Desenvolvedor de Apps

Android Studio V.3

Desenvolvedor de App Android Studio V3

Nome: _____

Quantidade de Aulas
16 aulas

Carga horária
24 horas

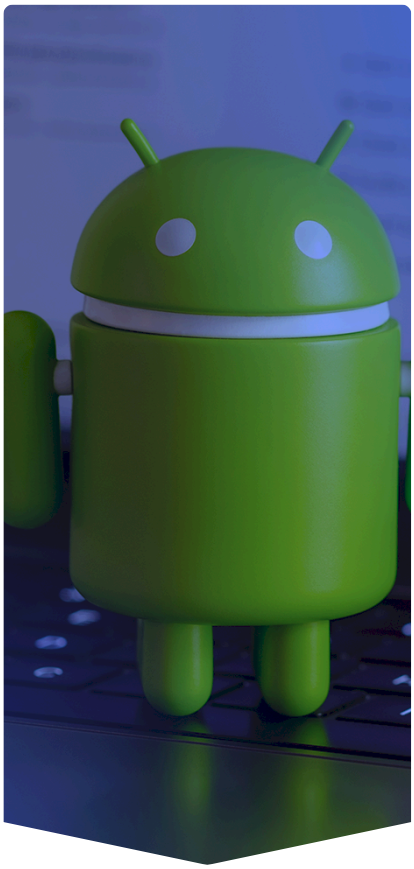
Programas Utilizados
Android Studio

Sobre o curso

Este curso é voltado para iniciantes no desenvolvimento de aplicativos Android. Você aprenderá a criar e publicar apps utilizando o Android Studio, desde a introdução à ferramenta até o desenvolvimento de um projeto completo.

O que aprender com este curso?

Neste curso, você aprenderá a desenvolver aplicativos Android usando o Android Studio, começando pela configuração do ambiente e criação de interfaces interativas. Iremos abordar a manipulação de dados, estruturas condicionais e a navegação entre telas. Você entenderá conceitos de programação e como usar XML para layouts adaptáveis. O curso também inclui integração de banco de dados.



Sumário

Olá, seja bem-vindo(a) à nossa primeira aula de nosso Curso de Desenvolvedor de Aplicativos para Android. Nesta aula, vamos conhecer como o Android surgiu e o que preciso para começar a criar um aplicativo.

1.1. O que é o Sistema Android

Android é o nome do sistema operacional que opera em celulares (smartphones), notebooks e tablets. É desenvolvido pela Open Handset Alliance, onde participam diversas empresas, dentre elas a Google.

O Android é projetado principalmente para dispositivos móveis com tela sensível ao toque como smartphones e tablets; com interface específica para TV (Android TV), carro (Android Auto) e relógio de pulso (Android Wear).

O sistema operacional utiliza-se da tela sensível ao toque para que o usuário possa manipular objetos virtuais e também de um teclado virtual. Apesar de ser principalmente utilizado em dispositivos com tela sensível ao toque, também é utilizado em consoles de videogames, câmeras digitais, computadores e outros dispositivos eletrônicos.

O funcionamento do Android é semelhante a outros sistemas operacionais, como o Windows, Mac OS, entre outros, cuja função é gerenciar todos os processos dos aplicativos e do hardware de um computador para que funcione de forma correta.

A Google impulsionou o Android para operar nos seus próprios dispositivos móveis, entrando, assim, em concorrência com outros sistemas operacionais como o Symbian (Nokia), iOS (Dispositivos Apple, como iPhone) e Blackberry OS.

Os smartphones, rodando o sistema Android, iniciaram em 2008. Mas a história começou em 2003 com a empresa Android Inc. A proposta foi apresentada por um grupo liderado pelo Andy Rubin.

Em 2005, o Google adquiriu o Android Inc, e com isso nasceu a Google Mobile Division, divisão de pesquisa em tecnologia móvel da maior empresa do mundo de tecnologia.

Apesar de ter causado desconfiança e dúvidas na época, já que muitos achavam difícil uma competição do Windows Mobile, da Microsoft, e o iOS, da Apple.

Os primeiros contratos de parceria surgiram com fabricantes de hardware e software, aos quais o Google prometeu um sistema flexível e atualizável.

1.2. O que é um Aplicativo

App é um apelido dado para o termo "aplicativo" (que vem do inglês application). Você também vai ouvir outras formas de chamar o App, tais como: aplicativo para celular, aplicativo móvel, aplicativo mobile. Nós, na Fábrica de Aplicativos, preferimos chamar simplesmente de App.

Você já deve ter visto aquelas pessoas que não tiram os olhos e os dedinhos da tela de um celular. Se você conhece esta cena, pode ter certeza, esta pessoa é apaixonada por eles, os Apps.

Os Apps são aqueles quadradinhos que estão espalhados no celular de cada vez mais pessoas, e que, com um único toque, te conecta com um universo inteiro de possibilidades. As pessoas também os chamam por aplicativos ou aplicações, mas o termo App já é comum no nosso dia a dia.

Um aplicativo bem comum nos celulares é a calculadora.

Uma tela que mostra a previsão do tempo, o jogo ou aquela câmera cheia de efeitos, entre outros, são considerados Apps. Todos os APPs podem ser baixados nas lojas de aplicativos, também conhecidas como “App Stores”.

As duas principais lojas são a AppStore da Apple e a Google Play, do Android. Diversos “apps” são disponibilizados gratuitamente, mas outros são vendidos por diversos preços, de centavos até os mais caros, caso seja um aplicativo especializado.

Toda vez que você compra um “app” ele fica em sua biblioteca e pode receber atualizações, que na grande maioria são gratuitas.

1.3. Comparativo entre os sistemas

O Android é o sistema operacional mais popular do mundo e mantém essa liderança também no Brasil.

De acordo com a empresa de análise de dados Stat Counter, o sistema do Google domina atualmente 82,08% do mercado nacional de dispositivos móveis, seguindo do iOS, com 17,68% e outros 0.24%. Percentual esse que varia de acordo com as pesquisas (08/2024).

1.4. Mercado de trabalho

O mercado para desenvolvedor Android está completamente aberto. Isso porque, segundo pesquisa da Stat Counter, o Android dominou o mercado brasileiro.

Todos esses números e estatísticas são reflexos diretos da preferência do brasileiro. O uso do smartphone virou como uma extensão do braço, e os aplicativos um grande passatempo. Um desenvolvedor Android se tornou uma das profissões mais promissoras do mercado.

As oportunidades de crescimento são imensas. O universo mobile cresce como nunca, e o desenvolvedor virou peça-chave da engrenagem.

Cada vez mais os aplicativos fazem parte de nosso cotidiano.

Quando temos algum problema como converter moedas, buscar uma nova dieta, fazer um pagamento ou qualquer coisa para facilitar a nossa vida de alguma forma, basta pesquisarmos no Google para descobrirmos algum aplicativo Android ideal para solucionar a questão.

1.5. Como lucrar

Atualmente as pessoas não conseguem ficar longe do seu smartphone, utilizando para resolver os mais variados problemas do dia a dia.

Para conseguir ganhar dinheiro com seu aplicativo é preciso ter uma boa ideia, desenvolver uma solução interessante e ainda planejar suas estratégias de monetização. Confira as cinco formas de ganhar dinheiro com seu app:

1.5.1. Aplicativo pago

Seguindo o modelo tradicional de vendas, o desenvolvedor cobra um valor para o usuário conseguir baixar o app. Uma boa estratégia é criar uma versão gratuita, com menos funções, para as pessoas entenderem o que ele faz e, se realmente gostarem, comprar o app principal. Os valores praticados variam de alguns centavos a centenas de dólares.

1.5.2. Compras dentro do aplicativo

As chamadas “in app” são compras que o usuário pode utilizar para melhorar a sua experiência dentro dos apps gratuitos. Normalmente são funções extras ou fases e vidas novas em jogos. É uma estratégia muito interessante, pois oferece uma solução gratuita, mas cria um senso de necessidade ou de merecimento com a aquisição das funcionalidades extras. Se o app for realmente bom e a compra no aplicativo tiver um valor justo, o usuário não se importa em pagar.

1.5.3. Banners e Anúncios

Muito parecido com o modelo de publicidade em sites e blogs, o app é oferecido gratuitamente para download, mas possui algumas propagandas como um banner ou inserção de vídeos no seu conteúdo. O desenvolvedor recebe de acordo com o número de visualizações ou de cliques nos anúncios exibidos. É uma forma simples de receber um retorno, mas é preciso ter cuidado para não exagerar: os usuários costumam desistir de apps com excesso de anúncios.

1.5.4. Assinatura

São apps que fornecem serviços contínuos para o usuário. Provavelmente os mais famosos são o Netflix e o Spotify, mas existem serviços de assinatura de diversos tipos, como de atividade física e meditação. O pagamento pode ser feito mensalmente ou por meio de um valor anual.

1.5.5. Serviço

Oferecer um serviço pelo seu app e cobrar uma taxa pelo uso também é uma opção. É assim que aplicativos como iFood e Uber funcionam.

O usuário paga apenas quando utiliza o app, que possui um sistema de cobrança que paga o seu parceiro — o motorista ou dono do restaurante — e retira a sua porcentagem. É uma alternativa interessante para as plataformas mais robustas.

O mais interessante é perceber que sempre existe uma forma de monetizar o seu trabalho, seja ele pequeno ou grande. Por isso, não tenha medo de começar! O mercado de aplicativos precisa de cada vez mais desenvolvedores.

Quando você vende seu aplicativo por R$ 0,99 nas lojas de aplicativos, esse é exatamente o valor que o cliente irá pagar. Mas, como você está usando as lojas como meio para fazer as vendas, as lojas terão 30% (Google e a Amazon) desse valor retido para eles. Como se fosse uma comissão.

Isso inclui tudo que utiliza os canais das lojas, como: aplicativos pagos, compras no aplicativo, redes publicitárias, assinaturas, etc.

O tempo de pagamento também varia um pouco nas lojas. Normalmente, são 30 dias.

Custo para criar um aplicativo

Saiba que não tem custo algum, tanto para utilizar as ferramentas de desenvolvimento como para fazer uso comercial do Android.

Outra coisa que precisamos fazer é uma conta no Google, e adivinha só quanto custa? Custo zero (R$0,00).

O único valor a ser pago é de US$ 25 (dólares), que corresponde a uma licença de desenvolvedor, para que você possa publicar seus aplicativos na Google Play.

Quais são as ferramentas para criar um aplicativo?

Existem algumas ferramentas para desenvolvimento Android nativo por aí, mas a única e oficial é o Android Studio.

Android Studio é um ambiente de desenvolvimento integrado (IDE) para desenvolver para a plataforma Android. Foi anunciado em 16 de maio de 2013 na conferência Google I/O. Android Studio é disponibilizado gratuitamente sob a Licença Apache 2.0.

Android Studio estava em estágio de acesso antecipado desde a versão 0.1 em Maio de 2013, entrando então em estágio beta a partir da versão 0.8 que foi lançada em Junho de 2014. A primeira compilação estável foi lançada em dezembro de 2014, começando da versão 1.0.

Baseado no software IntelliJ IDEA de JetBrains, Android Studio foi feito especificamente para o desenvolvimento para Android. Está disponível para download em Windows, Mac OS X e Linux, e substituiu Eclipse Android Development Tools (ADT) como a IDE primária do Google de desenvolvimento nativo para Android.

1.6. Android Emulator

Uma ferramenta de emulação de dispositivo baseada no QEMU que você pode usar para debugar e testar seus aplicativos em um ambiente real de execução do Android.

1.7. Exercícios

Para essa apostila, teremos dois exercícios, que serão baixar e instalar o Java JDK e o Android Studio.

1.7.1. Exercício 1

Como baixar o Java (a versão utilizada durante a aula está disponível na pasta de arquivos auxiliares)

O pacote, chamado JDK (Java Development Kit), é o que contém toda a infraestrutura necessária para o desenvolvimento de aplicações Java. Ao ser instalado, o JRE é instalado automaticamente.

Para instalar o JDK no Linux ou no Windows, primeiramente é necessário efetuar o download do arquivo de instalação. Para isso deve-se acessar o site da Oracle e baixar versão do JDK correspondente ao sistema operacional e arquitetura (32 ou 64 bits) utilizada.

Acesse o site **https://www.oracle.com/br/java/technologies/d** Nesta página encontraremos todos os recursos.

Então, escolha o sistema operacional de seu dispositivo e realize o download e instalação das versões necessárias.

1.7.2. Exercício 2

Agora em nosso segundo exercício, será feita a instalação do Android Studio.

Como baixar o Android (a versão utilizada durante a aula está disponível na pasta de arquivos auxiliares)

Acesse o site **https://developer.android.com/studio** e baixe a versão atual.

Java é a base para o desenvolvimento Android. Sendo assim, é muito importante que o desenvolvedor tenha domínio dessa linguagem.

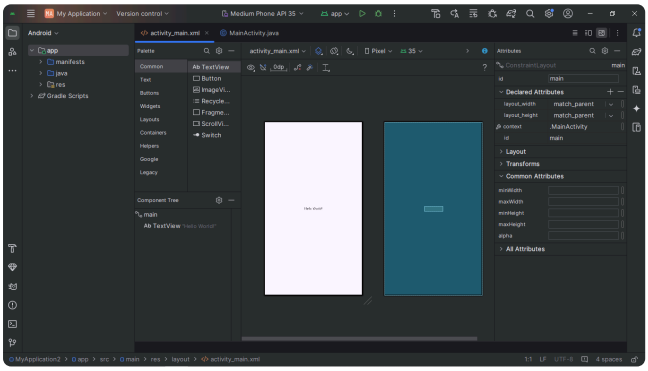
Muitos conhecem o Java como sinônimo de uma linguagem de programação orientada a objetos, mas o termo também se refere às inúmeras aplicações que utilizamos em nosso dia-a-dia quando navegamos na Internet. O próprio Android que utilizamos em nossos smartphones, assim como os aplicativos que instalamos nele, são desenvolvidos nessa linguagem.

Então, após o download, siga o passo a passo para a instalação do software igual foi mostrado em aula.

Seja bem-vindo(a)! Hoje, continuaremos explorando os fundamentos do desenvolvimento Android com o Android Studio, abordando a interface, a estrutura do projeto, os componentes e as ferramentas essenciais.

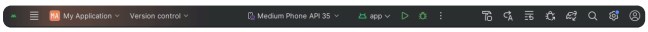
2.1. A interface do usuário

A janela principal do Android Studio é composta de diversas áreas.

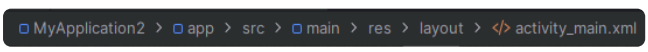


Vamos conhecer algumas de suas áreas.

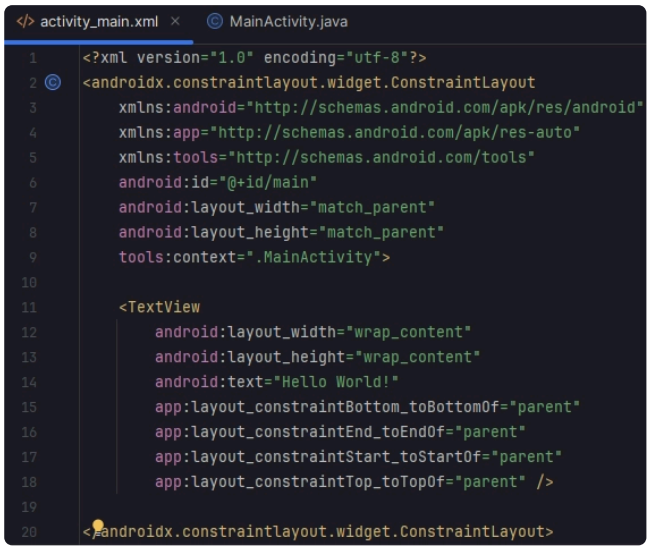
A barra de ferramentas permite executar diversas ações, incluindo executar aplicativos e inicializar ferramentas do Android.



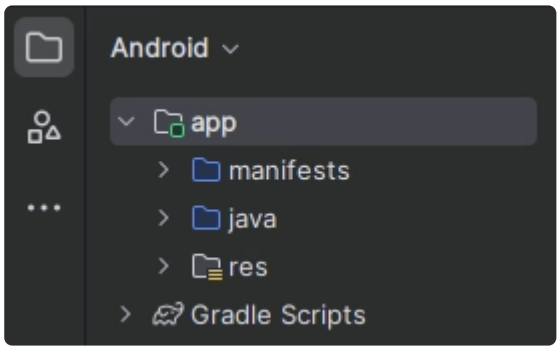
A barra de navegação ajuda na navegação pelo projeto e na abertura de arquivos para edição. Ela oferece uma visualização mais compacta da estrutura visível na janela Project.



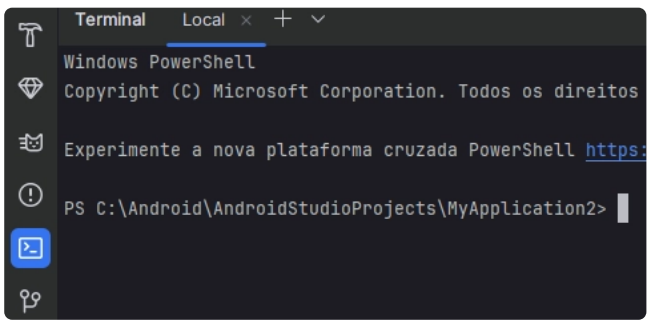
A janela do editor é onde você cria e modifica código, e se adapta ao tipo de arquivo. Por exemplo, ao abrir um layout, o Editor de Layout é exibido.

