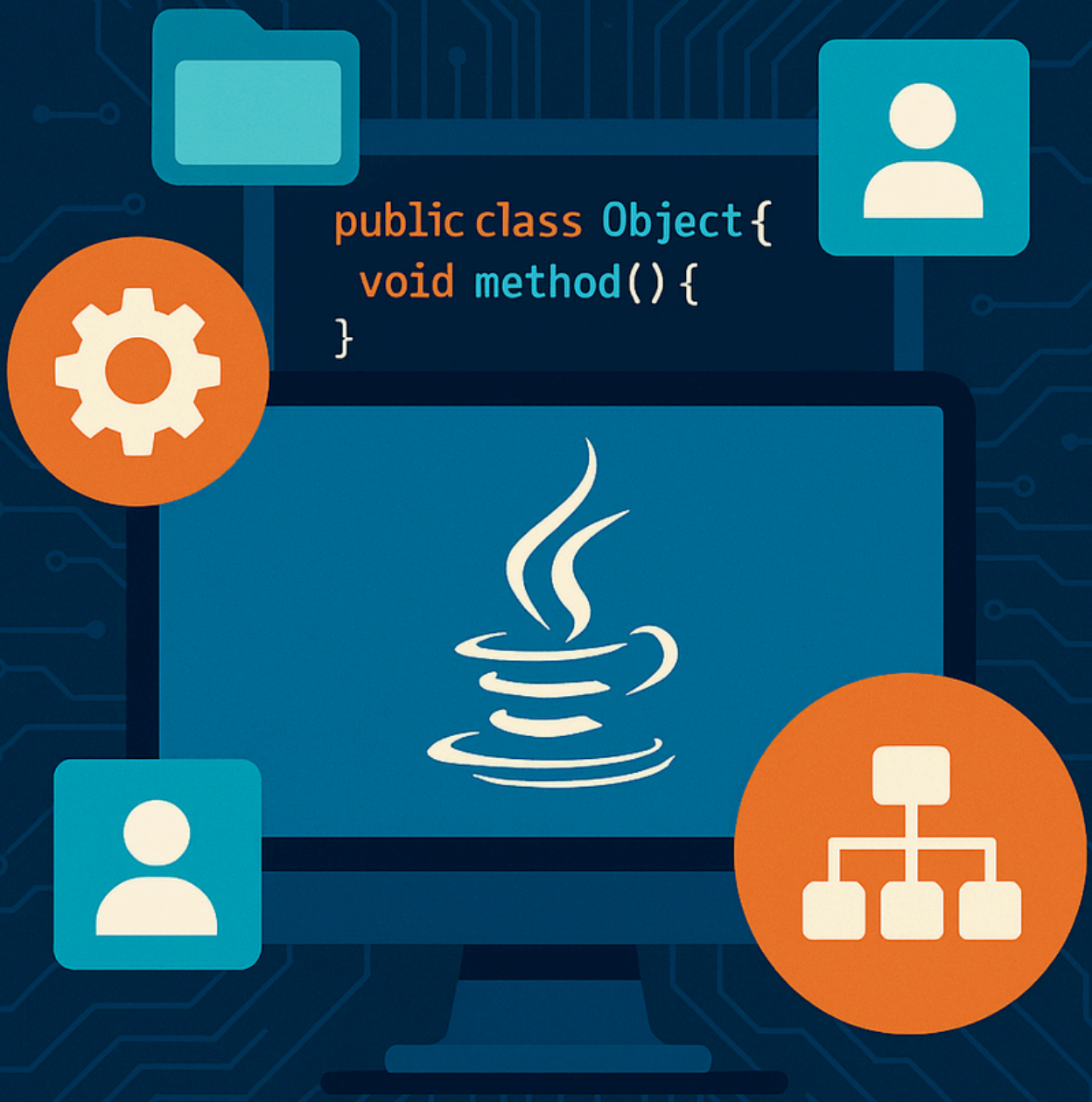
JAVA: CÓDIGO COM CLASSE



PROGRAMAÇÃO ORIENTADA A OBJETOS

1

Classes e Objetos

O que é uma classe?

Uma classe define como um **objeto** será — seus **atributos** (características) e **métodos** (comportamentos).

É como uma planta arquitetônica que descreve como uma casa será construída.

```java
public class Carro {
    String modelo;
    int ano;

    void buzinar() {
        System.out.println("Biiiii!");
    }
}
```

Neste exemplo, **Carro** é a classe. Ela tem dois atributos (modelo e ano) e um **método** `buzinar()`

O que é um objeto?

Um **objeto** é uma **instância** de uma classe, ou seja, algo real criado a partir do molde da classe.

```
public class Main {
    public static void main(String[] args)
{
        Carro meuCarro = new Carro();
        meuCarro.modelo = "Fusca";
        meuCarro.ano = 1979;


System.out.println(meuCarro.modelo); //
Fusca

System.out.println(meuCarro.ano);    //
1979

meuCarro.buzinar();                  //
Biiiii!
    }
}
```

Aqui, **meuCarro** é um **objeto** do tipo **Carro**. Ele tem seus próprios valores e pode executar ações definidas na classe.

2

Encapsulamento

Encapsulamento

É um dos pilares da programação orientada a objetos. Ele tem um objetivo claro: **proteger os dados e controlar o acesso a eles.** A forma correta de acessar ou alterar algo é através de **métodos controlados.**

```java
public class ContaBancaria {
    private double saldo;

    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
        }
    }

    public void sacar(double valor) {
        if (valor > 0 && valor <= saldo) {
            saldo -= valor;
        }
    }

    public double getSaldo() {
        return saldo;
    }
}
```

Aqui esta o encapsulamento em ação:
O atributo **saldo** é **privado**, **não pode ser acessado diretamente fora da classe.**

O acesso é feito **somente por meio de métodos públicos** como **depositar, sacar** e **getSaldo**

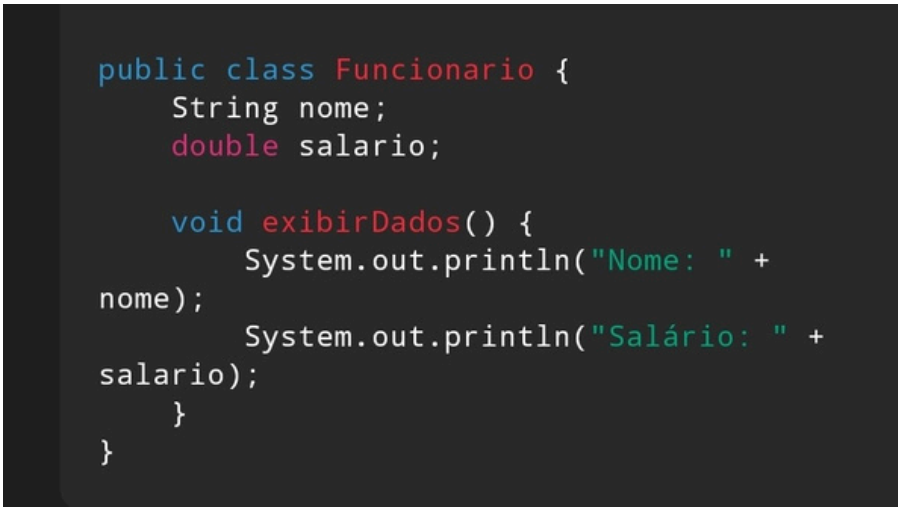
3

Herança

Herança

A herança permite que uma classe **herde atributos e métodos de outra**, promovendo reutilização de código e organização. No código, uma classe pode herdar de outra usando a palavra-chave **extends**.

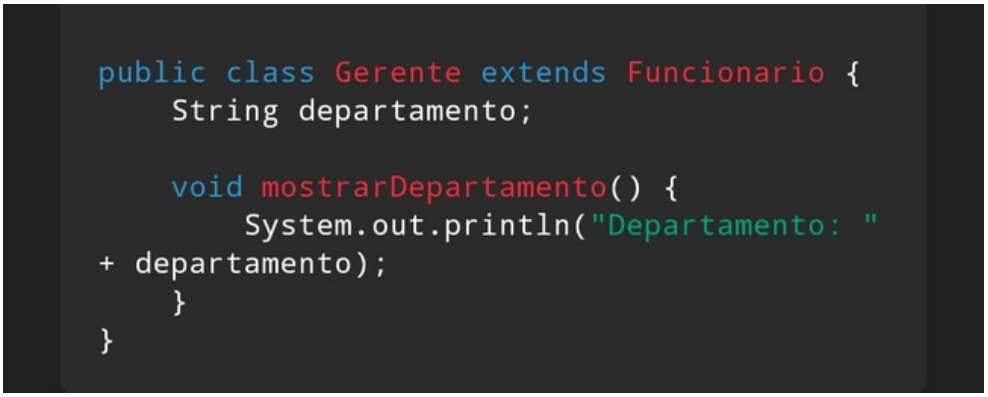
Classe base (pai):



```
public class Funcionario {
    String nome;
    double salario;

    void exibirDados() {
        System.out.println("Nome: " +
nome);
        System.out.println("Salário: " +
salario);
    }
}
```

Classe derivada (filha):



```
public class Gerente extends Funcionario {
    String departamento;

    void mostrarDepartamento() {
        System.out.println("Departamento: "
+ departamento);
    }
}
```

4

Polimorfismo

Polimorfismo

O polimorfismo permite que Uma mesma **ação** (método) se comporte **de maneira diferente** em **classes diferentes**

Existem dois tipos principais:
1 - Polimorfismo de **Sobrescrita (Override)**
2 - Polimorfismo de **Sobrecarga(Overload)**

Sobrescrita (Override):

```
public class Pagamento {
    public void processar() {
        System.out.println("Processando
pagamento genérico.");
    }
}
```

```
public class PagamentoBoleto extends
Pagamento {
    @Override
    public void processar() {
        System.out.println("Processando
pagamento via boleto.");
    }
}

public class PagamentoCartao extends
Pagamento {
    @Override
    public void processar() {
        System.out.println("Processando
pagamento via cartão.");
    }
}
```

Sobrecarga (Overload):

```
public class Calculadora {
    public int somar(int a, int b) {
        return a + b;
    }

    public double somar(double a, double b)
{
        return a + b;
    }

    public int somar(int a, int b, int c) {
        return a + b + c;
    }
}
```

5

Construtores

O que é construtor?

Construtores são métodos especiais usados para **criar objetos**. Eles são chamados automaticamente quando usamos **new**.

```java
public class Pessoa {
    String nome;

    // Construtor
    public Pessoa(String nome) {
        this.nome = nome;
    }
}
```

```java
Pessoa p = new Pessoa("João");
System.out.println(p.nome); // João
```