**Estudos Android**

**Infos iniciais:**

* Toast: esse componente apresenta uma mensagem e some
* O context representa o ponto que estamos no nosso app e a maioria dos componentes herdam dele. Ele também permite acessar recursos do Android.
* Evitar a prática de toda vez que quiser representar um valor numa tela, colocar no xml do layout. Não faça isso!
* **onCreate** serve para dar um ciclo de vida para nossa activity, ou seja, para criar um ciclo, mas ele só chama esse ciclo uma vez.
* **onResume** ele reexecuta o ciclo de vida da nossa activity.

**Sobre layouts:**

Para criar um pack para layout, vamos até pasta res e criamos o android resource directory e depois selecionamos layout.

Após isso, vamos no pack que criamos e vamos criar a Layout Resource File.

Esse arquivo segue algumas regras:

* Nome arquivo: prefixo activity\_nomedoarquivo

**Sobre Namespaces:**

**\*TODA GROUP VIEW QUE FOR A ROOT, ELA PRECISA ESTAR COM OS NAMESPACES DECLARADOS\***

Ou seja, se vc tem um group View (root) com os namespaces e outra group View virou a root, é só tirar os namespaces da antiga e colocar na nova.

Dentro da minha View Group constraintlayout, possuo algumas propriedades:

* Xmlns: namespaces (são apelidos para acessar algumas propriedades)

Android: são atributos que vem do sistema

App: faz modificações de uma propriedade do próprio componente

Tools: específico apenas para desenvolver e visualizar no preview (não dá para ver no app)

**Layouts\_width e height:**

* Match\_parent: a View vai crescer até o tamanho do pai, ou seja, conforme a tela (pegar altura quanto largura)
* Wrap\_content: vai crecer apenas o tamanho ao seu conteúdo

Se o eixo x (ou y) estiver preenchido tanto no início quanto no final, damos 0dp para os layouts (isso se tiver o startOf e o endOf).

**Texto:** inserir textos

**Hint:** placeholder

**textStyle:** Muda o estilo do texto para negrito, itálico ou sublinhado.

**maxLine:** Decide o número de linhas que será exibido de um texto

**ellipsize:** Decide onde colocar reticencias, ou seja, o limite de onde vai o meu texto

**inputType:** Seve para definir o tipo de preenchimento do campo, ou seja, o teclado. Ex: número, texto, decimal, password etc.

**View EditText:** Servem para preenchimento de campos

**Scroll View:** é uma group View que tem como função fazer outras views e groups ter a capacidade de rolar.

**Propriedades do constraint layout:**

* Layout\_constraintEnd\_toEndOf: regras de constraint do fim, vai ficar no fim de alguém.
* Layout\_constraintTop\_toTopOf: regras de constraint do topo, vai ficar no topo de alguém.
* Layout\_constraintBottom\_toBottomOf: ficar na parte de baixo

Sempre usamos esses constraint dentro de uma View que está em uma group View constraint.

**Sempre usar dp no lugar de px:**

Quando usamos px, determinamos valores fixos ao nosso componente e quando mudamos para um device diferente, corre o risco de ficar desproporcional.

Quando usamos dp, ele vai criar pixels virtuais e vai tentar se adaptar a tela diferente.

Casos que não utilizamos o dp: casos em aumentar a fonte (nesse caso usamos sp

Margin: utilizamos para definir uma margem.

* É bom evitar números ímpares
* **sp** dá acesso à funções de acessibilidade, que são recursos do próprio sistema do android (principalmente pessoas com problema de visão)

textSize: tamanho da fonte do texto

**Método padrão para buscar views:**

**Para representar valores sem usar o xml, usamos a técnica chamada binding.**

**Sobre Binding:**

Arquivos de layout são utilizados como uma referência para apresentar o conteúdo visual na tela. As informações que serão apresentadas no layout geralmente são produzidas pelo código fonte Kotlin/Java. Para que isso seja possível, fazemos o processo de binding de View que busca as Views do layout e disponibiliza no código fonte para a modificação dos dados.

findViewbyId<>() – ele busca uma View a partir do seu id

dentro do <> você coloca o tipo de View que você que buscar

cria uma const desse find e depois seta ele.

