

# **CURSOS TÉCNICOS**

## **DESENVOLVIMENTO DE APLICATIVOS I**

Eixo Informática para Internet

**SEMANA 4**

**SUMÁRIO**

1. USANDO ELEMENTOS COMPOSABLE (COMPOSIÇÃO) .....	3
1.1 Criando a estrutura do layout.....	4
1.2 Modifier .....	5
1.3 Pasta ‘res’ do Android/Kotlin .....	7
1.4 Trabalhando com elementos - Botão .....	10
1.5 Adicionar um elemento de composição Image.....	12
1.6 Inserindo a lógica no botão.....	12
1.7 Adicionando condições no App de jogar dados.....	14
1.8 State & mutableState().....	15
2. Acessando dados para ajuste e correção .....	17
3. Referências.....	18

## **UNIDADE 4**

### **1. USANDO ELEMENTOS COMPOSABLE (COMPOSIÇÃO)**

Na unidade anterior, estudamos como criar um layout base e, o que é mais importante, trabalhamos com o JetPack Compose, que na verdade é um framework que permite desenvolver Interfaces de Usuário (UI) sem a necessidade de manipulação do XML.

Também nos familiarizamos com as funções importantes onCreate e setContent, bem como as anotações das funções, os parâmetros passados para essas funções e suas tipificações.

Nesta unidade de estudo, iremos trabalhar com elementos de composição (Composable) sempre **utilizando o JetPack Compose** e dando foco nos desenvolvimentos de layouts, e componentes interativos por meio de um botão para um aplicativo mobile.

Vamos estudar, também, a importância das Activities em um projeto Kotlin Android Studio.

Então...bora focarmos em:

- ✚ O que é o **Modifier e quais as propriedades** que podemos declarar nos componentes.
- ✚ Declaração de **funções e lambdas**. O que são os lambdas?
- ✚ Mais manipulação do **JetPack Compose**.
- ✚ Trabalhando com a pasta '**res**', seus **recursos visuais** e seus valores por meio do uso do arquivo **strings.xml**.
- ✚ **Composable Button** e declaração da sua função.
- ✚ **States e função mutableStateOf()**.
- ✚ Elemento **Remember**.

### 1.1 Criando a estrutura do layout

A partir deste momento, crie um novo projeto Android com Kotlin, e vamos reestruturar o nosso código da seguinte forma:

- ✓ Acesse a classe **MainActivity**.
- ✓ Vamos remover a função padrão **Preview()**.
- ✓ Vamos criar uma função chamada **DadosComBtnImagem()** com a notação **@Composable**. Esta função irá representar os componentes da UI (User interface) e também irá conter a lógica do clique no botão e exibição de uma imagem que mais adiante iremos incluir na aplicação.
- ✓ Vamos agora, remover a função **Greeting(name: String)**.
- ✓ E vamos criar outra função **RolandoDadosApp()** com as notações **@Preview** e **@Composable**.

Nosso código da MainActivity estará da forma como a próxima imagem:

```

40  @Preview
41  @Composable
42  fun RolandoDadosApp() {
43
44  }
45
46  @Composable
47  fun DadosComBtnImagem() {
48
49  }

```

Fonte da imagem: autoria própria

Por enquanto, o código da MainActivity está desta forma:

```

1  package com.example.rolandodados
2
3  import ...
4
14
15  class MainActivity : AppCompatActivity() {
16      override fun onCreate(savedInstanceState: Bundle?) {
17          super.onCreate(savedInstanceState)
18          setContent {
19              RolandoDadosTheme {
20                  RolandoDadosApp()
21              }
22          }
23      }
24  }
25
26
27  @Preview
28  @Composable
29  fun RolandoDadosApp() {
30      DadosComBtnImagem()
31  }
32
33
34  @Composable
35  fun DadosComBtnImagem(modifier: Modifier = Modifier) {
36
37  }
38
  
```

Fonte da imagem: autoria própria

Agora estamos com o código estruturado e pronto para podermos trabalhar e estudar alguns tópicos importantes.

## 1.2 Modifier

O Compose usa um objeto Modifier, que é uma coleção de elementos usados para decorar ou modificar o comportamento dos elementos da IU do Compose.

Na unidade anterior já declaramos o Modifier, porém neste momento vamos entender mais este objeto, que é de extrema importância para o desenvolvimento de interfaces no Android.

O objeto Modifier é de fato um conceito fundamental. Ele é usado para aplicar modificações visuais ou comportamentais aos elementos de interface do usuário construídos com o Compose.

**O Modifier é uma parte essencial da sintaxe declarativa do Compose, permitindo que você especifique o estilo, o layout e o comportamento dos componentes de maneira concisa.**

O objeto Modifier contém várias funções que você pode encadear para aplicar várias modificações a um componente. Algumas das modificações que podemos aplicar incluem:

- ✓ Definir o tamanho do componente (**width, height**).
- ✓ Adicionar margens (**padding, margin**).
- ✓ Definir cores de fundo (**background**).
- ✓ Ajustar o espaçamento entre componentes (**spacer**).
- ✓ Controlar o comportamento de clique (**clickable, onClick**).
- ✓ Definir limites máximos ou mínimos (**fillMaxWidth, fillMaxHeight**).

Exemplo:

```

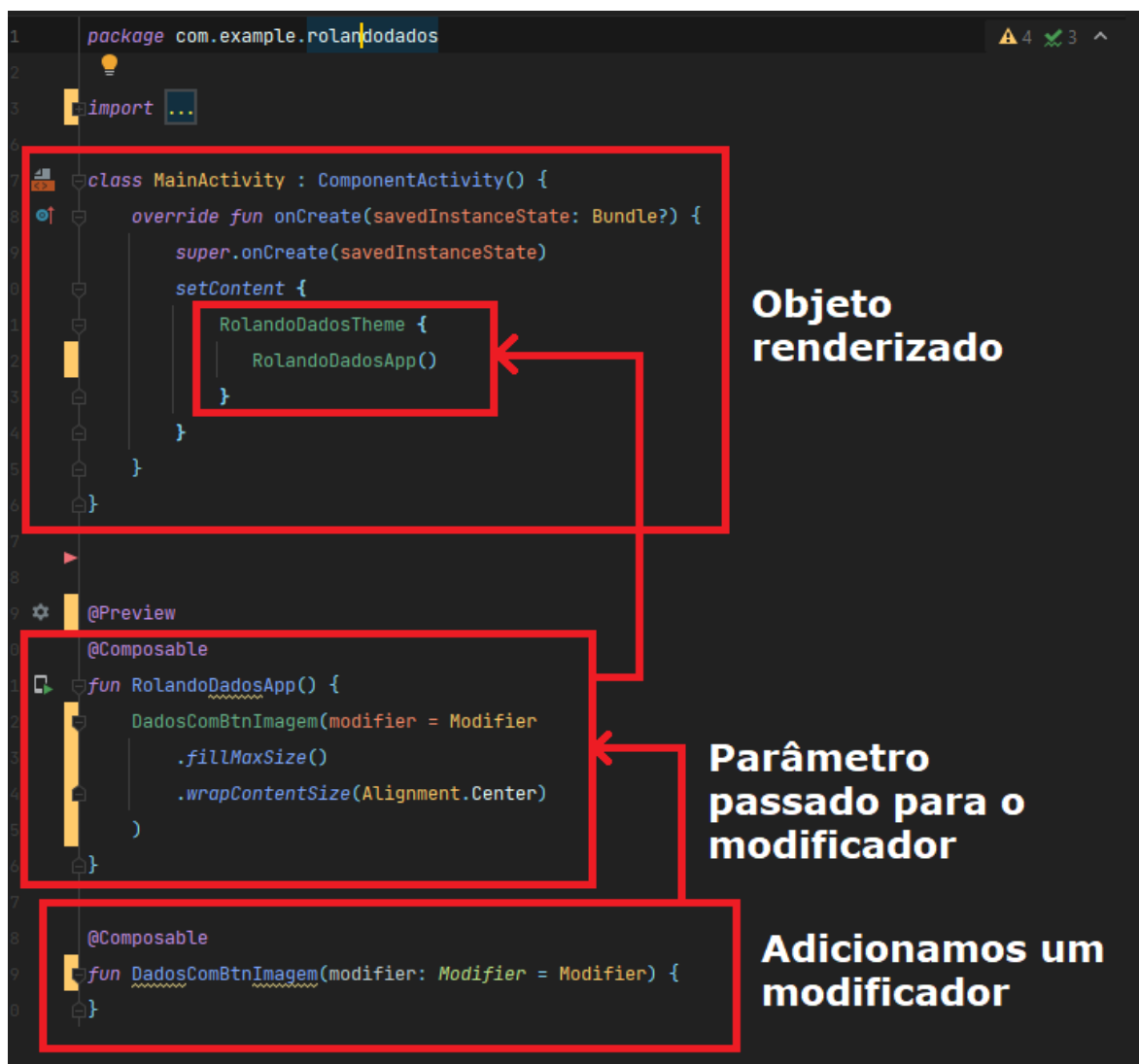
40  Box(
41      modifier = Modifier
42      .size(100.dp)
43      .background(Color.Blue)
44  ) {
45      Text( text: "Hello, Compose!", color = Color.White)
46  }
    
```

Fonte da imagem: autoria própria

Toda vez que usamos o Modifier, a instrução **import androidx.compose.ui.Modifier** importa o pacote **androidx.compose.ui.Modifier**, o que permite fazer referência ao objeto Modifier.

Vamos então fazer as declarações necessárias para deixar o nosso botão funcional e devidamente ajustado à interface que desejamos desenvolver. Lembrando que estas modificações vamos declarar na função com notação `@Preview`.

Observe a próxima imagem:



Fonte da imagem: autoria própria

### 1.3 Pasta 'res' do Android/Kotlin

A pasta '**res**' no desenvolvimento de aplicativos Android com Kotlin **contém recursos estáticos utilizados na construção da interface do usuário e em outros aspectos do aplicativo.** "res" é uma abreviação de "resources" (recursos).

Esses recursos são separados em diferentes subpastas, cada uma com um propósito específico. Alguns dos subdiretórios comuns dentro da pasta res incluem:

✚ **Drawable:** contém imagens, ícones e outros recursos gráficos utilizados na interface do usuário.

✚ **Layout:** aqui são armazenados arquivos XML que definem a estrutura e o posicionamento dos elementos na interface do usuário. Isso inclui a disposição de botões, textos, imagens e outros componentes visuais.

✚ **Values:** contém arquivos XML que definem valores constantes, como strings, cores, dimensões e estilos. Isso ajuda a manter esses valores centralizados e reutilizáveis em todo o aplicativo.

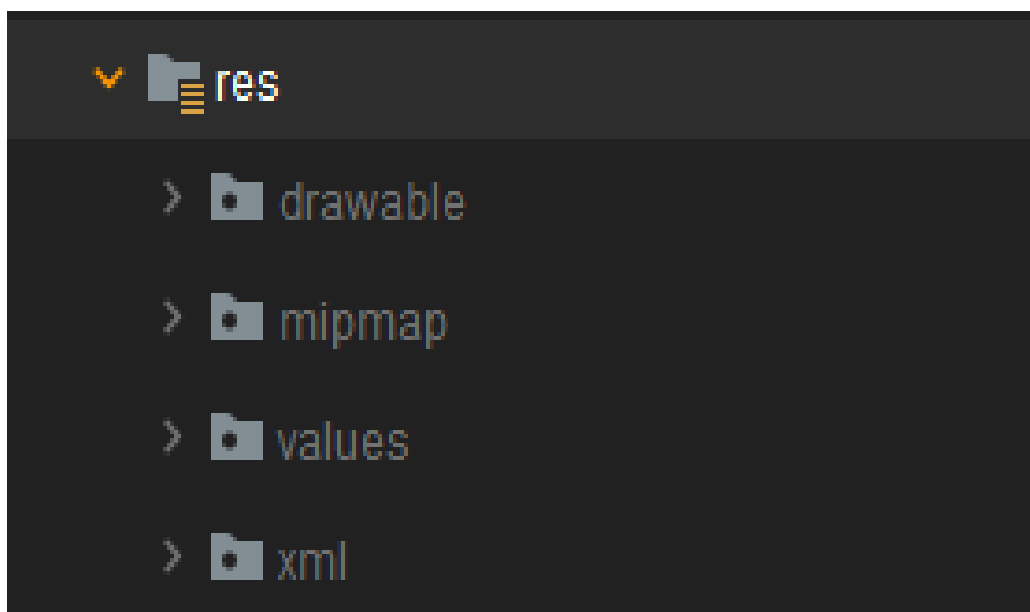
✚ **Menu:** guarda arquivos XML que definem menus de opções para ações dentro do aplicativo.

