Universidade POSITIVO

Desenvolvimento de Aplicativos Móveis

Professor Maurício Buess

mbuess@up.edu.br

https://github.com/mauriciobuess

Universidade POSITIVO

Introdução ao Desenvolvimento Android:

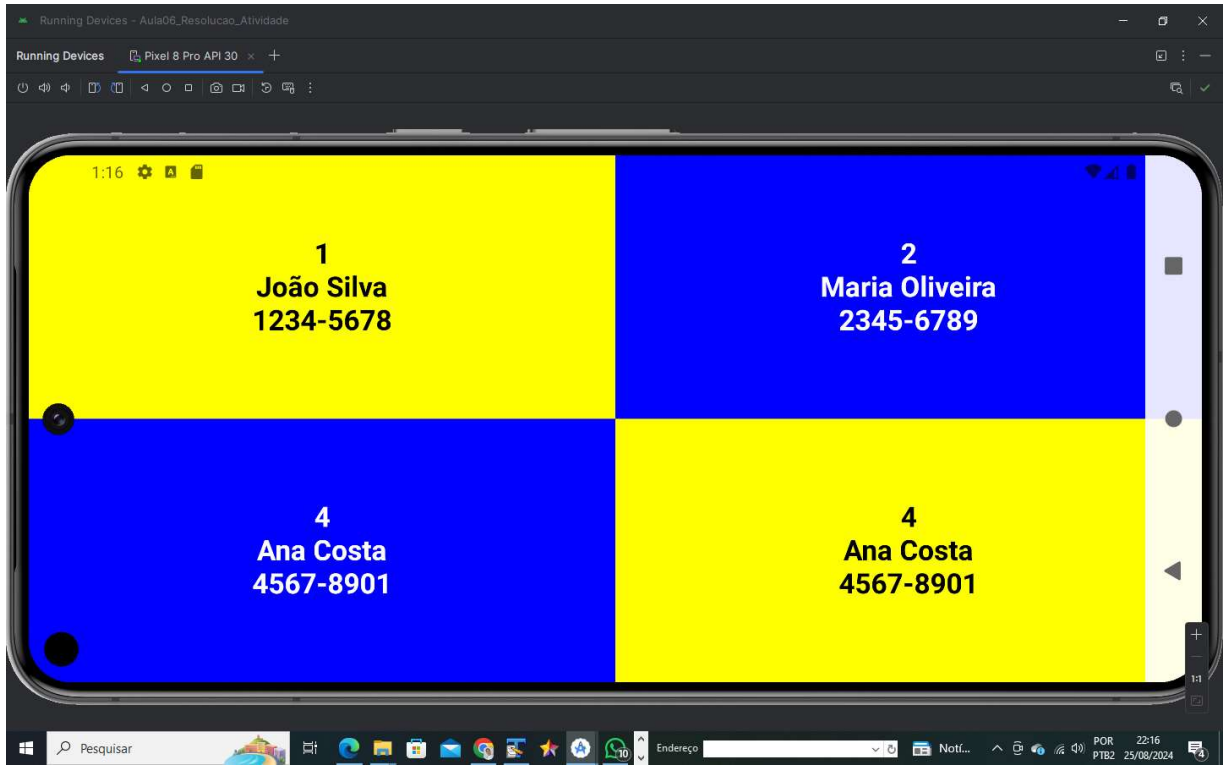
- Resolução atividade extra classe da aula anterior
- Compreender o funcionamento dos elementos Card e LazyColumn

Atividade extra-aula:

- Desenvolver uma UI em que a função composta divida a área disponível do dispositivo em quatro partes iguais, cada uma de uma cor diferente, e que no centro de cada quadrante apresente o nome e o número do telefone da pessoa, dados esses que deverão ser passados como parâmetro para a função composta.

Universidade POSITIVO

Atividade extra-aula:



Resolução atividade extra-aula:

- Criar a classe de dados que será o tipo de dados da lista a ser passada para a função @Composable;
  - A classe conterá os seguintes dados:
    - id : tipo Int
    - nome : tipo String
    - telefone : tipo String
- Criar uma lista que receberá quatro ocorrências da classe de dados criada

Resolução atividade extra-aula:

```
// Criação da classe de dados
data class Cadastro (
    val idCadastro      : Int,
    val nomeCadastro    : String,
    val foneCadastro    : String
)
```

Resolução atividade extra-aula:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Aula06_Resolucao_AtividadeTheme {
                val listaCadastros = listOf(
                    Cadastro(idCadastro = 1, nomeCadastro = "João Silva", foneCadastro = "1234-5678"),
                    Cadastro(idCadastro = 2, nomeCadastro = "Maria Oliveira", foneCadastro = "2345-6789"),
                    Cadastro(idCadastro = 3, nomeCadastro = "Pedro Santos", foneCadastro = "3456-7890"),
                    Cadastro(idCadastro = 4, nomeCadastro = "Ana Costa", foneCadastro = "4567-8901")
                )

            }
        }
    }
}
```

Resolução atividade extra-aula:

- listOf - função de nível superior em Kotlin que retorna uma instância de List;
  - É uma interface da coleção imutável.
  - A lista criada é imutável, o que significa que você não pode adicionar, remover ou alterar os elementos após a criação da lista.
  - A imutabilidade ajuda a garantir que a lista permaneça consistente e previsível.
  - Síntaxe básica:

```
val lista = listOf(elemento1, elemento2, elemento3, ...)
```

```
val numeros = listOf(1, 2, 3, 4, 5)
println(numeros) // Output: [1, 2, 3, 4, 5]
```

```
data class Pessoa(val nome: String, val idade: Int)

val pessoas = listOf(
    Pessoa("João", 25),
    Pessoa("Maria", 30),
    Pessoa("Pedro", 22)
)
println(pessoas)
```

Resolução atividade extra-aula:

- Criar uma função @Composable – a qual eu nomeei como CardColumn – que será a responsável por reproduzir o *layout* de tela esperado e tratar a lista de dados conforme o solicitado;
- A função @Composable CardColumn usará os seguintes elementos:
  - Column
  - Row
  - Box
  - Text

Resolução atividade extra-aula:

- Column: organiza verticalmente o conteúdo do contexto seu contexto.

```
@Composable
fun MyColumnExample() {
    Column(
        modifier = Modifier.fillMaxSize(), // Ocupa toda a tela
        verticalArrangement = Arrangement.Top, // Alinhamento vertical dos itens
        horizontalAlignment = Alignment.CenterHorizontally // Alinhamento horizontal dos itens
    ) {
        Text(text = "Primeiro Item")
        Text(text = "Segundo Item")
        Text(text = "Terceiro Item")
    }
}
```

Resolução atividade extra-aula:

- Row: organiza horizontalmente o conteúdo do contexto seu contexto.

```
@Composable
fun MyRowExample() {
    Row(
        modifier = Modifier.fillMaxSize(), // Ocupa toda a tela
        horizontalArrangement = Arrangement.SpaceBetween, // Alinhamento horizontal dos itens
        verticalAlignment = Alignment.CenterVertically // Alinhamento vertical dos itens
    ) {
        Text(text = "Item 1")
        Text(text = "Item 2")
        Text(text = "Item 3")
    }
}
```

Resolução atividade extra-aula:

- Box: Empilhar os elementos sobrepostos. O último elemento adicionado será exibido por cima dos anteriores.

```
@Composable
fun MyBoxExample() {
    Box( modifier = Modifier.size(200.dp) // Define o tamanho do Box
    ) {
        Box(modifier = Modifier
                .matchParentSize()
                .background(Color.Red) // Cor de fundo do Box
        )
        Text(text = "Texto Sobreposto")
        )
    }
}
```

Resolução atividade extra-aula:

```
fun CardColumn(cadastros : List<Cadastro>) {
   Column(modifier = Modifier.fillMaxSize())
   {   Row(modifier = Modifier.fillMaxSize().weight(1f)
      ) {Box(modifier = Modifier.weight(1f)
               .background(Color.Yellow).fillMaxHeight()
         ) { Column(modifier = Modifier.fillMaxSize()
               ,verticalArrangement = Arrangement.Center
               , horizontalAlignment = Alignment.CenterHorizontally) {
               Text(text = cadastros[0].idCadastro.toString()
                  , color = Color.Black
                  , style = TextStyle(fontStyle = FontStyle.Normal
                     ,fontSize = 24.sp
                     ,fontWeight = FontWeight.Bold)
               )
               Text(text = cadastros[0].nomeCadastro
                  , color = Color.Black
                  , style = TextStyle(fontStyle = FontStyle.Normal
                     ,fontSize = 24.sp
                     ,fontWeight = FontWeight.Bold))
               Text(text = cadastros[0].foneCadastro
                  , color = Color.Black
                  , style = TextStyle(fontStyle = FontStyle.Normal
                     ,fontSize = 24.sp
                     ,fontWeight = FontWeight.Bold))
         }
      }
```

Resolução atividade extra-aula:

```
Box(modifier = Modifier.weight(1f)
        .background(Color.Blue).fillMaxHeight()
 ) { Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(text = cadastros[1].idCadastro.toString()
            , color = Color.White
            , style = TextStyle(fontStyle = FontStyle.Normal
                ,fontSize = 24.sp
                ,fontWeight = FontWeight.Bold)
        )
        Text(text = cadastros[1].nomeCadastro
            , color = Color.White
            , style = TextStyle(fontStyle = FontStyle.Normal
                ,fontSize = 24.sp
                ,fontWeight = FontWeight.Bold)
        )
        Text(text = cadastros[1].foneCadastro
            , color = Color.White
            , style = TextStyle(fontStyle = FontStyle.Normal
                ,fontSize = 24.sp
                ,fontWeight = FontWeight.Bold)
        )
    }
}
```

Resolução atividade extra-aula:

```
}
//--
Row(modifier = Modifier.fillMaxSize().weight(1f)
) { Box(modifier = Modifier.weight(1f)
        .background(Color.Blue).fillMaxHeight()
   ){ Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
      ) {
        Text(text = cadastros[3].idCadastro.toString()
           , color = Color.White
           , style = TextStyle(fontStyle = FontStyle.Normal
              ,fontSize = 24.sp
              ,fontWeight = FontWeight.Bold)
        )
        Text(text = cadastros[3].nomeCadastro
           , color = Color.White
           , style = TextStyle(fontStyle = FontStyle.Normal
              ,fontSize = 24.sp
              ,fontWeight = FontWeight.Bold)
        )
```

Resolução atividade extra-aula:

```
        Text(text = cadastros[3].foneCadastro
            , color = Color.White
            , style = TextStyle(fontStyle = FontStyle.Normal
                ,fontSize = 24.sp
                ,fontWeight = FontWeight.Bold)
        )
    }
}
Box(modifier = Modifier.weight(1f)
        .background(Color.Yellow).fillMaxHeight()
) { Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) { Text(text = cadastros[3].idCadastro.toString()
            , color = Color.Black
            , style = TextStyle(fontStyle = FontStyle.Normal
                ,fontSize = 24.sp
                ,fontWeight = FontWeight.Bold)
        )
```

Resolução atividade extra-aula:

```
                Text(text = cadastros[3].nomeCadastro
                    , color = Color.Black
                    , style = TextStyle(fontStyle = FontStyle.Normal
                        ,fontSize = 24.sp
                        ,fontWeight = FontWeight.Bold)
                )
                Text(text = cadastros[3].foneCadastro
                    , color = Color.Black
                    , style = TextStyle(fontStyle = FontStyle.Normal
                        ,fontSize = 24.sp
                        ,fontWeight = FontWeight.Bold)
                )
            }
        }
    }
}
```

Resolução atividade extra-aula:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Aula06_Resolucao_AtividadeTheme {
                val listaCadastros = listOf(
                    Cadastro(idCadastro = 1, nomeCadastro = "João Silva", foneCadastro = "1234-5678"),
                    Cadastro(idCadastro = 2, nomeCadastro = "Maria Oliveira", foneCadastro = "2345-6789"),
                    Cadastro(idCadastro = 3, nomeCadastro = "Pedro Santos", foneCadastro = "3456-7890"),
                    Cadastro(idCadastro = 4, nomeCadastro = "Ana Costa", foneCadastro = "4567-8901")
                )
        CadastroList(cadastros = listaCadastros)
            }
        }
    }
}
```

O elemento Card

- O **Card** é um componente que encapsula o conteúdo dentro de uma área de forma arredondada e com sombra.
- Muito útil para criar uma hierarquia visual clara e destacar diferentes partes da interface.

```
@Composable
fun MyCardExample() {
    Card(
        modifier = Modifier.padding(16.dp), // espaço ao redor do Card
        elevation = 4.dp, // Define a sombra do Card
        shape = RoundedCornerShape(8.dp) // arredondando as bordas
    ) {
        // Conteúdo do Card
        Column(modifier = Modifier.padding(16.dp)
        ) {Text(text = "Título do Card", fontWeight = FontWeight.Bold)
            Spacer(modifier = Modifier.height(8.dp))
            Text(text = "Conteúdo do Card")
        }
    }
}
```

O elemento Card – Principais Parâmetros

- **modifier**: define o tamanho, o padding, e outras propriedades de layout para o Card.
- **elevation**: Define a sombra do Card. O valor é uma distância em dp que controla a profundidade da sombra.
- **shape**: Define o formato das bordas do Card. Pode ser um RoundedCornerShape ou qualquer outra forma personalizada.
- **backgroundColor**: Define a cor de fundo do Card.
- **contentColor**: Define a cor do conteúdo do Card, como o texto.

O elemento Card

```
@Composable
fun SimpleCard() {
    Card(modifier = Modifier.padding(8.dp),
        elevation = 2.dp,
        shape = RoundedCornerShape(8.dp)
    ) {
        Column(modifier = Modifier.padding(16.dp)
        ) {
            Text(text = "Título", fontWeight = FontWeight.Bold)
            Text(text = "Descrição do conteúdo do card.")
        }
    }
}
```

O elemento Card

```
@Composable
fun CardWithImage() {
    Card(modifier = Modifier.padding(8.dp),
        elevation = 4.dp,
        shape = RoundedCornerShape(16.dp)
    ) {Column {Image(painter = painterResource(id = R.drawable.ic_task_completed),
                contentDescription = null,
                modifier = Modifier.fillMaxWidth())
            Column(modifier = Modifier.padding(16.dp)
            ) {Text(text = "Título com Imagem", fontWeight = FontWeight.Bold)
                Text(text = "Descrição abaixo da imagem.")
            }
        }
    }
}
```

O elemento LazyColumn

- Um dos componentes de layout fundamentais no Jetpack Compose;
- Usado para exibir listas de itens que podem ser roladas verticalmente.
- É uma versão otimizada do Column para grandes conjuntos de dados, carregando itens sob demanda para melhorar o desempenho e a eficiência de memória.

- É um tipo de lista vertical que exibe apenas os itens que estão visíveis na tela e aqueles próximos, carregando e descarregando itens conforme você rola a lista. Isso é conhecido como "lazy loading" e ajuda a reduzir o uso de memória e a melhorar o desempenho quando lidamos com grandes conjuntos de dados.

LazyColumn – Síntaxe Básica

```
@Composable
fun MyLazyColumn() {
    LazyColumn {
        items(listOf("Item 1", "Item 2", "Item 3")) { item ->
            Text(text = item)
        }
    }
}
```

items: Uma função que mapeia uma lista de dados para a UI. Cada item é renderizado usando o lambda fornecido.

item: Uma função que pode ser usada para criar um item individual na lista. Ideal para listas pequenas onde a quantidade de itens é fixa e conhecida.

modifier: Permite adicionar modificadores como padding ou fillMaxSize para o LazyColumn.

contentPadding: Adiciona espaço extra ao redor do conteúdo da lista. Útil para adicionar padding ao início e ao fim da lista.

LazyColumn – Síntaxe Básica

```
@Composable
fun SimpleLazyColumn() {
    LazyColumn {
        items(listOf("Item 1", "Item 2", "Item 3", "Item 4")) { item ->
            Text(
                text = item,
                modifier = Modifier.padding(16.dp)
            )
        }
    }
}
```

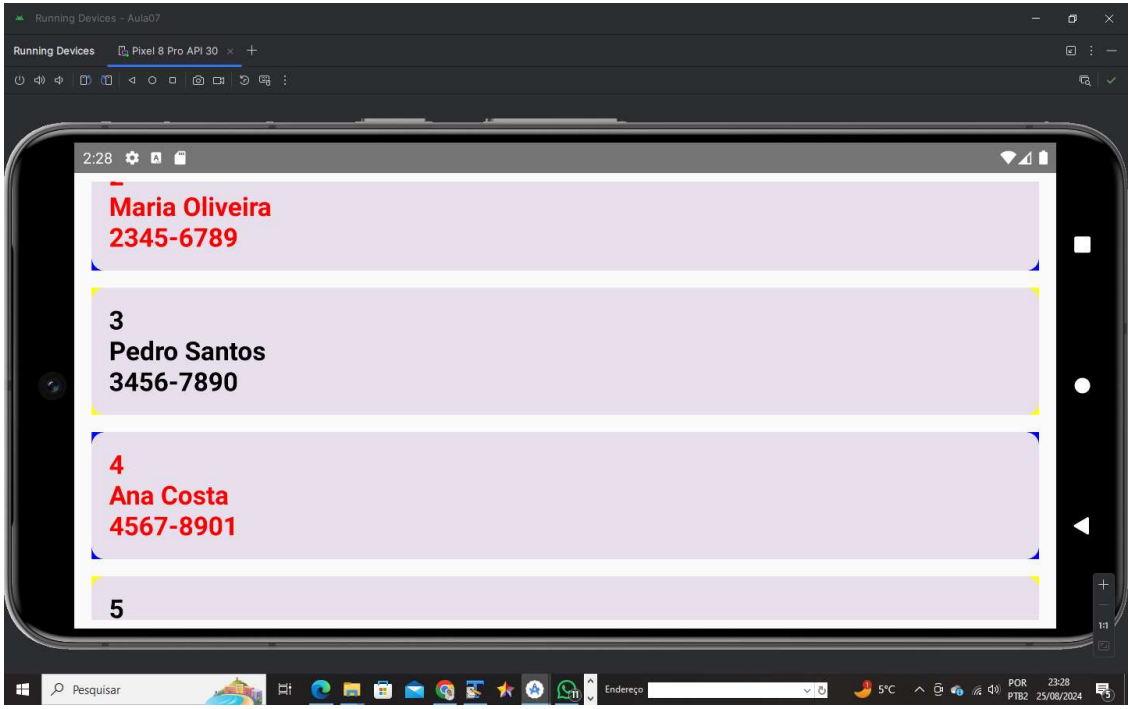
LazyColumn – Síntaxe Básica

```
@Composable
fun ImageTextLazyColumn() {
    LazyColumn {items(listOf("Item 1", "Item 2", "Item 3")) { item ->
            Row(modifier = Modifier.fillMaxWidth().padding(8.dp)
            ) {Image(painter = painterResource(id = R.drawable.ic_launcher_foreground),
                    contentDescription = null,
                    modifier = Modifier.size(40.dp)
                )
                Spacer(modifier = Modifier.width(8.dp))
                Text(text = item, fontSize = 18.sp)
            }
        }
    }
}
```

Universidade POSITIVO

Atividade Aula

- Reproduzir o seguinte comportamento:



Exercícios;

1) Você está desenvolvendo uma tela inicial para um aplicativo e sua tarefa é criar uma tela simples que exiba um título e uma descrição usando Column e Text.

- Ambos conteúdos devem aparecer centralizados vertical e horizontalmente na tela do dispositivo, sendo que o título deve anteceder a descrição e estar destacado graficamente;
- A descrição deve estar abaixo do título, de forma mais discreta porém facilmente perceptível ao usuário.
- O analista responsável pelo projeto o alertou que essa tela será reutilizada em oportunidades futuras, logo a mesma não deverá utilizar “hard code” e sim estar pronta para tratar tais atributos como parâmetros.

Dicas:

- Use Modifier.padding para adicionar espaço ao redor dos textos.
- Utilize fontWeight e fontSize para estilizar o título.

Universidade POSITIVO

Exercícios;

2) De fato você conseguiu impressionar o analista, tanto que ele requisitou que se mantenha o código recém criado e que desenvolva uma nova função composta que mantenha as funcionalidade da anterior porém a nova função recebará como parâmetro uma imagem de fundo de tela, imagem essa que poderá ser alterada no decorrer da execução do aplicativo.

Dicas:

- Importe as duas imagens de fundo (saharaDesert.jpg e fundoFesta.jpg) contidas no endereço a seguir:

https://github.com/mauriciobuess/Universidade-Positivo/tree/main/2024/**Desenvolvimento_Aplicativos_Moveis/Aulas/images**

- Crie uma classe de dados
- Use a classe de dados como parâmetro de entrada da função composta

Exercícios;

3) Você está desenvolvendo uma tela para um aplicativo de perfil de usuário. A tela deve exibir um cabeçalho com uma foto de perfil e um nome de usuário, seguido por uma lista de postagens do usuário. Use uma combinação de Box, Row, Column, LazyColumn, e Card.

Objetivo:

- Crie uma tela que contenha um cabeçalho (Box) com uma imagem de perfil e um nome de usuário (Row e Column).
- Abaixo do cabeçalho, exiba uma lista de postagens usando LazyColumn e Card.

Dicas:

- Use Box para sobrepor a imagem de perfil e o nome de usuário.
- Organize a imagem e o texto do cabeçalho em uma Row.
- Para a lista de postagens, use uma data class chamada Post.