### **1. Classes e Objetos Básicos**

#### **Python**

class Carro:

def \_\_init\_\_(self, marca, modelo, ano):

self.marca = marca

self.modelo = modelo

self.ano = ano

def detalhes(self):

return f"{self.marca} {self.modelo}, Ano: {self.ano}"

# Instanciar 3 objetos

carro1 = Carro("Ford", "Mustang", 2020)

carro2 = Carro("Chevrolet", "Camaro", 2019)

carro3 = Carro("Tesla", "Model 3", 2021)

# Exibir detalhes de cada carro

print(carro1.detalhes())

print(carro2.detalhes())

print(carro3.detalhes())

#### **Java**

class Carro {

String marca;

String modelo;

int ano;

Carro(String marca, String modelo, int ano) {

this.marca = marca;

this.modelo = modelo;

this.ano = ano;

}

void detalhes() {

System.out.println(marca + " " + modelo + ", Ano: " + ano);

}

public static void main(String[] args) {

Carro carro1 = new Carro("Ford", "Mustang", 2020);

Carro carro2 = new Carro("Chevrolet", "Camaro", 2019);

Carro carro3 = new Carro("Tesla", "Model 3", 2021);

carro1.detalhes();

carro2.detalhes();

carro3.detalhes();

}

}

#### **Golang**

package main

import "fmt"

type Carro struct {

Marca string

Modelo string

Ano int

}

func (c Carro) Detalhes() {

fmt.Printf("%s %s, Ano: %d\n", c.Marca, c.Modelo, c.Ano)

}

func main() {

carro1 := Carro{"Ford", "Mustang", 2020}

carro2 := Carro{"Chevrolet", "Camaro", 2019}

carro3 := Carro{"Tesla", "Model 3", 2021}

carro1.Detalhes()

carro2.Detalhes()

carro3.Detalhes()

}

### **2. Métodos**

#### **Python**

class Carro:

def \_\_init\_\_(self, marca, modelo, ano):

self.marca = marca

self.modelo = modelo

self.ano = ano

self.velocidade = 0

def acelerar(self, incremento):

self.velocidade += incremento

def frear(self, decremento):

self.velocidade -= decremento

def exibir\_velocidade(self):

return f"Velocidade atual: {self.velocidade} km/h"

# Teste

carro = Carro("Honda", "Civic", 2022)

carro.acelerar(50)

print(carro.exibir\_velocidade()) # Velocidade atual: 50 km/h

carro.frear(20)

print(carro.exibir\_velocidade()) # Velocidade atual: 30 km/h

#### **Java**

class Carro {

String marca;

String modelo;

int ano;

int velocidade = 0;

Carro(String marca, String modelo, int ano) {

this.marca = marca;

this.modelo = modelo;

this.ano = ano;

}

void acelerar(int incremento) {

velocidade += incremento;

}

void frear(int decremento) {

velocidade -= decremento;

}

void exibirVelocidade() {

System.out.println("Velocidade atual: " + velocidade + " km/h");

}

public static void main(String[] args) {

Carro carro = new Carro("Honda", "Civic", 2022);

carro.acelerar(50);

carro.exibirVelocidade(); // Velocidade atual: 50 km/h

carro.frear(20);

carro.exibirVelocidade(); // Velocidade atual: 30 km/h

}

}

#### **Golang**

package main

import "fmt"

type Carro struct {

Marca string

Modelo string

Ano int

Velocidade int

}

func (c \*Carro) Acelerar(incremento int) {

c.Velocidade += incremento

}

func (c \*Carro) Frear(decremento int) {

c.Velocidade -= decremento

}

func (c Carro) ExibirVelocidade() {

fmt.Printf("Velocidade atual: %d km/h\n", c.Velocidade)

