/21737>

3.4 Separadores

Os separadores são sinais que separam, ou seja, modificam a ordem das operações que podem ou não ser diferentes da comum. Java define quatro separadores, mostrados na Tabela 3.3 abaixo:

Separador	Descrição
;	Ponto e vírgula, o qual é usado para terminar uma instrução.
()	Parênteses , os quais separam as expressões.
[]	Colchetes , que são utilizados para indicar/separar os índices de vetores.
{}	Chaves , as quais separam blocos de programação.

Tabela 3.3 - Separadores.

Fonte: o autor



Na linguagem de programação Java, o separador ponto e vírgula é utilizado para finalizar uma instrução do código, ou seja, cada instrução individual deve ser finalizada com um ponto e vírgula. Já o separador de chaves utiliza um bloco é um conjunto de instruções conectadas logicamente que são delimitados por chaves de abertura e fechamento, ou seja, o início e o fim de cada bloco é indicado pela chave de abertura e pela chave fechamento.



Saiba mais sobre Separadores

<https://pt.wikibooks.org/wiki/Java/Operadores#Separadores>

3.5 Operadores

Um operador é um símbolo que solicita ao compilador que execute uma operação matemática ou lógica específica. Java tem quatro classes gerais de operadores: aritmético, bitwise (bit a bit), relacional e lógico. Nós iremos conhecer agora os operadores aritmético, relacional e lógico.

3.5.1 Operadores Aritméticos

Os operadores aritméticos definidos por Java podem ser visualizados a seguir, na Tabela

3.4 :

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
++	Incremento
--	Decremento

Tabela 3.4 - Operadores aritméticos.

Fonte: o autor



Os operadores soma, subtração, multiplicação e divisão funcionam em Java da mesma maneira que qualquer outra linguagem de programação. Eles podem ser aplicados a qualquer tipo de dado numérico e podem ser usados em objetos do tipo *char*.

Em Java o operador % (módulo ou resto) gera o resto de uma divisão. Pode ser aplicado a tipos inteiros e de ponto flutuante. Por exemplo, o resto da divisão de 10 por 3, representado em Java como (10 % 3) é igual a 1.

Aqui está um exemplo (Figura 3.17) que demonstra o operador módulo:

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        int modulo;  
        modulo = 10 % 3;  
  
        TextView c = (TextView) findViewById(R.id.campo);  
  
        c.setText("A variável modulo contém o valor: " + modulo);  
  
    }  
}
```

Figura 3.17 – Demonstração do operador módulo.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando o operador módulo.

Ao executar este programa será gerada esta saída:

A variável modulo contém: 1

3.5.1.1 Incremento e decremento

Em Java, o operador de incremento adiciona 1 ao seu operando e o operador de decremento subtrai 1. Por exemplo:

```
i = i + 1;
```

é o mesmo que:

```
i++;
```

e



```
i = i - 1;
```

é o mesmo que:

```
i--;
```

Tanto o operador de incremento quanto o operador de decremento podem preceder (prefixar) ou vir após (posfixar) o operando. Por exemplo:

```
i = i + 1;
```

pode ser escrito como:

```
++i; //forma prefixada
```

ou como:

```
i++; //forma posfixada
```

Quando um operador de incremento ou decremento precede seu operando, Java executa a operação correspondente antes de obter o valor do operando a ser usado pelo resto da expressão. Se o operador vier após o seu operando, Java obterá o valor do operando antes dele ser incrementado ou decrementado. Por exemplo nas Figuras 3.18 e Figura 3.19 com a demonstração do operador módulo incrementado ou decrementado:

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        int x;  
        int y;  
  
        x = 10;  
        y = ++x; // Java executa a operação correspondente antes de obter o valor  
do operando  
  
        TextView c = (TextView) findViewById(R.id.campo);  
  
        c.setText("O resultado de y é: " + y);  
  
    }  
}
```

Figura 3.18 – Demonstração do operador módulo incrementado



Fonte: o autor

Descrição: a imagem apresenta um fundo preto com o código utilizando o operador de incremento de forma prefixada.

Aqui está a saída gerada por esse programa:

O resultado de y é: 11

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        int x;  
        int y;  
  
        x = 10;  
        y = x++; // Java obterá o valor do operando antes dele ser incrementado  
  
        TextView c = (TextView) findViewById(R.id.campo);  
  
        c.setText("O resultado de y é: " + y);  
  
