INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: LISTANDO ELEMENTOS (LAZYCOLUMN E VIEWMODEL)

Prática 03

ATENÇÃO: Continuação da Prática 02; Use controle de versões (Git).

Parte 1: Criando uma lista simples

Passo 1: Crie uma *data class* `City` com o código abaixo em **ListPage.kt**:

```
data class City(val name: String, var weather: String)
```

Essa classe contém as informações relacionadas as cidades favoritas, por enquanto apenas nome e uma descrição das condições climáticas.

Passo 2: Crie função privada `getCities()` em **ListPage.kt**:

```
private fun getCities() = List(30) { i ->
    City(name = "Cidade $i", weather = "Carregando clima...")
}
```

Essa função cria uma lista de 30 cidades temporárias para testes.

Passo 3: Crie o `@composable CityItem` em **ListPage.kt**:

```
@Composable
fun CityItem(
    city: City,
    onClick: () -> Unit,
    onClose: () -> Unit,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier.fillMaxWidth().padding(8.dp).clickable { onClick() },
        verticalAlignment = Alignment.CenterVertically
    ) {
        Icon(
            Icons.Rounded.FavoriteBorder,
            contentDescription = ""
        )
        Spacer(modifier = Modifier.size(12.dp))
        Column(modifier = modifier.weight(1f)) {
            Text(modifier = Modifier,
                text = city.name,
                fontSize = 24.sp)
            Text(modifier = Modifier,
                text = city.weather,
                fontSize = 16.sp)
        }
        IconButton(onClick = onClose) {
            Icon(Icons.Filled.Close, contentDescription = "Close")
        }
    }
}
```

Essa classe descreve os itens visuais que serão exibidos na lista, incluindo uma imagem temporária a esquerda e um botão X que será usado para excluir a cidade da lista.

Passo 4: Modifique o `@composable ListPage` para conter o código abaixo:

```
@Preview
@Composable
fun ListPage(modifier: Modifier = Modifier) {
    val cityList = remember { getCities().toMutableStateList() }
    LazyColumn(
        modifier = Modifier
            .fillMaxSize()
            .padding(8.dp)
    ) {
        items(cityList) { city ->
            CityItem(city = city, onClose = {
                /* TO DO */
            }, onClick = {city ->
                /* TO DO */
            })
        }
    }
}
```

O componente `LazyColumn` cria itens dinamicamente à medida que eles aparecem na tela. Em caso de problemas de compilação com `items(...)`, uses o `import` abaixo:

```
import androidx.compose.foundation.lazy.items
```

Passo 5: Altere o `ListPage` para lançar um `Toast` no `onClick()` e no `onClose()` do `CityItem`.

Use como referência o código de `LoginActivity`. Veja que é preciso uma referência à atividade atual para lançar o `Toast`.

Passo 6: Rode e teste o aplicativo. Lembre de comitar se estiver tudo certo.

Temos uma lista de 30 cidades. Quando uma cidade na lista é clicada/tocada, umas mensagem simples e apresentada abaixo; o botão X ainda não remove as cidades.

Parte 2: Removendo elementos dinamicamente usando ViewModel.

Passo 0: No arquivo **build.gradle.kts** do módulo *app*, adicione a seguinte dependência.

```
...
dependencies {

    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.7.0")
    ...
}
```

Faça o Android Studio sincronizar o projeto. Faça um *clean build* se tiver problemas.

Passo 1: Crie a classe `MainViewModel` em um arquivo apropriado.

```
class MainViewModel : ViewModel() {
    val cities = getCities().toMutableStateList()

    fun remove(city: City) {
        cities.remove(city)
    }

    fun add(city: String) {
        cities.add(City(city, "Carregando clima..."))
    }
}
```

A classe `ViewModel` faz o comunicação entre os dados do App os componentes da UI.

Passo 2: Mova a classe `City` e a função `getCities()` para esse arquivo.

