# Introdução a Kotlin e Programação Orientada a Objetos (POO)

Prof. Esp. Iuri Nascimento Santos

**Objetivo geral:** Apresentar os fundamentos da linguagem Kotlin e aplicar conceitos básicos de Programação Orientada a Objetos (POO) usando exemplos simples e exercícios práticos.

---

## Bloco 1 — Introdução ao Kotlin (15 minutos)

**Explicação teórica:**

- O que é Kotlin:
  - Linguagem moderna, interoperável com Java, muito usada para Android e backend.
- Por que aprender Kotlin para POO:
  - Sintaxe enxuta, moderna, elimina verbosidade comum em Java.
- O que é POO:
  - Paradigma baseado em classes, objetos, atributos, métodos, herança, polimorfismo, encapsulamento.

**Atividade rápida no Kotlin Playground (https://play.kotlinlang.org/):**

1. Executar `Hello, World!`:

```
fun main() {
    println("Hello, World!")
}
```

   → Explicar função `main()` e `println()`.

2. Declarar variáveis (`val` e `var`):

```
fun main() {
    val nome = "Ana"
    var idade = 20

    println("Nome: $nome")
    println("Idade: $idade")
}
```

→ Destacar diferença entre variável imutável (`val`) e mutável (`var`).

3. Criar uma função simples:

```
fun main() {
    val resultado = somar(5, 3)
    println("Resultado: $resultado")
}

fun somar(a: Int, b: Int): Int {
    return a + b
}
```

→ Explicar declaração de função, parâmetros e retorno.

---

# Bloco 2 — Conceitos Básicos de Classes e Objetos (30 minutos)

**Exemplos guiados:**

1. Criar uma classe simples:

```
class Pessoa(val nome: String, var idade: Int, val email: String) {
    fun apresentar() {
        println("Olá, sou $nome, tenho $idade anos, e meu e-mail é $emai
l.")
    }
}

fun main() {
    val pessoa1 = Pessoa("Ana", 20, "ana@email.com")
```

```
    pessoa1.apresentar()
}
```

→ Explicar atributos no construtor e método `apresentar()`.

2. Modificar atributos:

```
pessoa1.idade = 21
pessoa1.apresentar()
```

→ Destacar que só atributos `var` podem ser alterados.

3. Criar mais objetos:

```
val pessoa2 = Pessoa("Bruno", 30, "bruno@email.com")
pessoa2.apresentar()
```

4. Adicionar método `verificarMaioridade()`:

```
fun verificarMaioridade() {
    if (idade >= 18) {
        println("$nome é maior de idade.")
    } else {
        println("$nome é menor de idade.")
    }
}
```

**Atividade prática:**

- Alunos criam uma terceira pessoa.
- Alteram algum atributo.
- Implementam método `verificarMaioridade()` e testam.

---

# Bloco 3 — Trabalhando com Coleções e Laços (30 minutos)

1. Criar lista de pessoas:

```
val listaPessoas = listOf(pessoa1, pessoa2)
for (p in listaPessoas) {
    p.apresentar()
}
```

→ Explicar laço `for`.

2. Criar lista mutável e adicionar elementos:

```
val listaPessoas = mutableListOf<Pessoa>()
listaPessoas.add(Pessoa("Ana", 20, "ana@email.com"))
listaPessoas.add(Pessoa("Bruno", 30, "bruno@email.com"))
```

3. Inserir dados por entrada (para rodar localmente, não no Playground):

```
for (i in 1..2) {
    print("Digite o nome da pessoa $i: ")
    val nome = readLine() ?: ""

    print("Digite a idade: ")
    val idade = readLine()?.toIntOrNull() ?: 0

    print("Digite o e-mail: ")
    val email = readLine() ?: ""

    listaPessoas.add(Pessoa(nome, idade, email))
}
```

---

# Bloco 4 — Exercícios para Fixação (30 minutos)

**Propor aos alunos:**

1. Criar uma nova classe `Livro`:
   - Atributos: `titulo`, `autor`, `ano`.
   - Método: `mostrarInfo()` que imprime as informações.
2. Criar uma lista mutável de livros e percorrê-la com `for`.

3. Implementar leitura de dados do terminal para preencher a lista de livros (opcional para quem roda localmente).
4. Desafio extra:
    - Criar uma classe `Biblioteca` que contém uma lista de livros.
    - Adicionar método `listarLivros()` que imprime todos os livros cadastrados.

---

# Encerramento (15 minutos)

- Revisar rapidamente:
    - Diferença entre `val` e `var`.
    - Como declarar classes e métodos.
    - Como trabalhar com listas.
- Destacar que os conceitos aprendidos (classes, atributos, listas, métodos) são base para construir aplicativos reais.
- Orientar os alunos a salvar os códigos testados e guardar para consulta futura.

---

## Bloco 4 — Herança e Sobrescrita (30 min)

---

## 1 — Introdução à herança

No Kotlin, usamos `open` para indicar que uma classe pode ser herdada, e `override` para sobrescrever métodos.

---

## Exemplo básico

```
open class Animal(val nome: String) {
    open fun emitirSom() {
        println("$nome está fazendo um som.")
    }
}

class Cachorro(nome: String) : Animal(nome) {
    override fun emitirSom() {
        println("$nome está latindo! Au au!")
```

```
    }
}

class Gato(nome: String) : Animal(nome) {
    override fun emitirSom() {
        println("$nome está miando! Miau!")
    }
}

fun main() {
    val cachorro = Cachorro("Rex")
    val gato = Gato("Mimi")

    cachorro.emitirSom()
    gato.emitirSom()
}
```

Pontos a explicar:

- `open class`: permite ser herdada.
- `: Animal(nome)`: chama o construtor da superclasse.
- `override`: sobrescreve o método herdado.

---

## 2 — Usando polimorfismo

Mesmo tratando tudo como `Animal`, cada objeto se comporta de forma diferente.

---

## Exemplo

```
fun main() {
    val animais = listOf(
        Cachorro("Rex"),
        Gato("Mimi"),
        Cachorro("Bolt"),
        Gato("Luna")
    )
```

```
    for (animal in animais) {
        animal.emitirSom()
    }
}
```

Ponto a explicar:

- O tipo declarado é `Animal`, mas o método específico de cada um é chamado → polimorfismo.

---

## 3 — Atividade para os alunos

- Criar uma nova classe `Passaro`, herdando de `Animal`.
- Fazer `Passaro` sobrescrever `emitirSom()` para imprimir `"Piu piu!"`.

---

## Exemplo solução

```
class Passaro(nome: String) : Animal(nome) {
    override fun emitirSom() {
        println("$nome está cantando! Piu piu!")
    }
}
```

Depois, adicionar um `Passaro` à lista e testar no laço.

---

## 4 — Extra (se sobrar tempo)

Criar um método extra em `Animal`:

```
open fun mover() {
    println("$nome está se movendo.")
}
```

Sobrescrever nas subclasses:

- Cachorro → correndo.
- Gato → andando silenciosamente.
- Pássaro → voando.

---

## Lista de Exercícios de Revisão

## Parte 1 — Aquecimento: Classes e Objetos

1. Crie a classe `Usuario` com `nome`, `email`, `senha`.

   → Instancie alguns objetos e imprima os dados.

2. Adicione o método `fazerLogin()` que imprime `"Usuário {nome} logado com sucesso"`.

1. Simule uma lista (`mutableListOf`) de usuários e exiba o nome de todos em um loop.

## Parte 2 — Praticando Coleções

1. Crie a classe `Produto` com `nome`, `preco`, `quantidadeEstoque`.

   → Monte uma lista e mostre no console os produtos com `quantidadeEstoque > 0`.

2. Escreva uma função para calcular o valor total do estoque (`preco * quantidadeEstoque`).

## Parte 3 — Trabalhando com Entrada de Dados (simulado no código)

1. Simule o cadastro de três produtos no código.

   → Crie uma função para “comprar” (reduzir `quantidadeEstoque` em 1).

2. Escreva `buscarProdutoPorNome()` para buscar um produto pelo nome e informar se não encontrado.

## Parte 4 — Herança e Polimorfismo

1. Crie a classe `Tela` com o método `abrirTela()`.

   → Crie subclasses `TelaLogin`, `TelaCadastro`, `TelaPerfil` que sobrescrevam esse método com mensagens específicas.

2. Monte uma lista de `Tela` e percorra chamando `abrirTela()` em cada uma.

## Parte 5 — Mini Projeto (Desafio Final)

Desafio: criar um sistema simples de gerenciamento de tarefas.

- Classe `Tarefa`: `titulo`, `descricao`, `status` (“pendente” ou “concluída”).
- Funções para:
  - Adicionar tarefa.
  - Marcar tarefa como concluída.
  - Listar todas as pendentes.
  - Listar todas as concluídas.
- (Opcional) Subclasse `TarefaImportante` com prioridade extra.

---

Obrigado por você estar presente nessa aula :)