    }  
}
```

Figura 3.19 – Demonstração do operador módulo decrementado

Fonte: o autor

Descrição: a imagem apresenta um fundo preto com o código utilizando o operador de incremento de forma posfixada.

Vejamos a saída deste programa:

O resultado de y é: 10

Nos dois casos, x é igual a 11. A diferença é quando isso ocorre.



Saiba mais sobre as operações aritméticas
<https://www.embarcados.com.br/operacoes-aritmeticas/>



3.5.2 Operadores Relacionais e Lógicos

O operador relacional se refere aos relacionamentos que os valores podem ter uns com os outros. Os operadores relacionais são mostrados na Tabela 3.5 abaixo:

Operador	Significado
==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Tabela 3.5 - Operadores relacionais.
Fonte: o autor

Já o operador lógico se refere às maneiras como os valores verdadeiro e falso podem estar conectados. Estes, fazem operações com os operandos do tipo boolean e o resultado da operação lógica é também do tipo boolean. Já que os operadores relacionais produzem resultados verdadeiros ou falsos, possuem relação com o resultado dos operadores lógicos. Logo, o resultado dos operadores relacionais e lógicos é um valor booleano.

Vejamos os operadores lógicos mostrados na Tabela 3.6 abaixo:

Operador	Significado
&	AND
	OR
^	XOR (exclusive OR)
	OR de curto-circuito
&&	AND de curto-circuito
!	NOT

Tabela 3.6 - Operadores lógicos.
Fonte: o autor

Os operadores lógicos **&**, **|**, **^** e **!** dão suporte às operações lógicas básicas **AND**, **OR**, **XOR** e **NOT**. Observe na Tabela 3.7 os resultados das comparações.



a	b	a & b	a b	a^b	!a
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

Tabela 3.7 - Tabela verdade.
Fonte: o autor

Na tabela acima, é mostrado o resultado das operações lógicas básicas **AND**, **OR**, **XOR** (exclusive OR) e **NOT**. Na operação **exclusive OR**, pode-se observar que o resultado mostrado é diferente do **OR**. Isso ocorre porque o resultado de uma operação **exclusive OR** é verdadeiro quando exatamente um, e apenas um, operando é verdadeiro.



Saiba mais sobre as operações relacionais e lógicas
<https://www.embarcados.com.br/operacoes-relacionais-e-logicas/>

3.5.3 Operadores de atribuição

O operador de atribuição é o sinal de igual simples (=) . Esta é a sua forma geral em Java:

variavel = expressao;

Onde, o tipo de variável deve ser compatível com o tipo de expressão. O operador de atribuição permite a criação de uma cadeia de atribuições. Por exemplo:

```
int a, b, c;  
a = b = c = 8; //configura a, b e c com o valor 8.
```

3.5.3.1 Operadores de atribuições abreviadas

As atribuições abreviadas são operadores especiais que simplificam a codificação de certas instruções de atribuição. Por exemplo:

x = x + 8;

pode ser escrita, com o uso da atribuição abreviada em Java, como:

x += 8;



Vale ressaltar que o par de operadores += solicita ao compilador que atribua a x o valor de x mais 8.

Observe na Tabela 3.8 os operadores aritméticos de atribuição abreviada:

Significado	Par de Operadores
Soma	x += 8;
Subtração	x -= 8;
Multiplicação	x *= 8;
Divisão	x /= 8;
Módulo	x %= 8;

Tabela 3.8 - Operadores de atribuição abreviada

Fonte: o autor

Descrição:

Os operadores combinam uma operação com uma atribuição, e consequentemente são chamados de operadores de atribuição composta.

E aí, estudante, foi legal o caminho até aqui? Você já conhecia algo sobre criar, compilar e executar um projeto *Android*? E sobre as variáveis, os tipos de dados e os operadores em Java? O legal é que até aqui aprendemos sobre:

- Criar um projeto com o *Android Studio*.
- A estrutura de um projeto *Android*.
- O processo de compilação e execução de um projeto *Android*.
- Variáveis
- Tipos de dados
- Operadores



Pronto para inserir mais informações em seu banco de dados?
Acesse o AVA e assista a nossa primeira videoaula competência 3.



Agora, acesse o AVA e responda as questões da competência 3.



Ficou com alguma dúvida na competência 3? Acesse o Fórum - “**Competência 3**” para saná-las e discutir com seus colegas sobre os assuntos estudados.

Agora, o nosso próximo passo é mergulhar na competência 4, que trata das estruturas de controle condicional. Vamos lá?



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

Vamos iniciar a quarta competência? Nela, iremos apresentar as estruturas de controle condicional. Você aprenderá as instruções que controlam o fluxo de execução do programa e as de seleção que incluem as estruturas **if** e **switch**.

4.1 Estrutura if

Podemos executar seletivamente parte de um programa com o uso da instrução condicional nomeada como **if**. Esta instrução na linguagem Java funciona de maneira semelhante à instrução **if** de qualquer outra linguagem. Sua estrutura mais simples é mostrada abaixo:

if (condição) instrução;

Vamos entender essa estrutura:

if: também pode ser traduzido para o português como “se” e indica que uma situação condicional será estabelecida;

condição: é um parâmetro que recebe uma expressão booleana, a qual resulta em dois valores, verdadeiro ou falso.

instrução: especifica um bloco de programação e funciona a partir do resultado da condição. Se a condição for verdadeira, a instrução será executada. Se a condição for falsa, a instrução será ignorada.

Para demonstrar **if**, iremos criar e desenvolver um jogo de adivinhação simples. Na primeira versão do jogo, o programa pede ao jogador um número entre 0 e 9. Se o jogador pressionar o número 7 no teclado, o programa responderá exibindo a mensagem “**O número da adivinhação está certo**”. O código pode ser visualizado na Figura 4.1 abaixo:



```
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 adivinhacao = 7;  
  
        if (adivinhacao == 7) {  
            c.setText("o numero da adivinhação está certo");  
        }  
    }  
}
```

Figura 4.1 - Estrutura if no Android Studio.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando a estrutura if.

Esse programa interage com o jogador e então lê um número no teclado. Usando uma instrução **if**, ele compara o número com a resposta que, nesse caso, é 7. Se 7 for inserido, a mensagem **“O número da adivinhação está certo!”** será exibida.

Esse programa interage com o jogador e então lê um número no teclado. Usando uma instrução **if**, ele compara o número com a resposta que, nesse caso, é 7. Se 7 for inserido, a mensagem **“O número da adivinhação está certo!”** será exibida.

4.2 Estrutura if-else

A forma completa do if é:

*if (condição) instrução;
else instrução;*

Onde os alvos de **if** e **else** são instruções individuais. A cláusula **else** é opcional. Os alvos tanto de **if** quanto de **else** podem ser blocos de instrução. A forma geral de if, usando blocos de instruções é:

*if (condição) {
 sequência de instruções }
else {
 sequência de instruções }*



Na segunda versão do jogo, iremos usar **else** para exibir a mensagem **“O número da adivinhação está errado”** quando o número 9 é digitado no teclado. O código pode ser visualizado na Figura 4.2 abaixo:

```
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 adivinhacao = 9;  
  
        if (adivinhacao == 7) {  
            c.setText("o numero da adivinhação está certo");  
        }  
        else {  
            c.setText("o numero da adivinhação está errado");  
        }  
    }  
}
```

Figura 4.2 - Estrutura if-else no Android Studio.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando a estrutura if-else.

Esse programa interage com o jogador e então lê um número no teclado. Usando uma instrução **if**, ele compara o número com a resposta que, nesse caso, é 7. Se 7 for inserido, a mensagem **“O número da adivinhação está certo”** será exibida. Caso contrário, a cláusula **else** será executado, e a mensagem **“O número da adivinhação está errado”** será exibida.

Esse programa interage com o jogador e então lê um número no teclado. Usando uma instrução **if**, ele compara o número com a resposta que, nesse caso, é 7. Se 7 for inserido, a mensagem **“O número da adivinhação está certo”** será exibida. Caso contrário, a cláusula **else** será executado, e a mensagem **“O número da adivinhação está errado”** será exibida.

4.3 Estrutura if-else aninhada



Um *if aninhado* é uma instrução **if** que possui relação com outro **if** ou **else**. Em Java, uma instrução **else** será sempre referente à instrução **if** mais próxima que estiver dentro do mesmo bloco e ainda não estiver associada a um **else**.

Na terceira versão do jogo, iremos usar um **if-else aninhado** para melhorar ainda mais o jogo de adivinhação. Esse acréscimo irá fornecer um palpite exibindo ou a mensagem “**A resposta é um número menor!**” caso o número seja maior que 7, ou a mensagem “**A resposta é um número maior!**” caso o número seja menor que 7. O código pode ser visualizado na Figura 4.3 abaixo:

```
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 adivinhacao = 5;
        int resposta = 9;

        if (adivinhacao == resposta){
            c.setText("O número da adivinhação está certo");
        } else {

            //Este é um if-else aninhado
            if (adivinhacao > resposta){
                c.setText("a resposta é um número menor");
            } else {
                c.setText("a resposta é um número maior");
            }
        }
    }
}
```

Figura 4.3 - Estrutura *if aninhado* no Android Studio.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando a estrutura **if-else aninhado**.

Esse programa interage com o jogador e então lê um número no teclado. Usando uma instrução **if**, ele compara o número com a resposta que, nesse caso, é 7. Se 7 for inserido, a mensagem “**O número da adivinhação está certo**” será exibida. Caso contrário, **else** será executado. Na cláusula **else**, será executado o **if-else aninhado**, o qual irá fornecer um palpite.

Esse programa interage com o jogador e então lê um número no teclado. Usando uma instrução **if**, ele compara o número com a resposta que, nesse caso, é 7. Se 7 for inserido, a mensagem



“O número da adivinhação está certo” será exibida. Caso contrário, **else** será executado. Na a cláusula **else**, será executado o **if-else aninhado**, o qual irá fornecer um palpite.

4.4 Estrutura *Switch*

A instrução *switch* fornece uma ramificação com vários caminhos. Logo, ela permite que o programa faça uma seleção entre várias alternativas. Funciona desta forma: o valor de uma expressão é verificado sucessivamente em uma lista de constantes. Quando uma ocorrência é encontrada, a sequência de instruções associada a essa ocorrência é executada. A forma geral da estrutura *switch* é:

```
switch(expressão){  
    case constante1:  
        sequência de instruções  
        break;  
    case constante2:  
        sequência de instruções  
        break;  
    case constante3:  
        sequência de instruções  
        break;  
    ..  
    default:  
        sequência de instruções  
}
```

A sequência de instruções *default* é executada quando nenhuma constante *case* coincide com a expressão.

Para demonstrar **switch**, iremos criar e desenvolver um jogo simples para RPGs. Neste jogo, o programa irá pedir ao jogador um número entre 1 e 6, que corresponde a um dado de seis faces, e irá informar uma classe de personagem. Assim, ao pressionar no teclado o número de face 2, o programa responderá exibindo uma mensagem com a face do dado escolhida e uma mensagem com a classe que será atribuída ao personagem do jogador no RPG. O código pode ser visualizado na Figura 4.4 abaixo:

```
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 faceDado = 2;