Faça as importações necessárias em `ListPage`.

Passo 3: Modifique a `ListPage` para utilizar o `ViewModel`, como no código a seguir:

```
@Preview
@Composable
fun ListPage(
    modifier: Modifier = Modifier,
    viewModel: MainViewModel
) {
    val cityList = viewModel.cities
    ...
    LazyColumn(...) {
        ...
    }
}
```

`ListPage` agora recebe um `ViewModel` como parâmetro, usando a lista contida nele.

Passo 4: Altere o código do `ListPage` de forma que o `onClose()` do `CityItem` remova a cidade a lista de favoritos usando o `viewModel`:

```
    onClose = { viewModel.remove(city) }
```

Passo 5: Faça `MainNavHost` receber uma parâmetro do tipo `MainViewModel` que deve ser repassado para o `ListPage`. Modifique também a `MainActivity` para instanciar `MainViewModel` (dentro de `setContent`, como abaixo) e passar para o `MainNavHost`.

```
    val viewModel : MainViewModel by viewModels()
```

Isso é chamado de *state hoisting*: o estado (dados) deve ser instanciado no nível mais alto da aplicação, para não serem recriados a cada recomposição.

Passo 6: Rodar, testar e comitar.

Veja que agora é possível remover os itens da lista clicando no botão X.

Faça um novo commit se estiver tudo correto.

Parte 3: Adicionando elementos com um Diálogo.

Passo 1: Crie o `@composable CityDialog` abaixo no pacote `ui`.

```
@Composable
fun CityDialog(onDismiss: () -> Unit, onConfirm: (city: String) -> Unit) {
    val cityName = remember { mutableStateOf("") }

    Dialog(onDismissRequest = { onDismiss() } ) {
        Surface( shape = RoundedCornerShape(16.dp) ) {
            Column(modifier = Modifier.padding(20.dp)) {
                Row(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement = Arrangement.SpaceBetween,
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    Text(text = "Adicionar cidade favorita:")
                    Icon(imageVector = Icons.Filled.Close,
                        contentDescription = "",
                        modifier = Modifier.clickable { onDismiss() })
                }
                Spacer(modifier = Modifier.height(20.dp))
                OutlinedTextField(
                    modifier = Modifier.fillMaxWidth(),
                    label = { Text(text = "Nome da cidade") },
                    value = cityName.value,
                    onValueChange = { cityName.value = it })
                Spacer(modifier = Modifier.height(20.dp))
                Button(
                    onClick = { onConfirm(cityName.value) },
                    modifier = Modifier.fillMaxWidth().height(50.dp)
                ) { Text(text = "OK") }
            }
        }
    }
}
```

Esse diálogo será lançado ao clicar no botão flutuante "+" e permitirá ao usuário digitar o nome de uma nova cidade favorita.

Passo 2: Na `MainActivity`, adicione uma variável local dentro de `setContent()` com o código:

```
    var showDialog by remember { mutableStateOf(false) }
```

Passo 3: Na `MainActivity`, dentro de `WebAppTheme` e antes de `Scaffold`, adicione o código:

```
    if (showDialog.value) CityDialog(
        onDismiss = { showDialog = false },
        onConfirm = { city ->
            if (city.isNotBlank()) viewModel.add(city)
            showDialog = false
        })
```

Esse código lança o diálogo (dependendo da variável `showDialog`) e configura o que acontece em caso de cancelamento (`onDismiss`) ou botão Ok (`onConfirm`) ser clicado. Nesse caso, uma nova cidade é adicionada à lista via `viewModel`, o que atualiza a UI.

Passo 4: Na `MainActivity`, configure o tratador de *click* do botão flutuante "+" como abaixo:

```
     onClick = { showDialog = true }
```

A mudança em `showDialog` força uma recomposição, exibindo o diálogo na tela.

Passo 5: Rodar, testar e comitar.

Veja que agora é possível tanto adicionar quanto remover os itens da lista.