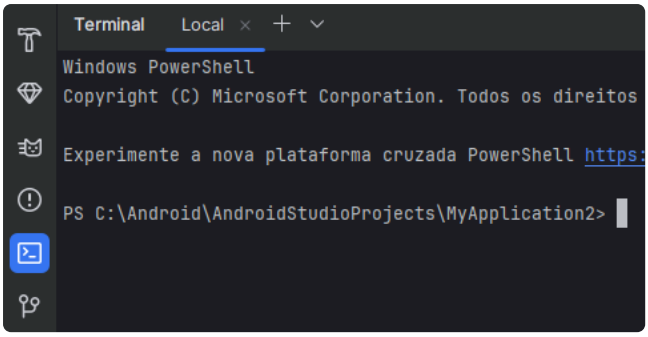


A barra de ferramentas, localizada fora da janela principal do IDE, possui botões que expandem ou recolhem as janelas de cada ferramenta.



A janela das ferramentas dá acesso a tarefas específicas, como gerenciamento de projetos, busca, controle de versão e muitos outros. Você pode expandi-las e recolhê-las.





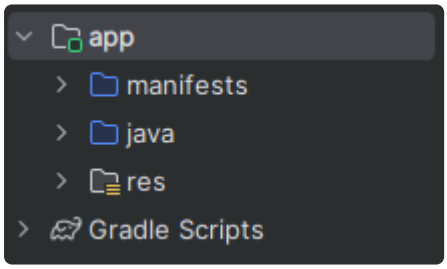
2.2. Estrutura do projeto

Todos os arquivos da compilação podem ser vistos no nível superior em Gradle Scripts e cada módulo de aplicativo contém as pastas a seguir:

Manifestos: contém o arquivo AndroidManifest.xml.

Java: contém os arquivos de código-fonte do Java, incluindo o código de teste do JUnit.

Recursos: contém todos os recursos que não são código, como layouts XML, strings de IU e imagens em bitmap.



MANIFESTS

Todo aplicativo tem que ter um arquivo AndroidManifest.xml (precisamente com esse nome) no diretório raiz. O arquivo de manifesto apresenta informações essenciais sobre o aplicativo ao sistema Android, necessárias para o sistema antes que ele possa executar o código do aplicativo.

Entre outras coisas, o arquivo do manifesto serve para:

Nomear o pacote Java para o aplicativo. O nome do pacote serve como identificador exclusivo para o aplicativo.

Descrever os componentes do aplicativo que abrangem atividades, serviços, receptores de transmissão e provedores de conteúdo que compõem o aplicativo. Ele também nomeia a classe que implementa cada um dos componentes e publica suas capacidades, como as mensagens Intent que podem lidar. Essas declarações informam ao sistema Android os componentes e as condições em que eles podem ser inicializados.

Determinar os processos que hospedam os componentes de aplicativo.

Declarar as permissões que o aplicativo deve ter para acessar partes protegidas da API e interagir com outros aplicativos. Ele também declara as permissões que outros devem ter para interagir com os componentes do aplicativo.

Listar as classes Instrumentation que fornecem geração de perfil e outras informações durante a execução do aplicativo. Essas declarações estão presentes no manifesto somente enquanto o aplicativo está em desenvolvimento e são removidas antes da publicação do aplicativo.

Declarar o nível mínimo da Android API que o aplicativo exige.

Listar as bibliotecas às quais o aplicativo deve se vincular.

JAVA:

Pasta com dois Pacotes:

1º. MainActivity (onde iremos criar nossos arquivos Java);

2º. Application Test (onde faremos testes da nossa Aplicação).

Drawable

Recursos gráficos que podem ser bitmaps ou XML.

Arquivos Bitmap (.png, .jpg, .gif) ou arquivos XML que são compilados nos seguintes resources:

Arquivos Bitmap

9-Patch (bitmaps redimensionáveis)

Listas de estado

Shapes (formas)

Animações (frame animations)

Outros tipos.

LAYOUT

Você pode definir vários layouts para sua UI e alterná-los de acordo com o estado da aplicação.

Arquivos XML que definem a sua UI (user interface).

MIPMAP

Arquivos "drawable" especificamente para ícones de aplicações de diferentes tipos de densidade.

VALUES

Arquivos XML que contêm valores, como strings, inteiros e cores. Dentro dessa pasta, você deve nomear cada arquivo XML com o nome do seu resource e dessa forma irá acessar com uma subclasse de R. Por exemplo, o arquivo string.xml será acessado por R.string. Abaixo, algumas convenções:

arrays.xml para Arrays.

colors.xml para valores de cores.

dimens.xml para dimensões.

strings.xml para todas as strings.

styles.xml para estilos.

Gradle

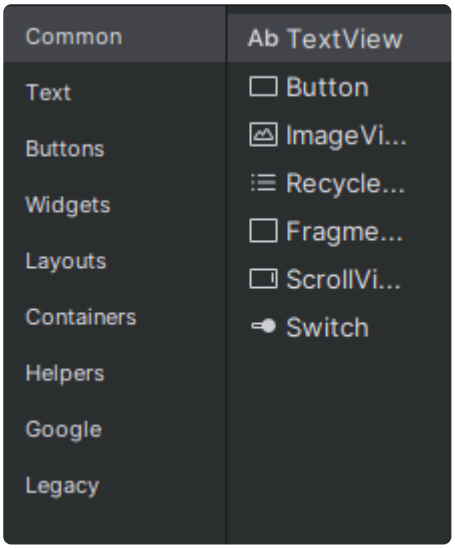
Desde que o Android Studio se tornou a ferramenta oficial do Google para desenvolvimento Android, o Gradle veio como principal aliado para ajudar no gerenciamento do projeto e controle de dependências.

Para entender como funciona o Gradle, antes precisamos entender um pouco sobre como funciona o processo de build de um aplicativo Android.

O build é o processo de compilação, ou seja, construção de recursos que vão integrar o projeto.

Quando executamos o Gradle para construir seus projetos e módulos, o processo do build é executado em segundo plano.

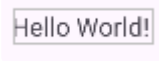
2.3. Componentes do Android Studio



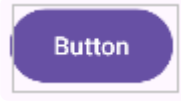
A primeira aba é a Common (Comum), onde ficam os componentes indicados como mais usados.

Vamos conhecer alguns elementos:

TextView: uma caixa de saída de texto, utilizada para textos, títulos ou para passar informações.



Button: um botão comum, utilizado em menus de opções.

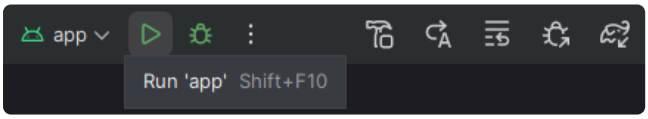


ImageView: uma imagem de demonstração, como ícones e fotos.

2.4. Emulador

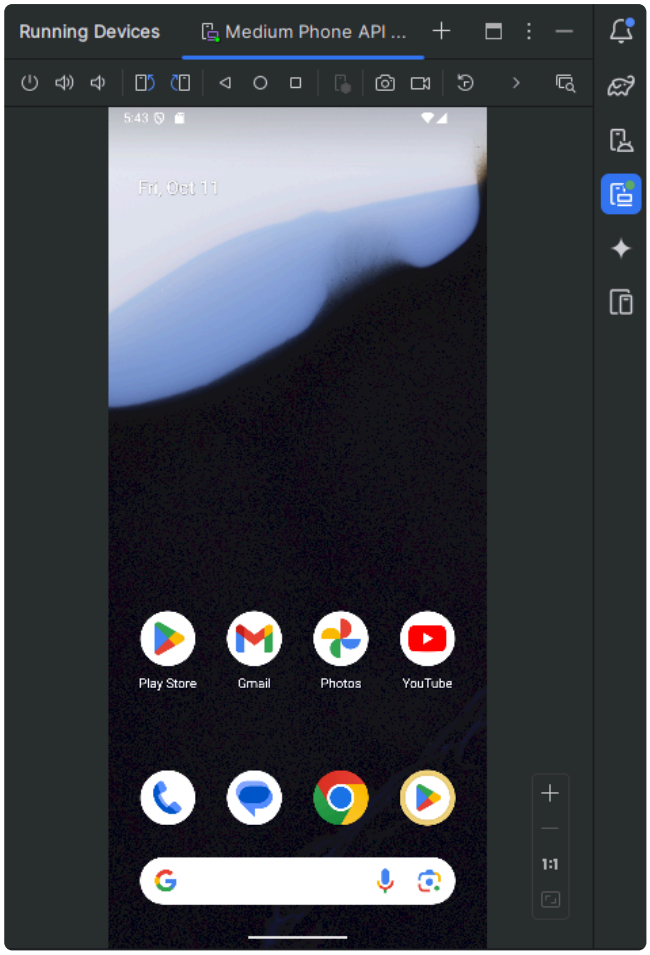
O Android Emulator permite executar aplicativos Android em um ambiente simulado dentro do Windows, simula um dispositivo e o exibe no seu computador de desenvolvimento. Com ele, você pode criar protótipos, desenvolver e testar aplicativos do Android sem usar um dispositivo de hardware. O emulador é compatível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

Barra de ferramentas



O botão **Run** (app), destacado, serve para executar o projeto.

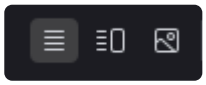
O Android Emulator é uma ferramenta rápida, eficaz e repleta de recursos. Ele pode transferir informações com mais rapidez do que um dispositivo de hardware conectado, o que agiliza o processo de desenvolvimento. O recurso de vários núcleos permite que o emulador aproveite os processadores de vários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



Você interage com o emulador da mesma forma que faria com um dispositivo de hardware, mas usando o mouse e o teclado, além dos botões e controles do emulador. O emulador é compatível com botões e touchscreens de hardware virtual, incluindo operações com dois dedos, pads direcionais, trackballs, rodas e diversos sensores. É possível redimensionar dinamicamente a janela do emulador conforme a necessidade, aumentar e reduzir o zoom, alterar a orientação e até registrar uma captura de tela.

2.5. Modos de visualização

O Android Studio possui três modos de visualização.



Modo Design: Esse modo permite adicionar diversos recursos. Por exemplo, a "Component Tree", no canto inferior esquerdo, exibe a hierarquia do layout, onde a visualização raiz é um ConstraintLayout contendo um TextView. A "Palette" possibilita inserir elementos de entrada de texto (simples, com senha, telefone, e-mail, etc.) e botões (como checkbox e radiobutton).

Modo Code: É o modo que permite trabalhar com o código da página, se já foram aplicados recursos no modo design, estes serão apresentados no código, onde de forma mais prática poderemos fazer alterações em suas propriedades, como tamanho e cor.

Modo Split: O modo Split é uma junção dos modos Design e Code, Separando um espaço na interface do Android Studio onde é mostrado cada modo em metade da tela.

Como alterar visual da UI

Caso seu Android Studio esteja com um visual diferente ao da aula, pode ser que a versão antiga da interface está habilitada. Não tem problema usar ela, mas, se quiser utilizar a mesma que a aula, vá para a engrenagem localizada ao lado direito superior do seu Android Studio e procure pela opção escrita "Enable new UI".

2.6. Exercício

O exercício desta apostila tem o simples objetivo de você treinar o que você viu durante a aula, ou seja, praticar, então além do exercício a seguir, você deve usar o Studio por conta própria e entender como cada opção funciona.

Para o exercício, você deverá criar um projeto novo no Android Studio e adicionar campos TextView modificados por você, esses campos deverão ser semelhantes e um botão para enviar as "informações" dos TextView que você criou.

Nome:

E-mail:

Telefone:

[Enviar]

Ideia de projeto.

anotações

Variáveis são espaços reservados na memória RAM do dispositivo para armazenamento temporário de dados. Para criar uma variável, precisamos definir um nome e o tipo de dados.

Confira a tabela com alguns tipos de dados:

Tipo	Descrição
Char	Tipo caractere, dados alfanuméricos
String	Tipo texto
Int	Tipo valor inteiro
Float	Números reais
Double	Números reais
Boolean	Tipo lógico (verdadeiro/true/1 ou (falso/false/0)

As variáveis são representadas, geralmente, por uma palavra simples ou composta que a identifica e a torna única. Pode-se utilizar quaisquer letras, sejam elas maiúsculas ou minúsculas. Variáveis não devem utilizar nomes reservados de classes, pacotes ou funções, ou seja, não devemos declarar uma variável com o nome "int", ou então "String", uma vez que essas palavras são reservadas, e o programa vai executá-las como um comando, retornando erro.

Variável de nome simples:

Nome, e-mail, endereço, cidade, idade ou idadeCliente, qtde (quantidade), valorUnitario, nota1, nota2, entre outras.

Para criar um identificador (nome da variável) em Java, precisamos seguir algumas regras, listadas a seguir:

Deve conter apenas letras, _ (underline), $ ou os números de 0 a 9;

Deve obrigatoriamente iniciar por uma letra (preferencialmente) _ ou $;

Deve iniciar com uma letra minúscula (boa prática – ver abaixo);

Não pode conter espaços;

Não podemos usar palavras-chave da linguagem;

O nome deve ser único dentro de um escopo.

Além disso, o Java é case sensitive, o que significa que os nomes de variáveis diferenciam maiúsculas de minúsculas.

Exemplos de nomes de variáveis.

Válidos:

nomeCliente

telefone_1

preco$

produtoAdquirido

Inválidos:

1Telefone

Nome Cliente

#Preço

Declaração de Variáveis

Vejamos como proceder para realizar a declaração de variáveis em Java.

Sintaxe:

tipo identificador [= valor];

Onde tipo é o tipo de dado que a variável irá armazenar, identificador é seu nome, e valor é o valor inicial atribuído à variável, sendo opcional (denotado pelos colchetes, que não devem ser digitados na declaração).

Exemplos de declaração:

byte a;

char t;

int valor;

float x, y;

int dia = 20;

char sexo = 'F';

String nome = "Fábio";

Atribuição de valores à variáveis

Para atribuir um valor a uma variável, devemos usar o operador de atribuição =

Veja os exemplos de atribuições a seguir:

byte a = 45;

char t = 'T';

int valor = 200;

float x = 98.80;

char sexo = 'F';

int dia; // variável declarada e não inicializada

dia = 20; // variável atribuída agora

Exemplo de variáveis

java

```java
public class VarEstudo {
     public static void main(String[] args){
        int num = 0;
        int minhaVariavel = 100;
        System.out.println(num);
        System.out.println(minhaVariavel);
        double dinheiro = 1.55;
        System.out.println(dinheiro);
     }
 }
```

java

```java
public class Aula01 {
     public static void main(String[] args)
{
        int num = 0;
        /* 1) um nome
        * 2) um tipo
        * 3) um tamanho
        * 4) um valor        */
        num = 10;
        /*   1) é temporário
        * 2) os dados são removidos da
memória
        * quando o PC é desligado
        * 3) volatil
        * 4) o valor antigo sempre é perdido
(para sempre)
        * 5) o Java, desaloca a memória
*/
     }
 }
```

3.1. Exercícios

3.1.1. Exercício 1

O objetivo do primeiro exercício é utilizar variáveis para criar um controle de custos. Neste controle, vamos precisar das seguintes variáveis:

Título = Controle de Custos Custo = 250

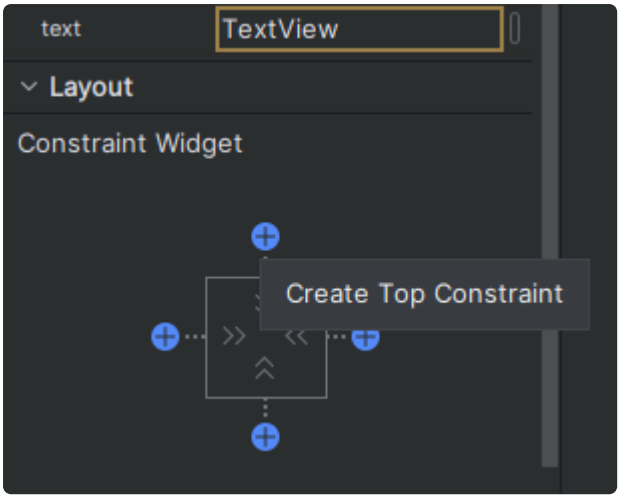
Lucro = criar a fórmula Venda = 310

1) Crie um novo projeto.

2) Mude a versão do Gradle se necessário.

3) Clique no arquivo xml, apague o TextView existente e então, arraste um novos componente TextView e um EditText.

4) Criando espaçamento entre os componentes. Clique no TextView e clique nos botões "Create Top Constraint" e "Create Left Constraint".



5) Clique no EditText e clique no botão "Create Top Constraint" e "Create Left Constraint".

6) Clique na aba MainActivity.java e crie as seguintes variáveis:

```
public class MainActivity extends AppCompatActivity {
    String titulo = "Controle de Custos";  no usages
    double custo = 250;  1 usage
    double venda = 310;  1 usage
    double lucro = venda - custo;  no usages
```

7) Localizando os componentes.

```
TextView tituloPagina = (TextView) findViewById(R.id.textView);
EditText lucroTotal = (EditText) findViewById(R.id.editTextText);
```

8) Defina o valor de cada componente.

```
tituloPagina.setText(titulo);
lucroTotal.setText(String.valueOf(lucro));
```

9) Veja o resultado clicando no botão Run.

3.1.2. Exercício 2

O objetivo do segundo exercício é criar um formulário onde um escritório possa registrar os dados de seus clientes. Neste caso, iremos arrastar componentes para a tela. Precisaremos dos seguintes componentes:

1) Inserir 1 TextView para o título da página = Cadastro de Clientes

2) Inserir 1 PlainText com o texto padrão de entrada = Nome do cliente

3) Inserir 1 PlainText com o texto padrão de entrada = Preencha o endereço

4) Inserir 1 PlainText com o texto padrão de entrada = Preencha a cidade

5) Inserir 2 RadioButtons, um para o sexo masculino e outro para o sexo feminino

6) Inserir 1 Button com o seguinte texto = Cadastrar

anotações

Os operadores aritméticos são operadores binários, ou seja, funcionam com dois operandos. Por exemplo, a expressão “a + 1” contém o operador binário “+” (mais) e os dois operandos “a” e “1”.