Exemplo:

val nome = findViewById<TextView>(R.id.*nome*)  
nome.*text* = "Lista de compras do Wagner"

**Recyclerview:**

Em situações que precisamos apresentar uma mesma View com informações diferentes, como é o caso de uma lista de produtos, podemos utilizar o RecyclerView. A partir dele temos a capacidade de criar Views dinamicamente a partir da nossa lógica de código-fonte sem a necessidade de modificar o layout cada vez que precisamos adicionar novas Views. O grande destaque do RecyclerView é a performance, tornando-o a recomendação para esse tipo de solução.

É um group View que serve basicamente para criar listas dinâmicas (mas existem outras soluções) Ele entrega uma performace independentemente da quantidade para apresentar na tela e usa uma técnica de reciclar essas views.

O recycler View vai vir com tamanho padrão, nesse caso apagamos eles e usamos as constraint layout se ele estiver dentro de uma.

* O listitem é um atributo que vc referência os ids da suas views e ele será representado dentro da lista.
* O componente que faz a lógica do Recyclerview é o adapter (sempre vamos usar ele) e colocar a referência básica que é a RecyclerView.Adapter(). Ao invés de usar essa referência, criamos uma classe e essa classe vai herdar desse recyclerview.adapter.

Texto

Descrição gerada automaticamenteTexto

Descrição gerada automaticamenteTexto

Descrição gerada automaticamente

**Builds Tools**

**O que é um processo de build:**

Quando estamos desenvolvendo, passamos por algumas fases no nosso projeto. Código fonte, uso de bibliotecas e frameworks, realização de testes, incompatibilidade entre IDEs e empacotamento e distribuição da aplicação.

Essas rotinas são trabalhosas e fazer de maneira manual não é muito lega. Para ajudar nesse trabalho, usaremos a Build Tools (ferramentas de construção).

**O que acontece quando utilizamos um build tool:**

Automatizar tarefas rotineiras:

* Crias source sets para desenvolvimento e testes;
* Adicionar e remover dependências;
* Executar testes e gerar relatórios;
* Empacotar e fazer deploy do projeto.

Manter a compatibilidade entre IDEs;

Executar rotinas utilitárias.

**Exemplos de build tools:**

* Apache ant
* Maven
* Gradle

**Sobre Gradle do projeto:**

No build.gradle temos configurações gerais do projeto: versão do kotlin, qual config do android em relação ao gradle, plugins etc.

Se eu quero ver algo especifico do projeto, vou no build.gradle(app). Sempre é bom eu verificar as dependências que eu uso.

Com ctrl + shift + a posso procurar Analyze dependencies e ela vai mostrar todas as bibliotecas e mostram as dependências.

**Float (pode ser imagens, ícones, botões etc.)**

Uma das funções dessa View é o **clickable**, que diz que ele é clicável, mas ela pede para adicionar uma outra que é a **focusable**, que indica que esse componente também é focável.

* srcCompat é a imagem que estou utilizando, o ícone.

**finish()** //esse método finaliza nossa activity e volta para a anterior

**Log.i**("FormularioProdutoAct", "onCreate: $produtoNovo")

É uma forma de adicionar logs para saber o que está acontecendo com seu programa.

//a primeira é uma chave que você escolhe, depois o valor que espera e a variavel

//lembrando que não pode ser mais que 23 caracteres

Assuntos pra estudar: Companion Object

**Adicionando imagens no app**

**ImageView:** é uma View de imagens (vc pode adicionar backgrounds ou avatares)

**Chains:** É uma técnica de adição de imagens. Além de permitir uma flexibilidade de alinhamento entre as Views, Chains possuem alguns comportamentos específicos que podem ser úteis na construção de layout, por exemplo, a distribuição proporcional entre as Views configuradas em Chain.

**ScaleType:** Serve para ajustar a escala de preenchimento de uma imagem, onde das várias opções

**srcCompat:** você pode simular uma cor

**Pasta Drawable:** Nessa pasta consigo adicionar novas imagens ao meu app

Para carregar imagens que vem da internet, nós usamos uma biblioteca pois o kotlin não possui tal função de forma nativa.

Alguns exemplos de bibliotecas são o Coil, Picasso e Bumbtech. No curso, recomendaram o coil pois além de ser mais prático, ele está desenvolvido em kotlin.

Mas quando inserir essas imagens, precisamos da permissão de internet, pois se não vai dar erro, então ativamos essa permissão no manifesto.

Obs:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente