Desenvolvimento para Dispositivos Móveis

Introdução à Programação Orientada a Objetos (POO)

Prof. Bruno Azevedo

UNIP – Universidade Paulista

Programação Orientada a Objetos (POO)

- POO é um paradigma de programação que organiza o código em torno de **objetos**.
- Cada objeto representa um elemento do mundo real com características e comportamentos.
- Exemplos do cotidiano:
  - `Carro`: atributos como `cor`, `modelo`, `velocidade`; métodos como `acelerar()`, `frear()`.
  - `Celular`: atributos como `marca`, `sistema`, `tamanho`; métodos como `ligar()`, `tocarMusica()`.
  - `ContaBancaria`: atributos como `saldo`, `titular`; métodos como `depositar()`, `sacar()`.

A importância da POO no Android

- O Android é baseado fortemente em componentes orientados a objetos.
- As telas, botões, caixas de texto, atividades, etc., são todos objetos.
- A POO permite criar aplicativos organizados, reutilizáveis e fáceis de manter.

Classe

- Uma **classe** é um modelo que define os atributos e comportamentos de um objeto.
- Ela funciona como um molde para criar objetos.
- Exemplo: Classe `Aluno` com atributos como `nome`, `curso`, `notaFinal`.
- A classe define as variáveis e funções que os objetos terão.

Objeto

- Um **objeto** é uma instância de uma classe.
- Quando criamos um objeto, ele passa a ter valores reais e únicos.
- Exemplo:
  - Objeto `aluno1` da classe `Aluno`.
  - Atributos: `nome = "Lucas"`, `curso = "Computação"`, `notaFinal = 8.5`.
- Cada objeto tem seus próprios dados, mas compartilha a estrutura definida pela classe.

Atributos

- Atributos são variáveis definidas dentro da classe que representam características dos objetos.
- Cada objeto (instância) pode ter valores diferentes para os atributos.
  - Exemplo da Classe Aluno: `nome`, `idade`, `notaFinal`.
- Atributos representam "quem o objeto é".

Exemplo de Atributo em um App Android

- Em um aplicativo Android, elementos de interface também possuem atributos que podem ser modificados.
- Suponha que temos um botão chamado `botaoOk`.
- Podemos alterar seus atributos assim:

```
botaoOk.text = "OK"
botaoOk.textSize = 18f
```

- `text` e `textSize` são atributos do objeto `botaoOk`, que é uma instância de `Button`.
- O `18` é o valor do tamanho da fonte e o `f` indica que o número é um float (número decimal), que é o tipo esperado por textSize.

Métodos

- Métodos são funções associadas a uma classe.
- Definem os comportamentos da classe.
  - Exemplo da Classe carro: `acelerar()`, `frear()`.
- Métodos representam "o que o objeto faz".

Métodos em uma Classe

- Métodos são funções associadas a uma classe e definem o comportamento dos objetos criados a partir dela.
- Eles permitem manipular atributos e realizar ações específicas.
- Um método pode usar os dados do próprio objeto (acessando os atributos) e também receber parâmetros externos.
- Exemplo com a classe `Carro`:

```
fun detalhesDoCarro() {
    println("Marca: $marca, Modelo: $modelo, Ano: $anoFabricacao")
}
```

- O método `detalhesDoCarro()` imprime os dados do carro (atributos).

Exemplo de Método em um App Android

- Em Android, usamos métodos para reagir a ações do usuário.
- O método `setOnClickListener` é associado a um botão:

```
botaoOk.text = "OK"
botaoOk.setOnClickListener {
    // Alterando o texto do botão quando clicado
    botaoOk.text = "Botão clicado!"
}
```

- Neste exemplo, o método `setOnClickListener` escuta o clique no botão e executa uma ação: alterar o texto do próprio botão.
- Usamos um método e um atributo.

Instanciação de Objetos

- Para usar uma classe, precisamos **instanciar** um objeto (criar).
- Um objeto é uma **instância** de uma classe.
- Em Kotlin, usamos o operador `val` ou `var` com o nome da classe:

```
val aluno1 = Aluno()
```

- Podemos acessar os atributos e métodos do objeto com o operador ponto:

```
aluno1.nome = "Fernanda"
aluno1.exibirStatus()
```

- Acima, criamos um objeto (`aluno1`) da classe `Aluno`.
- Cada instância (objeto) é independente das outras.

Implementação: Exemplo

- Vamos implementar em Kotlin as classes do primeiro slide:
  - `Carro`: atributos como `cor`, `modelo`, `velocidade`; métodos como `acelerar()`, `frear()`.
  - `Celular`: atributos como `marca`, `sistema`, `tamanho`; métodos como `ligar()`, `tocarMusica()`.
  - `ContaBancaria`: atributos como `saldo`, `titular`; métodos como `depositar()`, `sacar()`.

Implementação: Exemplo

- Classe Carro em Kotlin:

```
class Carro(val cor: String, val modelo: String, var velocidade: Int) {
    fun acelerar() {
        velocidade += 10
        println("O carro acelerou. Velocidade atual: $velocidade km/h")
    }
    fun frear() {
        if (velocidade > 0) {
            velocidade -= 10
        }
        println("O carro freou. Velocidade atual: $velocidade km/h")
    }
}
fun main() {
    val meuCarro = Carro("Vermelho", "Sedan", 0)
    meuCarro.acelerar()
    meuCarro.acelerar()
    meuCarro.frear()
}
```

Implementação: Exemplo

- Classe Celular em Kotlin:

```
class Celular(val marca: String, val sistema: String, val tamanho: Double) {
    fun ligar() {
        println("Ligando o celular da marca $marca...")
    }
    fun tocarMusica() {
        println("Reproduzindo música no $marca com sistema $sistema.")
    }
}
fun main() {
    val meuCelular = Celular("Samsung", "Android", 6.5)
    meuCelular.ligar()
    meuCelular.tocarMusica()
}
```

Implementação: Exemplo

- Classe Conta Bancária em Kotlin:

```
class ContaBancaria(val titular: String, var saldo: Double) {
    fun depositar(valor: Double) {
        saldo += valor
        println("Depósito de R$ $valor realizado. Saldo atual: R$ $saldo")
    }
    fun sacar(valor: Double) {
        if (valor <= saldo) {
            saldo -= valor
            println("Saque de R$ $valor realizado. Saldo atual: R$ $saldo")
        } else {
            println("Saldo insuficiente para saque de R$ $valor.")
        }
    }
}
fun main() {
    val conta = ContaBancaria("João", 1000.0)
    conta.depositar(500.0)
    conta.sacar(200.0)
    conta.sacar(1500.0)
}
```

Herança em Programação Orientada a Objetos

- A **herança** permite que uma classe herde atributos e métodos de outra classe.
- Representa uma relação do tipo é `um`.
  - Um `Cachorro` é `um` `Animal`.
  - Um `Estudante` é `uma` `Pessoa`.
- ▷ A classe `base` (superclasse, pai) representa um conceito mais genérico e contém comportamentos comuns.
- ▷ A classe `derivada` (subclasse, filha) estende a classe base, podendo reutilizar e especializar seus comportamentos.
- Em Kotlin, para que uma classe seja herdada, ela deve ser marcada com a palavra-chave **open**.

Exemplo de Herança: Carro

- Considere uma classe `Veiculo`, que define um método `acelerar()`.
- Exemplo de implementação:

```
open class Veiculo {
    fun acelerar() {
        println("Acelerando o veículo!")
    }
}
class Carro : Veiculo() {
    // Carro herda o método acelerar
}
class Moto : Veiculo() {
    // Moto herda o método acelerar
}
```

- As classes `Carro` e `Moto` herdam de `Veiculo` e podem usar o método `acelerar()` herdado da classe base.

Exemplo de Herança: Botão Contador

- A classe `BotaoContador` herda de `Button` e adiciona um novo comportamento.
- O método `incrementaContaClick()` altera o texto do botão cada vez que ele é clicado.
- Exemplo:

```
class BotaoContador : Button {
    private var contaClick = 0
    // Outros Atributos e Métodos da classe BotaoContador
    fun incrementaContaClick() {
        contaClick++
        text = "Clicado $contaClick vezes"
    }
}
```

- O `BotaoContador` usa todos os atributos e métodos da classe `Button` e pode adicionar novos atributos e comportamentos.

Polimorfismo em Programação Orientada a Objetos

- Polimorfismo significa “muitas formas”.
- É a capacidade de um objeto assumir diferentes formas dependendo do contexto.
- Em POO, um método pode ter comportamentos diferentes em classes relacionadas por herança.
- Exemplo:
  - O método `falar()` pode ter implementações distintas em diferentes classes.
  - Em `Pessoa`: `falar()` diz “Olá!”.
  - Em `Cachorro`: `falar()` faz “Au au!”.
- O método tem o mesmo nome, mas comportamentos diferentes nas subclasses.

Exemplo de Polimorfismo

```
open class Animal {
    open fun emitirSom() {
        println("O animal emite um som")
    }
}
class Pessoa : Animal() {
    override fun emitirSom() {
        println("Olá!")
    }
}
class Cachorro : Animal() {
    override fun emitirSom() {
        println("Au au!")
    }
}
fun main() {
    val pessoa: Animal = Pessoa()
    val cachorro: Animal = Cachorro()
    pessoa.emitirSom()   // Saída: Olá!
    cachorro.emitirSom() // Saída: Au au!
}
```

- `emitirSom()` tem comportamentos diferentes dependendo do tipo do objeto (Pessoa ou Cachorro).

Criando o Projeto Android

- Vamos criar nosso primeiro projeto Android utilizando o Android Studio.
- Utilizaremos o que aprendemos de Programação Orientada a Objetos.
- Criaremos um aplicativo que instancia objetos de uma classe representando filmes e exibe a categoria de cada filme instanciado.

Criando o Projeto Android

- Abra o Android Studio e crie um novo projeto:
  - Nome do projeto: `FilmeApp`.
  - Linguagem: Kotlin.
  - Tipo de atividade: `Empty Activity`.
- Editaremos apenas o arquivo `MainActivity.kt`.

Editando o MainActivity.kt

- Apague todo conteúdo do `MainActivity.kt`.
- Adicione o seguinte trecho no início.
- Esses imports são necessários para configurar a tela, aplicar o tema e usar componentes do Jetpack Compose.

```
package com.example.filmeapp
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.sp
import androidx.compose.ui.unit.dp
import com.example.filmeapp.ui.theme.FilmeAppTheme
```

Editando o MainActivity.kt

- A classe `MainActivity` herda de `ComponentActivity` e configura o conteúdo da tela no método `onCreate()`.
- Dentro de `setContent()`, o tema do aplicativo `FilmeAppTheme` é aplicado, envolvendo a função composable `TelaFilmes()`.
- A função `TelaFilmes()` exibe a lista de filmes.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            FilmeAppTheme {
                TelaFilmes()  // Chama o composable TelaFilmes
            }
        }
    }
}
```

Editando o MainActivity.kt

- A classe `Filme` representa um filme com título e ano de lançamento.
- O método `categoria()` retorna a categoria do filme com base no ano.
- O método `descricao()` gera uma descrição formatada com título, ano e categoria.

```
class Filme(val titulo: String, val ano: Int) {
    fun categoria(): String {
        return when {
            ano >= 2025 -> "Lançamento"
            ano in 2010..2024 -> "Moderno"
            ano in 2000..2009 -> "Anos 2000"
            ano in 1980..1999 -> "Anos 80 e 90"
            else -> "Clássico"
        }
    }
    fun descricao(): String {
        return "$titulo ($ano) é um filme da categoria: ${categoria()}."
    }
}
```

Instanciando filmes e exibindo resultados

- A função `@Composable TelaFilmes()` é responsável por exibir os filmes na tela.
- Uma lista `resultado` é criada, unindo as descrições de cada filme utilizando `joinToString()`.
- A função `Column()` é usada para organizar os elementos verticalmente.
- `Text()` exibe a string (`resultado`) na tela.

```
@Composable
fun TelaFilmes() {
    // Criando os objetos Filme
    val filme1 = Filme("Matrix", 1999)
    val filme2 = Filme("Interestelar", 2014)
    val filme3 = Filme("Duna", 2021)
    val filme4 = Filme("Lobisomem", 2025)
    // Criando uma lista com as descrições dos filmes
    val resultado = listOf(filme1, filme2, filme3, filme4)
        .joinToString("\n") { it.descricao() }
    // Exibindo os resultados na tela
    Column(modifier = Modifier.padding(16.dp)) {
        Text(
            text = resultado,
            fontSize = 12.sp
        )
    }
}
```

Exibindo o resultado na tela

- Finalmente, adicione uma função preview para acompanhar o conteúdo durante o desenvolvimento.

```
@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    FilmeAppTheme {  // Usando o tema do seu app na visualização
        TelaFilmes()  // Chamando a tela com os filmes
    }
}
```

Resultado Esperado

- Ao executar o aplicativo, o usuário verá na tela:
  - `Matrix (1999) é um filme da categoria: Anos 80 e 90.`
  - `Interestelar (2014) é um filme da categoria: Moderno.`
  - `Duna (2021) é um filme da categoria: Moderno.`
  - `Lobisomem (2025) é um filme da categoria: Lançamento.`

Atividade Prática 5

1. Criar um aplicativo de gestão de produtos:
   - Definir uma classe `Produto` com os atributos: `nome`, `preço`, e `quantidade em estoque`.
   - Criar um método `exibirInformacoes()` que exiba os detalhes do produto (nome, preço e estoque disponível).
   - Instanciar pelo menos três objetos de produtos diferentes e exibir suas informações em uma lista.
2. Ampliação: adicionar as seguintes funcionalidades:
   - Criar um método que atualize o estoque com base em uma venda simulada pelo usuário.
   - Implementar um método para calcular e exibir o valor total em estoque (`preço × quantidade`).
   - Exibir uma mensagem de alerta (usando `AlertDialog`) caso algum produto esteja com o estoque zerado.
3. Cada aluno deve produzir um relatório de 1 a 3 páginas contendo:
   - Resumo teórico: Explicação conceitual sobre classes, objetos, métodos e manipulação de dados.
   - Código-fonte comentado: Explicação detalhada do código, destacando a lógica de cada funcionalidade implementada.