Operação	Operador	Expressão algébrica	Expressão Java
Adição	+	a + 1	a + 1
Subtração	-	b - 2	b - 2
Multiplicação	*	cm	c * m
Divisão	/	d / e	d / e
Resto	%	f mod g	f % g

Observação importante: a divisão de inteiros produz um quociente do tipo inteiro. Quando possuímos o número 1 maior que o número 2, por exemplo, a expressão 9 / 6 no resultado é interpretado como 1, e a expressão 23 / 8 é avaliada como 2, ou seja, a parte fracionária em uma divisão de inteiros é descartada, não contendo nenhum arredondamento.

O módulo (%) fornece o resto da divisão, na expressão “x % y”, o resultado é o restante depois que x é dividido por y. Sendo assim, na expressão “7 % 4”, o resultado é 3 e “17 % 5” o resultado produz 2. Esse operador é mais utilizado com operandos inteiros, mas também pode ser utilizado com outros tipos.

Precedência de operadores

Os operadores possuem regras que são aplicadas nas expressões aritméticas do Java, que são as mesmas seguidas em álgebra. Quando dizemos que os operadores são aplicados da esquerda para a direita, estamos nos referindo à sua associatividade.

Operadores de multiplicação, divisão e módulo são aplicadas primeiro. Por exemplo, quando aparecer uma expressão com várias dessas operações, elas serão aplicadas da esquerda para a direita.

As operações de adição e subtração são aplicadas em seguidas operações de adição e subtração são aplicadas em seguida.

Abaixo uma tabela de referência dos operadores e suas ordens de avaliação.

Operador	Operação	Ordem de avaliação (precedência)
* / %	Multiplicação Divisão Resto	Avaliado primeiro. Se houver vários operadores desse tipo serão avaliados da esquerda para a direita.
+-	Adição Subtração	Avaliado em seguida. Se houver vários operadores desse tipo, serão avaliados da esquerda para a direita.
=	Atribuição	Avaliado por último.

Listagem: Avaliação da precedência dos operadores.

java

```
public class Avalia_Precedencia {
 public static void main(String[] args) {
 int a = 30;
 int b = 5;
 int c = 10;
 int total = (a + b + c) / 10;
 System.out.println("O resultado = "+total);
 }
}
```

4.1. Operadores

Os operadores de igualdade verificam se o valor ou o resultado da expressão lógica à esquerda é igual (“==”) ou diferente (“!=”) ao da direita, retornando um valor booleano.

java

```java
int idadeJoao=20;
int idadeMaria=22;

if(idadeJoao == idadeMaria){
      System.out.println("idades iguais");
}else{
      System.out.println("idades
diferentes");
}
```

Esse código verifica se duas variáveis contêm o mesmo valor e imprimem o resultado. Uma vez que as variáveis IdadeJoao e IdadeMaria possuem valores diferentes, o trecho de código presente no else será executado.

A tabela abaixo apresenta os **operadores de igualdade** do Java:

==	Utilizado quando desejamos verificar se uma variável é igual a outra.
!=	Utilizado quando desejamos verificar se uma variável é diferente de outra.

== em textos

Como você deve ter visto, durante a aula utilizamos a função "Equals()" para verificar se um conteúdo digitado em um PlainText era equivalente a um valor pré-definido. Então, essa função basicamente verifica se dois objetos são logicamente equivalentes ao invés de verificar se são a mesma instância na memória (o que o operador == faz).

Operadores relacionais

Os operadores relacionais, assim como os de igualdade, avaliam dois operandos. Nesse caso, mais precisamente, definem se o operando à esquerda é menor, menor ou igual, maior ou maior ou igual ao da direita, retornando um valor booleano.

java

```java
int idadeJoao = 20;
int idadeMaria = 22;

if (idadeJoao > idadeMaria){
  System.out.println("maior");
}
if (idadeJoao >= idadeMaria){
  System.out.println(“maior ou igual”);
}
if (idadeJoao < idadeMaria){
 System.out.println(“menor”);
}
if (idadeJoao <= idadeMaria){
 System.out.println(“menor ou igual”);
}
```

Esse código realiza uma série de comparações entre duas variáveis para determinar o que será impresso no console. Uma vez que o valor da variável idadeJoao é menor que idadeMaria serão impressas as mensagens “menor” e “menor ou igual”.

Opções de operadores relacionais

A tabela abaixo apresenta os operadores relacionais do Java:

>	Utilizado quando desejamos verificar se uma variável é maior que a outra.
>=	Utilizado quando desejamos verificar se uma variável é maior ou igual a outra.
<	Utilizado quando desejamos verificar se uma variável é menor que a outra.
<=	Utilizado quando desejamos verificar se uma variável é menor ou igual a outra.

Operadores lógicos

Os operadores lógicos representam o recurso que nos permite criar expressões lógicas maiores a partir da junção de duas ou mais expressões. Para isso, aplicamos as operações lógicas E (representado por “&&”) e OU (representado por “||”).

Exemplo de uso:

```java
if((1 == (2 -1)) && (2 == (1 + 1))){
    System.out.println(“Ambas as expressões são verdadeiras”);
}
```

Opções de operadores de lógicos

A tabela abaixo apresenta os operadores lógicos do Java:

&&	Utilizado quando desejamos que as duas expressões sejam verdadeiras.
||	Utilizado quando precisamos que pelo menos uma das expressões sejam verdadeiras.

4.2. Estrutura condicional IF()

As estruturas de condição possibilitam ao programa tomar decisões e alterar o seu fluxo de execução. É por meio delas que podemos dizer ao sistema: “execute a instrução A caso a expressão X seja verdadeira; caso contrário, execute a instrução B”.

A estrutura condicional if/else permite ao programa avaliar uma expressão como sendo verdadeira ou falsa e, de acordo com o resultado dessa verificação, executar uma ou outra rotina.

Na linguagem Java, o tipo resultante dessa expressão deve ser sempre um booleano, pois diferentemente das demais, o Java não converte null ou inteiros como 0 e 1 para os valores true ou false.

Sintaxe do if/else:

```java
if (expressão booleana) {
 // bloco de código 1
}else {
 // bloco de código 2
}
```

As instruções presentes no bloco de código 1 serão executadas caso a expressão booleana seja verdadeira. Do contrário, serão executadas as instruções presentes no bloco de código 2.

O Java utiliza as chaves como delimitadores de bloco, sendo que elas têm a função de agrupar um conjunto de instruções. Apesar do uso desses delimitadores ser opcional caso haja apenas uma linha de código, ele é recomendado, pois facilita a leitura e manutenção do código, tornando-o mais legível.

Complementar ao if/else temos o operador else if. Esse recurso possibilita adicionar uma nova condição à estrutura de decisão para atender a lógica sendo implementada.

Sintaxe do if/else com else if:

```java
if (expressão booleana 1) {
 // bloco de código 1
} else if (expressão booleana 2) {
 // bloco de código 2
} else {
 // bloco de código 3
}
```

Dessa forma, se a expressão booleana 1 for verdadeira, o bloco de código 1 será executado. Caso seja falsa, o bloco de código 1 será ignorado e será testada a expressão booleana 2. Se ela for verdadeira, o bloco de código 2 será executado. Caso contrário, o programa vai ignorar esse bloco de código e executar o bloco 3, declarado dentro do else.

Podemos utilizar quantos else if forem necessários. Entretanto, o else deve ser adicionado apenas uma vez, como alternativa ao caso de todos os testes terem falhado.

Exemplo de condicional IF simples:

java

```
int qtdeVenda;
int qtdeEstoque;
int estoqueMinimo;

estoqueMinimo=50;
qtdeVenda=140;
qtdeEstoque=170;

int total=qtdeEstoque - qtdeVenda;
System.out.println("Quantidade atual é de:
"+total);
if(total>estoqueMinimo)
{System.out.println("Repor estoque mínimo de
50 unidades");
}else{
System.out.println("Estoque em dia");
}
```

No exemplo acima, foram criadas três variáveis, quantidade de venda, quantidade no estoque e estoque mínimo, uma outra variável foi criada para calcular a diferença entre a quantidade vendida e o que tem no estoque.

Uma condição foi criada para verificar se a quantidade vendida baixou o estoque. Se isso aconteceu, uma mensagem informa que o mínimo é de 50 unidades, caso contrário, o estoque está em dia.

Criando um exemplo utilizando o operador lógico "&&".

No exemplo abaixo, estaremos fazendo um teste em um cadastro de revendedor, onde o usuário deve morar na cidade de Montenegro e ser maior de idade para ser aprovado no sistema.

java

```
String cidade;
Integer idade;
EditText resultado;

resultado=(EditText)
findViewById(R.id.resultado);
cidade="Montenegro";
idade=17;

if(cidade=="Porto Alegre" && idade>17){
 resultado.setText("Dados confirmados;
}else{
 resultado.setText("Cidade ou idade não
conferem com o exigido");
}
```

Para o exemplo acima funcionar, deve ser inserido um componente do tipo Plain Text.

4.3. Exercícios

O objetivo do **primeiro** exercício é criar um controle de vendas de segunda, terça e quarta, assim como calcular o total dos valores vendidos nestes dias.

1) Crie um novo projeto e faça as alterações necessárias para o funcionamento do mesmo.

2) Clique na aba activity_main.xml e clique na categoria Text e arraste 1 TextView e 4 PlainText.

3) Clique na categoria Buttons e arraste 1 Button até entre o terceiro e o quarto PlainText.

-Definindo o ID para cada componente (tirando o TextView): edSeg, edTer, edQua, btSomar e edRes.

4) Criando espaçamento entre os componentes. Clique em cada componente e clique no botão "Create Top Constraint".

5) Exclua os textos dos componentes PlainText e defina o texto do TextView para "Vendas da semana".

6) Clique na aba MainActivity.java e crie as variáveis abaixo:

```
EditText edSeg;
EditText edTer;
EditText edQua;
EditText edRes;
Button btSomar;
```

7) Localizando os componentes.

```
edSeg=(EditText)findViewById(R.id.edSeg);
edTer=(EditText)findViewById(R.id.edTer);
edQua=(EditText)findViewById(R.id.edQua);
edRes=(EditText)findViewById(R.id.edRes);
btSomar=(Button)findViewById(R.id.btSomar);
```

8) Criando evento para o botão somar e definindo o valor de cada componente.

```
btSomar.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view) {
        Double qtdeSeg;
        Double qtdeTer;
        Double qtdeQua;
        Double resultado;

        qtdeSeg=Double.parseDouble(edSeg.getText().toString());
        qtdeTer=Double.parseDouble(edTer.getText().toString());
        qtdeQua=Double.parseDouble(edQua.getText().toString());
        resultado=qtdeSeg+qtdeTer+qtdeQua;
        edRes.setText(String.valueOf(resultado));
    }
});
```

9) Veja o resultado clicando no botão Run.

Este **segundo** exercício tem como objetivo criar um espaço para o usuário preencher o valor e o mês da parcela. Também iremos criar a seguinte situação: se o cliente pagou no mês de janeiro, a mensagem “Pago” será exibida, caso contrário, a mensagem “Em aberto” será exibida.

1) Crie um novo projeto;

2) Clique na aba activity_main.xml e clique na categoria Text e arraste 1 TextView e 3 PlainText.

3) Clique na categoria Buttons e arraste 1 Button até entre o terceiro e o quarto PlainText.

-Definindo IDs: edValor, edMes, btConfirmar e edRes.

-Apague os textos de todos PlainTexts, defina o texto do botão como "Confirmar" e o do TextView como "Controle de pagamento".

4) Clique em MainActivity.java e crie as seguintes variáveis:

```
EditText edValor;  2 usages
EditText edMes;  2 usages
EditText edRes;  3 usages
Button btConfirmar;  2 usages
```

5) Localizando os componentes:

```
edValor=(EditText)findViewById(R.id.edValor);
edMes=(EditText)findViewById(R.id.edMes);
edRes=(EditText)findViewById(R.id.edRes);
btConfirmar=(Button)findViewById(R.id.btConfirmar);
```

6) Criando um evento no botão:

```
btConfirmar.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view) {
        Double valor;
        String mes;
        valor=Double.parseDouble(edValor.getText().toString());
        mes=edMes.getText().toString();
        if(mes.equals("Janeiro")){
            edRes.setText("Pago");
        }else{
            edRes.setText("Em aberto");
        }
    }
});
```

7) Veja o resultado clicando no botão Run.

anotações

Outro comando para tomada de decisões é o switch. Seu funcionamento é bem simples, testamos uma expressão e, de acordo com o seu resultado, executamos um determinado bloco de comandos.

O switch vai funcionar como um interruptor, pois, dependendo da entrada que você der a ele, serão acionados somente certo (s) comando (s) dentre os que você disponibilizou.

É como se você criasse um menu, ou cardápio, e com o switch você escolhesse o que vai querer.

Declaração e Sintaxe do comando switch

Em Java, usamos e declaramos o comando switch da seguinte maneira:

java

```
switch( opção ){
case opção1:
    comandos caso a opção 1 tenha sido
escolhida
break;
case opção2:
    comandos caso a opção 2 tenha sido
escolhida
break;
case opção3:
    comandos caso a opção 3 tenha sido
escolhida
break;
default:
    comandos caso nenhuma das opções
anteriores tenha sido escolhida
}
```

A variável 'opção' geralmente é um inteiro ou caractere.

Se 'opção' receber 'opção1' como entrada, são os códigos contidos na 'case opção1' que serão executados.

Se 'opção' receber 'opção2' como entrada, são os códigos contidos na 'case opção2' que serão executados.

Se 'opção' receber 'opção3' como entrada, são os códigos contidos na 'case opção3' que serão executados.

Se 'opção' receber algum valor diferente das opções 'opção1', 'opção2' ou 'opção3', são os códigos contidos em 'default' que serão executados.

Veja um exemplo onde estaremos simulando a escolha do dia da semana.

java

```
Integer diaDaSemana;
EditText resultado1;
resultado1=
(EditText)findViewById(R.id.resultado1);
diaDaSemana=2;

 switch(diaDaSemana){
case 1:
  resultado1.setText("Domingo");
break;
case 2:
  resultado1.setText("Segunda");
break;
case 3:
  resultado1.setText("Terça");
break;
case 4:
  resultado1.setText("Quarta");
break;
case 5:
  resultado1.setText("Quinta");
break;
case 6:
  resultado1.setText("Sexta");
break;
case 7:
  resultado1.setText("Sábado");
break;
default:
  resultado1.setText("Este não é um dia
válido");
```

```
break;
}
```

Curiosidade

A estrutura switch-case pode oferecer uma performance melhor que múltiplos if-else quando se trata de avaliar muitas condições, especialmente com tipos primitivos e enums.

5.1. Tratamento de texto

Sobre tratamento de texto, estaremos melhorando o desempenho do código e facilitando a sua manutenção utilizando alguns métodos.

Os métodos da classe String permitem unir textos, comparar conteúdos e buscar dados, entre outras funcionalidades.

Em Java, String é uma sequência de caracteres utilizada para representação e manipulação de texto.

Para representar informações textuais em aplicações (seja para Desktop ou Web), como nomes, endereços ou comentários, usamos instâncias da classe String. Elas são indispensáveis para exibir dados em um sistema.

Conhecendo alguns métodos:

toUpperCase()

O método toUpperCase() retorna uma nova String com o mesmo conteúdo da original, só que com todos os caracteres em letras maiúsculas.

java

```
String nomeCliente;EditText resultado1;

resultado1=
(EditText)findViewById(R.id.resultado1);

nomeCliente="joão da silva";

resultado1.setText(nomeCliente.toUpperCase()
);
```

O resultado deste método vai ser:

JOÃO DA SILVA

toLowerCase()

O método toLowerCase converte toda a String para caixa baixa e o toUpperCase faz o inverso, convertendo toda a String para caixa alta.

java

```
String nomeCliente;EditText resultado1;

resultado1=
(EditText)findViewById(R.id.resultado1);

nomeCliente="JOÃO DA SILVA";

resultado1.setText(nomeCliente.toLowerCase()
);
```

O resultado deste método vai ser:

João da silva.

trim()

O método trim remove espaços em branco no inicial e no final da String.

java

```
String nomeCliente;EditText resultado1;

resultado1=
(EditText)findViewById(R.id.resultado1);

nomeCliente="    JOÃO DA SILVA   ";

resultado1.setText(nomeCliente.trim());
```

O resultado deste método será a remoção dos espaços, conforme o digitado acima:

JOÃO DA SILVA

5.2. Layout

Alguns atributos são essenciais para determinar como exemplo a largura e altura do layout, o alinhamento das informações, como a utilização de cores no fundo da tela, cor de texto, tamanho de letra e o modo de exibição dos dados com relative ou linear, além de definição de margem.

O layout define a estrutura visual para uma interface do usuário, como a IU de uma atividade ou de um widget de aplicativo. É possível declarar um layout de dois modos:

Declarar elementos da IU em XML. O Android fornece um vocabulário XML direto que corresponde às classes e subclasses de View, como as de widgets e layouts.

Instanciar elementos do layout em tempo de execução. O aplicativo pode criar objetos View e ViewGroup (e processar suas propriedades) programaticamente.

A estrutura do Android dá a flexibilidade de usar um desses métodos ou ambos para declarar e gerenciar a IU do aplicativo.

Por exemplo, você pode declarar os layouts padrão do aplicativo em XML, incluindo os elementos da tela que aparecerão neles e em suas propriedades. Em seguida, você poderia adicionar código ao aplicativo que modificaria o estado dos objetos da tela, inclusive os declarados em XML, em tempo de execução.

Todos os grupos de exibições contêm largura e altura (layout_width e layout_height), e cada exibição é obrigatória para defini-las.

