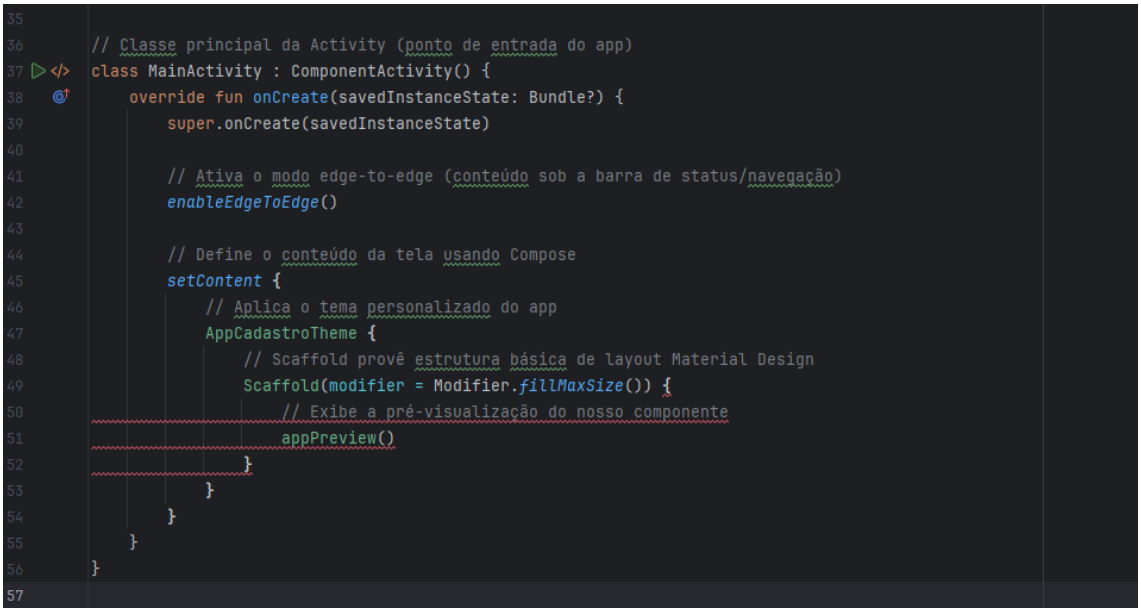
MainActivity

MainActivity.kt

```kotlin
// Declaração do pacote (organização do código)
package com.example.appcadastro

// Importações de bibliotecas Android e Compose necessárias
import android.os.Bundle  // Para manipulação de estado da Activity
import android.widget.Toast  // Para mostrar mensagens popup
import androidx.activity.ComponentActivity  // Classe base para Activities
import androidx.activity.compose.setContent  // Para definir conteúdo com Compose
import androidx.activity.enableEdgeToEdge  // Para layout edge-to-edge
import androidx.compose.foundation.Image  // Componente de imagem
import androidx.compose.foundation.background  // Para cor de fundo
import androidx.compose.foundation.layout.*  // Todos os layouts (Column, Row, etc)
import androidx.compose.foundation.rememberScrollState  // Para scroll
import androidx.compose.foundation.shape.CircleShape  // Forma circular
import androidx.compose.foundation.text.KeyboardOptions  // Opções de teclado
import androidx.compose.foundation.verticalScroll  // Scroll vertical
import androidx.compose.material3.*  // Componentes Material Design 3
import androidx.compose.runtime.*  // Para estado e efeitos
import androidx.compose.ui.Alignment  // Alinhamento de componentes
import androidx.compose.ui.Modifier  // Modificadores de estilo
import androidx.compose.ui.draw.clip  // Para cortar elementos
import androidx.compose.ui.graphics.Brush  // Para gradientes
import androidx.compose.ui.layout.ContentScale  // Escala de imagem
import androidx.compose.ui.platform.LocalContext  // Contexto Android
import androidx.compose.ui.res.painterResource  // Para carregar imagens
import androidx.compose.ui.text.font.FontFamily  // Família de fontes
import androidx.compose.ui.text.font.FontWeight  // Peso da fonte
import androidx.compose.ui.text.input.KeyboardType  // Tipo de teclado
import androidx.compose.ui.unit.dp  // Unidade de medida independente de densidade
import androidx.compose.ui.unit.sp  // Unidade de medida para texto
import androidx.compose.ui.tooling.preview.Preview  // Para pré-visualização
import com.example.appcadastro.ui.theme.AppCadastroTheme  // Tema personalizado
import com.example.appcadastro.ui.theme.Darkblue  // Cor do tema
import com.example.appcadastro.ui.theme.Lightblue  // Cor do tema
```

Este bloco inicial representa os fundamentos necessários para construir a tela do aplicativo. A primeira linha com `package` define o local exato onde este arquivo está organizado dentro do projeto, seguindo a convenção padrão de desenvolvimento Android. Já as diversas declarações `import` funcionam como uma lista de ferramentas essenciais que serão utilizadas: desde componentes básicos como `Bundle` para gerenciar estados e `Toast` para exibir mensagens rápidas, até elementos de interface modernos do Jetpack Compose como `Image` para imagens, `Button` para botões interativos e `Column`/`Row` para organizar o layout. Também estão presentes recursos de estilização como `Modifier` para ajustes visuais, `clip` para formatar elementos e `AppCadastroTheme` que define a identidade visual do aplicativo com suas cores personalizadas. Esta seção não executa nenhuma ação visível, mas prepara todo o ambiente necessário para a construção da interface que será definida nas próximas partes do código.

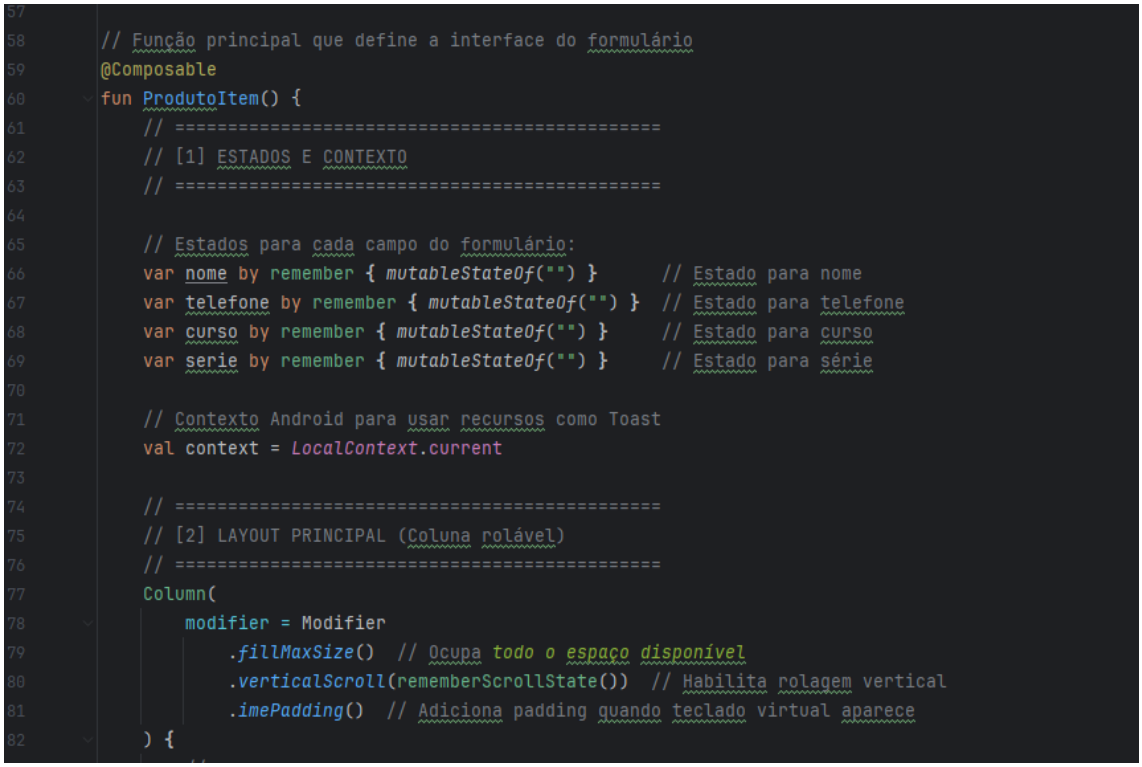


```
// Classe principal da Activity (ponto de entrada do app)
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Ativa o modo edge-to-edge (conteúdo sob a barra de status/navegação)
        enableEdgeToEdge()

        // Define o conteúdo da tela usando Compose
        setContent {
            // Aplica o tema personalizado do app
            AppCadastroTheme {
                // Scaffold provê estrutura básica de layout Material Design
                Scaffold(modifier = Modifier.fillMaxSize()) {
                    // Exibe a pré-visualização do nosso componente
                    appPreview()
                }
            }
        }
    }
}
```

A classe MainActivity representa a tela inicial do aplicativo, funcionando como ponto de entrada quando o usuário abre o programa. Ela estende ComponentActivity, que fornece a estrutura básica para atividades no Android moderno. O método onCreate é acionado automaticamente ao iniciar a tela e realiza três configurações essenciais: primeiro, ativa o modo edge-to-edge através do enableEdgeToEdge(), permitindo que o conteúdo se estenda por toda a área disponível da tela, incluindo as regiões abaixo das barras de sistema. Em seguida, o bloco setContent define a interface usando Jetpack Compose, onde o AppCadastroTheme aplica consistentemente as cores e estilos personalizados em todos os componentes. Por fim, o Scaffold serve como container principal, seguindo os princípios do Material Design para organizar os elementos visuais de forma padronizada, enquanto o modificador fillMaxSize() garante que o layout ocupe todo o espaço disponível. Essa estrutura assegura que o aplicativo tenha uma base sólida e moderna, com aproveitamento otimizado do espaço da tela e consistência visual em todos os dispositivos Android.



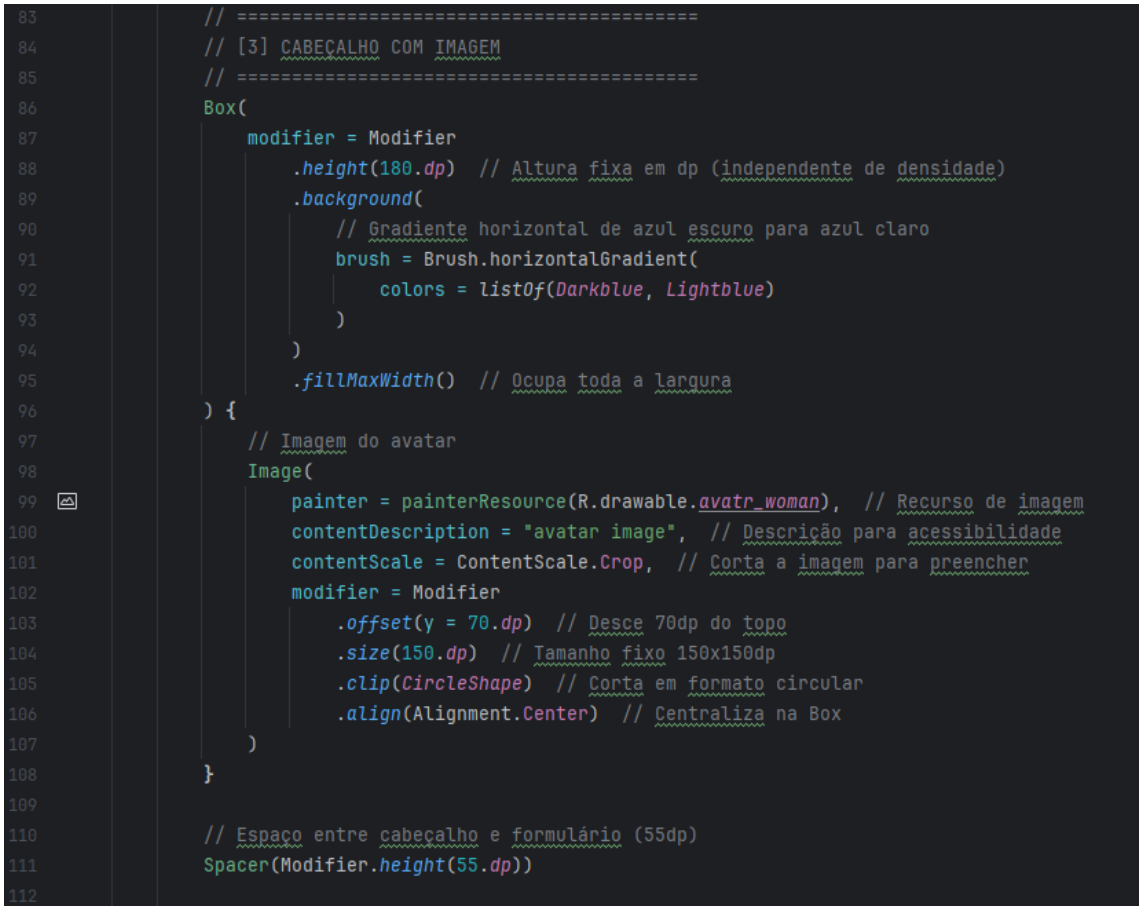
```
// Função principal que define a interface do formulário
@Composable
fun ProdutoItem() {
    // ================================================
    // [1] ESTADOS E CONTEXTO
    // ================================================

    // Estados para cada campo do formulário:
    var nome by remember { mutableStateOf("") }      // Estado para nome
    var telefone by remember { mutableStateOf("") }  // Estado para telefone
    var curso by remember { mutableStateOf("") }     // Estado para curso
    var serie by remember { mutableStateOf("") }     // Estado para série

    // Contexto Android para usar recursos como Toast
    val context = LocalContext.current

    // ================================================
    // [2] LAYOUT PRINCIPAL (Coluna rolável)
    // ================================================
    Column(
        modifier = Modifier
            .fillMaxSize()  // Ocupa todo o espaço disponível
            .verticalScroll(rememberScrollState())  // Habilita rolagem vertical
            .imePadding()  // Adiciona padding quando teclado virtual aparece
    ) {
```

A função ProdutoItem implementa um formulário dinâmico utilizando os princípios do Jetpack Compose. Na primeira seção, são declarados quatro estados reativos para armazenar os valores dos campos (nome, telefone, curso e série), utilizando mutableStateOf combinado com remember - essa abordagem garante que os dados digitados pelo usuário sejam preservados durante as recomposições e automaticamente atualizem a interface quando modificados. O contexto Android é obtido através de LocalContext.current para permitir a exibição de notificações do sistema. A segunda seção define a estrutura visual através de uma Column principal que organiza os elementos verticalmente, com três modificadores essenciais: fillMaxSize para ocupar todo o espaço disponível, verticalScroll para habilitar rolagem quando o conteúdo exceder o tamanho da tela, e imePadding que ajusta automaticamente o layout quando o teclado virtual é exibido, prevenindo que campos fiquem ocultos. Essa implementação segue as melhores práticas do desenvolvimento moderno com Compose, oferecendo uma experiência de usuário fluida e responsiva.



```
// ========================================
// [3] CABEÇALHO COM IMAGEM
// ========================================
Box(
    modifier = Modifier
        .height(180.dp)  // Altura fixa em dp (independente de densidade)
        .background(
            // Gradiente horizontal de azul escuro para azul claro
            brush = Brush.horizontalGradient(
                colors = listOf(Darkblue, Lightblue)
            )
        )
        .fillMaxWidth()  // Ocupa toda a largura
) {
    // Imagem do avatar
    Image(
        painter = painterResource(R.drawable.avatr_woman),  // Recurso de imagem
        contentDescription = "avatar image",  // Descrição para acessibilidade
        contentScale = ContentScale.Crop,  // Corta a imagem para preencher
        modifier = Modifier
            .offset(y = 70.dp)  // Desce 70dp do topo
            .size(150.dp)  // Tamanho fixo 150x150dp
            .clip(CircleShape)  // Corta em formato circular
            .align(Alignment.Center)  // Centraliza na Box
    )
}

// Espaço entre cabeçalho e formulário (55dp)
Spacer(Modifier.height(55.dp))
```

A seção de cabeçalho foi cuidadosamente projetada para criar uma primeira impressão visual impactante. O componente principal é uma `Box` que define uma área com altura fixa de 180dp, ocupando toda a largura disponível. Seu plano de fundo apresenta um gradiente horizontal elegante que transiciona suavemente do azul escuro para o azul claro, estabelecendo a identidade visual do aplicativo. Sobre este gradiente, posiciona-se estrategicamente uma imagem de avatar circular, que recebe tratamento especial: carregada a partir dos recursos do projeto, a imagem é recortada em formato circular através do modificador `clip`, dimensionada para 150dp e posicionada 70dp abaixo do topo, criando um efeito visual equilibrado. A propriedade `contentScale` garante que a imagem preencha adequadamente o espaço designado, enquanto a `contentDescription` atende aos requisitos de acessibilidade. Para finalizar, um espaçador de 55dp é adicionado, proporcionando uma transição harmoniosa entre o cabeçalho e os elementos subsequentes do formulário. Esta implementação exemplifica a atenção aos detalhes de design e usabilidade, combinando estética e funcionalidade de maneira coesa.

```
        // ========================================
        // [4] FORMULÁRIO DE CADASTRO
        // ========================================
        Column(Modifier.padding(16.dp)) {  // Padding interno de 16dp
            // Título do formulário
            Text(
                text = "Cadastre-se",
                fontFamily = FontFamily.SansSerif,  // Fonte sem serifa
                fontWeight = FontWeight(500),  // Peso médio (500)
                fontSize = 35.sp,  // Tamanho em sp (escala com preferências do usuário)
            )

            // Espaço após título (25dp)
            Spacer(Modifier.height(25.dp))

            // --- CAMPO NOME ---
            Text("Nome:", fontSize = 25.sp, fontWeight = FontWeight(250))
            Spacer(Modifier.height(15.dp))  // Espaço após label
            TextField(
                value = nome,
                onValueChange = { nome = it },  // Atualiza estado quando texto muda
                label = { Text("Digite seu nome completo") },  // Texto de hint
                maxLines = 2  // Máximo de 2 linhas
            )

            Spacer(Modifier.height(20.dp))  // Espaço entre campos

            // --- CAMPO TELEFONE ---
            Text("Telefone:", fontSize = 25.sp, fontWeight = FontWeight(250))
            Spacer(Modifier.height(15.dp))
            TextField(
                value = telefone,
                onValueChange = { telefone = it },
                label = { Text("Digite o telefone...") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number)  // Teclado numérico
            )

            Spacer(Modifier.height(20.dp))
```

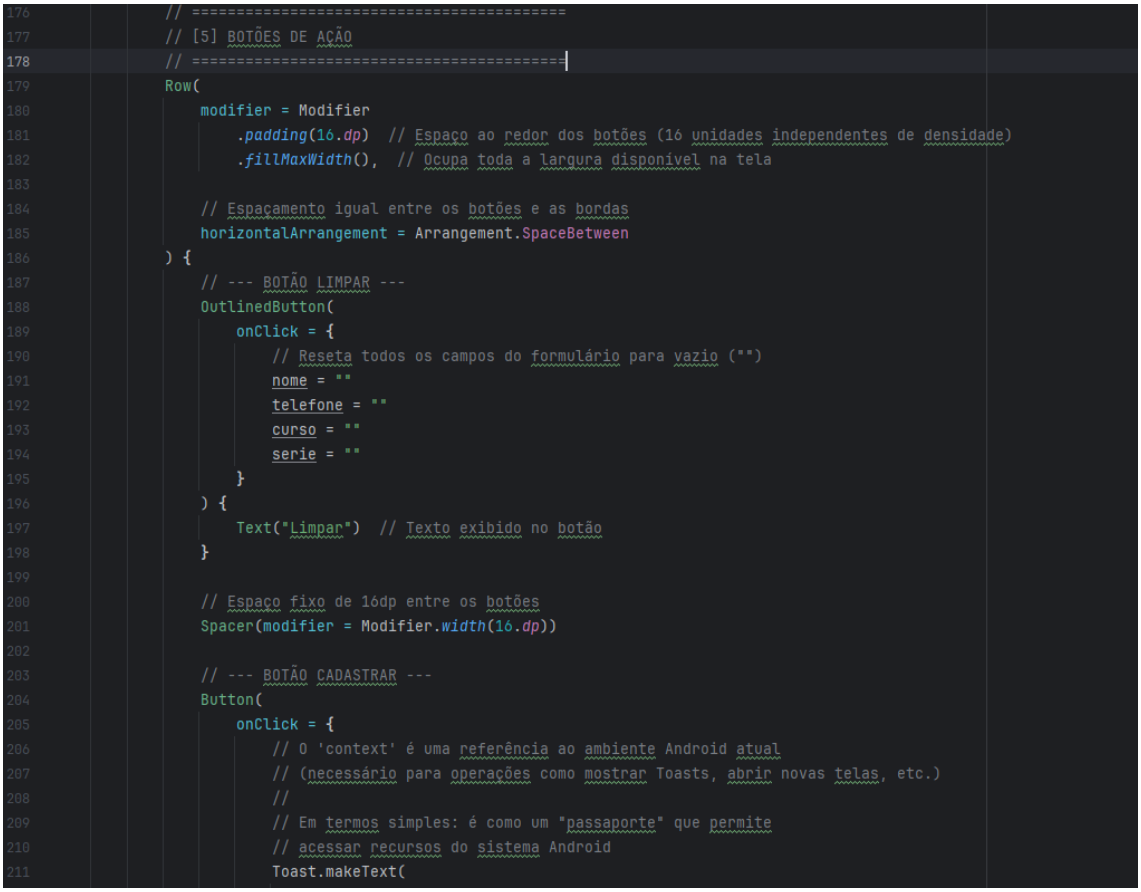
```
            Spacer(Modifier.height(20.dp))

            // --- CAMPO CURSO ---
            Text("Curso:", fontSize = 25.sp, fontWeight = FontWeight(250))
            Spacer(Modifier.height(15.dp))
            TextField(
                value = curso,
                onValueChange = { curso = it },
                label = { Text("Digite o nome do curso") },
                maxLines = 2
            )

            Spacer(Modifier.height(20.dp))

            // --- CAMPO SÉRIE ---
            Text("Série:", fontSize = 25.sp, fontWeight = FontWeight(250))
            Spacer(Modifier.height(15.dp))
            TextField(
                value = serie,
                onValueChange = { serie = it },
                label = { Text("Digite o número da sua série") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
                maxLines = 2
            )
        }
```

A seção de cadastro é organizada em uma estrutura vertical clara e bem espaçada, começando com o título principal "Cadastre-se" em fonte Sans Serif de tamanho 35sp, que se adapta às configurações de acessibilidade do usuário. Cada campo segue um padrão consistente: primeiro aparece o rótulo do campo (como "Nome:" ou "Telefone:") em tamanho 25sp, seguido por um espaço de 15dp e depois o campo de texto editável. Os campos de texto possuem dicas internas (hints) que desaparecem automaticamente quando o usuário começa a digitar. Para melhorar a experiência de preenchimento, os campos numéricos como telefone e série exibem automaticamente o teclado numérico, enquanto os campos de texto comum permitem até duas linhas de entrada quando necessário. Entre cada grupo de campo, um espaçamento generoso de 20dp cria uma separação visual clara, tornando o formulário mais legível e organizado. Todos os campos estão conectados aos respectivos estados da interface, atualizando-se instantaneamente conforme o usuário digita, o que garante uma experiência de preenchimento fluida e responsiva.

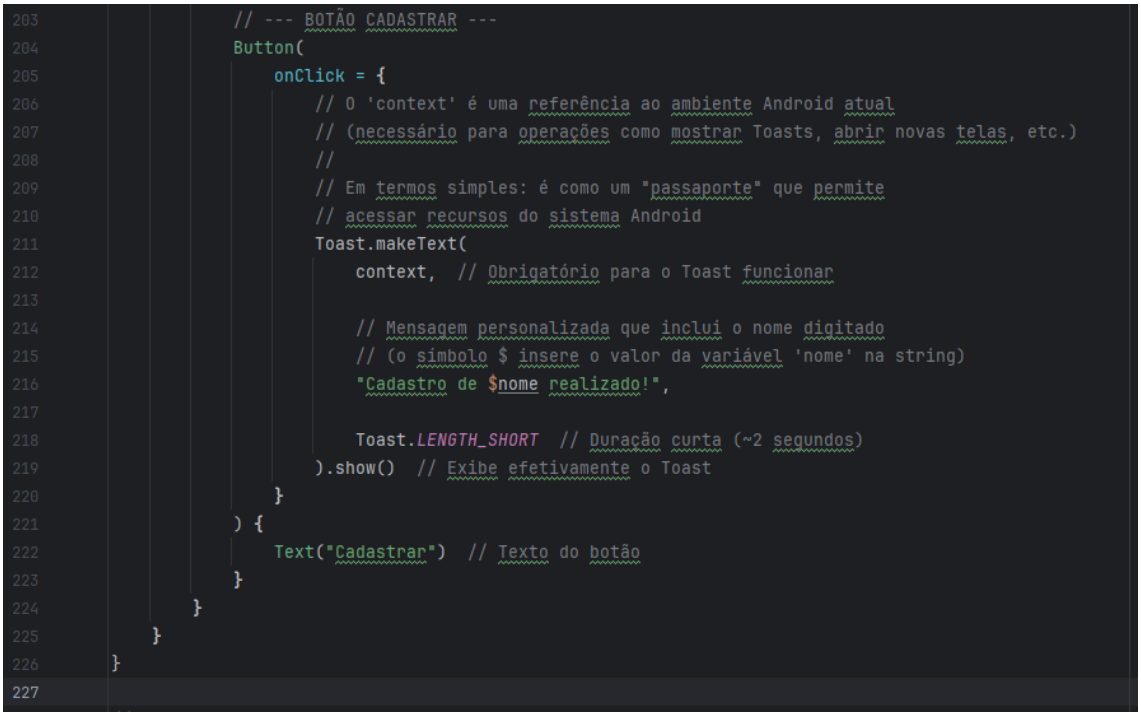


```
        // ========================================
        // [5] BOTÕES DE AÇÃO
        // ========================================
        Row(
            modifier = Modifier
                .padding(16.dp)  // Espaço ao redor dos botões (16 unidades independentes de densidade)
                .fillMaxWidth(),  // Ocupa toda a largura disponível na tela

            // Espaçamento igual entre os botões e as bordas
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            // --- BOTÃO LIMPAR ---
            OutlinedButton(
                onClick = {
                    // Reseta todos os campos do formulário para vazio ("")
                    nome = ""
                    telefone = ""
                    curso = ""
                    serie = ""
                }
            ) {
                Text("Limpar")  // Texto exibido no botão
            }

            // Espaço fixo de 16dp entre os botões
            Spacer(modifier = Modifier.width(16.dp))

            // --- BOTÃO CADASTRAR ---
            Button(
                onClick = {
                    // O 'context' é uma referência ao ambiente Android atual
                    // (necessário para operações como mostrar Toasts, abrir novas telas, etc.)
                    //
                    // Em termos simples: é como um "passaporte" que permite
                    // acessar recursos do sistema Android
                    Toast.makeText(
```



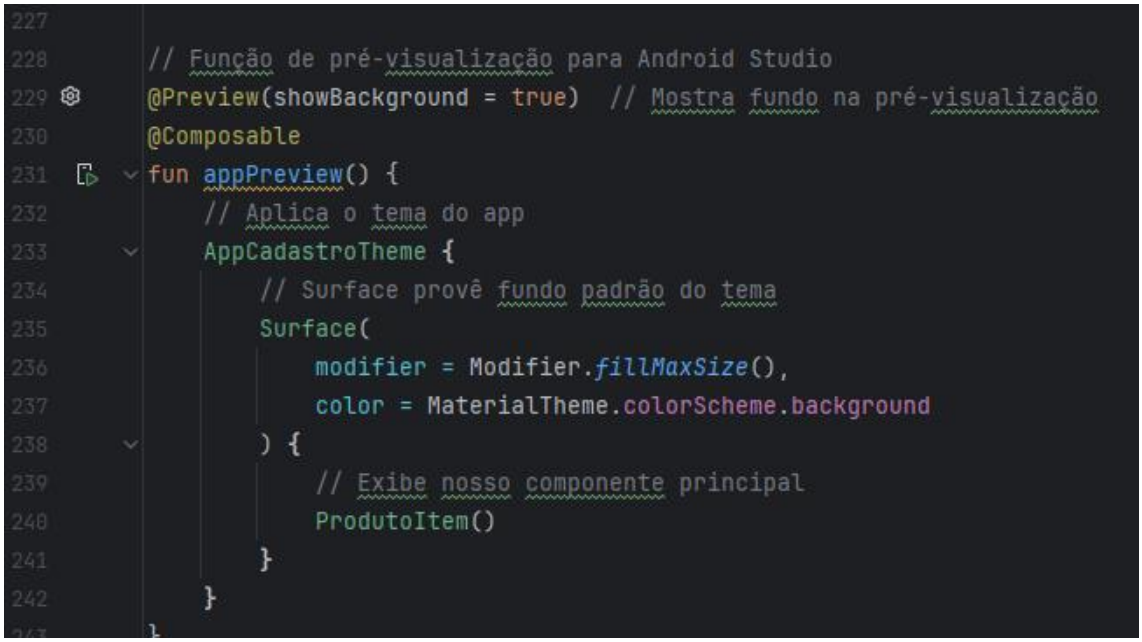
```
            // --- BOTÃO CADASTRAR ---
            Button(
                onClick = {
                    // O 'context' é uma referência ao ambiente Android atual
                    // (necessário para operações como mostrar Toasts, abrir novas telas, etc.)
                    //
                    // Em termos simples: é como um "passaporte" que permite
                    // acessar recursos do sistema Android
                    Toast.makeText(
                        context,  // Obrigatório para o Toast funcionar

                        // Mensagem personalizada que inclui o nome digitado
                        // (o símbolo $ insere o valor da variável 'nome' na string)
                        "Cadastro de $nome realizado!",

                        Toast.LENGTH_SHORT  // Duração curta (~2 segundos)
                    ).show()  // Exibe efetivamente o Toast
                }
            ) {
                Text("Cadastrar")  // Texto do botão
            }
        }
    }
}

```

A seção final do formulário apresenta dois botões com funções complementares, cuidadosamente dispostos em uma linha horizontal. À esquerda, o botão "Limpar" aparece com um visual discreto de contorno, que ao ser pressionado apaga instantaneamente todos os dados digitados nos campos do formulário - uma funcionalidade prática para quando o usuário precisa começar novamente. À direita, o botão "Cadastrar" se destaca com seu preenchimento sólido, e quando acionado exibe uma mensagem flutuante na parte inferior da tela confirmando o registro, incluindo de forma personalizada o nome que a pessoa digitou. Os botões são posicionados nas extremidades da tela com um espaçamento equilibrado entre eles, criando uma disposição visual harmoniosa e intuitiva. Essa implementação oferece ao usuário tanto a opção de reiniciar o preenchimento quanto de confirmar seu cadastro, com um feedback visual imediato em ambos os casos, tornando a experiência de uso fluida e satisfatória.



```
227
228     // Função de pré-visualização para Android Studio
229     @Preview(showBackground = true)  // Mostra fundo na pré-visualização
230     @Composable
231     fun appPreview() {
232         // Aplica o tema do app
233         AppCadastroTheme {
234             // Surface provê fundo padrão do tema
235             Surface(
236                 modifier = Modifier.fillMaxSize(),
237                 color = MaterialTheme.colorScheme.background
238             ) {
239                 // Exibe nosso componente principal
240                 ProdutoItem()
241             }
242         }
```

Esta função representa uma ferramenta essencial para o desenvolvimento com Jetpack Compose, criando um ambiente simulado que permite visualizar o formulário ProdutoItem diretamente no Android Studio. Através da anotação @Preview, o ambiente de desenvolvimento reconhece que esta é uma tela que deve ser exibida no painel de pré-visualização, com a opção showBackground ativada para melhor contextualização visual. O código envolve o componente principal com o AppCadastroTheme, garantindo que todas as personalizações de estilo e cores sejam aplicadas fielmente durante o processo de design. O container Surface cumpre um papel importante ao estabelecer o fundo padrão do tema Material Design, enquanto o modificador fillMaxSize assegura que o formulário ocupe todo o espaço disponível na pré-visualização. Esta abordagem oferece aos desenvolvedores um feedback visual imediato durante a implementação, permitindo ajustes rápidos no layout e na aparência dos componentes, significativamente acelerando o ciclo de desenvolvimento e refinamento da interface do usuário, tudo isso sem a necessidade de compilar e executar o aplicativo repetidamente em emuladores ou dispositivos físicos.