switch (faceDado) {
    case 1:
        c.setText("O número escolhido da face do dado foi: " + faceDado);
        c.setText("Wow! A classe do seu personagem é: Guerreiro");
        break;
    case 2:
        c.setText("O número escolhido da face do dado foi: " + faceDado);
        c.setText("Wow! A classe do seu personagem é: Berserker");
        break;
    case 3:
        c.setText("O número escolhido da face do dado foi: " + faceDado);
        c.setText("Wow! A classe do seu personagem é: Assassino");
        break;
    case 4:
        c.setText("O número escolhido da face do dado foi: " + faceDado);
        c.setText("Wow! A classe do seu personagem é: Ninja");
        break;
    case 5:
        c.setText("O número escolhido da face do dado foi: " + faceDado);
        c.setText("Wow! A classe do seu personagem é: Arqueiro");
        break;
    case 6:
        c.setText("O número escolhido da face do dado foi: " + faceDado);
        c.setText("Wow! A classe do seu personagem é: Mago");
        break;
    default:
        c.setText("What!?! Não lembra os valores de um D6?");
}
}
```

Figura 4.4 - Estrutura switch no Android Studio.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando a estrutura switch.

Tecnicamente, a instrução *break* é opcional. Quando encontrada dentro da sequência de instruções de um *case*, a instrução *break* faz o fluxo do programa sair da instrução *switch* e continuar na próxima instrução externa.

E aí, estudante, foi legal o caminho até aqui? Você já conhecia algo sobre as estruturas de controle condicional? E sobre as estruturas **if** e **switch** em Java? O legal é que até aqui aprendemos sobre:

- As estruturas de controle condicional.
- A forma completa da estrutura **if**.

Competência 04



- Usar a instrução **if**.
- A forma completa da estrutura **switch**.
- Usar a instrução **switch**.



Pronto para inserir mais informações em seu banco de dados?
Acesse o AVA e assista a nossa primeira videoaula competência 4.



Agora, acesse o AVA e responda as questões da competência 4.



Ficou com alguma dúvida na competência 4? Acesse o Fórum - “Competência 4” para saná-las e discutir com seus colegas sobre os assuntos estudados.

Agora, o nosso próximo passo é mergulhar na competência 5, que trata das estruturas de repetição. Vamos juntos?



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

Vamos iniciar a quinta competência? Nela, iremos aprender as instruções que controlam o fluxo de execução do programa, as instruções de interação que incluem os laços **for**, **while** e **do-while**.

5.1 Estrutura For

Podemos executar repetidamente uma sequência de código criando um laço. Java fornece um grupo poderoso de estrutura de laços. A que examinaremos primeiro é o laço **for**. A forma mais simples do laço **for** é mostrada a seguir:

```
for(inicialização; condição; interação) instrução;
```

Para a repetição de um bloco, a forma geral é:

```
for(inicialização; condição; interação)  
{  
sequência de instruções;  
}
```

Onde:

- A *inicialização* do laço define uma variável de controle de laço com um valor inicial.
- A *condição* é uma expressão booleana que testa a variável de controle de laço. Se o resultado desse teste for verdadeiro, o laço **for** continuará a iterar. Se for falso, o laço será encerrado.
- A *interação* determina como a variável de laço é alterada sempre que o laço itera.

Para demonstrar **for**, iremos inicialmente pensar em como você irá guiar o seu personagem no mundo. Onde, o laço de repetição irá fazer seu o personagem andar x passos em uma direção ou y passos em outra direção. Por exemplo, como podemos guiar o personagem para “andar cinco passos para o leste”? O código pode ser visualizado na Figura 5.1 abaixo:

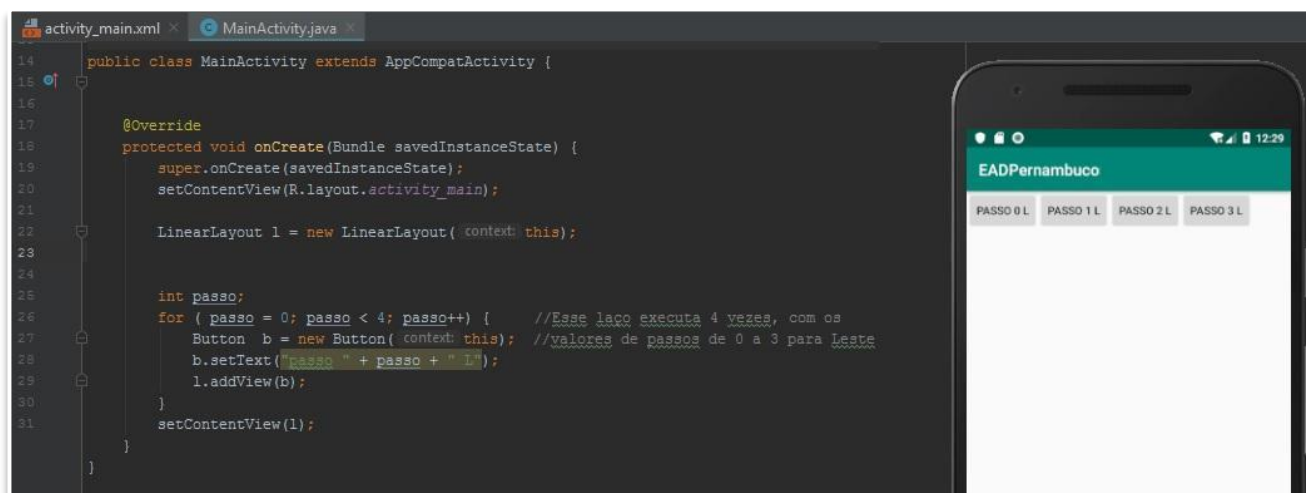


Figura 5.1 – Estrutura for no Android Studio.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando a estrutura for. No lado esquerdo tem-se um celular com as palavras PASSO 0L, PASSO 1L, PASSO 2L, PASSO 3L.

Nesse exemplo, **passo** é a variável de controle do laço. Ela é configurada com zero na parte de *inicialização* do **for**. No começo de cada iteração (inclusive a primeira), o teste condicional **passo < 4** é executado. Se o resultado desse teste for verdadeiro, serão executadas as instruções **b.setText()** e **l.addView()**, e então a parte de iteração do laço, que aumentará **passo** em uma unidade. Esse processo continua até o teste condicional ser falso, momento em que a execução é retomada no final do laço.

5.2 Estrutura While

Outro laço Java é o while. A forma geral do laço while é:

while (condição) instrução;

Onde:

- A *instrução* pode ser um único trecho de código ou um bloco de instruções.
- A *condição* pode ser qualquer expressão booleana válida. O laço se repete enquanto a condição é verdadeira. Quando a condição se torna falsa, o controle do programa passa para a linha imediatamente posterior ao laço.

Para demonstrar o laço **while**, iremos novamente guiar o seu personagem no mundo. Onde, o laço de repetição irá fazer seu o personagem andar x passos em uma direção ou y



passos em outra direção. Por exemplo, como podemos guiar o personagem para “andar quatro passos para o oeste”? O código pode ser visualizado na Figura 5.2 abaixo:

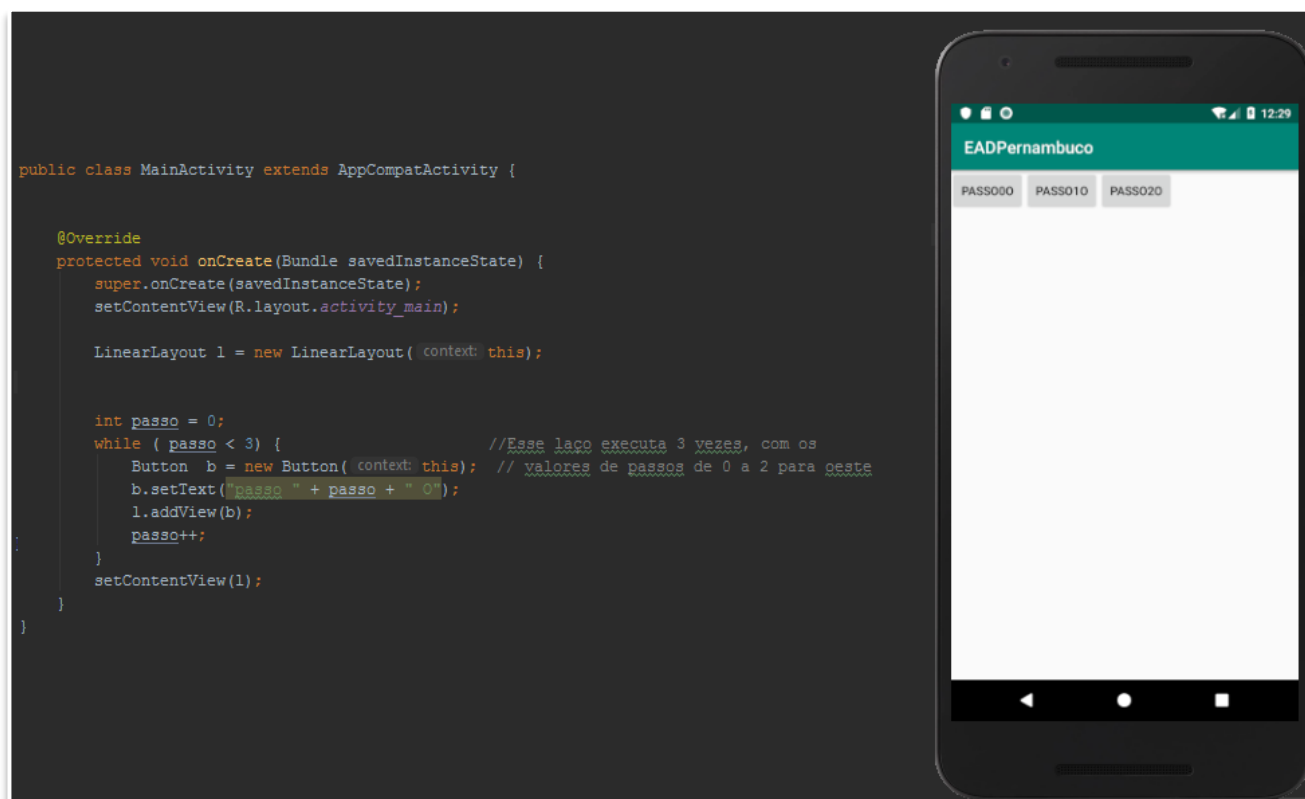


Figura 5.2 – Estrutura while no Android Studio.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando a estrutura while. No lado esquerdo tem-se um celular com as palavras PASSO0, PASSO1, PASSO2.

No exemplo, a variável **passo** é inicializado com 0. A cada passagem pelo laço, a mensagem é exibida e a variável **passo** é incrementada. Esse processo continua até **passo** ser maior que 4.

Como no laço **for**, **while** verifica a expressão condicional no início do laço, ou seja, o código do laço pode não ser executado. Isso elimina a necessidade de execução de um teste separado do laço.

5.3 Estrutura Do-While

O último dos laços Java é o **do-while**. Diferentemente dos laços **for** e **while**, em que a condição é testada no início do laço, o laço **do-while** verifica sua condição no fim do laço, ou seja, um laço **do-while** será sempre executado pelo menos uma vez. A forma geral do laço do-while é:



```
do {  
    instruções;  
} while(condição);
```

Embora as chaves não sejam necessárias quando há apenas uma instrução presente, elas são usadas com frequência para melhorar a legibilidade da estrutura **do-while**, evitando, assim, confusão com **while**. O Laço do-while é executado enquanto a expressão condicional for verdadeira.

Para demonstrar o laço **do-while**, iremos novamente guiar como guiar um personagem em diversas direções ~~e seu personagem no mundo~~. Onde, o laço de repetição irá fazer seu o personagem andar x passos em uma direção ou y passos em outra direção. Por exemplo, como podemos guiar o personagem para “voltar três passos para o leste”? O código pode ser visualizado na Figura 5.3 abaixo:

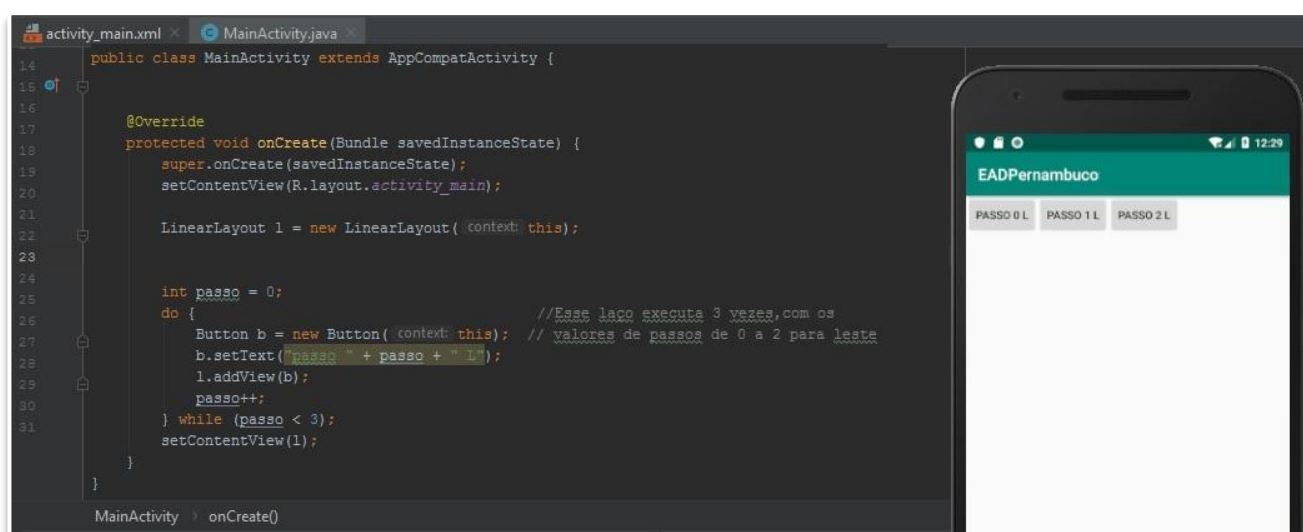


Figura 5.3 – Estrutura do-while no Android Studio.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando a estrutura do-while. No lado esquerdo tem-se um celular com as palavras PASSO0, PASSO1, PASSO2.

E aí, estudante, foi legal o caminho até aqui? Você já conhecia algo sobre as estruturas de repetição? E sobre as estruturas **for**, **while** e **do-while** em Java? O legal é que até aqui aprendemos sobre:

- As estruturas de repetição.
- A forma completa da estrutura **for**.



- Usar a instrução **for**.
- A forma completa da estrutura **while**.
- Usar a instrução **while**.
- A forma completa da estrutura **do-while**.
- Usar a instrução **do-while**.



Pronto para inserir mais informações em seu banco de dados? Acesse o AVA e assista a nossa primeira videoaula competência 5.



Agora, acesse o AVA e responda as questões da competência 5.



Ficou com alguma dúvida na competência 5? Acesse o Fórum - “Competência 5” para saná-las e discutir com seus colegas sobre os assuntos estudados.

Agora, o nosso próximo passo é mergulhar na sexta e última competência, que trata da utilização de *arrays*. Vamos juntos?



6. Competência 06 | Formatar uma Activity para utilização de Listas e Arrays

Vamos iniciar a sexta competência? Nela, voltaremos ao assunto de tipos de dados. Iremos apresentá-los ao tipo de dado *array*. Começaremos conhecendo os *arrays*. Em seguida, vamos entender e criar um *array* unidimensionais. E por fim, iremos entender e criar um *array* multidimensionais.

6.1 Arrays

Um *array* é um tipo de dado formado por um conjunto de variáveis do mesmo tipo, referenciadas por um nome comum. Em Java, os *arrays* podem ter uma ou mais dimensões e são usados para diversos fins, porque oferecem um meio conveniente de agrupar variáveis relacionadas. Como exemplo, você pode utilizar um *array* para registrar a temperatura máxima diária, uma lista de medias de preços e uma lista com coleção de livros de programação.

Por exemplo, imagine que durante o desenvolvimento de um programa fosse preciso declarar variáveis para armazenar um registro da temperatura máxima diária durante um mês, ou seja, você teria que declarar tempmax01, tempmax02, tempmax03, tempmax040, , tempmax30 e tempmax31. Nesse caso, para resolver este problema, você pode usar um *array* para armazenar o registro diário da temperatura máxima de um mês.

A principal vantagem de um *array* é que ele organiza os dados de tal forma que é fácil tratá-los. Por exemplo, com um *array* contendo o registro diário da temperatura máxima de um mês, será fácil calcular a temperatura média percorrendo-o.



Você sabia que um vetor é uma estrutura de dados formada por um conjunto de elementos de mesmo tipo. Em programação, os vetores são chamados de arrays.



6.1.1 Arrays Unidimensionais

São considerados uma lista de variáveis relacionadas e são muito comuns em programação. Para declarar um *array* unidimensional, você pode usar esta forma geral:

tipo nome_array[] = new tipo[tamanho];

Onde:

- *tipo*: declara o tipo de dado do *array*;
- *nome_array*: determinará o nome do *array*;
- *new*: permite criar um objeto *array*;
- *tamanho*: determinará o número de posições (índices) que o *array* conterá. Cada posição contém um elemento do *array*.

Em Java, a criação de um *array* se dá em duas etapas: primeiro você declara uma variável de referência de *array*. Depois, aloca memória para o *array*, atribuindo uma referência dessa memória a variável de *array*. Portanto, esses *arrays* em java são alocados na memória com o uso do operador *new*.

Para demonstrar o **array unidimensional**, iremos inicialmente pensar como armazenar um conjunto de palavras secretas para um jogo de adivinhação simples, o jogo da forca. Por exemplo, como podemos criar um *array String* de 15 elementos e o vincular a uma variável de referência de *array* chamado **palavra**. A linha a seguir cria um *array String* de 15 elementos e o vincula a **palavra**:

String palavra[] = new String [15];

A variável **palavra** contém uma referência a memória alocada por **new**. Essa memória é suficientemente grande para conter 15 elementos do tipo *String*.

Um elemento individual de um *array* é acessado com o uso de um índice. Um índice descreve a posição de um elemento dentro de um *array*. Em Java, geralmente os *arrays* tem zero como o índice de seu primeiro elemento. Já que a variável **palavra** tem 15 elementos, ela tem valores de índice que vão de 0 a 14.

Para indexar um *array*, devemos especificar o número do elemento desejado, inserido em colchetes. Portanto o primeiro elemento de *palavra* é *palavra[0]* e o último é *palavra[14]*.



Por exemplo, o programa a seguir (Figura 6.1) carrega **palavra** com 15 elementos (palavras secretas):