Width = largura

Height = altura

DP:

A unidade de medida utilizada é o DP (Density-independent Pixel), que se ajusta à resolução da tela. Por exemplo, em uma tela de 160 dpi, 1 dp equivale a 1 pixel.

USO: aconselhamos que, ao invés de usar o px, sempre use o dp.

Curiosidade

Ao trabalhar com layouts no Android, você pode defini-los tanto via XML quanto programaticamente. Essa flexibilidade permite separar a lógica da interface (usando XML) ou criar componentes dinamicamente conforme necessário.

SP:

Scale-independent Pixels (sp) são como o dp, mas consideram também o tamanho da fonte definido pelo usuário. Recomenda-se usá-los para definir tamanhos de fonte, garantindo ajuste conforme as preferências do usuário.

Dicas

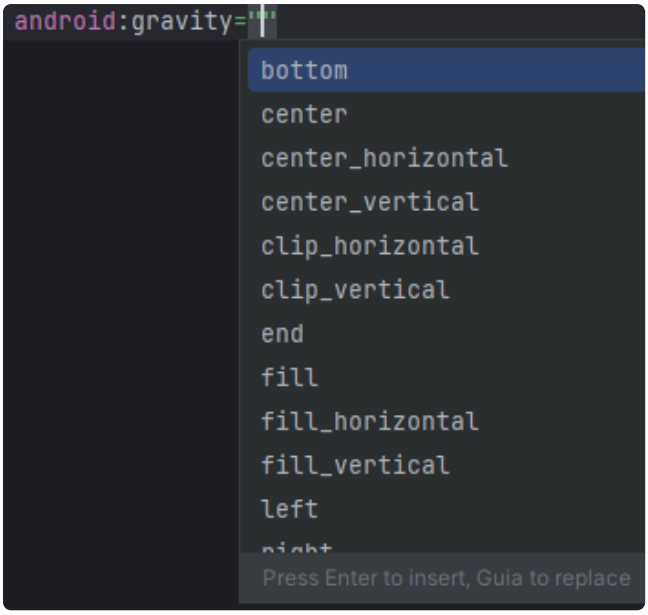
Sempre use DP para dimensões (largura, altura, margens) e SP para definir tamanhos de fonte. Essas unidades garantem que sua interface se adapte corretamente a diferentes dispositivos e respeite as preferências do usuário.

Gravidade Android

android: gravidade é um atributo que define a gravidade do conteúdo da exibição usada.

O android: gravidade específica como um objeto deve posicionar seu conteúdo nos eixos X e Y.

Os valores possíveis do android: gravidade (superior, inferior, esquerda, direita, centro, center_vertical, center_horizontal, etc).

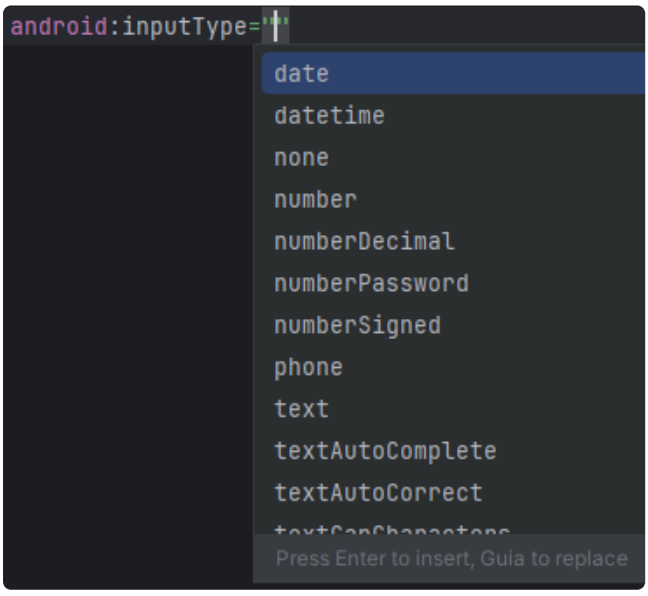


Para alinhar componentes na tela horizontalmente e centralizando, utilizamos o atributo abaixo.

O "android:layout_centerHorizontal" alinha na horizontal os componentes, apenas passando o valor “true” ele já ativa a propriedade.

Para definir cor de fundo no componente, utilizamos o atributo abaixo.

android:background=”#FF00FF” ou padrões do sistema.



Para definir cor no texto, utilizamos o atributo abaixo:

android:textColor=”#FFFFFF”

Para definir o tamanho do texto, utilizamos o atributo abaixo:

android:textSize=”20sp”;

5.3. Exercício

O objetivo do exercício desta apostila é criar um controle de promoções, alguns produtos terão desconto, ocorrendo da seguinte forma:

Calça – 3%, Camiseta – 5%, Chinelo – 7%, Bermuda – 10%, outros – sem desconto.

1) Criar um novo projeto:

- Abra o Android Studio.
- Crie um novo projeto selecionando "Empty Activity".
- Defina o nome do projeto, local de armazenamento e linguagem como Java ou Kotlin, de acordo com sua preferência.
- Certifique-se de que a versão do SDK está configurada corretamente (API mínima recomendada: 21 - Android 5.0 Lollipop).
- Verifique o arquivo build.gradle (app) e confirme se as versões do Gradle e plugins estão atualizadas. Altere se necessário, para garantir o funcionamento do projeto.

2) Configurando o layout (activity_main.xml):

- No arquivo activity_main.xml, altere o layout padrão para ConstraintLayout (se não for o padrão) para facilitar o posicionamento manual dos componentes ou utilize o layout linear na vertical para automatizar.

Adicione os componentes necessários:

- 1 TextView para o título ou instrução: "Digite o produto:"
- 1 EditText (Plain Text): um para entrada do nome do produto (ID: edProd).
- 1 Button: para confirmar o cálculo do desconto (ID: btConfirmar).
- 1 outro EditText para mostrar o resultado (ID: edRes).

3) Criando espaçamento entre os componentes (Caso esteja utilizando o espaçamento manual). Clique em cada componente e clique no botão “Create Top Constraint”.

4) Clique na aba MainActivity.java e crie as variáveis abaixo:

```
EditText edProd;  no usages
EditText edRes;  no usages
Button btConfirmar;  no usages
```

5) Localizando os componentes.

```
edProd=(EditText)findViewById(R.id.edProd);
edRes=(EditText)findViewById(R.id.edRes);
btConfirmar=(Button)findViewById(R.id.btConfirmar);
```

6) Criando evento para o botão confirmar e setando o valor de cada componente.

```
btConfirmar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String produto;
        produto=edProd.getText().toString();
        switch (produto){
            case "Calça":
                edRes.setText("Desconto de 3%");
                break;
            case "Camiseta":
                edRes.setText("Desconto de 5%");
                break;
            case "Chinelo":
                edRes.setText("Desconto de 7%");
                break;
            case "Bermuda":
                edRes.setText("Desconto de 10%");
                break;
            default:
                edRes.setText("Sem desconto");
                break;
        }
    }
});
```

7) Veja o resultado clicando no botão Run.

anotações

Matrizes são usadas para armazenar vários valores em uma única variável, em vez de declarar variáveis separadas para cada valor.

Para declarar uma matriz, defina o tipo de variável com colchetes:

String[] cars;

Nós agora declaramos uma variável que contém um array de strings. Para inserir valores, podemos usar um literal de matriz - coloque os valores em uma lista separada por vírgula, dentro de chaves:

java

```
String[] cars = {"Volvo", "BMW", "Ford",
"Mazda"};
```

Você acessa um elemento da matriz referindo-se ao número do índice.

Esta declaração acessa o valor do primeiro elemento em carros:

java

```
String[] cars = {"Volvo", "BMW", "Ford",
"Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

Os arrays ou matrizes, como são conhecidos pelo Java, fazem parte do pacote java.util, na coleção da API do Java. São objetos de recipientes que contém um número fixo de valores de um único tipo. O comprimento de um array é estabelecido quando criado, sendo que, após a criação, o seu comprimento fica fixo.

Cada item em um array é chamado de elemento, e cada elemento é acessado pelo número, o índice.

A seguir é mostrado como é acessado os seus elementos, lembrando que sempre sua numeração começa em 0.

Opala	Chevette	Rural Willys	Golf
0	1	2	3

O nosso exemplo é um array de quatro carros, sendo que vamos exibir o elemento de índice 2.

java

```
String[] carros={"Opala","Chevette","Corcel
II","Rural"};
EditText resultado1;

resultado1=
(EditText)findViewById(R.id.resultado1);
resultado1.setText(carros[2]);
```

O resultado deste array é:

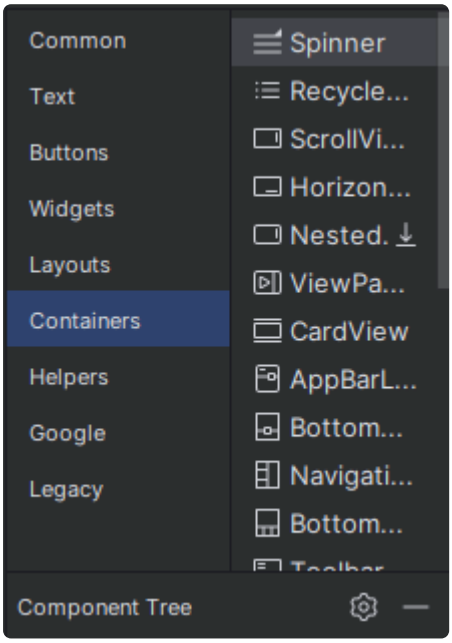
Corcel II

Componente Spinner

O componente Spinner é uma versão do Select (HTML) ou ComboBox.

Muitas vezes, temos a necessidade de personalizar componentes no decorrer do desenvolvimento de algum projeto. Na realidade, é isso que deixa nosso software com um layout diferente dos outros. Quando trabalhamos com a programação para Android, é inevitável não nos depararmos com uma situação dessas.

Neste artigo, irei personalizar um dos principais componentes, o “Spinner”. Ele fornece uma maneira rápida de selecionar um valor a partir de um conjunto de valores. No estado padrão, um “Spinner” mostra seu valor atualmente selecionado. Ao selecioná-lo, é exibido um menu suspenso com todos os outros valores disponíveis, a partir do qual o usuário pode selecionar um novo. Ele se parece com o “Combobox”.



Encontramos o componente na categoria Containers.

Após o componente ser inserido na tela, ele recebe um ID automático com base em quantos componentes iguais você adicionou em seu projeto, sempre somando um número (Spinner, Spinner2, Spinner 3, ...).

Criando uma variável:

java

```
Spinner cidades;

Variável recebendo conteúdo do componente:
cidades=(Spinner)findViewById(R.id.spinner);

cidades – é a variável.
(Spinner) – é o componente.
findViewById – é para localizar um ID.
```

Curiosidade

Funciona como um ComboBox ou select do HTML, permitindo que o usuário escolha um valor de uma lista suspensa.

ArrayAdapter

No desenvolvimento Android, para exibir uma lista vertical com rolagem em nossa tela, utilizamos o componente ListView, e uma maneira eficiente de popular os dados neste componente é através do Adapter.

Uma manera simples de criar um adapter é usar um ArrayAdapter, pois o adapter converte cada objeto do ArrayList em itens (View) da ListView que iremos popular.

java

```
ArrayAdapter adapter =
ArrayAdapter.createFromResource(this,R.array
.materiais,android.R.layout.simple_spinner_i
tem);
```

Segundo item do array é a variável “adapter”, que vai receber o conteúdo que vai ser localizado com o createFromResource, é passado o array.

Layout de item – escolhemos o simple_spinner_item, que é um simples TextView, como o layout para cada um dos itens.

setAdapter().

Permite listar o conteúdo do array.

6.1. Activity

A navegação entre telas/páginas é uma necessidade comum da maioria das aplicações, sejam elas mobile, web ou desktop. No Android, as telas da aplicação são representadas por activities.

A primeira situação a ser mostrada é a de navegação entre activities. Com ela, é possível iniciar novas activities, além de poder passar parâmetros para as activities iniciadas.

Veja o exemplo:

java

```
Intent intent = new Intent(this,
NewActivity.class);
startActivity(intent);
```

Na segunda linha, basta chamar o startActivity, passando a intent por parâmetro para que seja executada pelo Android a criação da nova activity.

6.2. Exercício

O objetivo do exercício de fixação desta apostila é criar uma lista de bandas utilizando Array String.

1) Crie um novo projeto e faça os ajustes necessários para o mesmo funcionar sem problemas.

2) Clique na aba activity_main.xml, arraste um LinearLayout Vertical e depois, na categoria Containers, arraste o componente Spinner para dentro desse layout.

3) Clique na seta “app/res/values” e abra o arquivo styles.xml.

4) No arquivo styles, crie um "String Array" com o nome "Discos" e adicione os seguintes itens: "Selecione", "Beatles", "Creedance", "A-ha" e "U2".

5) Clique na aba MainActivity.java e atribua o id do componente Spinner à uma nova variável chamada "Spinner sistema;".

6) Agora, você deverá definir um ArrayAdapter definindo o nome "Discos" que você concedeu ao String Array criado anteriormente juntamente com o layout do Spinner (simple_spinner_item).

7) Para finalizar, defina o 'adapter' que você criou na variável chamada "sistema".

8) Veja o resultado clicando no botão Run.

anotações

Orientação a objetos

Programas grandes são difíceis de manter, por isso é um bom hábito separá-los em unidades mais ou menos isoladas. Em Java, isso é feito utilizando objetos, que são compostos por atributos e métodos definidos a partir de classes que, por sua vez, são organizadas em pacotes. Esses conceitos são tão centrais em Java que não se pode programar na linguagem sem utilizá-los.

Todo programa em Java usa classes e objetos, por isso compreender esses conceitos é fundamental para entender a própria linguagem. Na prática, sistemas de software reais são grandes e precisam ser fatorados em partes relativamente independentes para serem viáveis. Como em Java isso é feito com classes e objetos, compreendê-los é imprescindível para escrever qualquer programa significativo.

Dica

Utilize os conceitos de orientação a objetos (encapsulamento, herança, polimorfismo e abstração) para estruturar seu código de forma modular e reutilizável. Isso facilita a manutenção e evolução do seu app, permitindo que componentes sejam facilmente adaptados ou substituídos conforme novas funcionalidades são implementadas.

7.1. Classes

Em Java, os programas são escritos em pequenos pedaços separados, chamados de objetos. Objetos são pequenos programas que guardam dentro de si os dados, em suma, as variáveis que precisam para executar suas tarefas. Os objetos também trazem em si, como sub-rotinas, as instruções para processar esses dados. As variáveis que um objeto guarda são chamadas de atributos, as suas sub-rotinas, de métodos.

Em Java, objetos são criados a partir de modelos que os descrevem. Esses modelos são chamados de classes. É dentro dessas classes que definimos que atributos os objetos conterão e que métodos os objetos fornecerão.

Exemplo:

Vamos começar apenas com o que uma **Conta** tem, e não com o que ela faz (veremos logo em seguida).

Um tipo desses, como o especificado de Conta acima, pode ser facilmente traduzido para Java:

java

```java
class Conta {
int numero;
String titular;
double saldo;
}
```

String

String é uma classe em Java. Ela guarda uma cadeia de caracteres, uma frase completa.

Por enquanto, declaramos o que toda conta deve ter. Estes são os atributos que toda conta, quando criada, vai ter.

Características das classes

-Toda classe possui um nome;

-Possuem visibilidade, exemplo: public, private, protected;

-Possuem membros como: Características e Ações;

-Para criar uma classe, basta declarar a visibilidade + digitar a palavra reservada class + NomeDaClasse + abrir e fechar chaves { }.

java

```
public class Teste{
     //ATRIBUTOS OU PROPRIEDADES
     //MÉTODOS
}
```

7.2. Métodos

Dentro da classe, definimos os comportamentos (métodos) de cada objeto, ou seja, o que ele faz e como faz. Por exemplo, a forma como uma conta realiza um saque é especificada na própria classe, mantendo as informações centralizadas. Esses comportamentos são chamados de métodos, pois representam ações realizadas pelos objetos.

Queremos criar um método que saca uma determinada quantidade e não devolve nenhuma informação para quem acioná-lo:

java

```
class Conta {
 double salario;
  // ... outros atributos ...
 void saca(double quantidade)
    double novoSaldo = this.saldo -
quantidade;
   this.saldo = novoSaldo;
 }
}
```

A palavra-chave void diz que, quando você pedir para a conta sacar uma quantia, nenhuma informação será enviada de volta a quem pediu.

Ao realizar um saque, é necessário informar o valor a ser sacado, que é passado como argumento do método. Esse argumento é uma variável temporária ou local, que só existe durante a execução do método.

Dentro do método, ao declarar uma nova variável, ela existirá apenas durante a execução do método. Para acessar um atributo da classe, usamos a palavra-chave **this** para diferenciá-lo de uma variável local, embora seu uso seja opcional.

Métodos com retorno

Um método sempre tem que definir o que retorna, nem que defina que não há retorno, como nos exemplos anteriores onde estávamos usando o void.

Um método pode retornar um valor para o código que o chamou. No caso do nosso método saca, podemos devolver um valor booleano indicando se a operação obteve sucesso.

java

```
class Conta {
 // ... outros métodos e atributos ...
 boolean saca(double valor) {
   if (this.saldo < valor) {
    return false;
   }else {
    this.saldo = this.saldo - valor;
    return true;
   }
 }
}
```

A declaração do método mudou! O método saca não tem void na frente. Isso quer dizer que, quando é acessado, ele devolve algum tipo de informação. No caso, um booleano. A palavra-chave return indica que o método vai terminar ali, retornando tal informação.

Exemplo de uso:

java

```
minhaConta.saldo = 1000;
boolean consegui = minhaConta.saca(2000);
if (consegui) {
   System.out.println("Consegui sacar");
```