✚ **Anim:** aqui você pode colocar arquivos XML para animações que podem ser aplicadas a elementos da interface do usuário.

✚ **Raw:** armazena recursos que não são processados automaticamente pelo Android, como arquivos de mídia não compactados.

✚ **Xml:** usado para outros recursos XML que não se encaixam em nenhuma das categorias anteriores.

✚ **Mipmap:** contém ícones em várias densidades de pixels para a tela de início e outras partes do sistema Android.

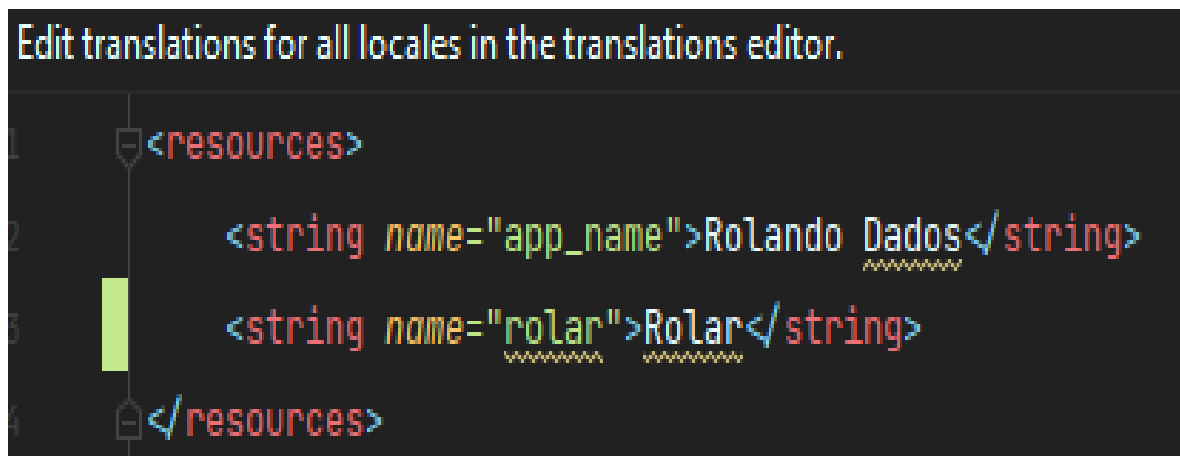


Fonte da imagem: autoria própria



Desta forma, alguns recursos precisamos ainda definir em arquivos XML mesmo trabalhando com o JetPack Compose e para reforçar: XML significa "Extensible Markup Language" (Linguagem de Marcação Extensível). É uma linguagem de marcação amplamente utilizada para estruturar e armazenar dados de forma hierárquica e legível por máquinas e humanos. **O XML não é uma linguagem de programação, mas sim uma linguagem de descrição de dados.**

Vamos acessar o arquivo `strings.xml` que se encontra dentro de: **res/values/strings.xml** e vamos adicionar uma string com valor de Rolar → **<string name='rolar'>Rolar</string>**



```

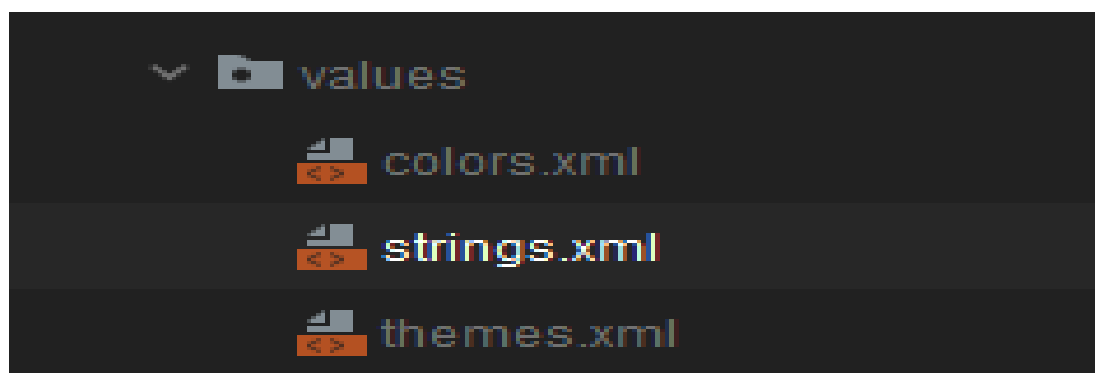
Edit translations for all locales in the translations editor.

1  <resources>
2      <string name="app_name">Rolando Dados</string>
3      <string name="rolar">Rolar</string>
4  </resources>
  
```

Fonte da imagem: autoria própria

### Por que estamos inserindo uma string no arquivo value.xml?

Porque estamos nos referindo à necessidade de definir um atributo de texto para o botão. Por exemplo, podemos definir o texto exibido em um botão usando um valor de string pré-definido. Isso permite que possamos manter o texto do botão em um único lugar, tornando mais fácil a alteração em toda a interface do usuário se necessário.



Fonte da imagem: autoria própria

O arquivo `strings.xml` é um componente fundamental em desenvolvimento Android com Kotlin (ou qualquer outra linguagem suportada pelo Android).

Ele é usado especificamente para armazenar strings, que são usadas em seu aplicativo.

A principal função do arquivo `strings.xml` é separar o conteúdo textual do código-fonte, permitindo que as strings sejam gerenciadas, localizadas e atualizadas de maneira mais eficiente.

Ao utilizar o arquivo `strings.xml`, estamos seguindo as melhores práticas de desenvolvimento Android, que incluem a separação de conteúdo estático (como strings) do código, tornando o aplicativo mais organizado, personalizável e adaptável a diferentes idiomas e localizações.

### 1.4 Trabalhando com elementos - Botão

Button é um elemento combinável do Android Compose que cria um botão clicável. Você passa uma função anônima (sem nome) para o parâmetro/evento `onClick` para especificar o que deve acontecer quando o botão for clicado.

O modifier é usado para aplicar modificações visuais ao botão, como adicionar preenchimento (`padding`), mas você também pode usar o modifier para aplicar várias outras modificações, conforme necessário.

Vamos configurar o nosso botão, observe a próxima imagem:



Para que possamos fazer o nosso dado rolando na tela quando clicamos no botão, precisamos da imagem específica. A imagem que vamos trabalhar é uma SVG.

**SVG significa "Scalable Vector Graphics" em inglês, que pode ser traduzido para o português como "Gráficos Vetoriais Escaláveis".** É um formato de arquivo de imagem baseado em XML (Extensible Markup Language), o que significa que ele descreve gráficos em forma de vetores usando uma linguagem de marcação.

**Ao contrário de formatos de imagem raster, como JPEG ou PNG, onde as imagens são compostas por pixels, as imagens em formato SVG são compostas por elementos vetoriais, como linhas, curvas, formas geométricas e texto. Isso permite que as imagens SVG sejam redimensionadas para qualquer tamanho sem perda de qualidade, pois os elementos vetoriais são recriados matematicamente quando a imagem é ampliada ou reduzida.**

Para obter esta imagem, clique [neste link](#) e faça download das imagens. Para importar, siga os seguintes passos:

- ➦ No Android Studio, clique em **View > Tool Windows > Resource Manager**.
- ➦ Clique em + > **Import Drawables** para abrir um navegador de arquivos.
- ➦ **Clique em Import** para confirmar que queremos importar as seis imagens.
- ➦ As imagens vão aparecer no painel **Resource Manager**.

Após a importação, é preciso observar que temos 6 imagens SVG para que possamos formar a 'animação' do dado que será jogado.