```
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 palavra[] = new String [15];
        int i;

        palavra[0] = "Deadpool";
        palavra[1] = "Wolverine";
        palavra[2] = "Black Widow";
        palavra[3] = "Ironman";
        palavra[4] = "Thor";
        palavra[5] = "Loki";
        palavra[6] = "Spawn";
        palavra[7] = "Kal-El";
        palavra[8] = "Wonder Woman";
        palavra[9] = "Batman";
        palavra[10] = "Gamora";
        palavra[11] = "Death";
        palavra[12] = "Kratos";
        palavra[13] = "Yoda";
        palavra[14] = "Drácula";
        for (i = 0; i < 15; i++) { //o array é indexado a partir de zero.
            c.setText(" A palavra secreta "+ i +" :  "+ palavra[i]);
        }
    }
}
```

Figura 6.1 - Array unidimensional. Neste exemplo o array carrega palavra com 15 elementos (palavras secretas).

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando array unidimensional.

Conceitualmente, o *array* palavra tem a seguinte aparência:

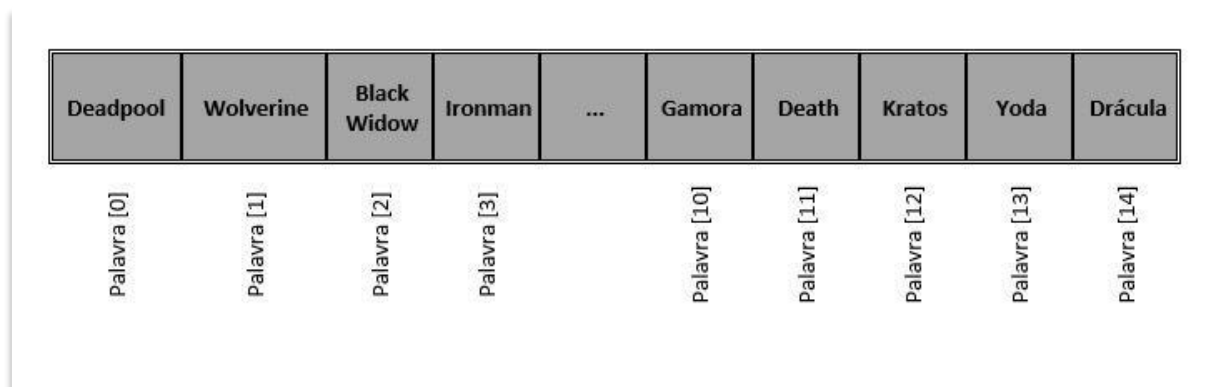


Figura 6.2 - Aparência conceitual do array palavra.

Fonte: o autor

Descrição da imagem: a imagem apresenta um retângulo contendo outros quadrados, cada um representando o elemento individual de um array unidimensional. Cada quadrado contendo os seguintes textos: Deadpool, Wolverine, Black Widow, Ironman, ..., Gamora, Death, Kratos, Yoda e Drácula. Em seguida temos abaixo de cada quadrado textos descritivos do índice da posição de cada elemento dentro de um array.

6.1.2 Arrays Multidimensionais

O *array* unidimensional é o mais comum em programação, porém os *arrays* multidimensionais (*arrays* de duas ou mais dimensões) não são raros. Em Java, o *array* multidimensional é um *array* composto por *arrays*.

6.1.2.1 Arrays Bidimensionais

A forma mais simples de *array* multidimensional é o *array* bidimensional. Para declarar um *array* bidimensional, você pode usar esta forma geral:

```
tipo nome_array[ ] [ ] = new tipo[tamanho] [tamanho];
```

Diferentemente de outras linguagens de programação, que usam vírgulas para separar as dimensões do *array*, Java insere cada dimensão em seu próprio conjunto de colchetes.

Para demonstrar o **array bidimensional**, iremos inicialmente pensar como criar um tabuleiro para o desenvolvimento do jogo da velha. Por exemplo, como podemos criar um *array int* de 9 elementos e o vincular a uma variável de referência de *array* chamado **tabuleiro**.

Por exemplo, o programa a seguir (Figura 6.4) cria um *array int* de 9 elementos e o vincula a **tabuleiro**:



```
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 t, i;  
        int tabuleiro[][] = new int[3][3];  
        for (t = 0; t < 3; ++t) {  
            for (i = 0; i < 3; ++i) {  
                tabuleiro[t][i] = (t * 3) + i + 1;  
                c.setText(tabuleiro[t][i] + "X");  
            }  
        }  
    }  
}
```

Figura 6.3 - Array bidimensional. Neste exemplo o array carrega tabuleiro com 9 elementos.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código utilizando array bidimensional.

Neste exemplo, `tabuleiro[0][0]` terá o valor 1, `tabuleiro[0][1]` terá o valor 2, `tabuleiro[0][2]` terá o valor 3, e assim por diante. O valor de `tabuleiro[2][2]` será 9. Conceitualmente, o array ficará parecido como mostrado na Figura 6.6:

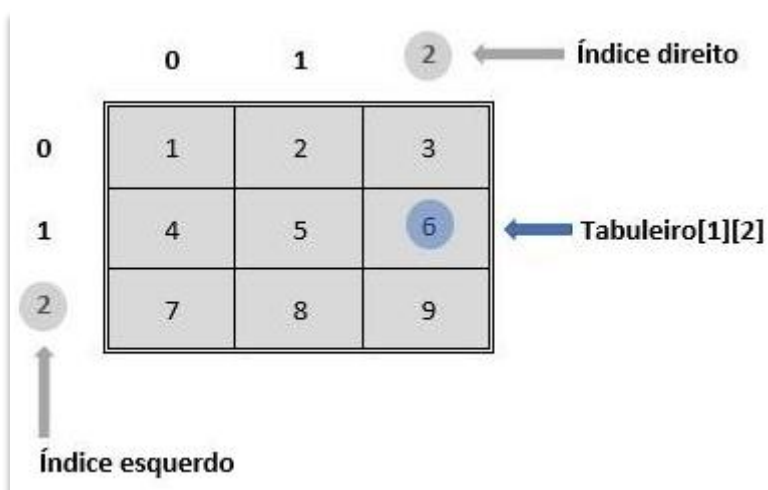


Figura 6.4 - Visão conceitual do array tabuleiro.

Fonte: o autor

Descrição da imagem: a imagem apresenta um retângulo contendo outros quadrados, cada um representando o elemento individual de um array bidimensional. Cada quadrado contendo os seguintes textos: 1, 2, 3, 4, 5, 6, 7, 8 e 9.

Em seguida temos acima de cada quadrado textos descritivos do índice direito, no lado esquerdo de cada quadrado temos textos descritivos do índice esquerdo, e no lado direito temos o texto descritivos do índice da posição `Tabuleiro[1][2]` e o valor do elemento.



E aí, estudante, foi legal o caminho até aqui? Você já conhecia algo sobre o tipo de dado array? E sobre arrays unidimensionais e arrays bidimensionais? O legal é que até aqui aprendemos sobre:

- Entender e criar arrays.
- A forma completa de um array unidimensional.
- Criar arrays unidimensional.
- A forma completa de um array bidimensional.
- Criar arrays bidimensional.



Pronto para inserir mais informações em seu banco de dados? Acesse o AVA e assista a nossa primeira videoaula competência 6.



Agora, acesse o AVA e responda as questões da competência 6.



Ficou com alguma dúvida na competência 6? Acesse o Fórum - “Competência 6” para saná-las e discutir com seus colegas sobre os assuntos estudados.

Agora, o nosso próximo passo é focar na avaliação da disciplina de IPDM. Vamos lá?



Conclusão

Olá novamente, desenvolvedor!

Chegamos ao final de nossa jornada! Espero que os conhecimentos acumulados em nosso passeio tenham lhe ajudado a alcançar a compreensão das noções iniciais e essenciais do desenvolvimento de aplicativos para a plataforma *Android* e sejam a base para eventuais experiências.

Não se acanhe em retornar para este caderno caso você tenha dúvidas em relação às noções iniciais do desenvolvimento de aplicativos para *Android*, pois o conhecimento necessário em lições futuras é cumulativo, ou seja, o que foi visto aqui não pode ser tratado de forma isolada: servirá como base em todo o processo da próxima disciplina e em outras aplicações que você talvez queira experimentar fora das lições abarcadas neste material didático.

Falando em experimentação, lhe incentivo a tentar de forma criativa utilizar os conhecimentos deste caderno para criar aplicações com propostas diferentes dos exemplos construídos aqui neste conteúdo. Desafie-se ir além!

Cordialmente



Referências

LECHETA, Ricardo. **Google Android**. Aprenda A Criar Aplicações Para Dispositivos Móveis Com O Android SDK. São Paulo: Novatec, 2010.

GLAUBER, Nelson. **Dominado o Android**. São Paulo: Novatec, 2015.

GLAUBER, Nelson. **Android Essencial com Kotlin**. São Paulo: Novatec, 2018.

GLAUBER, Nelson. **Dominado o Android com Kotlin**. São Paulo: Novatec, 2019.

QUEIRÓS, Ricardo. **Android Profissional**. Desenvolvimento Moderno de Aplicações. Lisboa: FCA, 2018.

SCHILDT, Herbert. **Java**. Para Iniciantes. Porto Alegre: Bookman, 2015.

<https://developer.android.com/?hl=pt-br>



Minicurrículo do Professor

Rebecca Cristina Linhares de Carvalho

Mestre em Ciências da Computação no Centro de Informática (CIn) na universidade Federal de Pernambuco (UFPE). Possui graduação em Bacharelado em Sistemas de Informação pela Faculdades Integradas Barros Melo (2009). Atualmente é professora do curso técnico de Desenvolvimento de Sistemas, na modalidade EAD, na Secretaria de Educação do estado de Pernambuco, professora formadora das disciplinas: Edição e Processamento de Imagens e Programação para Dispositivos Móveis, e conteudista da disciplina de Introdução a Programação para Dispositivos Móveis.