```
}else {
   System.out.println("Não consegui sacar");
}
```

7.3. Herança

O que é herança na vida real?

É quando uma pessoa deixa seus bens para outra. No Java também.

Geralmente a herança ocorre entre membros de uma mesma família. No Java também.

Herança, em Java, nada mais é do que criar classes usando outras já existentes.

Obviamente, você vai fazer uma classe herdar as características de outra se essas tiverem uma relação (se forem parecidas).

Outro ponto importante é que, quando fazemos uso da herança, nós podemos adicionar mais atributos à classe.

Dica

Ao desenvolver para Android, estenda classes como Activity e Fragment para aproveitar funcionalidades já implementadas, mas combine com a composição quando precisar de maior flexibilidade e desacoplamento.

Exemplo 1: Carros e motos

Imagine que você tem uma revenda de veículos: carros e motos.

Todos são veículos. Ora, crie a classe "Veículo".

O que esses veículos têm em comum?

Motor, preço, marca, nome do cliente que vai comprar, quantos quilômetros fazem com 1 litro de combustível, etc.

Todos os objetos da classe “Veículo” têm essas características.

No entanto, existem algumas características que as motos têm que um carro não tem: capacete, somente duas rodas, cilindrada, etc.

Também existem características que os carros têm que as motos não têm: podem ter 4 portas, banco de couro, ar-condicionado, etc.

Para resolver esse problema e deixar a aplicação MUITO MAIS ORGANIZADA, vamos fazer duas classes: "Moto" e "Carro".

Cada uma dessas irá herdar a classe "Veículo", pois também são veículos.

Dizemos que "Veículo" é a superclasse, "Moto" e "Carro" são subclasses.

7.4. Exercício

No exercício da nossa apostila vamos fixar o que aprendemos durante a aula, que foram as classes e suas heranças. O objetivo deste exercício é criar três classes: Funcionários, Gerente (que herda atributos de Funcionários), e Principal, que instancia objetos das classes anteriores.

Etapa 1: Configurando o novo projeto

- Crie um novo projeto.

- Aplique as configurações necessárias para seu funcionamento.

Etapa 2: Configurando o ambiente

- Abra o Android Studio e localize a pasta app/java/com/ dentro do seu projeto.

- Certifique-se de que o arquivo MainActivity.java está acessível, pois será usado como referência para testar suas classes.

Etapa 3: Criando a classe Funcionários

- Crie um novo arquivo Java Class.

- Nomeie a classe como Funcionarios (lembre-se de iniciar com letra maiúscula).

- Dentro da classe Funcionarios, crie as seguintes variáveis públicas:

nome, cidade e salario.

Etapa 4: Criando a classe Gerente com herança

- Crie um novo arquivo Java Class.

- Nomeie a classe como Gerente e pressione Enter.

- Para que a classe Gerente herde os atributos da classe Funcionarios, use a palavra-chave extends.

- Adicione uma variável pública chamada 'cargo' específica para a classe Gerente.

Etapa 5: Configurando o arquivo principal para testar(MainActivity)

- Crie uma variável para se comunicar com a classe Gerente.

- Defina valores para os atributos herdados e próprios.

- Exiba os valores no console através do comando println.

Etapa 6: Compilando e executando

- Clique no botão Run (ícone de seta verde) para compilar e executar o código.

-Verifique o LogCat para confirmar se os valores definidos foram exibidos corretamente:

anotações

São padrões de visibilidade de acessos às classes, atributos e métodos. Esses modificadores são palavras-chaves reservadas pelo Java, ou seja, palavras reservadas não podem ser usadas como nome de métodos, classes ou atributos.

Como boas práticas do Java, na maioria das declarações de variáveis de instância são definidos os seus atributos com a palavra-chave private para garantir a segurança de alterações acidentais, sendo somente acessíveis através dos métodos. Essa ação tem como efeito ajudar no encapsulamento dos dados, preservando ainda mais a segurança e a aplicação de programação orientada a objetos do Java.

Public

Uma declaração com o modificador public pode ser acessada de qualquer lugar e por qualquer entidade que possa visualizar a classe a que ela pertence.

Private

Os membros da classe definidos como private não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.

Protected

O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

Default (padrão):

A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo esse identificado pelo compilador.

Para utilizarmos estes modificadores de acesso, basta que nós os digitemos antes do nome da variável, atributo, método, função ou classe, com exceção de package-private, que é entendido como padrão, portanto, é qualquer membro ou classe que não tenha modificador especificado.

Exemplo:

java

```java
public class MinhaClasse {
  //classe public

  private int inteiro;
  //atributo inteiro private

  protected float decimal;
  //atributo float protected

  boolean ativado;
  //atributo booleano package-private
}
```

Por padrão, a linguagem Java permite acesso aos membros apenas ao pacote em que ele se encontra.

De forma ilustrativa, abaixo está uma tabela demonstrando todas estas características.

Modificador	Classe	Pacote	Subclasse	Globalmente
Public	Sim	Sim	Sim	Sim
Protected	Sim	Sim	Sim	Não
Sem modificador (Padrão)	Sim	Sim	Não	Não
Private	Sim	Não	Não	Não

Fonte: Controlling Access to Members of a Class

Um método com modificador de acesso public pode ser acessado por qualquer classe em qualquer pacote. Já um método private pode ser acessado somente pela classe que o implementa. Ou seja, se o método for public, você consegue enxergá-lo fora da classe, se for private, não conseguirá, porque apenas a classe que gerou este método pode usá-lo.

java

```
class A{
   public void methodPublic(){
      methodPrivate();
   }
   private void methodPrivate(){
   }
}
   class B{
      public void teste(){
      A a = new A();
      //Correto
      a.methodPublic();
      //Erro de Compilação
      a.methodPrivate();
   }
}
```

8.1. Exercício

Etapa 1: Configuração Inicial

- Abra o Android Studio e prepare um projeto funcional para teste. Certifique-se de que a estrutura do projeto está corretamente configurada e que você tem acesso à pasta app/java/com/seu_pacote.

- Localize o arquivo MainActivity.java, que será usado como ponto de entrada para as interações.

Etapa 2: Definindo a Classe Carro

- Crie uma nova classe chamada Carro. Lembre-se de seguir a convenção de nomenclatura em maiúsculas para o início da classe.

- Na classe Carro, inclua:

Uma variável pública ou var, chamada marca, para armazenar o nome do fabricante do carro.

Outra variável chamada valor, inicializada com um valor padrão de 10000.0.

Um método acessível que permita aplicar um desconto. Esse método deve receber o valor do desconto como parâmetro, subtrair do valor do carro e atualizar a propriedade valor.

Etapa 3: Implementação na MainActivity

- Instancie a classe Carro dentro da função onCreate da MainActivity.

- Defina a marca do carro como desejado.

- Aplique um desconto de 350.0 utilizando o método previamente criado.

- Exiba o valor final do carro no Logcat, utilizando o comando println.

Etapa 4: Testando a Implementação

- Execute o projeto clicando em Run no Android Studio.

- Verifique o Logcat para validar que o valor atualizado corresponde ao esperado após a aplicação do desconto.

anotações

Um Constraint Layout é uma descrição de como deve ser posicionada uma View em relação aos outros elementos do layout. Pode-se definir uma constraint para um ou mais views, ligando-se a view a:

-Um ponto de ancoragem em um outro ponto da view;

-Uma lateral do layout;

-Uma invisível linha de instrução.

Como o layout de cada exibição é definido por associação com outros elementos, pode-se criar um layout complexo com uma hierarquia horizontal. Embora seja conceitualmente semelhante ao RelativeLayout, ConstraintLayout é mais flexível e é projetado para o uso inteiramente a partir do novo editor de layouts.

Para melhor posicionamento dos elementos na tela, utilizamos o botão Infer Constraints.

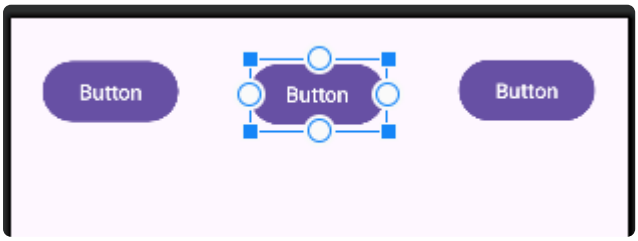
INFER CONSTRAINTS

DELETE CONSTRAINTS

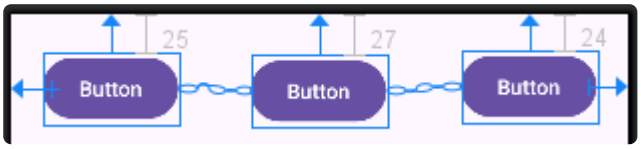
O exemplo é o seguinte:

Distribuir três botões, independentemente se o layout for vertical ou horizontal.

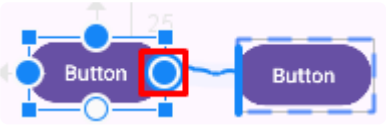
Primeiramente, inserimos três botões.



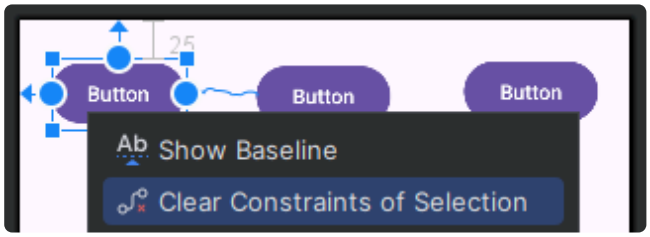
Em segundo lugar, clicamos no botão Infer Constraints, ativando a ligação entre eles.



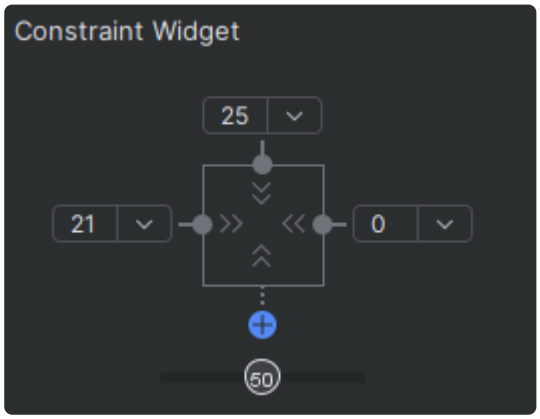
Através deste botão, conseguimos conectar os objetos entre si.



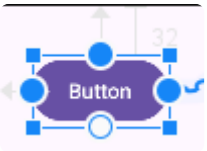
Para excluir a conexão, clique no botão Clear Constraints of Selection.



Podemos ajustar ou excluir as distâncias utilizando o componente ao lado.

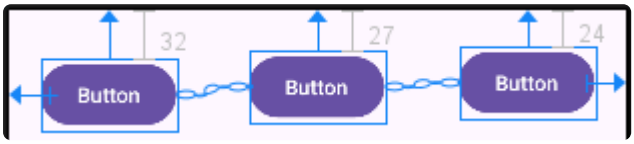


No próximo exemplo, alteramos a distância do topo até o componente para 32dp.

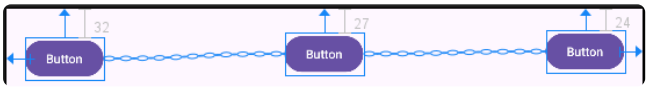


Veja o comportamento em dois modos de visualização.

Veja o modo Portrait:

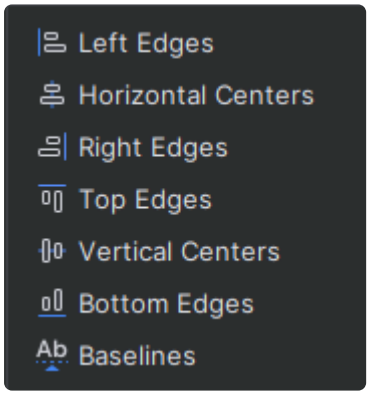


Veja o modo Landscape:



Muito bom, o recurso Infer Constraints fez os ajustes de distribuição automática de acordo com o tipo de layout.

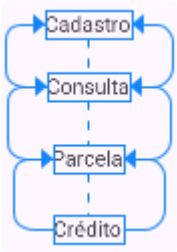
Alinhamento dos componentes na tela.



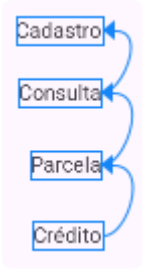
Veja o seguinte exemplo em que digitamos alguns títulos e os bagunçamos na tela.

O que vamos fazer é alinhar todos à esquerda. Para isso, devemos clicar na opção Left Edges.

Para centralizar, clique na opção Horizontal Centers.



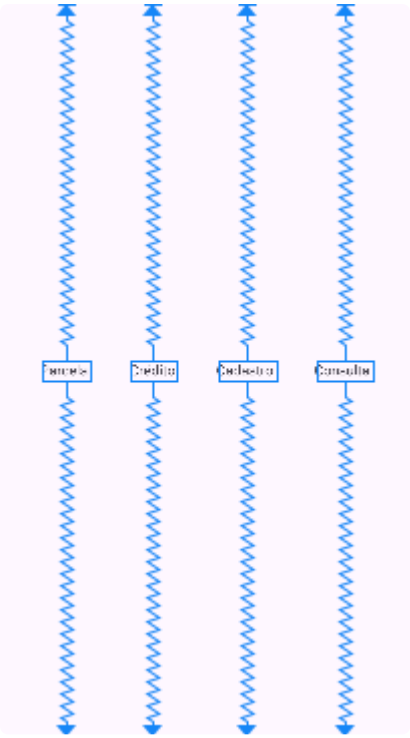
Para alinhar à direita, clique na opção Right Edges.



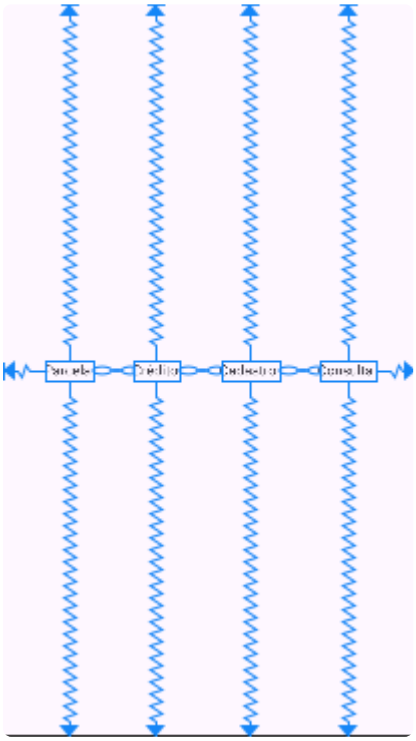
Deslocamos os elementos em outra posição na tela. Com isso, vamos alinhar no topo, escolhendo a opção Top Edges.

Utilizando a opção Vertically ou Horizontally, teremos a distribuição de espaço igualmente entre as extremidades e os componentes.

Exemplo com o uso da opção Vertically.



No exemplo abaixo, utilizamos os recursos Vertically e Horizontally.



9.1. Exercício

O objetivo desse exercício é ajustar os componentes a seguir na tela, conforme o modelo, com o recurso Infer Constraints.

Você deverá reproduzir uma tela semelhante à esta:

1) Crie um novo projeto e realize as configurações necessárias para o funcionamento do mesmo.

2) Clique na categoria "Text" na área de Design e arraste seis componentes "TextView" até a tela.

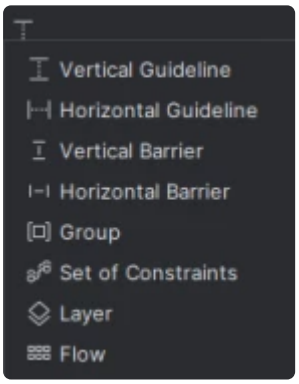
3) Digite e ajuste os TextViews conforme a imagem demonstrativa.

4) Copie da pasta Arquivos Auxiliares/09 a imagem do Notebook e cole na pasta Drawable, clique na categoria Widgets e insira a imagem da pasta Project.

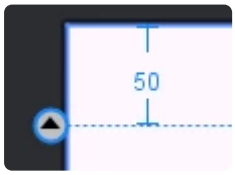
5) Após realizar o posicionamento de todos os componentes, utilize os recursos necessários para torná-los responsivos á tela.

As Guidelines têm como finalidade nos orientar para um melhor posicionamento dos componentes na tela. Aqueles que estão familiarizados com as ferramentas de design gráfico já estarão familiarizados com o conceito de diretrizes porque são comumente usados. Mas, para aqueles que não são: uma diretriz é um guia visual que não será visto em tempo de execução, que é usado para alinhar outras visualizações.

As orientações podem ser aparecer horizontalmente ou verticalmente.



Podemos alterar o tipo de Guideline para unidade de medida em “DP” ou “%”.

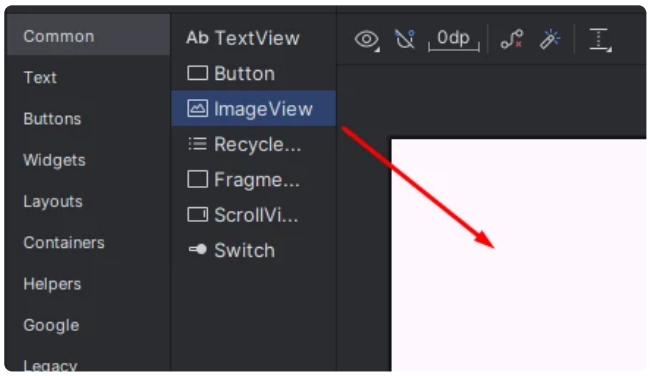


Para modificar, basta clicar no botão de acesso da Guideline que aparece no topo da linha tracejada.

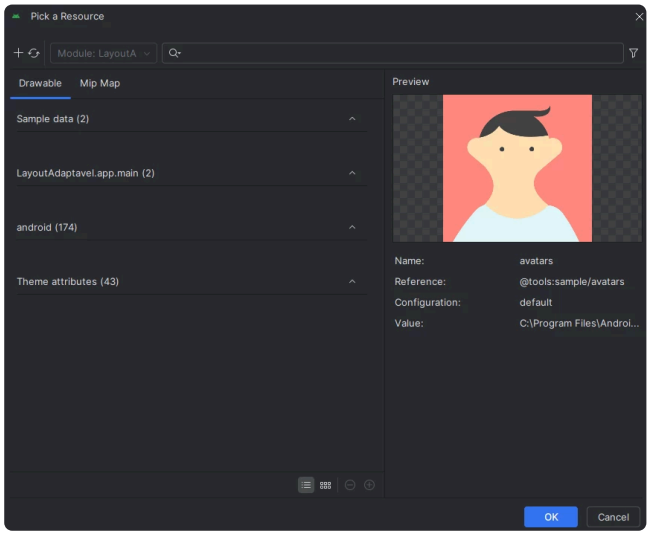
Componente ImageView

O ImageView, como o próprio nome diz, é um componente que nos permite trabalhar com imagens. Iremos mostrar um exemplo em que podemos aplicar esse componente, utilizaremos duas imagens.

Para fazer um teste, iremos inserir dois ícones. Devemos acessar a categoria Commom e arrastar o componente ImageView até a tela.

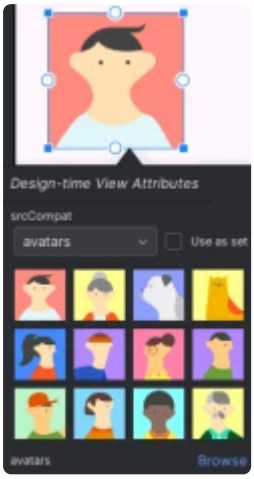


Logo, aparece a caixa de diálogo Resource, onde poderemos selecionar imagens no componente.



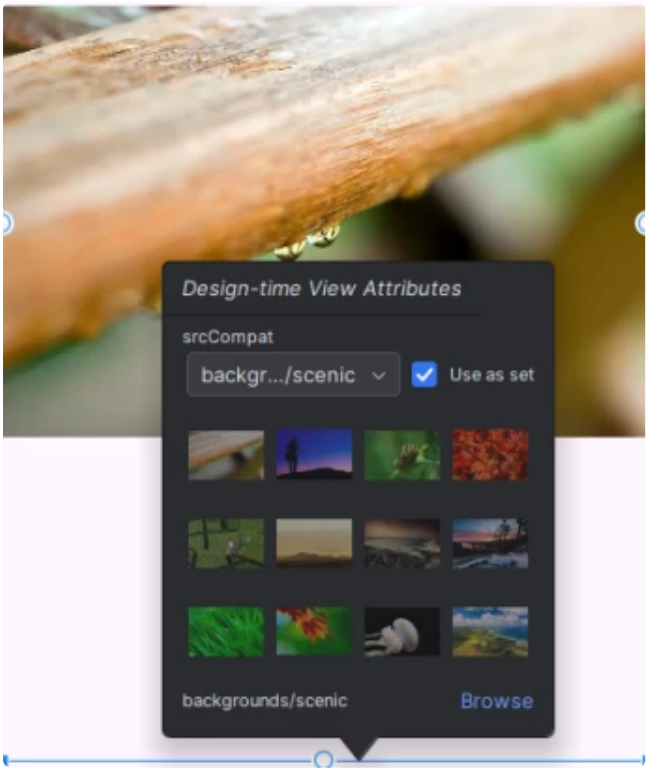
A categoria Drawable exibe as seguintes opções:

Avatars – aqui iremos encontrar algumas imagens de exemplo.

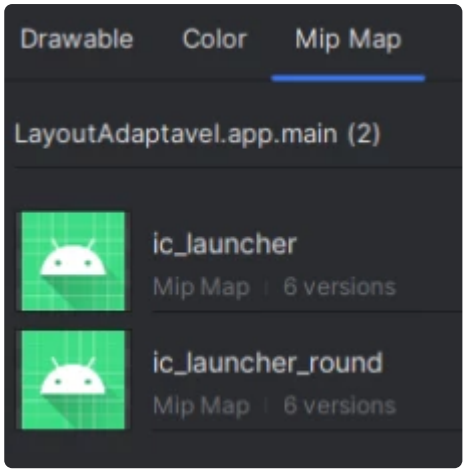


Para selecionar uma imagem da lista, devemos desmarcar a opção Use as set.

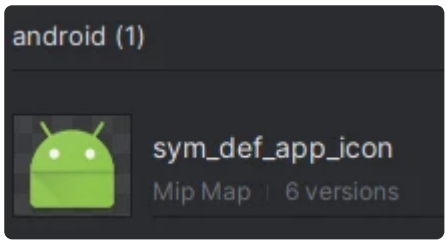
Backgrounds/scenic – aqui iremos encontrar algumas paisagens de exemplo.



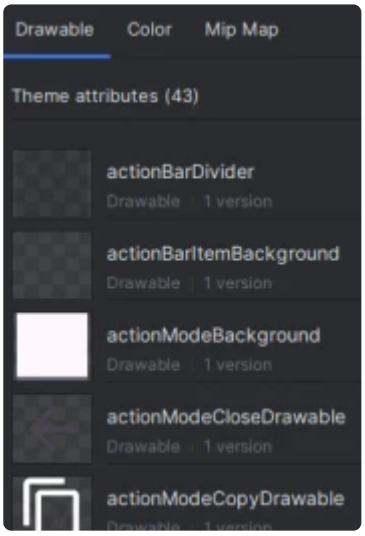
Project – aqui iremos encontrar imagens que copiamos para o Android Studio e outras padrão.



Android – exibe ícone como padrão para personalizar o seu layout.



Theme attributes – traz outros ícones padrão do Android Studio.



Curiosidade

Combinar o ImageView com bibliotecas como Glide ou Picasso permite carregar imagens de forma assíncrona e otimizada, melhorando significativamente a performance do app.

10.1. Exercício

O objetivo será criar um aplicativo simples onde você vai usar a ferramenta Guidelines para alinhar e ajustar os componentes na tela.

O que você precisa fazer:

1. Abra o Android Studio e crie um novo projeto.

2. Faça os ajustes necessários para o funcionamento do mesmo.

3. Adicione alguns componentes na tela, como TextView, Button ou EditText (você escolhe!).

4. Use as Guidelines para organizar esses componentes de forma alinhada e proporcional. As Guidelines podem ser colocadas tanto na horizontal quanto na vertical para ajudar no posicionamento.

5. Experimente mover os componentes e observe como as Guidelines facilitam o ajuste.

anotações

11.1. Chains

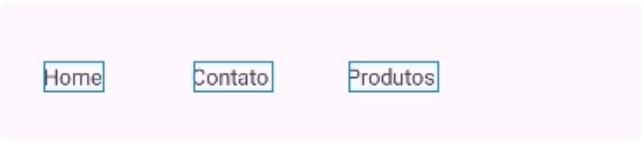
O que são chains (correntes)?

As correntes são um tipo específico de restrição que nos permite compartilhar o espaço entre as visualizações dentro da cadeia e controlar como o espaço disponível é dividido entre elas. O analógico mais próximo dos layouts tradicionais do Android é o peso LinearLayout, mas as cadeias fazem muito mais do que isso, como veremos.

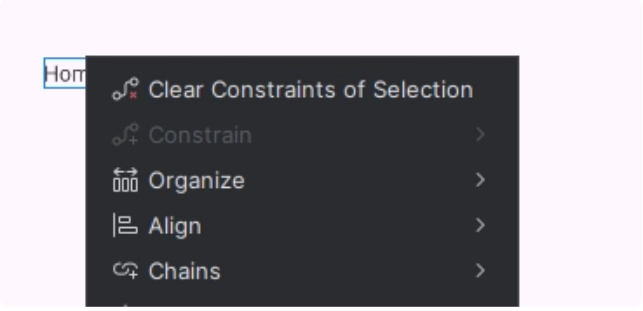
Criando uma corrente

Como já mencionamos, uma cadeia consiste em múltiplas visualizações, portanto, para criar uma cadeia, devemos selecionar as exibições que desejamos encadear e, em seguida, selecionar 'Centralizar na horizontal' para criar uma cadeia horizontal ou 'Centralizar verticalmente' para criar uma corrente vertical.

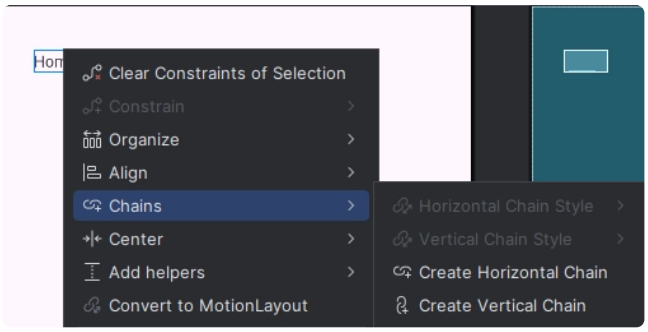
Em primeiro lugar, selecione os elementos da tela.



Em segundo lugar, clique com o botão direito do mouse em qualquer local na tela.

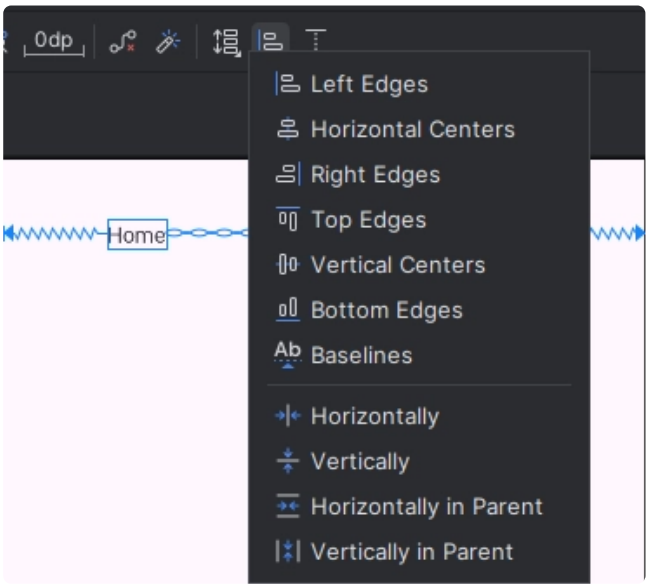


Na lista que surgiu, clique em Chains.

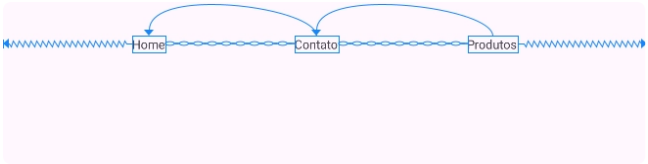


Nesse caso, foi selecionada a opção **Create Horizontal Chain**.

Para auxiliar no alinhamento, vamos clicar em **Align** e escolher **Top Edges**.



Visualizando no modo Landscape.



Dica

Ao usar Chains no ConstraintLayout, experimente os diferentes estilos de chain (spread, spread_inside e packed) para encontrar a distribuição ideal para seus elementos. Dessa forma, você garante um layout responsivo e equilibrado em diferentes tamanhos de tela.

11.2. Componentes de formulário 1

A criação de formulários pode ser feita tanto através da edição do arquivo XML como através do layout gráfico deste arquivo.

Utilizando o layout gráfico, sua construção se dá de maneira intuitiva. Os componentes são arrastados e soltos na tela, podendo ser editados no campo Properties.

Para descrever os campos do nosso formulário, utilizamos a tag **TextView**. Essa tag é utilizada quando é preciso exibir algum texto na tela.

Criando os campos de entrada

O campo que haverá entrada de dados propriamente dita é estabelecido pela tag EditText. Essa tag cria o campo de texto no qual o usuário colocará algum dado.

O componente é o **PlainText**.

Criando campo de entrada para senha.

O campo no qual utilizaremos é **Password**, destinado à inserção de senhas pelo usuário.

Outro componente para digitar a senha é o **Password (Numeric)**, que diferente do **Password**, aceita apenas números.

Barra de ferramentas – Attributes

ID – permite criar uma identificação para o componente.

Layout_width – permite definir a largura de um componente.

Layout_height – permite definir a altura de um componente.

Hint – permite definir um texto de entrada para alguns componentes que some quando clicamos dentro do mesmo.

Text – permite definir um texto padrão de entrada que não apaga quando clicamos dentro do componente.

11.3. Exercício

Neste exercício, você vai criar um aplicativo simples, bem parecido com o exercício da última apostila, porém agora utilizando o recurso Chain para organizar os componentes da interface. Além disso, vamos adicionar alguns detalhes extras para deixar o exercício mais completo e desafiador. Vamos lá!

Passo a Passo:

1. Criar o Projeto

- Abra o Android Studio e crie um novo projeto.

- Escolha uma Empty Activity e dê um nome ao projeto, como "ExercicioChains".

2. Configurar o Layout

- No arquivo activity_main.xml, altere o layout padrão para ConstraintLayout, se necessário.

3. Adicionar Componentes

Insira pelo menos 4 componentes na tela, como:

- TextView para exibir um título.

- EditText para entrada de texto.

- Button para envio.

- ImageView para adicionar uma imagem.

4. Criar as Chains

Agrupe os componentes em uma Chain:

- Crie uma Chain horizontal para organizar os botões.

- Crie uma Chain vertical para alinhar o título, o campo de texto e a imagem.

5. Configurar os Tipos de Chains

Experimente configurar diferentes tipos de Chains:

- Spread: para distribuir os elementos igualmente.

- Spread Inside: para distribuir igualmente, mas com os elementos das extremidades fixos.

- Packed: para agrupar os elementos no centro.

6. Ajustar Espaçamentos e Margens

- Defina espaçamentos e margens personalizadas para os elementos na Chain, garantindo que o layout fique organizado e esteticamente agradável.

7. Adicionar Restrições para Outros Elementos

- Inclua Guidelines para ajustar o alinhamento de alguns elementos fora das Chains, como a imagem ou o botão de envio.

8. Testar e Refatorar

- Execute o aplicativo no emulador ou dispositivo real e verifique como os componentes se comportam na tela.

- Ajuste margens, espaçamentos ou propriedades da Chain, se necessário.

anotações

AutoCompleteTextView

O AutoCompleteTextView é uma subclasse de EditText. Ele mostra automaticamente uma lista de sugestões de conclusão enquanto o usuário está digitando.

MultiAutoCompleteTextView

O autocomplete preenche o texto enquanto o usuário digita e, se encontrar uma correspondência na lista, permite completar a palavra com um toque, usando o **AutoCompleteTextView**.

O segundo método permite ao usuário inserir itens separados por vírgulas, utilizando o **MultiAutoCompleteTextView** para ativar o preenchimento automático de cada item.

Button	Permite ativar uma ação quando pressionado.
Toggle Button	Exibe estados marcados / desmarcados (ou ligado / desligado) usando um indicador luminoso.
CheckBox	O CheckBox é um botão especial com marcação e texto, que pode estar marcado ou desmarcado, permitindo selecionar vários itens simultaneamente.
RadioButton	O RadioButton tem dois estados e, com o RadioGroup, permite selecionar apenas uma alternativa de várias opções agrupadas.

12.1. Ocultação de senha

Agora, vamos rever um tutorial básico do conteúdo que vimos durante a aula, contendo mais explicação e também como realizar a construção da função etapa por etapa.

1. Declaração das Variáveis

Primeiro, são criadas referências para os componentes da interface:

EditText Senha: Representa o campo de texto onde o usuário digita a senha.

CheckBox verSenha: Representa a caixa de seleção (CheckBox) que permite alternar entre exibir ou ocultar a senha.

Essas variáveis são vinculadas aos componentes do layout por meio do método findViewById().

2. Detecção da Alteração no CheckBox

O método setOnCheckedChangeListener() é usado para monitorar as mudanças de estado do CheckBox:

- O evento é disparado toda vez que o CheckBox é marcado ou desmarcado.
- Dentro desse método, o parâmetro isChecked indica se o CheckBox está marcado **(true)** ou desmarcado **(false)**.

3. Lógica para Mostrar ou Ocultar a Senha

Dentro do listener, é implementada a lógica para alternar entre os modos de exibição do texto no campo de senha:

Se o CheckBox estiver marcado (isChecked == true):

O tipo de entrada do EditText é definido como "TYPE_TEXT_VARIATION_VISIBLE_PASSWORD" o que exibe a senha digitada de forma legível.

```java
Senha.setInputType(InputType.
TYPE_TEXT_VARIATION_VISIBLE_PASSWORD);
```

Se o CheckBox estiver desmarcado (isChecked == false):

O tipo de entrada é definido como uma combinação de TYPE_CLASS_TEXT e TYPE_TEXT_VARIATION_PASSWORD, que oculta o conteúdo da senha.

java

```
Senha.setInputType(InputType.TYPE_CLASS_TEXT
| InputType.TYPE_TEXT_VARIATION_PASSWORD);
```

4. Ajuste da Posição do Cursor

Após alterar o tipo de entrada do campo, o cursor é reposicionado no final do texto para manter a experiência de usuário fluida. Isso é feito com o método:

java

```
Senha.setSelection(Senha.getText().length())
;
```

12.2. Exercício

Você foi contratado por um e-commerce semelhante à Amazon para desenvolver a tela de cadastro de novos usuários. Essa tela será usada para que futuros clientes possam criar suas contas de forma simples e prática. Seu desafio é criar essa tela de formulário utilizando o Android Studio. *__Use sua imaginação__*

Requisitos Básicos do Formulário

O formulário deve permitir que o usuário insira informações básicas, como:

- Nome completo, E-mail, Senha, Telefone e Endereço

O layout precisa ser organizado, claro e funcional, priorizando a experiência do usuário.

Funcionalidades Específicas a Serem Implementadas

*Visibilidade da Senha:

O campo de senha deve permitir que o usuário visualize ou oculte o que está digitando, de acordo com sua preferência. Adicione uma funcionalidade interativa que habilite essa escolha.

*Preferências de Ofertas Exclusivas:

Inclua uma seção para que o usuário escolha se deseja ou não receber ofertas exclusivas da loja. Essa funcionalidade deve ser configurada de forma interativa.

Dicas para a Criação do Formulário

- Lembre-se de utilizar componentes apropriados para cada tipo de informação (por exemplo, EditText para texto, CheckBox para escolhas, e RadioButton para opções exclusivas).

- Considere utilizar grupos de componentes para organizar melhor as informações.

- Planeje a funcionalidade do botão "Cadastrar", que deve validar a opção de preferência de notificação de ofertas e exibir uma mensagem em um TextView na segunda tela.

Pontos Importantes

- Adapte o visual do formulário, ajustando cores, fontes e tamanhos para torná-lo mais profissional.

- Teste o comportamento dos componentes interativos (como o campo de senha e as opções de ofertas) para garantir que funcionem corretamente.

anotações

O ListView é um componente visual utilizado para o Android com a finalidade de armazenar uma ampla quantidade de informações e com o poderio de atualizar, receber e enviar eventos dinamicamente.

Neste artigo, o ambiente de desenvolvimento a ser utilizado é o Android Studio que, além de possuir uma estrutura poderosa dividida em layouts de tela e de códigos, é muito leve e de fácil manipulação do usuário. O objetivo deste artigo é criar um aplicativo através de uma lista, contendo menus e eventos em cada um, configurar permissões para internet e chamadas telefônicas, como também desenvolver uma tela “Sobre” para a prática na navegação entre telas.

O Android Studio possui diversos modelos de telas e dispositivos para o seu próprio emulador integrado, dando a possibilidade do desenvolvedor testar e visualizar como sua aplicação funciona nos mais diferentes dispositivos. Também é necessário ter o JDK (Java SE Development Kit) com a versão 21 ou superior instalado na máquina.

Layouts para projetos

Assim como as Views, os ViewGroups também podem ser customizados alterando cor, tamanho, posicionamento interno e espaços entre si.

Cada Android Layout tem suas particularidades e funcionam de uma forma diferente, e é isso que vamos ver nos próximos tópicos.

LinearLayout: Horizontal e Vertical

Legal, então agora nós sabemos sobre como os ViewGroups podem conter várias Views. Cada ViewGroup Pai tem regras específicas sobre como ele irá posicionar as Views Filhas dentro dele.

Agora vamos ver como o LinearLayout funciona. Ele pode posicionar as Views em uma única coluna vertical ou horizontal.

Para definirmos qual o tipo de orientação que queremos posicionar as nossas Views dentro do LinearLayout, nós usamos o atributo orientation. Esse atributo aceita dois valores, vertical e horizontal.

Veja um exemplo de um LinearLayout posicionando as Views de forma vertical.

```
<LinearLayout
    android:layout_width="409dp"
    android:layout_height="729dp"
    android:orientation="vertical"
    tools:layout_editor_absoluteX="1dp"
    tools:layout_editor_absoluteY="1dp"
    tools:ignore="MissingConstraints">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:text="Linear" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="22sp"
        android:text="Layout" />
</LinearLayout>
```

Repare como utilizamos o atributo android:orientation passando o valor vertical para posicionar as Views uma embaixo da outra.

Outro atributo importante do LinearLayout é o android:layout_weight, ele é responsavél por definir o peso que cada View tem referente à distribuição dentro do LinearLayout.

Precisamos utilizar o atributo android:layout_weight para distribuir igualmente as Views em toda a tela e assumir mais do espaço disponível.

A utilização desse atributo depende de como você está utilizando o LinearLayout. Para um layout vertical, precisamos definir a altura igual a zero e um peso para cada uma das Views.

Caso esteja usando um layout horizontal, precisamos definir a largura igual a zero e um peso para cada uma das Views.

13.1. Exercício

No decorrer dos exercícios das próximas apostilas, você acompanhará a criação do projeto final que estamos vendo em aula.

1. Criar um novo projeto:
Abra o Android Studio e inicie um novo projeto. Escolha o tipo de layout "Linear" com orientação "Vertical". Salve o projeto com um nome adequado ("ProjetoCadastro").

2. Adicionar os componentes de entrada de dados:
Arraste e solte os seguintes componentes na tela, um abaixo do outro:

Number: para o código do cliente (defina o ID como "edtCodigo").

Plain Text: para o nome do cliente (ID: "edtNome").

Plain Text: para a cidade do cliente (ID: "edtCidade").

Phone: para o telefone do cliente (ID: "edtFone").

E-mail: para o e-mail do cliente (ID: "edtEmail").

3. Configurar os componentes de entrada:

Para cada componente, defina o atributo "hint" com o texto que você deseja que apareça dentro do campo (ex: "Nome", "Cidade", "Telefone", "E-mail").

Apague o texto do atributo "Text" para que o "hint" seja exibido.

Desabilite a edição do campo "edtCodigo", pois ele será preenchido automaticamente.

4. Adicionar os botões de controle:

Arraste e solte um componente LinearLayout (Horizontal) abaixo dos campos de entrada.

Dentro do LinearLayout, arraste e solte três componentes Button.

Defina os IDs dos botões como "btLimpar", "btSalvar" e "btExcluir".

Defina o texto de cada botão como "Limpar", "Salvar" e "Excluir", respectivamente.

5. Adicionar a lista de clientes:

Arraste e solte um componente ListView abaixo dos botões.

Defina o ID do ListView como "lvClientes".

6. Ajustar o layout:

Clique no componente LinearLayout (Horizontal) e defina o atributo "layout_height" como "wrap_content" para que ele se ajuste ao tamanho dos botões.

7. Executar o projeto:

Clique no botão "Run" para visualizar o layout no emulador ou dispositivo Android.

anotações

Existem vários tipos de banco de dados e eles estão presentes na nossa vida há muito tempo. A lista telefônica pode ser considerada um bom exemplo.

Antigamente, as empresas armazenavam informações em arquivos físicos. Com a evolução dos computadores, o armazenamento digital tornou os bancos de dados o núcleo dos sistemas de informação. A definição de Banco de dados encontrada na internet é essa:

"Banco de dados são coleções de informações que se relacionam de forma que crie um sentido. São de vital importância para empresas e há duas décadas se tornaram a principal peça dos sistemas de informação."

O SQLite é o banco de dados interno e oficial da plataforma Android. Com ele, é possível modelar uma estrutura de tabelas relacionadas entre si para representar os dados do mundo real.

Se fizermos uma comparação, o SQLite é muito parecido com o MySQL, porém com algumas limitações por ser um banco de dados muito mais leve e simples.

O SQLite é um banco de dados open source com recursos relacionais, que usa SQL e transações. Devido à sua baixa exigência de memória (cerca de 250 KByte), ele funciona perfeitamente no Android.

Por ser simples, o SQLite suporta apenas TEXT, INTEGER e REAL, exigindo a conversão de outros tipos antes do armazenamento. Além disso, ele não valida se os valores correspondem ao tipo definido, permitindo inserções incompatíveis.

Ao contrário de outros bancos de dados que ficam em servidores, como Oracle e Microsoft SQL Server, cada banco de dados SQLite é armazenado em um único arquivo no disco.

O SQLite está incorporado em todos os dispositivos Android, não exigindo configurações adicionais. Basta definir as estruturas das tabelas e as operações necessárias, pois o banco de dados é gerenciado automaticamente pelo sistema. Como o acesso ao SQLite pode ser lento por envolver o sistema de arquivos, é recomendável usar AsyncTask para operações assíncronas.

Criar o Banco de Dados

Uma vez definida a estrutura do banco de dados, vamos implementar os métodos para criar e alterar as tabelas.

Utilizamos a classe SQLiteOpenHelper para criar o banco de dados, permitindo que o sistema realize automaticamente as operações de criação e atualização conforme necessário, sem depender da inicialização do aplicativo.

Manipulando os Dados

Agora vamos ver como manipular os dados em nosso SQLite utilizando os métodos do SQLiteOpenHelper.

Inserir

Para inserir dados no banco de dados, vamos utilizar um objeto ContentValues, passando para o método insert():

Ler

Para ler as informações de um banco de dados, use o método query(), passando os critérios de seleção e as colunas desejadas. Os resultados da consulta são retornados em um objeto Cursor.

Atualizar

Quando precisar modificar os valores dos dados do seu banco de dados, use o método update().

Excluir

Para excluir linhas de uma tabela, utilize os filtros para selecionar as linhas que devem ser excluídas.

A API do banco de dados oferece um mecanismo para criar critérios e filtros de seleção.

Quando criamos um banco de dados, criamos também tabelas, que servem para armazenar os dados por categoria, exemplo:

O banco de dados de uma loja vai possuir o cadastro dos funcionários, clientes, produtos, fornecedores, entre outras opções.

Para organizar essas categorias, fazemos o uso de tabelas.

Entendendo os atributos de uma tabela. Quando criamos uma tabela, precisamos definir os campos em cada coluna e com eles o tipo de dados que vai armazenar.

Dados do tipo String:

Char – tamanho fixo, completado com espaço em branco.

Varchar – tamanho variável.

Text – tamanho variável, até 2GB de dados.

O varchar é usado para armazenar até 255 caracteres, exemplo: Nome, endereço, cidade.

Dados do tipo numérico.

Int – permite números inteiros, como código, quantidade e idade.

Decimal – é ideal para cálculo de valores.

Double – é mais adequado para cálculos científicos gerais.

Float – é o double com menos bytes para representação.

Dados do tipo data.

Date – armazena apenas uma data. De 1 de janeiro de 0001 a 31 de dezembro de 9999.

Time – armazena um tempo apenas para uma precisão de 100 nanosegundos.

Depois das informações que uma tabela precisa conter, vamos entender como criar uma:

Comando para criar uma tabela.

java

```
CREATE TABLE nome_da_tabela (campos e tipos
de dados).

Exemplo com o cadastro de clientes.

CREATE TABLE cadClientes (
codigo INTEGER PRIMARY KEY,
nome TEXT,
cidade TEXT,
idade INTEGER)
```

14.1. Exercício

No exercício, você deverá aplicar ao projeto anterior os conceitos abordados em aula. Se encontrar dificuldades, consulte o passo a passo visto durante a aula para facilitar a compreensão.

14.1.1. Passo 1: Criação da Classe “Clientes”

1. Nova Classe: Clique com o botão direito do mouse no pacote do seu projeto e selecione "New" > "Java Class".

2. Nome da Classe: Dê o nome "Clientes" para a classe e pressione Enter.

3. Campos e Tipos de Dados: Dentro da classe, declare os seguintes campos com seus respectivos tipos de dados:

- `codigo` (tipo int)
- `nome` (tipo String)
- `cidade` (tipo String)
- `telefone` (tipo String)
- `email` (tipo String)

4. Construtores:

Construtor Vazio: Crie um construtor vazio (sem parâmetros).

Construtor para Atualização: Crie um construtor que receba todos os campos como parâmetros (incluindo `codigo`) e atribua esses valores aos campos da classe.

Construtor para Inserção: Crie um construtor que receba todos os campos como parâmetros, exceto o `codigo`.

5. Getters e Setters: Use a ferramenta de geração automática da IDE (clique com o botão direito, "Generate", "Getter and Setter") para criar os métodos getters e setters para todos os campos da classe.

14.1.2. Passo 2: Criação da Classe "Banco01"

1. Nova Classe: Crie uma nova classe Java chamada "Banco01".

2. Extensão da Classe SQLiteOpenHelper: Faça a classe "Banco01" estender a classe `SQLiteOpenHelper`.

3. Importação da Biblioteca: Importe a biblioteca `SQLiteOpenHelper`.

4. Constantes: Defina as seguintes constantes dentro da classe:

- `VERSAO_BANCO` (tipo int, com o valor 1)
- `NOME_BANCO` (tipo String, com o nome do seu banco de dados, por exemplo, "db_clientes")
- `TABELA_CLIENTE` (tipo String, com o nome da tabela, por exemplo, "tb_clientes")
- Constantes para os nomes das colunas (ex: `COLUNA_CODIGO`, `COLUNA_NOME`, etc.)

5. Método onCreate:

Criação da Tabela: Dentro do método `onCreate`, escreva o comando SQL para criar a tabela, especificando os nomes das colunas e seus tipos de dados.

Execução do Comando: Use `db.execSQL()` para executar o comando SQL de criação da tabela.

14.1.3. Passo 3: Método para Adicionar Cliente

1. Novo Método: Na classe "Banco01", crie um método chamado `addCliente` que recebe um objeto da classe `Clientes` como parâmetro.

2. Permissão de Escrita: Obtenha permissão de escrita no banco de dados usando `getWritableDatabase()`.

3. ContentValues: Crie um objeto `ContentValues` para armazenar os dados do cliente.

4. Inserção de Valores: Use `values.put()` para adicionar os valores de cada campo do cliente (nome, cidade, telefone, email) ao objeto `ContentValues`.

5. Inserção no Banco: Use `db.insert()` para inserir os dados na tabela, passando o nome da tabela, `null` e o objeto `ContentValues`.

6. Fechamento do Banco: Feche o banco de dados usando `db.close()`.

anotações

Os bancos de dados estão presentes em nossa vida há muito tempo e se manifestam de diversas formas. Um exemplo clássico é a lista telefônica, que organiza informações como nomes, endereços e números de telefone de maneira a facilitar a consulta e o contato. No passado, as empresas costumavam armazenar seus dados em arquivos físicos, como papéis e pastas, o que tornava o gerenciamento das informações trabalhoso e sujeito a erros. Com o surgimento e a evolução dos computadores, tornou-se possível armazenar esses dados de forma digital, o que revolucionou a forma como as informações são organizadas, acessadas e manipuladas.

Atualmente, os bancos de dados são o coração dos sistemas de informação. Eles permitem a organização de grandes volumes de dados de forma estruturada e relacionável, o que é essencial para que empresas e instituições possam tomar decisões baseadas em informações precisas e atualizadas. Uma definição comum encontrada na internet é a seguinte:

"Bancos de dados são coleções de informações que se relacionam de forma que criem um sentido. São de vital importância para empresas, e há duas décadas se tornaram a principal peça dos sistemas de informação."

Esse conceito reflete a importância dos bancos de dados como ferramentas fundamentais para o armazenamento e a manipulação de informações, servindo tanto para operações simples do dia a dia quanto para análises complexas que suportam estratégias de negócios.

15.1. O Papel do SQLite no Android

Na plataforma Android, o SQLite é o banco de dados interno e oficial. Ele é integrado ao sistema e permite que os desenvolvedores modelem uma estrutura de tabelas relacionadas, representando dados do mundo real de maneira eficiente. Por exemplo, ao criar um aplicativo que gerencia contatos, você pode utilizar o SQLite para armazenar informações como nome, telefone e endereço em tabelas organizadas, facilitando tanto a inserção quanto a consulta desses dados.

Apesar de sua semelhança com outros sistemas de gerenciamento de banco de dados, como o MySQL, o SQLite é projetado para ser mais leve e simples. Essa característica o torna ideal para aplicações móveis, onde o armazenamento local e o desempenho eficiente são prioridades, embora ele possua algumas limitações em termos de escalabilidade e recursos avançados quando comparado a sistemas mais robustos.

15.2. Manipulação de Dados com SQL

A manipulação de dados em um banco de dados é realizada por meio de comandos SQL (Structured Query Language), que permitem inserir, recuperar, atualizar e excluir informações das tabelas. A seguir, alguns comandos básicos e exemplos que ilustram como essas operações são realizadas:

15.2.1. Inserindo Dados

Para adicionar novos registros a uma tabela, utiliza-se o comando **INSERT**. A sintaxe básica é:

```sql
INSERT INTO nome_tabela (lista_de_campos)
VALUES (lista_de_dados);
```

Exemplo:

```sql
INSERT INTO cadClientes (codigo, nome,
cidade, idade) VALUES (1, 'João', 'Triunfo',
22);
```

Nesse exemplo, um novo cliente é inserido na tabela cadClientes com os valores especificados para cada campo.

15.2.2. Recuperando Dados

Para consultar os dados armazenados em uma tabela, o comando **SELECT** é utilizado. Sua forma básica é:

```sql
SELECT lista_de_campos FROM nome_tabela;
```

Exemplo:

```sql
SELECT nome, idade FROM cadClientes;
```

Esse comando retorna os valores das colunas nome e idade de todos os registros da tabela cadClientes.

15.2.3. Filtrando Dados com WHERE

A cláusula **WHERE** permite refinar as consultas, especificando condições que os registros devem atender para serem retornados:

```sql
SELECT nome, cidade FROM cadClientes WHERE
cidade = 'Triunfo';
```

Aqui, apenas os clientes que moram na cidade de "Triunfo" serão selecionados.

15.2.4. Removendo Dados

Para excluir registros de uma tabela, utiliza-se o comando **DELETE** com uma condição para determinar quais registros serão removidos:

```sql
DELETE FROM nome_tabela WHERE condição;
```

Essa instrução é essencial para manter a integridade dos dados, permitindo a remoção de informações desatualizadas ou incorretas.

15.3. O Conceito de Query

O termo query refere-se a qualquer comando SQL que seja executado no banco de dados, seja para consulta (**SELECT**), inserção (**INSERT**), atualização (**UPDATE**) ou exclusão (**DELETE**). As queries são a forma de interagir com o banco de dados, permitindo que os dados sejam manipulados de acordo com as necessidades do aplicativo ou sistema.

15.4. Exercício

Vamos dar sequência ao projeto criando funções que permitirão manipular os dados dos clientes. Cada etapa indica as instruções que você deverá digitar e, em seguida, pressionar Enter (Na maioria das instruções).

15.4.1. 1. Função para Excluir Dados

Objetivo: Criar uma função que exclua um cliente do banco de dados com base no seu

código. (deverá ser construído no arquivo Banco01)

15.4.1.1. 1. Criação da Função delCliente:

- Digite a seguinte assinatura e pressione Enter:

```java
void delCliente(Clientes cliente) {
```

15.4.1.2. 2. Obter o Banco de Dados no Modo de Escrita:

- Dentro da função, chame o banco de dados para escrita:

```java
SQLiteDatabase db =
this.getWritableDatabase();
```

15.4.1.3. 3. Definir o Comando de Exclusão:

- Digite o comando que utiliza a função delete, passando o nome da tabela, a condição baseada no código e o valor a ser excluído:

```java
db.delete(TABELA_CLIENTE, COLUNA_CODIGO +
"=?", new String[]
{String.valueOf(cliente.getCodigo())});
```

Explicação: O operador "=?" serve para receber o valor do código do cliente quando o usuário selecionar o registro a ser apagado.

15.4.1.4. 4. Fechar o Banco de Dados:

- Finalize a função fechando o banco:

```java
db.close();
```

15.4.2. 2. Função para Selecionar um Cliente

Objetivo: Criar uma função que, dado um código, retorne o objeto Clientes com os dados correspondentes (deverá ser feito no arquivo Banco01).

15.4.2.1. 1. Criação da Função selecionaCliente:

- Digite a assinatura e pressione Enter:

```java
Clientes selecionaCliente(int codigo) {
```

15.4.2.2. 2. Ativar o Banco no Modo de Leitura:

- Digite:

```java
SQLiteDatabase db =
this.getReadableDatabase();
```

15.4.2.3. 3. Utilizar o Comando Cursor para Seleção dos Campos:

- Inicie a criação do comando query para recuperar os dados:

```java
Cursor cursor = db.query(TABELA_CLIENTE, new
String[]{
```

Quando solicitado, importe o componente Cursor (Alt + Enter se necessário).

15.4.2.4. 4. Definir os Campos a Serem Selecionados:

- Digite os nomes dos campos (em um array) e pressione Enter:

```java
COLUNA_CODIGO, COLUNA_NOME, COLUNA_CIDADE,
COLUNA_TELEFONE, COLUNA_EMAIL},
```

15.4.2.5. 5. Definir a Condição de Seleção:

- Utilize o código como referência para buscar o cliente:

```java
COLUNA_CODIGO + "=?", new String[]
{String.valueOf(codigo)},
```

15.4.2.6. 6. Os próximos parâmetros podem ser deixados como null:

```java
null, null, null, null);
```

15.4.2.7. 7. Verificar se Há Registros e Posicionar o Cursor:

- Digite:

```java
if(cursor != null) {
    cursor.moveToFirst();
}
```

15.4.2.8. 8. Definir e Retornar o Objeto Clientes:

- Crie um objeto Clientes preenchendo os campos com os valores recuperados (lembre-se de converter o índice 0 para inteiro):

```java
Clientes grade = new
Clientes(Integer.parseInt(cursor.getString(0
)),
cursor.getString(1), cursor.getString(2),
cursor.getString(3), cursor.getString(4));
return grade;
}
```

15.4.3. 3. Função para Atualizar Registros

Objetivo: Criar uma função que atualize os dados de um cliente existente no banco de dados (deverá ser feito no arquivo Banco01).

15.4.3.1. 1. Criação da Função atualizaCliente:

- Digite a assinatura:

```java
void atualizaCliente(Clientes cliente) {
```

15.4.3.2. 2. Obter o Banco no Modo de Escrita:

- Digite:

```java
SQLiteDatabase db =
this.getWritableDatabase();
```

15.4.3.3. 3. Criar um Objeto ContentValues para os Novos Valores:

- Digite:

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

15.4.3.4. 4. Inserir os Novos Valores para Cada Campo:

- Para o campo nome:

```java
values.put(COLUNA_NOME, cliente.getNome());
```

- Para o campo cidade:

```java
values.put(COLUNA_CIDADE,
cliente.getCidade());
```

- Para o campo telefone:

```java
values.put(COLUNA_TELEFONE,
cliente.getTelefone());
```

- Para o campo e-mail:

```java
values.put(COLUNA_EMAIL,
cliente.getEmail());
```

15.4.3.5. 5. Atualizar o Registro Usando o Código como Referência:

- Digite:

```java
db.update(TABELA_CLIENTE, values,
COLUNA_CODIGO + "=?",
new String[]
{String.valueOf(cliente.getCodigo())});
}
```

15.4.4. 4. Função para Listar Todos os Clientes

Objetivo: Criar uma função que retorne uma lista com todos os clientes cadastrados no banco de dados.

15.4.4.1. 1. Criação da Função listaTodosClientes:

- Digite a assinatura:

```java
public List listaTodosClientes() {
```

- Caso solicitado, pressione Alt+Enter para importar a biblioteca **java.util.List** e **java.util.ArrayList**.

15.4.4.2. 2. Declarar a Variável para Armazenar a Lista:

- Digite:

```java
List listaCli = new ArrayList();
```

15.4.4.3. 3. Definir a Query para Selecionar os Registros:

- Digite:

```java
String query = "SELECT * FROM " +
TABELA_CLIENTE;
```

15.4.4.4. 4. Ativar o Banco no Modo de Escrita (ou Leitura):

- Digite:

```java
SQLiteDatabase db =
this.getWritableDatabase();
```

15.4.4.5. 5. Criar o Objeto Cursor para Percorrer os Registros:

- Digite:

```java
Cursor c = db.rawQuery(query, null);
```

15.4.4.6. 6. Verificar se Existem Registros e Iterar Sobre Eles:

- Se o cursor tiver registros, posicione-o no primeiro:

```java
if(c.moveToFirst()) {
do {
Clientes cliente = new Clientes();
cliente.setCodigo(Integer.parseInt(c.getStri
ng(0)));
cliente.setNome(c.getString(1));
cliente.setCidade(c.getString(2));
cliente.setTelefone(c.getString(3));
cliente.setEmail(c.getString(4));
listaCli.add(cliente);
} while(c.moveToNext());
}
```

15.4.4.7. 7. Retornar a Lista de Clientes:

- Digite:

```java
return listaCli;
}
```

15.4.5. 5. Integração no MainActivity

Objetivo: Implementar funções na MainActivity para listar clientes na interface, selecionar um item para edição e limpar os campos.

15.4.5.1. 5.1. Função para Listar Clientes

1. Criação da Função listarCli:

- No MainActivity, digite:

```java
public void listarCli() {
```

2. Obter a Lista de Clientes do Banco:

- Digite:

```java
List<Clientes> clientes =
db.listaTodosClientes();
```

3. Criar e Configurar o ArrayList e o Adapter:

- Declare o ArrayList:

java

```
ArrayList<String> arrayList = new
ArrayList<String>();
```

4. Configure o adapter:

java

```
ArrayAdapter<String> adapter = new
ArrayAdapter<String>(
MainActivity.this,
android.R.layout.simple_list_item_1,
arrayList);
lvClientes.setAdapter(adapter);
```

5. Percorrer os Clientes e Adicionar ao ArrayList:

- Utilize um loop:

java

```
for(Clientes c : clientes) {
arrayList.add(c.getCodigo() + "-" +
c.getNome());
}
adapter.notifyDataSetChanged();
}
```

15.4.5.2. 5.2. Função para Selecionar um Cliente da Lista

1. Configurar o OnItemClickListener na Lista:

- Digite:

java

```
lvClientes.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView parent,
View view, int i, long id) {
String conteudo = (String)
lvClientes.getItemAtPosition(i);
String codigo = conteudo.substring(0,
conteudo.indexOf("-"));
Clientes cliente =
db.selecionaCliente(Integer.parseInt(codigo)
);
edtCodigo.setText(String.valueOf(cliente.get
Codigo()));
edtNome.setText(cliente.getNome());
edtCidade.setText(cliente.getCidade());
edtFone.setText(cliente.getTelefone());
edtEmail.setText(cliente.getEmail());
}
});
```

15.4.5.3. 5.3. Função para Limpar os Campos

1. Criar a Função limparCampos:

- Digite:

java

```
void limparCampos() {
edtCodigo.setText("");
edtNome.setText("");
edtCidade.setText("");
edtFone.setText("");
edtEmail.setText("");
edtNome.requestFocus();
}
```

2. Vincular a Função ao Botão Limpar:

- No MainActivity, configure o OnClickListener:

java

```
btLimpar.setOnClickListener(new
View.OnClickListener() {
@Override
public void onClick(View v) {
limparCampos();
}
});
```

16.1. Processo para publicar um projeto

16.1.1. 1. Criar ou Acessar a Conta de Desenvolvedor

Primeiro, é necessário ter sua conta de desenvolvedor no Google Play Developer Console.

Importante: A taxa de abertura da conta tem o custo de $ 25 (**dólares**), valor cobrado pela própria Google. Esse valor só precisa ser pago para abrir a conta, é uma taxa única, **não tem renovação**.

Preencha as informações solicitadas (nome do desenvolvedor, método de pagamento, dados de contato etc.).

Dica

Mantenha um e-mail profissional ou dedicado ao desenvolvimento para facilitar o gerenciamento de comunicações futuras.

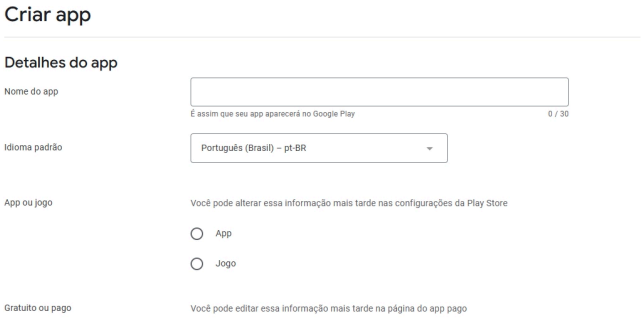
16.1.2. 2. Iniciar um Novo Projeto (App)

- No painel inicial do Google Play Console, clique em “Criar app” ou “Adicionar novo app”.

- Defina o nome do aplicativo (como será exibido na Play Store).

- Escolha o idioma padrão e selecione se o app será gratuito ou pago.

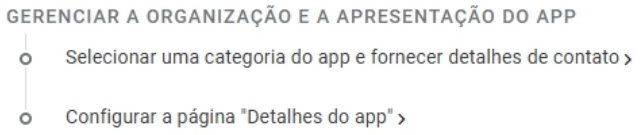
- Confirme as informações e avance para as próximas etapas.



Observação

Se o seu app for pago, será necessário configurar mais detalhes de pagamento e impostos na conta de desenvolvedor.

16.1.3. 3. Preencher as Informações Básicas do App



Na seção "**Detalhes do app**", na aba "**Configurar o app**", você vai encontrar e deverá preencher as seguintes informações:

Título e Descrição: Adicione uma descrição curta e outra detalhada para o app.

Ícone e Gráficos: Faça upload do ícone do app, capturas de tela e, se quiser, vídeos promocionais.

16.1.4. 4. Configurações de Conteúdo e Classificação

Políticas de Conteúdo: Em "Painel" (ou "Configuração do app"), há tópicos como "Política de Privacidade", "Acesso ao app", "Anúncios", "Conteúdo gerado por usuários", "Apps direcionados a crianças" etc.

Marque e preencha cada informação pedida segundo as características do seu app.

Classificação do Conteúdo: O Google pedirá que você responda a um questionário para obter a classificação indicativa (ex.: Livre, 10+, 12+, 18+).

Segurança de Dados (Data Safety): Dependendo da versão do console, você também precisará preencher o formulário de segurança de dados, indicando como seu app coleta, armazena e compartilha informações dos usuários.

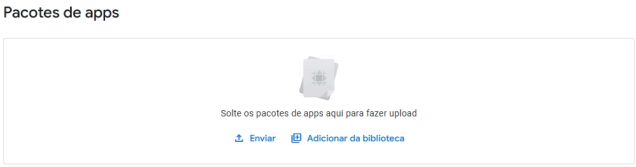
16.1.5. 5. Definir as Configurações de Lançamento

Criar Versão: Em "Criar e publicar uma versão", selecione se deseja lançar na faixa de produção, alfa ou beta.

Faixa de Teste: Se escolher alfa ou beta, você poderá testar o app com um grupo limitado de usuários antes de lançar publicamente.

Países e Regiões: Selecione os países onde deseja distribuir o aplicativo. Você pode adicionar ou remover regiões a qualquer momento.

16.1.6. 6. Fazer o Upload do AAB



Upload: Clique em "Enviar" ou "Upload" e selecione o arquivo gerado pelo Android Studio.

Verificar Erros: O Play Console analisará o pacote para verificar se há problemas de compatibilidade ou segurança.

16.1.7. 7. Revisar e Concluir o Lançamento

Revisão das Configurações: Confira todas as seções (ícone, descrição, políticas, classificação, países de distribuição).

Salvar e Enviar para Análise: Ao finalizar, clique em "Revisar Resumo" ou "Enviar para revisão".

Tempo de Aprovação: A Google analisará o app e poderá aprová-lo ou solicitar ajustes (esse processo costuma levar de algumas horas a alguns dias).

16.1.8. 8. Publicar e Acompanhar Desempenho

Aprovação: Se o app for aprovado, ele ficará disponível na Play Store conforme as configurações de distribuição definidas.

Acompanhamento: No painel do Play Console, você pode monitorar estatísticas de downloads, avaliações, receita (se for pago) e feedback dos usuários.

Atualizações: Sempre que fizer alterações importantes no app, gere uma nova versão (AAB) e repita o processo de upload, mantendo a versão atualizada na loja.

16.1.8.1. Dicas Finais

- Mantenha o app em conformidade com as Políticas do Google Play.
- Verifique os requisitos para apps específicos (por exemplo, direcionados a crianças ou que utilizem dados sensíveis).
- Use faixas de teste (alfa ou beta) para obter feedback antes de lançar ao público geral.
- Monitore avaliações e comentários para corrigir bugs e melhorar a experiência do usuário.

16.2. Exercício

O nosso último exercício tem como objetivo concluir o desenvolvimento do aplicativo final do nosso curso, confira o passo a passo a seguir.

16.2.1. Passo a Passo para Configurar as Funcionalidades de Cadastro, Atualização e Exclusão

16.2.1.1. 1. Configurar o Evento do Botão Salvar:

- Com o projeto aberto, encontre um local para criar a função do botão salvar (btSalvar).

- Configure o evento de clique:

java

```
btSalvar.setOnClickListener(new
View.OnClickListener(){
    public void onClick(View v){

    }
});
```

16.2.1.2. 2. Capturar os Dados dos Componentes:

- Dentro do método onClick, crie variáveis do tipo String para cada campo:

java

```
String codigo =
edtCodigo.getText().toString();
String nome = edtNome.getText().toString();
String cidade =
edtCidade.getText().toString();
String telefone =
edtFone.getText().toString();
String email =
edtEmail.getText().toString();
```

16.2.1.3. 3. Validar a Entrada de Dados:

- Verifique se o campo “nome” está vazio. Se estiver, exiba uma mensagem de erro:

java

```
if(nome.isEmpty()){
    edtNome.setError("Campo Obrigatório");
}
```

16.2.1.4. 4. Processar Cadastro e Atualização:

- Para Cadastro: Se o campo código estiver vazio, significa que é um novo cliente. Então:

- Adicione o cliente:

java

```
else if(codigo.isEmpty()){
    db.addCliente(new Clientes(nome, cidade,
telefone, email));
```

- Exiba uma mensagem de sucesso:

java

```
Toast.makeText(MainActivity.this, "Cliente
Cadastrado", Toast.LENGTH_LONG).show();
```

- Limpe os campos e atualize a lista:

java

```
    limparCampos();
    listarCli();
```

```
}
```

- Para Atualização: Se o campo código não estiver vazio, atualize o registro existente:

java

```
else{
    db.atualizaCliente(new
Clientes(Integer.parseInt(codigo), nome,
cidade, telefone, email));
    Toast.makeText(MainActivity.this,
"Cliente Atualizado",
Toast.LENGTH_LONG).show();
    limparCampos();
    listarCli();
}
```

16.2.1.5. 5. Configurar o Evento do Botão Excluir:

- Configure o clique no botão excluir (btExcluir):

java

```
btExcluir.setOnClickListener(new
View.OnClickListener() {
    public void onClick(View v){

    }
});
```

- Dentro do evento, capture o código do cliente:

java

```
String codigo =
edtCodigo.getText().toString();
```

- Verifique se o campo código está vazio:

java

```
if(codigo.isEmpty()){
    Toast.makeText(MainActivity.this,
"Nenhum cliente selecionado",
Toast.LENGTH_LONG).show();
}
```

- Se um cliente foi selecionado, prossiga com a exclusão:

- Crie um objeto do tipo Clientes e atribua o código:

java

```
else{
    Clientes cliente = new Clientes();

cliente.setCodigo(Integer.parseInt(codigo));
```

- Chame a função de exclusão:

java

```
db.delCliente(cliente);
```

- Exiba uma mensagem confirmando a exclusão, limpe os campos e atualize a lista:

java

```
    Toast.makeText(MainActivity.this,
"Cliente excluido",
Toast.LENGTH_LONG).show();
    limparCampos();
    listarCli();
}
```

16.2.1.6. 6. Testar o Aplicativo:

Após implementar as rotinas, clique no botão Run e teste se o cadastro, atualização e exclusão estão funcionando conforme esperado.

Muito obrigado por nos acompanhar durante o curso. Esperamos que os conhecimentos adquiridos sejam valiosos para sua jornada. Até a próxima!