Importante seguir as referências a essas imagens no código Kotlin com os IDs dos recursos conforme as especificações abaixo:

- ✓ **R.drawable.dice\_1**
- ✓ **R.drawable.dice\_2**
- ✓ **R.drawable.dice\_3**
- ✓ **R.drawable.dice\_4**
- ✓ **R.drawable.dice\_5**
- ✓ **R.drawable.dice\_6**

### 1.5 Adicionar um elemento de composição Image

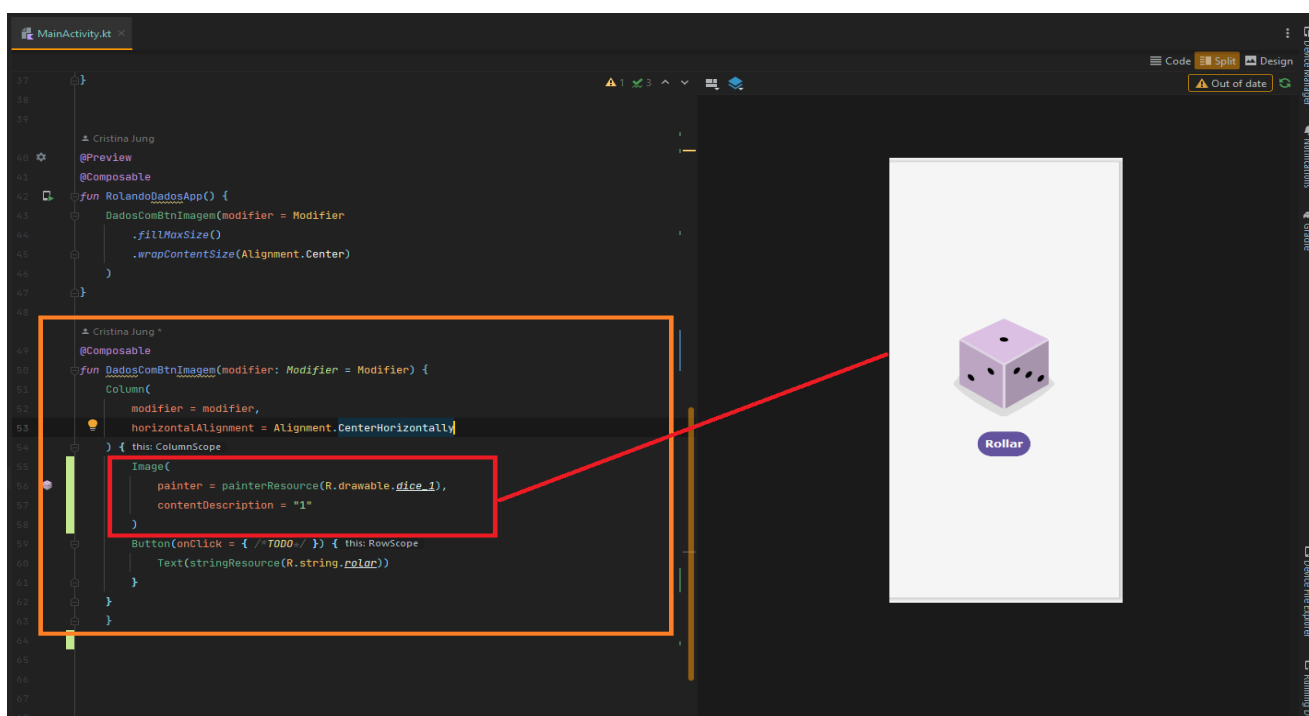
A imagem do dado precisa aparecer acima do botão Rolar. O Compose coloca naturalmente os componentes da IU de maneira sequencial.

**Em outras palavras, o elemento de composição é declarado e exibido primeiro. Isso pode indicar que a primeira declaração é exibida acima ou antes do elemento de composição declarado depois dela.**

Os elementos de composição dentro de um Column vão ser exibidos acima/abaixo um do outro no dispositivo. Neste app, estamos usando uma Column para empilhar os elementos de composição na vertical.

Portanto, o elemento de composição que é declarado primeiro dentro da função Column() é exibido antes dos elementos de composição que são declarados depois dele na mesma função Column().

