FIAP ON

APP WORLD

GESTÃO DE **ESTADO NO JETPACK COMPOSE**

9A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 GESTÃO DE ESTADO NO JETPACK COMPOSE

A Interface do Usuário (IU) em uma aplicação Android é formada por diversos elementos. Para que possamos organizar melhor nosso código, é necessário “quebrarmos” a interface em diversos componentes, que depois serão combinados para criarmos a tela como um todo. Essa técnica é bastante interessante, já que permite a reutilização de componentes.

Outro fator bastante importante quando trabalhamos com a componentização é podermos manter o estado da aplicação em seus diversos componentes. Para isso, podemos utilizar o “State Hoisting” que significa manter o estado em um componente de hierarquia mais alta, ou podemos utilizar uma arquitetura de projeto chamada Model-View-ViewModel, MVVM.

1.1 State Hoisting - Elevação de estado

A Elevação de Estado ou “*State Hoisting*” é um *design pattern* que orienta a implementação de componentes “stateless”, ou seja, que não gerenciam o próprio estado. O estado deverá ser mantido no menor ancestral comum entre todos os componentes combináveis.

No Jetpack Compose, cada composable é uma função independente que descreve a aparência e o comportamento de um componente. Deste modo, podemos reutilizar este componente em diversas partes da aplicação, o que poderia causar duplicidade de estado. O “State Hoisting” garante que o estado será gerenciado de forma centralizada, garantindo que compartilhem o mesmo estado atualizado.

Conforme exemplificado na figura “Estrutura hierárquica da aplicação”, o componente “AppScreen” é o componente de hierarquia mais alta. Os componentes filhos: Componente 1, Componente 2 e Componente 3 são os componentes de hierarquia mais baixa, assim, quem deve manter o estado da tela é o “AppScreen”, ou seja, as variáveis de estado devem ser declaradas neste componente e os componentes “filhos” recebem o estado e comportamento do componente AppScreen.

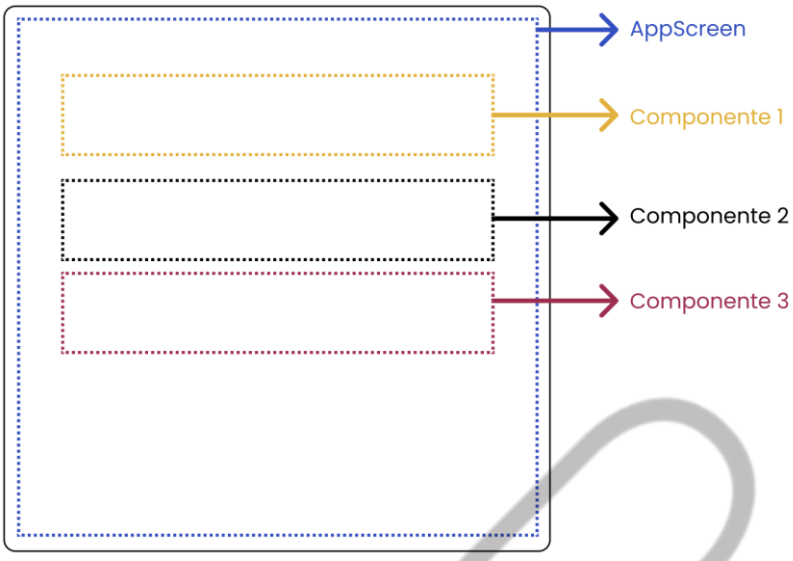

Figura 1 – Estrutura hierárquica da aplicação
Fonte: Elaborado pelo autor (2023)

1.1.1 Aplicativo para cálculo de juros simples

Para praticarmos o “State Hoisting”, vamos criar uma aplicação para cálculo de juros simples. Vamos começar criando uma única função com todos os componentes e estados. Depois, aplicaremos o conceito de elevação de estado e modularização da aplicação.

Crie um projeto no Android Studio com o nome “Cálculo de Juros”. Substitua o código do arquivo “MainActivity.kt” que foi gerado automaticamente, pelo código da listagem abaixo:

```
package br.com.fiap.calculodejuros

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import br.com.fiap.calculodejuros.calculos.calcularJuros
import br.com.fiap.calculodejuros.calculos.calcularMontante
import br.com.fiap.calculodejuros.ui.theme.CalculoDeJurosTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      CalculoDeJurosTheme {
        // A surface container using the 'background' color from the theme
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          JurosScreen()
        }
      }
    }
  }
}

@Composable
fun JurosScreen() {

  var capital by remember { mutableStateOf("") }
  var taxa by remember { mutableStateOf("") }
  var tempo by remember { mutableStateOf("") }
```

```
var juros by remember { mutableStateOf(0.0) }
var montante by remember { mutableStateOf(0.0) }

Box(
  modifier = Modifier.padding(16.dp),
  contentAlignment = Alignment.Center
) {
  Column() {
    Text(
      text = "Cálculo de Juros Simples",
      modifier = Modifier.fillMaxWidth(),
      fontSize = 20.sp,
      color = Color.Red,
      fontWeight = FontWeight.Bold,
      textAlign = TextAlign.Center
    )
    Spacer(modifier = Modifier.height(32.dp))
    // Formulário para entrada de dados
    Card(
      modifier = Modifier
        .fillMaxWidth()
    ) {
      Column(modifier = Modifier.padding(16.dp)) {
        Text(
            text = "Dados do Investimento",
            fontWeight = FontWeight.Bold
        )
        // Caixas de entrada da aplicação
        OutlinedTextField(
          value = capital,
          onValueChange = { capital = it },
          modifier = Modifier
              .fillMaxWidth()
              .padding(top = 16.dp),
          placeholder = {
            Text(text = "Quanto deseja investir?")
          },
          label = {
            Text(text = "Valor do investimento")
          },
          keyboardOptions = KeyboardOptions(
              keyboardType = KeyboardType.Decimal
          )
        )
        OutlinedTextField(
          value = taxa,
          onValueChange = { taxa = it },
          modifier = Modifier
              .fillMaxWidth()
              .padding(top = 16.dp),
          placeholder = {
            Text(text = "Qual a taxa de juros mensal?")
          },
          label = {
            Text(text = "Taxa de juros mensal")
          },
          keyboardOptions = KeyboardOptions(
              keyboardType = KeyboardType.Decimal
          )
        )
        OutlinedTextField(
```

```
            value = tempo,
            onValueChange = { tempo = it },
            modifier = Modifier
                .fillMaxWidth()
                .padding(top = 16.dp),
            placeholder = {
              Text(text = "Qual o tempo em meses?")
            },
            label = {
              Text(text = "Período em meses")
            },
            keyboardOptions = KeyboardOptions(
                keyboardType = KeyboardType.Decimal
            )
          )
          Button(
            onClick = {
              juros = calcularJuros(
                capital = capital.toDouble(),
                taxa = taxa.toDouble(),
                tempo = tempo.toDouble()
              )
              montante = calcularMontante(
                capital = capital.toDouble(),
                juros = juros
              )
            },
            modifier = Modifier
                .fillMaxWidth()
                .padding(top = 32.dp)
          ) {
            Text(text = "CALCULAR")
          }
        }
      }
      Spacer(modifier = Modifier.height(16.dp))
      // Resultado da aplicação
      Card(
        modifier = Modifier
          .fillMaxWidth(),
        colors = CardDefaults.cardColors(
          containerColor = Color(0xFF4CAF50)
        )
      ) {
        Column(
          modifier = Modifier
            //.fillMaxSize()
            .padding(16.dp)
        ) {
          Text(
            text = "Resultado",
            fontSize = 18.sp,
            fontWeight = FontWeight.Bold,
            color = Color.White
          )
          Spacer(modifier = Modifier.height(16.dp))
          Row(modifier = Modifier.fillMaxWidth()) {
            Text(
              text = "Juros",
              modifier = Modifier.padding(end = 8.dp),
              fontSize = 16.sp,
```

```
                fontWeight = FontWeight.Bold
              )
              Text(
                text = juros.toString(),
                modifier = Modifier.padding(end = 8.dp),
                fontSize = 16.sp,
                fontWeight = FontWeight.Bold,
                color = Color.White
              )
            }
            Spacer(modifier = Modifier.height(8.dp))
            Row(modifier = Modifier.fillMaxWidth()) {
              Text(
                text = "Montante",
                modifier = Modifier.padding(end = 8.dp),
                fontSize = 16.sp,
                fontWeight = FontWeight.Bold
              )
              Text(
                text = montante.toString(),
                modifier = Modifier.padding(end = 8.dp),
                fontSize = 16.sp,
                fontWeight = FontWeight.Bold,
                color = Color.White
              )
            }
          }
        }
      }
    }
}
```

Código-fonte 1 – Aplicação Cálculo de Juros Simples
Fonte: Elaborado pelo autor (2023)

Vamos criar um pacote em nossa aplicação com o nome “calculos". Crie neste pacote um arquivo com o nome “CalcularJuros.kt”. A estrutura de pastas do seu projeto deverá se parecer com a figura “Estrutura do projeto”:

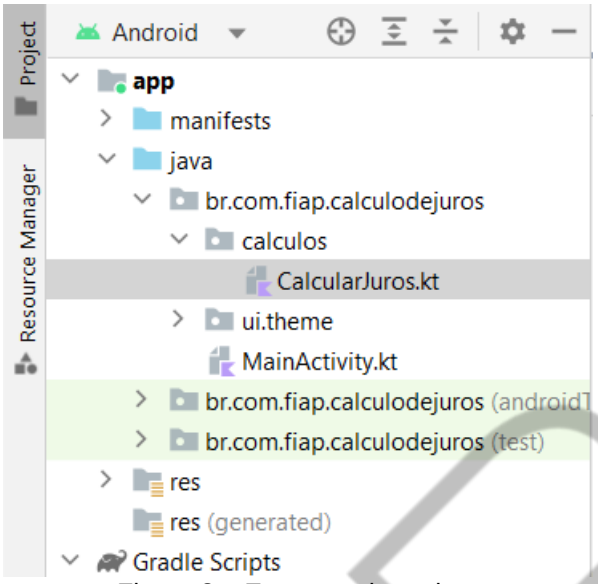

Figura 2 – Estrutura do projeto
Fonte: Elaborado pelo autor (2023)

O arquivo “CalcularJuros.kt” será responsável por efetuar os cálculos de juros e montantes. Para isso, implemente neste arquivo as funções necessárias. O código do arquivo “CalcularJuros.kt” deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros.calculos

fun calcularJuros(capital: Double, taxa: Double, tempo: Double): Double {
  return capital * taxa / 100 * tempo
}

fun calcularMontante(capital: Double, juros: Double): Double {
  return capital + juros
}
```

Código-fonte 2 – Arquivo CalcularJuros.kt
Fonte: Elaborado pelo autor (2023)

Após a inclusão do novo código, execute a aplicação em um emulador. O resultado esperado deverá ser parecido com a figura “Layout do aplicativo Cálculo de Juros”:

Figura 3 - Layout do aplicativo Cálculo de Juros
Fonte: Elaborado pelo autor (2023)

1.1.2 Modularizando a aplicação

Como podemos notar, nossa aplicação ficou muito extensa e isso não é uma boa prática. Devemos modularizar nossa aplicação sempre que possível, pois facilitará a manutenção.

Outra coisa que percebemos também foi a repetição de código. Você percebeu como as caixas de entrada são praticamente iguais? O que muda são, basicamente, os textos e valores.

Ao modularizar a aplicação, podemos dividir a interface do usuário em componentes menores, como botões, listas, cartões, entre outros. Cada componente pode ser projetado de forma independente, com sua própria lógica e aparência específica.

Vamos começar produzindo uma função responsável por criar caixas de entrada personalizadas. Vamos lá!

Crie um pacote em nosso projeto chamado “components”. A estrutura de pastas do projeto deverá se parecer com a figura “Pacote components”:

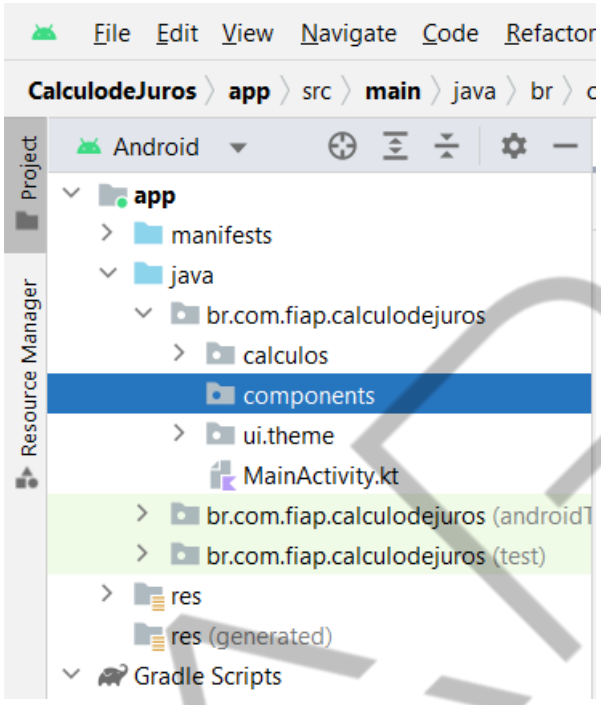

Figura 4 - Pacote "components"
Fonte: Elaborado pelo autor (2023)

Vamos criar um arquivo chamado “CaixaDeEntrada.kt” no pacote “components”. Neste arquivo, iremos criar um “OutlinedTextField” personalizado que poderá ser utilizado sempre que quisermos. E qualquer mudança nesta função será aplicada imediatamente em toda a nossa aplicação.

O arquivo “CaixaDeEntrada.kt” deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros.components

import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp

@Composable
fun CaixaDeEntrada(
  value: String,
  placeholder: String,
  label: String,
  modifier: Modifier,
  keyboardType: KeyboardType
```

```
) {
  OutlinedTextField(
    value = value,
    onValueChange = {},
    modifier = modifier
      .fillMaxWidth()
      .padding(top = 16.dp),
    placeholder = {
      Text(text = placeholder)
    },
    label = {
      Text(text = label)
    },
    keyboardOptions = KeyboardOptions(keyboardType = keyboardType)
  )
}
```

Código-fonte 3 - Arquivo CaixaDeEntrada.kt
Fonte: Elaborado pelo autor (2023)

Vamos utilizar a nossa função “CaixaDeEntrada” em nossa aplicação. Para isso, abra o arquivo “MainActivity.kt”, comente o código da caixa de texto responsável pela digitação do capital e substitua pelo código da listagem a seguir:

```
// Caixas de entrada da Aplicação
CaixaDeEntrada(
  value = "",
  placeholder = "Quanto deseja investir",
  label = "Valor do investimento",
  modifier = Modifier,
  keyboardType = KeyboardType.Decimal
)
```

Código-fonte 4 – Implementação da caixa de entrada capital
Fonte: Elaborado pelo autor (2023)

Notou que a criação da caixa de entrada ficou muito mais limpa?

Execute a aplicação no emulador novamente. A sua IU deve funcionar sem nenhum problema, mas se você tentar digitar irá perceber que nada acontece.

1.1.3 Elevando o estado

O que precisamos entender neste momento é que agora nosso componente está sendo renderizado pela função “CaixaDeEntrada”, e ela será utilizada para renderizar todas as outras. Mas, como iremos controlar o estado e comportamento de cada uma delas? Agora entra a aplicação do “State Hoisting”.

O parâmetro “value” da função “CaixaDeEntrada” deverá receber a variável de estado que está sendo mantida na função “JurosScreen”, que é hierarquicamente

superior, ou seja, é ela que mantém o estado da nossa tela. A função “CaixaDeEntrada” é “stateless”, já que não precisa manter o estado.

Quanto ao comportamento, precisamos passar uma função para a função de composição “CaixaDeEntrada” que será utilizada pelo parâmetro “onValueChange” do “OutlinedTextField”. Isso permitirá que possamos passar comportamentos diferentes para cada caixa de entrada que criarmos. Percebeu as vantagens?

Vamos refatorar a função “CaixaDeEntrada” para que implemente o estado e o comportamento para as caixas de entrada da nossa aplicação. Seu código deverá se parecer com a listagem abaixo:

```
@Composable
fun CaixaDeEntrada(
  value: String,
  placeholder: String,
  label: String,
  modifier: Modifier,
  keyboardType: KeyboardType,
  atualizarValor: (String) -> Unit
) {
  OutlinedTextField(
    value = value,
    onValueChange = {
      atualizarValor(it)
    },
    modifier = modifier
      .fillMaxWidth()
      .padding(top = 16.dp),
    placeholder = {
      Text(text = placeholder)
    },
    label = {
      Text(text = label)
    },
    keyboardOptions = KeyboardOptions(keyboardType = keyboardType)
  )
}
```

Código-fonte 5 – Função lambda para atualizar valor da caixa de entrada
Fonte: Fonte: Elaborado pelo autor (2023)

Agora, devemos passar o parâmetro “atualizarValor” durante a chamada para a função “CaixaDeEntrada”. No arquivo “MainActivity.kt”, o código da caixa de texto responsável pela entrada do capital deverá se parecer com a listagem abaixo:

```
// Caixas de entrada da Aplicação
CaixaDeEntrada(
  value = capital,
  placeholder = "Quanto deseja investir",
  label = "Valor do investimento",
  modifier = Modifier,
```

```
  keyboardType = KeyboardType.Decimal
){
  capital = it
}
```

Código-fonte 6 – Uso da função lambda da função CaixaDeEntrada
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador. Agora você já deverá ser capaz de digitar o valor do capital novamente. Preencha todos os valores e clique no botão calcular; a aplicação deverá funcionar corretamente e calcular os juros e montante.

Vamos alterar agora as outras caixas de texto. Seu código deverá se parecer com a listagem abaixo:

```
// Caixas de entrada da Aplicação
CaixaDeEntrada(
  value = capital,
  placeholder = "Quanto deseja investir",
  label = "Valor do investimento",
  modifier = Modifier,
  keyboardType = KeyboardType.Decimal
){
  capital = it
}
CaixaDeEntrada(
  value = taxa,
  placeholder = "Qual a taxa de juros mensal?",
  label = "Taxa de juros mensal",
  modifier = Modifier,
  keyboardType = KeyboardType.Decimal
){
  taxa = it
}
CaixaDeEntrada(
  value = tempo,
  placeholder = "Qual o período do investimento em meses?",
  label = "Período em meses",
  modifier = Modifier,
  keyboardType = KeyboardType.Decimal
){
  tempo = it
}
```

Código-fonte 7 – Uso da função lambda por todas as caixas de entrada
Fonte: Elaborado pelo autor (2023)

Execute a aplicação e tudo deverá funcionar corretamente.

Neste exemplo fizemos a modularização de um único componente, mas podemos modularizar componentes mais complexos. Vamos agora criar uma função de composição para o card que exibe o resultado do nosso investimento. Então, crie um arquivo no pacote “components” com o nome “CardResultado”. Assim como no

exemplo do “OutlinedTextField”, o estado ficará na função hierarquicamente superior que é a “JurosScreen”. O código do arquivo “CardResultado.kt” deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros.components

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Composable
fun CardResultado(juros: Double, montante: Double) {
  Card(
    modifier = Modifier
      .fillMaxWidth(),
    colors = CardDefaults.cardColors(
      containerColor = Color(0xFF4CAF50)
    )
  ) {
    Column(
      modifier = Modifier
        //.fillMaxSize()
        .padding(16.dp)
    ) {
      Text(
        text = "Resultado",
        fontSize = 18.sp,
        fontWeight = FontWeight.Bold,
        color = Color.White
      )
      Spacer(modifier = Modifier.height(16.dp))
      Row(modifier = Modifier.fillMaxWidth()) {
        Text(
          text = "Juros",
          modifier = Modifier.padding(end = 8.dp),
          fontSize = 16.sp,
          fontWeight = FontWeight.Bold
        )
        Text(
          text = juros.toString(),
          modifier = Modifier.padding(end = 8.dp),
          fontSize = 16.sp,
          fontWeight = FontWeight.Bold,
          color = Color.White
        )
      }
      Spacer(modifier = Modifier.height(8.dp))
```

```
        Row(modifier = Modifier.fillMaxWidth()) {
          Text(
            text = "Montante",
            modifier = Modifier.padding(end = 8.dp),
            fontSize = 16.sp,
            fontWeight = FontWeight.Bold
          )
          Text(
            text = montante.toString(),
            modifier = Modifier.padding(end = 8.dp),
            fontSize = 16.sp,
            fontWeight = FontWeight.Bold,
            color = Color.White
          )
        }
      }
    }
  }
}
```

Código-fonte 8 – Arquivo CardResultado.kt
Fonte: Elaborado pelo autor (2023)

Agora vamos substituir todo o trecho de código responsável por renderizar o card de resultado da “MainActivity.kt” pela chamada da nossa função de composição. O resultado da classe “MainActivity.kt” deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
```

```
import androidx.compose.ui.unit.sp
import br.com.fiap.calculodejuros.calculos.calcularJuros
import br.com.fiap.calculodejuros.calculos.calcularMontante
import br.com.fiap.calculodejuros.components.CaixaDeEntrada
import br.com.fiap.calculodejuros.components.CardResultado
import br.com.fiap.calculodejuros.ui.theme.CalculoDeJurosTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      CalculoDeJurosTheme {
        // A surface container using the 'background' color from the theme
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          JurosScreen()
        }
      }
    }
  }
}

@Composable
fun JurosScreen() {

  var capital by remember { mutableStateOf("") }
  var taxa by remember { mutableStateOf("") }
  var tempo by remember { mutableStateOf("") }
  var juros by remember { mutableStateOf(0.0) }
  var montante by remember { mutableStateOf(0.0) }

  Box(
    modifier = Modifier.padding(16.dp),
    contentAlignment = Alignment.Center
  ) {
    Column {
      Text(
        text = "Cálculo de Juros Simples",
        modifier = Modifier.fillMaxWidth(),
        fontSize = 20.sp,
        color = Color.Red,
        fontWeight = FontWeight.Bold,
        textAlign = TextAlign.Center
      )
      Spacer(modifier = Modifier.height(32.dp))
      // Formulário para entrada de dados
      Card(
        modifier = Modifier
          .fillMaxWidth()
      ) {
        Column(modifier = Modifier.padding(16.dp)) {
          Text(
              text = "Dados do Investimento",
              fontWeight = FontWeight.Bold
          )
          // Caixas de entrada da Aplicação
          CaixaDeEntrada(
            value = capital,
            placeholder = "Quanto deseja investir",
```

```
                label = "Valor do investimento",
                modifier = Modifier,
                keyboardType = KeyboardType.Decimal
            ){
                capital = it
            }
            CaixaDeEntrada(
                value = taxa,
                placeholder = "Qual a taxa de juros mensal?",
                label = "Taxa de juros mensal",
                modifier = Modifier,
                keyboardType = KeyboardType.Decimal
            ){
                taxa = it
            }
            CaixaDeEntrada(
                value = tempo,
                placeholder = "Qual o período do investimento em meses?",
                label = "Período em meses",
                modifier = Modifier,
                keyboardType = KeyboardType.Decimal
            ){
                tempo = it
            }
            Button(
                onClick = {
                    juros = calcularJuros(
                        capital = capital.toDouble(),
                        taxa = taxa.toDouble(),
                        tempo = tempo.toDouble()
                    )
                    montante = calcularMontante(
                        capital = capital.toDouble(),
                        juros = juros
                    )
                },
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(top = 32.dp)
            ) {
                Text(text = "CALCULAR")
            }
        }
    }
    Spacer(modifier = Modifier.height(16.dp))
    // Resultado da aplicação
    CardResultado(juros = juros, montante = montante)
  }
 }
}
```

Código-fonte 9 – Uso da função CardResultado em MainActivity.kt
Fonte: Elaborado pelo autor (2023)

Execute novamente a aplicação no emulador e tudo deverá estar funcionando corretamente.

Neste exemplo aprendemos como modularizar a aplicação em pequenas partes. No início, nossa aplicação era um arquivo com centenas de linhas onde tudo

acontecia em um único lugar. Agora conseguimos separar as funções de modo que cada função tenha um papel específico em nossa aplicação. Que tal criar uma função para componentizar o card com o formulário?

1.2 Model View ViewModel – MVVM

Já sabemos que é possível modularizar a aplicação de modo que cada componente seja uma pequena parte customizável que possuirá seu próprio estado e comportamento, mas precisamos ir um pouco mais fundo e aplicar uma arquitetura que separe as responsabilidades do nosso código de modo a melhorar a manutenção e a escalabilidade da aplicação como um todo; precisamos começar a pensar nas "arquiteturas". E quando pensamos em arquitetura no desenvolvimento Android logo vem a nossa mente o padrão MVVM, *Model-View-ViewModel*.

1.2.1 O que é o padrão MVVM?

O *Model-View-ViewModel* é um padrão arquitetural utilizado no desenvolvimento de aplicações Android, em que o objetivo principal é separar as responsabilidades da aplicação em camadas. O MVVM é organizado em três partes. São elas:

- Model: aqui temos a representação da camada de dados do aplicativo. Essa camada é responsável pelo acesso aos dados em um banco de dados ou requisições através da rede. A Model é a camada responsável por fornecer os dados que serão exibidos pela IU;
- View: esta camada é responsável por exibir os dados ao usuário, além de permitir que o usuário interaja com a aplicação. A *View* é a IU da aplicação.
- ViewModel: esta camada conecta a *View* e a *Model*. A *ViewModel* fornece os dados que serão exibidos pela *View*, assim como processa as entradas de usuário que podem resultar em atualização dos dados na *Model*. A ViewModel também fornece suporte a dados observáveis através do "LiveData", que atualiza os dados da View quando um dado é atualizado.

A comunicação entre as camadas pode ser observada na figura “Arquitetura MVVM”:

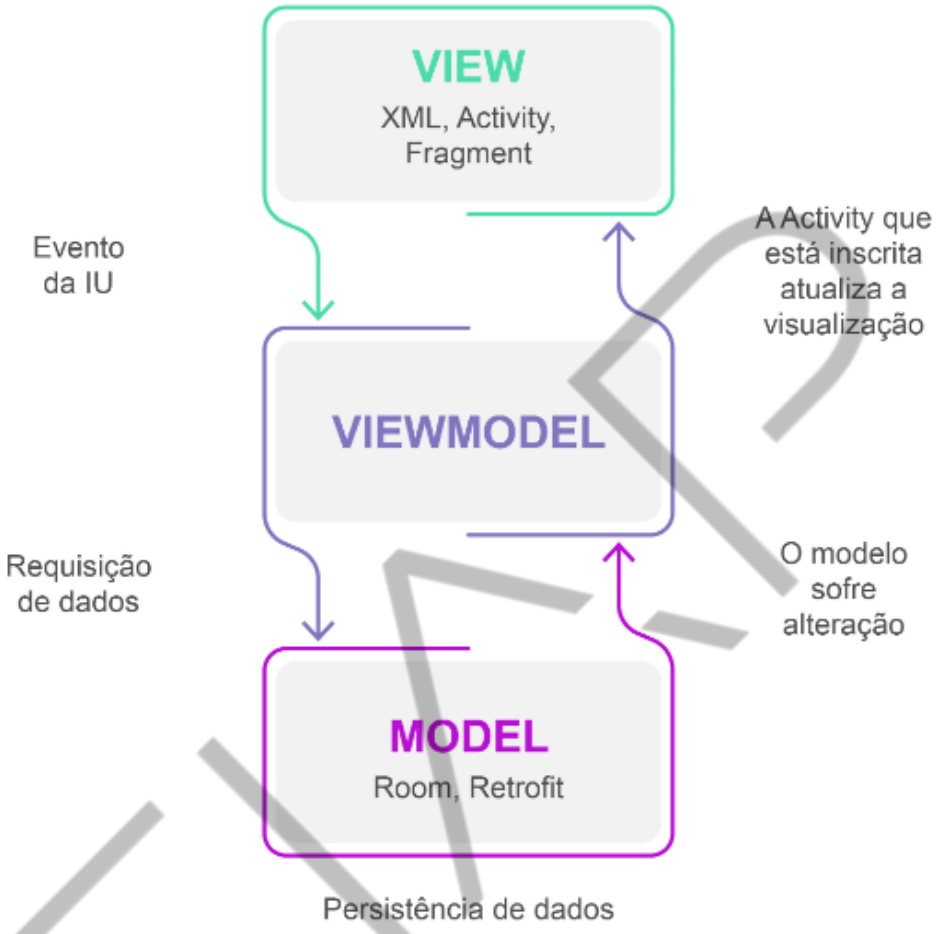

Figura 5 - Arquitetura MVVM
Fonte: Elaborado pelo autor (2023)

1.2.2 Utilizando o padrão MVVM

Para demonstrar a aplicação do padrão MVVM em um projeto Android, iremos refatorar nossa calculadora de juros simples, adotando uma abordagem modular e organizada.

Iniciaremos criando um pacote chamado "juros" que abrigará os arquivos responsáveis pela renderização da interface e pela classe *ViewModel* da tela.

Ao dividir nossa implementação em camadas distintas, obteremos os seguintes benefícios:

Separação de responsabilidades: Com o pacote "juros", teremos um local dedicado para lidar com a lógica de apresentação e a interação com o usuário. A *ViewModel* será responsável por fornecer os dados necessários para a interface, enquanto a tela (*View*) será responsável por exibir esses dados e responder às ações do usuário.

Manutenção facilitada: Com os arquivos organizados em pacotes distintos, será mais fácil localizar, modificar e adicionar novos recursos à calculadora de juros. Cada componente terá uma função bem definida, tornando o código mais legível e de fácil compreensão.

Testes aprimorados: Com a separação clara entre a *ViewModel* e a tela, poderemos testar a lógica de negócios de forma isolada, sem a necessidade de simular interações com a interface do usuário. Isso nos permitirá criar testes unitários eficientes e garantir a qualidade do nosso código.

Após a criação do pacote e dos arquivos, a estrutura do seu projeto deverá se parecer com a figura “Estrutura do projeto MVVM”:

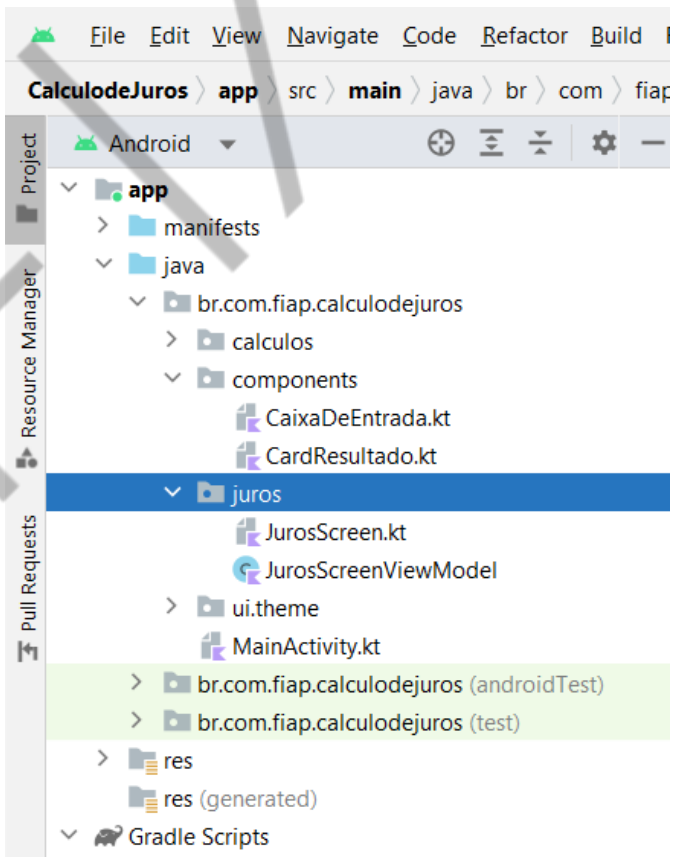

Figura 6 - Estrutura do projeto com MVVM
Fonte: elaborado pelo autor (2023)

Vamos recortar a função “JurosScreen” da classe “MainActivity.kt” e colar no arquivo “JurosScreen.kt”. Seu código deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros.juros

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import br.com.fiap.calculodejuros.calculos.calcularJuros
import br.com.fiap.calculodejuros.calculos.calcularMontante
import br.com.fiap.calculodejuros.components.CaixaDeEntrada
import br.com.fiap.calculodejuros.components.CardResultado

@Composable
fun JurosScreen() {

  var capital by remember { mutableStateOf("") }
  var taxa by remember { mutableStateOf("") }
  var tempo by remember { mutableStateOf("") }
  var juros by remember { mutableStateOf(0.0) }
  var montante by remember { mutableStateOf(0.0) }

  Box(
    modifier = Modifier.padding(16.dp),
    contentAlignment = Alignment.Center
  ) {
    Column {
      Text(
        text = "Cálculo de Juros Simples",
        modifier = Modifier.fillMaxWidth(),
        fontSize = 20.sp,
        color = Color.Red,
        fontWeight = FontWeight.Bold,
        textAlign = TextAlign.Center
      )
      Spacer(modifier = Modifier.height(32.dp))
      // Formulário para entrada de dados
      Card(
        modifier = Modifier
          .fillMaxWidth()
      ) {
```

```
        Column(modifier = Modifier.padding(16.dp)) {
          Text(
            text = "Dados do Investimento",
            fontWeight = FontWeight.Bold
          )
          // Caixas de entrada da Aplicação
          CaixaDeEntrada(
            value = capital,
            placeholder = "Quanto deseja investir",
            label = "Valor do investimento",
            modifier = Modifier,
            keyboardType = KeyboardType.Decimal
          ){
            capital = it
          }
          CaixaDeEntrada(
            value = taxa,
            placeholder = "Qual a taxa de juros mensal?",
            label = "Taxa de juros mensal",
            modifier = Modifier,
            keyboardType = KeyboardType.Decimal
          ){
            taxa = it
          }
          CaixaDeEntrada(
            value = tempo,
            placeholder = "Qual o período do investimento em meses?",
            label = "Período em meses",
            modifier = Modifier,
            keyboardType = KeyboardType.Decimal
          ){
            tempo = it
          }
          Button(
            onClick = {
              juros = calcularJuros(
                capital = capital.toDouble(),
                taxa = taxa.toDouble(),
                tempo = tempo.toDouble()
              )
              montante = calcularMontante(
                capital = capital.toDouble(),
                juros = juros
              )
            },
            modifier = Modifier
              .fillMaxWidth()
              .padding(top = 32.dp)
          ) {
            Text(text = "CALCULAR")
          }
        }
      }
      Spacer(modifier = Modifier.height(16.dp))
      // Resultado da aplicação
      CardResultado(juros = juros, montante = montante)
    }
  }
}
```

Código-fonte 10 – Arquivo JurosScreen.kt
Fonte: Elaborado pelo autor (2023)

Na classe “MainActivity.kt”, devemos importar a função “JurosScreen”. O código da classe “MainActivity.kt” deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import br.com.fiap.calculodejuros.juros.JurosScreen
import br.com.fiap.calculodejuros.ui.theme.CalculoDeJurosTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      CalculoDeJurosTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          JurosScreen()
        }
      }
    }
  }
}
```

Código-fonte 11 – Uso da função JurosScreen pela classe MainActivity
Fonte: Elaborado pelo autor (2023)

Após os ajustes, execute o aplicativo em um emulador. Tudo deverá funcionar corretamente.

O arquivo “JurosScreen.kt” representa a tela da nossa aplicação, ou seja, uma “feature” que é responsável por renderizar a tela com o formulário da nossa aplicação. Se nossa aplicação tiver uma tela de login, por exemplo, criaríamos um pacote chamado “login” para armazenar a *feature* login, além da *ViewModel* para a tela de login.

O arquivo “JurosScreenViewModel” será a classe responsável por toda a lógica da nossa tela “JurosScreen”, ou seja, esta classe é a detentora do estado e da lógica de negócios.

A classe “JurosScreenViewModel” deve estender a classe “ViewModel”. Assim, seu código deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros.juros

import androidx.lifecycle.ViewModel

class JurosScreenViewModel: ViewModel() {

}
```

Código-fonte 12 – JurosScreenViewModel herdando ViewModel
Fonte: Elaborado pelo autor (2023)

A utilização desta classe é possível por conta da dependência da biblioteca “lifecycle-runtime” que já está configurada no arquivo “build.gradle” em nível de módulo. O conteúdo deste arquivo na sessão “dependencies” deve se parecer com a listagem abaixo:

```
dependencies {
    implementation 'androidx.core:core-ktx:1.8.0'
    implementation platform('org.jetbrains.kotlin:kotlin-bom:1.8.0')
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
    implementation 'androidx.compose.material3:material3:1.1.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation platform('androidx.compose:compose-
bom:2022.10.00')
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'
}
```

Código-fonte 13 – Dependência da biblioteca lifecycle-runtime
Fonte: Elaborado pelo autor (2023)

Se por algum motivo você não tiver esta configuração, basta acrescentá-la e sincronizar o projeto.

1.2.3. Gerenciando o estado da aplicação com LiveData

O LiveData é uma classe que armazena dados observáveis, sendo assim, os atributos que estão observando um atributo do tipo “LiveData” serão notificados sempre que o valor de um “LiveData” for alterado.

Deste modo, na classe “JurosScreenViewModel” criaremos os atributos observáveis, e na função “JurosScreen” criaremos os atributos observadores e assim, vamos manter o estado na “ViewModel” e não mais na tela.

Para utilizarmos o “LiveData”, precisamos adicionar uma dependência no arquivo “build.gradle” em nível de módulo. A sessão “dependencies” do arquivo “build.gradle” deve se parecer com a listagem abaixo:

```
dependencies {
    implementation 'androidx.core:core-ktx:1.8.0'
    implementation platform('org.jetbrains.kotlin:kotlin-bom:1.8.0')
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
    implementation 'androidx.compose.material3:material3:1.1.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation platform('androidx.compose:compose-
bom:2022.10.00')
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'

    // Dependência do LiveData
    implementation "androidx.compose.runtime:runtime-livedata:1.4.3"
}
```

Código-fonte 14 – Inclusão da dependência da biblioteca LiveData
Fonte: Elaborado pelo autor (2023)

Não se esqueça de sincronizar a aplicação clicando em “Sync Now”, conforme a figura “Sincronizando o Gradle”:

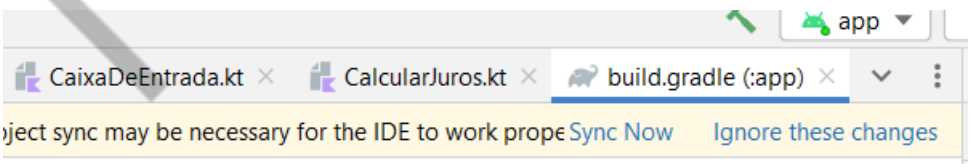

Figura 7 - Sincronizando o Gradle
Fonte: Elaborado pelo autor (2023)

Vamos começar criando um observável e um observador para o atributo “capital”. O observável deve ser criado na classe “JurosScreenViewModel”, que deverá se parecer com a listagem abaixo:

```
package br.com.fiap.calculodejuros.juros

import androidx.lifecycle.LiveData
```

```
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class JurosScreenViewModel: ViewModel() {

  private val _capital = MutableLiveData<String>()
  val capital: LiveData<String> = _capital

}
```

Código-fonte 15 – Criação do atributo observável capital na ViewModel
Fonte: Elaborado pelo autor (2023)

No código acima, criamos um atributo privado, que só pode ser acessado da "ViewModel" chamado "_capital". Este atributo é do tipo "MutableLiveData", ou seja, seu valor é mutável. Observe que utilizamos o caractere underscore (_) antes do nome do atributo, que é uma convenção adotada para indicarmos que este atributo é privado e que não deve ser acessado de outras classes. Essa convenção é chamada de "underscore prefix" ou "underscore notation".

O atributo "capital" foi definido de forma pública, ou seja, é através dele que os observadores saberão que a variável teve seu valor alterado. Note que ele recebe o valor do atributo "_capital".

A função "JurosScreen" é a nossa função principal, ou seja, ela é responsável por manter o estado da nossa tela, então, é necessário que ela receba um parâmetro com a sua "ViewModel" que é a "JurosScreenViewModel". Altere a função "JurosScreen" de modo que ela se pareça com a listagem abaixo:

```
@Composable
fun JurosScreen(jurosScreenViewModel: JurosScreenViewModel) {

  var capital by remember { mutableStateOf("") }
  var taxa by remember { mutableStateOf("") }

  … trecho de Código omitido
```

Código-fonte 16 – Criação do parâmetro jurosScreenViewModel
Fonte: Elaborado pelo autor (2023)

Após este ajuste, começaremos a receber uma notificação de erro na classe "MainActivity", que é responsável por chamar a função "JurosScreen". Isso ocorre devido ao fato da função "JurosScreen" pedir um parâmetro do tipo "JurosScreenViewModel". Vamos abrir o arquivo "MainActivity.kt" e passar o parâmetro durante a chamada da nossa função. Seu código deverá se parecer com a listagem abaixo:

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      CalculoDeJurosTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          JurosScreen(JurosScreenViewModel())
        }
      }
    }
  }
}
```

Código-fonte 17 – Passagem do parâmetro JurosScreenViewModel
Fonte: Elaborado pelo autor (2023)

Agora, vamos alterar a declaração do atributo “capital” da função “JurosScreen” para que utilize o “LiveData”. Seu código deverá se parecer com a listagem abaixo:

```
@Composable
fun JurosScreen(jurosScreenViewModel: JurosScreenViewModel) {

//  var capital by remember { mutableStateOf("") }
  val capital by jurosScreenViewModel
    .capital
    .observeAsState(initial = "")
```

Código-fonte 18 – Criação da variável observadora capital.
Fonte: Elaborado pelo autor (2023)

No código apresentado, temos a declaração de uma variável chamada “capital” que é inicializada com o valor observado do estado “capitalState” da função “JurosScreenViewModel”, referenciada pelo parâmetro “jurosScreenViewModel”. A função “observeAsState” é utilizada para criar um estado observável a partir do valor do estado “capitalState” e atribuí-lo à variável “capital”.

A função “observeAsState” permite observar e acompanhar as mudanças de um estado ao longo do tempo. Ele cria um estado que pode ser utilizado dentro de um *composable* para atualizar a interface do usuário automaticamente sempre que o valor do estado observado sofrer alterações.

Dessa forma, sempre que houver uma alteração no valor do estado “capitalState” em “JurosScreenViewModel”, o estado “capital” será atualizado e o *composable* que utiliza esse estado será recomposto, refletindo as alterações na interface do usuário. Isso proporciona uma atualização automática da tela com base nas mudanças de estado, sem a necessidade de manipulação manual ou repetitiva.

Após esse ajuste, vamos obter mais uma mensagem de erro. Lembre-se que a função “CaixaDeEntrada” possui uma função lambda como parâmetro que é usada para atualizar o atributo “value” do composable “OutlinedTextField”. Isso ocorre porque não podemos mudar o valor do atributo “capital” diretamente. Quem deve fazer isso é a “ViewModel”, então, vamos criar uma função em “JurosScreenViewModel” que será responsável por atualizar o estado do atributo “capitalState” e consequentemente do atributo “capital” da função “JurosScreen”, que é o observador. Abra o arquivo “JurosScreenViewModel” e acrescente a função responsável por atualizar v valor de “capitalState”. Seu código deverá se parecer com a listagem abaixo:

```
class JurosScreenViewModel: ViewModel() {

  private val _capital = MutableLiveData<String>()
  val capitalState: LiveData<String> = _capital

  fun onCapitalChanged(novoCapital: String){
    _capital.value = novoCapital
  }

}
```

Código-fonte 19 – Função onCapitalChanged
Fonte: Elaborado pelo autor (2023)

A função “onCapitalChanged” recebe o parâmetro “novoCapital” do tipo String, que será atribuído ao objeto “MutableLiveData” referenciado pela variável “_capital”, que por sua vez irá atualizar o valor do atributo “capitalState” que está sendo observado pelo atributo “capital” da função “JurosScreen”. Agora, vamos alterar a chamada para a função “CaixaDeEntrada” responsável pela entrada do capital. Seu código deverá se parecer com a listagem abaixo:

```
CaixaDeEntrada(
  value = capital,
  placeholder = "Quanto deseja investir",
  label = "Valor do investimento",
  modifier = Modifier,
  keyboardType = KeyboardType.Decimal
){
  jurosScreenViewModel.onCapitalChanged(it)
}
```

Código-fonte 20 – Utilização da função onCapitalChanged
Fonte: Elaborado pelo autor (2023)

Rode a aplicação em um emulador e verifique se a aplicação continua funcionando corretamente.

Vamos modificar as variáveis “taxa” e “tempo” da função “JurosScreen” para que utilizem a “ViewModel”. O código da “JurosScreenViewModel” deverá se parecer com a listagem abaixo:

```
class JurosScreenViewModel: ViewModel() {

  private val _capital = MutableLiveData<String>()
  val capitalState: LiveData<String> = _capital

  private val _taxa = MutableLiveData<String>()
  val taxaState: LiveData<String> = _taxa

  private val _tempo = MutableLiveData<String>()
  val tempoState: LiveData<String> = _tempo

  fun onCapitalChanged(novoCapital: String){
    _capital.value = novoCapital
  }

  fun onTaxaChanged(novaTaxa: String){
    _taxa.value = novaTaxa
  }

  fun onTempoChanged(novoTempo: String) {
    _tempo.value = novoTempo
  }

}
```

Código-fonte 21 – Acréscimo dos atributos taxa e tempo na ViewModel
Fonte: Elaborado pelo autor (2023)

E, finalmente, vamos ajustar os atributos de estado na função “JurosScreen”. Seu código deverá se parecer com a listagem abaixo:

```
@Composable
fun JurosScreen(jurosScreenViewModel: JurosScreenViewModel) {

//  var capital by remember { mutableStateOf("") }
  val capital by jurosScreenViewModel
    .capitalState
    .observeAsState(initial = "")

//  var taxa by remember { mutableStateOf("") }
  val taxa by jurosScreenViewModel
    .taxaState
    .observeAsState(initial = "")

//  var tempo by remember { mutableStateOf("") }
  val tempo by jurosScreenViewModel
    .tempoState
    .observeAsState(initial = "")
```

```
. . . trecho omitido
```

Código-fonte 22 – Criação dos observadores taxa e tempo em JurosScreen
Fonte: Elaborado pelo autor (2023)

Não podemos nos esquecer de ajustar a chamada para as funções “CaixaDeEntrada” que estão renderizando a entrada do usuário. No arquivo “JurosScreen”, seu código deve se parecer com a listagem abaixo:

```
. . . trecho omitido
CaixaDeEntrada(
  value = taxa,
  placeholder = "Qual a taxa de juros mensal?",
  label = "Taxa de juros mensal",
  modifier = Modifier,
  keyboardType = KeyboardType.Decimal
){
  jurosScreenViewModel.onTaxaChanged(it)
}
CaixaDeEntrada(
  value = tempo,
  placeholder = "Qual o período do investimento em meses?",
  label = "Período em meses",
  modifier = Modifier,
  keyboardType = KeyboardType.Decimal
){
  jurosScreenViewModel.onTempoChanged(it)
}
. . . trecho omitido
```

Código-fonte 23 – Atualização das caixas de entrada em JurosScreen
Fonte: Elaborado pelo autor (2023)

Precisamos agora refatorar o código do botão responsável por efetuar o cálculo dos juros e montante. O código atual deve se parecer com a listagem a seguir:

```
Button(
  onClick = {
    juros = calcularJuros(
      capital = capital.toDouble(),
      taxa = taxa.toDouble(),
      tempo = tempo.toDouble()
    )
    montante = calcularMontante(
      capital = capital.toDouble(),
      juros = juros
    )
  },
  modifier = Modifier
    .fillMaxWidth()
    .padding(top = 32.dp)
) {
  Text(text = "CALCULAR")
}
```

Código-fonte 24 – Botão calcular sem uso da ViewModel

Fonte: Elaborado pelo autor (2023)

Como podemos observar, a lógica de negócio para efetivação dos cálculos está sendo implementada na função “JurosScreen”, e, apesar de ser um código simples, devemos nos lembrar que a lógica de negócios também deve ser implementada na “ViewModel”, portanto, faremos os ajustes necessários começando pela “ViewModel”. Então, abra o arquivo “JurosScreenViewModel” e implemente as alterações conforme a listagem abaixo:

```
class JurosScreenViewModel: ViewModel() {

  private val _capital = MutableLiveData<String>()
  val capitalState: LiveData<String> = _capital

  private val _taxa = MutableLiveData<String>()
  val taxaState: LiveData<String> = _taxa

  private val _tempo = MutableLiveData<String>()
  val tempoState: LiveData<String> = _tempo

  private val _juros = MutableLiveData<Double>()
  val jurosState: LiveData<Double> = _juros

  private val _montante = MutableLiveData<Double>()
  val montanteState: LiveData<Double> = _montante

  fun onCapitalChanged(novoCapital: String){
    _capital.value = novoCapital
  }

  fun onTaxaChanged(novaTaxa: String){
    _taxa.value = novaTaxa
  }

  fun onTempoChanged(novoTempo: String) {
    _tempo.value = novoTempo
  }

  fun calcularJurosInvestimento(){
    _juros.value = calcularJuros(
      capital = _capital.value!!.toDouble(),
      taxa = _taxa.value!!.toDouble(),
      tempo = _tempo.value!!.toDouble()
    )
  }

  fun calcularMontanteInvestimento(){
    _montante.value = calcularMontante(
      _capital.value!!.toDouble(),
      _juros.value!!.toDouble()
    )
  }
}
```

Código-fonte 25 – Implementação de regra de negócio na ViewModel
Fonte: Elaborado pelo autor (2023)

Com a “ViewModel” refatorada, vamos ajustar a função “JurosScreen”. A declaração das variáveis “juros” e “montante” deve ser ajustada conforme a listagem abaixo:

```
. . . trecho omitido
//  var juros by remember { mutableStateOf(0.0) }
  val juros by jurosScreenViewModel
    .jurosState
    .observeAsState(initial = 0.0)

//  var montante by remember { mutableStateOf(0.0) }
  val montante by jurosScreenViewModel
    .montanteState
    .observeAsState(initial = 0.0)
. . . trecho omitido
```

Código-fonte 26 – Criação dos observadores para juros e montante
Fonte: Elaborado pelo autor (2023)

Finalmente, o evento de clique do botão “Calcular” deve se parecer com a listagem abaixo:

```
Button(
  onClick = {
    jurosScreenViewModel.calcularJurosInvestimento()
    jurosScreenViewModel.calcularMontanteInvestimento()
  },
  modifier = Modifier
    .fillMaxWidth()
    .padding(top = 32.dp)
) {
  Text(text = "CALCULAR")
}
```

Código-fonte 27 – Clique do botão calcular utilizando a ViewModel
Fonte: Elaborado pelo autor (2023)

Após todos os ajustes, execute a aplicação em um emulador, faça os testes necessários e verifique se tudo está funcionando corretamente.

CONCLUSÃO

A utilização de padrões de projeto desempenha um papel fundamental no desenvolvimento de aplicações Android, proporcionando benefícios significativos em termos de manutenção e escalabilidade. Esses padrões nos orientam na criação de aplicações que podem aproveitar efetivamente a reutilização de código.

Um dos padrões arquiteturais mais amplamente adotados no desenvolvimento Android é o Model-View-ViewModel (MVVM). Ele oferece uma abordagem que promove o desacoplamento entre a interface do usuário, o estado e a lógica de negócios. No MVVM, a camada ViewModel é responsável por gerenciar o estado e a lógica, enquanto a View representa a interface do usuário. Além disso, a camada Model é responsável por fornecer o acesso aos dados utilizados pelo aplicativo.

A adoção do padrão MVVM traz diversos benefícios. Ele permite a criação de interfaces de usuário mais modularizadas e testáveis, facilitando a manutenção e a evolução do código. O desacoplamento entre a interface do usuário e a lógica de negócios também torna mais simples a adição de novos recursos e a personalização da interface.

Além disso, o MVVM incentiva a separação de responsabilidades, melhorando a legibilidade e a organização do código. Ele promove uma arquitetura mais limpa e facilita a colaboração entre membros da equipe de desenvolvimento.

REFERÊNCIAS

DEVELOPERS. **Onde elevar o estado**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/state-hoisting?hl=pt-br>. Acesso em: 04 jul. 2023.

DEVELOPERS: **Visão Geral do LiveData**. 2023. Disponível em: <https://developer.android.com/topic/libraries/architecture/livedata>. Acesso em: 04 jul. 2023.

DEVELOPERS. **Visão Geral do ViewModel**. 2023. Disponível em: <https://developer.android.com/topic/libraries/architecture/viewmodel?hl=pt-br>. Acesso em: 04 jul. 2023.