Para adicionar um elemento de composição Image:



Fonte da imagem: autoria própria

### 1.6 Inserindo a lógica no botão

A lógica será definida por meio de: quando clicamos o botão, nosso dado irá ser girado para que possa retornar um valor aleatório dentro das faces do dado.

▪ Na função: DadosComBtnImagem e antes da função Column, iremos adicionar uma variável com nome de resultado e com valor de 1 atribuído.

▪ Em seguida, vamos acessar a função Button(). Próxima etapa será remover o comentário `/*TODO*/`. Importante aqui, conhecermos alguns conceitos de programação e não só do Kotlin. Quando vemos a seguinte sintaxe:

```
Button(onClick = { /*TODO*/ })
```

A função Button é um elemento de composição. Observe que ele está passando um **parâmetro onClick**, que está definido como um par de chaves com o comentário `/*TODO*/` dentro delas.

**Neste caso, as chaves representam o que é conhecido como lambda e a área dentro delas é o corpo da lambda.**

**Quando uma função é passada como um argumento, ela também pode ser chamada de "callback".** Para conhecer mais, é possível ler mais neste link: [Fluxos em Kotlin no Android](#)

Um lambda é uma função literal e funciona da mesma maneira que qualquer outra função.

No entanto, em vez de ser declarado separadamente com a palavra-chave "fun", ele é escrito inline e passado como uma expressão.

O componente Button, que pode ser combinado, espera que uma função seja passada como o parâmetro onClick. Este é o lugar ideal para utilizar um lambda.

A jogada do dado é aleatória. Para refletir isso no código, é preciso usar a sintaxe correta para gerar um número aleatório.

No Kotlin, podemos usar o método random() em um intervalo numérico.



Fonte da imagem: autoria própria

Importante que ainda o botão já está certo, porém não está funcional, ou seja, não causa mudanças visuais porque ainda precisamos finalizar a estruturação desta aplicação.

### 1.7 Adicionando condições no App de jogar dados

No tópico anterior, criamos uma variável chamada "resultado" e inicializamos com o valor 1.

Posteriormente, o valor da variável "resultado" é atualizado quando o usuário interage com o botão e a imagem a ser exibida é determinada com base nesse valor.

Por padrão, os componentes compostos estão sem estado, o que significa que eles não possuem um valor intrínseco e podem ser redefinidos a qualquer momento pelo sistema, resultando em uma nova redefinição de seus estados.

No entanto, o Compose oferece uma abordagem conveniente para lidar com isso. Os componentes compostos têm a capacidade de armazenar informações na memória usando o elemento "remember".

Agora, vamos transformar a variável "resultado" em um elemento de composição "remember". "Remember" é uma função do Jetpack Compose que permite armazenar e manter o estado de um composável durante as recomposições, o que é fundamental para o funcionamento adequado de muitos componentes de interface do usuário.

Portanto, é uma parte importante do sistema de composição do Jetpack Compose. O elemento de composição "remember" requer a passagem de uma função como argumento.

### **1.8 State & mutableState()**

Dentro do bloco de código do elemento de composição "remember", passe a função "mutableStateOf()" com o argumento 1. A função "mutableStateOf()" retorna um elemento observável.

Aos poucos iremos explorar conceitos mais aprofundados relacionados aos elementos observáveis mais adiante, mas, por ora, compreenda que sempre que o valor da variável "resultado" for alterado, uma recomposição será disparada.

Isso resultará na atualização do valor exibido e na atualização da interface do usuário.

Esta análise e utilização da função "mutableStateOf()", nos leva diretamente a um conceito extremamente importante no desenvolvimento de aplicativos mobile nativos (Android/Kotlin). Este conceito nós chamamos de State.

State se refere ao conceito de "estados" que um componente de interface do usuário pode ter.

Isso é particularmente relevante ao trabalhar com a arquitetura de Jetpack Compose, que é uma biblioteca moderna para a criação de interfaces de usuário em aplicativos Android.

No Jetpack Compose, os "states" são usados para representar valores que podem mudar ao longo do tempo e afetar a exibição da interface do usuário. Um "state" é uma fonte única de verdade que é reativa: sempre que o valor do "state" é atualizado, a interface do usuário é automaticamente atualizada para refletir essa mudança.

```

65  @Composable
66  fun DadosComBtnImagem(modifier: Modifier = Modifier) {
67      var resultado by remember { mutableStateOf( value: 1) }
68      val imageResource = when(resultado) {
69          1 → R.drawable.dice_1
70          2 → R.drawable.dice_2
71          3 → R.drawable.dice_3
72          4 → R.drawable.dice_4
73          5 → R.drawable.dice_5
74          else → R.drawable.dice_6
75      }
76      Column(
77          modifier = modifier,
78          horizontalAlignment = Alignment.CenterHorizontally
79      ) { this: ColumnScope
80          Image(painter = painterResource(imageResource), contentDescription = resultado.toString())
81          Button(onClick = { resultado = (1 ≤ .. ≤ 6).random() }) { this: RowScope
82              Text(text = stringResource(R.string.rolar), fontSize = 24.sp)
83          }
84      }
85  }

```

Fonte da imagem: autoria própria

Importante observar que no trecho de código (próxima imagem), usamos a instrução “when”. A instrução “when” é usada para criar expressões condicionais mais legíveis e concisas em Kotlin.

Ela é semelhante a um switch-case em outras linguagens, mas oferece mais flexibilidade e recursos.

```

val imageResource = when(resultado) {
    1 → R.drawable.dice_1
    2 → R.drawable.dice_2
    3 → R.drawable.dice_3
    4 → R.drawable.dice_4
    5 → R.drawable.dice_5
    else → R.drawable.dice_6
}

```

Fonte da imagem: autoria própria



Analisando este código, podemos ver que utilizamos a instrução "when" para mapear um valor chamado resultado em recursos de imagem em um contexto Android usando Kotlin.

Neste trecho, o resultado é uma variável (ou expressão) que contém o resultado de um lançamento de dado.

A instrução "when" avalia o valor de resultado e seleciona o recurso de imagem correspondente com base no valor. Aqui está como os casos são mapeados:

- ✚ Se o resultado for 1, `imageResource` receberá o ID do recurso `dice_1`.
- ✚ Se o resultado for 2, `imageResource` receberá o ID do recurso `dice_2`.
- ✚ Se o resultado for 3, `imageResource` receberá o ID do recurso `dice_3`.
- ✚ Se o resultado for 4, `imageResource` receberá o ID do recurso `dice_4`.
- ✚ Se o resultado for 5, `imageResource` receberá o ID do recurso `dice_5`.
- ✚ Para qualquer outro valor, `imageResource` receberá o ID do recurso `dice_6`.

Esse código é específico para atribuir uma imagem de dado específica com base no valor do lançamento do dado.

## ***2. Acessando dados para ajuste e correção***

Por fim, temos as etapas de verificação e correção do código, inclusive o foco desta unidade foi inspirado e adaptado do código de documentação do site Android Developer.

Portanto:

- ✚ Para que você possa fazer a verificação e correção do seu código, pode acessar o projeto por [este link](#).
- ✚ Este projeto desenvolvido neste livro, foi inspirado e adaptado do código de documentação do site Android Developers e você pode acessar por [este link](#).
- ✚ O código do projeto da Vídeo Aula está [neste link](#) para que você possa acompanhar mais detalhadamente.

### **3. Referências**

---

Múltiplos autores: **KOTLIN DOCS**. 2023. Disponível em:  
<<https://kotlinlang.org/docs/home.html>>. Acesso em: 17 de julho de 2023

Múltiplos autores: **JETBRAINS**. 2022. Disponível em:  
<<https://www.jetbrains.com/pt-br/>>. Acesso em: 18 de julho de 2023

Múltiplos autores: **DOCUMENTAÇÃO ANDROID STUDIO**. 2023. Disponível em:  
<<https://developer.android.com/studio>>. Acesso em: 18 de julho de 2023

Múltiplos autores: ESTADO E JETPACK COMPOSE. 2023. Disponível em: <[Estado e Jetpack Compose | Android Developers](#)>. Acesso em: 13 de agosto de 2023