}

func main() {

carro := Carro{Marca: "Honda", Modelo: "Civic", Ano: 2022}

carro.Acelerar(50)

carro.ExibirVelocidade() // Velocidade atual: 50 km/h

carro.Frear(20)

carro.ExibirVelocidade() // Velocidade atual: 30 km/h

}

### **3. Encapsulamento**

#### **Python**

class ContaBancaria:

def \_\_init\_\_(self, titular, saldo):

self.titular = titular

self.\_\_saldo = saldo # Atributo encapsulado

def depositar(self, valor):

self.\_\_saldo += valor

def sacar(self, valor):

if valor <= self.\_\_saldo:

self.\_\_saldo -= valor

else:

print("Saldo insuficiente!")

def exibir\_saldo(self):

return f"Saldo atual: R$ {self.\_\_saldo}"

# Teste

conta = ContaBancaria("João", 1000)

conta.depositar(500)

print(conta.exibir\_saldo()) # Saldo atual: R$ 1500

conta.sacar(2000) # Saldo insuficiente!

print(conta.exibir\_saldo()) # Saldo atual: R$ 1500

#### **Java**

class ContaBancaria {

private String titular;

private double saldo;

ContaBancaria(String titular, double saldo) {

this.titular = titular;

this.saldo = saldo;

}

void depositar(double valor) {

saldo += valor;

}

void sacar(double valor) {

if (valor <= saldo) {

saldo -= valor;

} else {

System.out.println("Saldo insuficiente!");

}

}

void exibirSaldo() {

System.out.println("Saldo atual: R$ " + saldo);

}

public static void main(String[] args) {

ContaBancaria conta = new ContaBancaria("João", 1000);

conta.depositar(500);

conta.exibirSaldo(); // Saldo atual: R$ 1500

conta.sacar(2000); // Saldo insuficiente!

conta.exibirSaldo(); // Saldo atual: R$ 1500

}

}

#### **Golang**

package main

import "fmt"

type ContaBancaria struct {

Titular string

saldo float64

}

func (c \*ContaBancaria) Depositar(valor float64) {

c.saldo += valor

}

func (c \*ContaBancaria) Sacar(valor float64) {

if valor <= c.saldo {

c.saldo -= valor

} else {

fmt.Println("Saldo insuficiente!")

}

}

func (c ContaBancaria) ExibirSaldo() {

fmt.Printf("Saldo atual: R$ %.2f\n", c.saldo)

}

func main() {

conta := ContaBancaria{Titular: "João", saldo: 1000}

conta.Depositar(500)

conta.ExibirSaldo() // Saldo atual: R$ 1500

conta.Sacar(2000) // Saldo insuficiente!

conta.ExibirSaldo() // Saldo atual: R$ 1500

}

### **4. Herança**

#### **Python**

class Animal:

def som(self):

pass

class Cachorro(Animal):

def som(self):

return "Latido"

class Gato(Animal):

def som(self):

return "Miau"

# Teste

cachorro = Cachorro()

gato = Gato()

print(cachorro.som()) # Latido

print(gato.som()) # Miau

#### **Java**

class Animal {

public void som() {

}

}

class Cachorro extends Animal {

@Override

public void som() {

System.out.println("Latido");

}

}

class Gato extends Animal {

@Override

public void som() {

System.out.println("Miau");

}

public static void main(String[] args) {

Animal cachorro = new Cachorro();

Animal gato = new Gato();

cachorro.som(); // Latido

gato.som(); // Miau

}

}

#### **Golang**

package main

import "fmt"

type Animal interface {

Som() string

}

type Cachorro struct{}

func (c Cachorro) Som() string {

return "Latido"

}

type Gato struct{}

func (g Gato) Som() string {

return "Miau"

}

func main() {

var cachorro Animal = Cachorro{}

var gato Animal = Gato{}

fmt.Println(cachorro.Som()) // Latido

fmt.Println(gato.Som()) // Miau

}

### **5. Polimorfismo**

#### **Python**

class Animal:

def som(self):

pass

class Cachorro(Animal):

def som(self):

return "Latido"

class Gato(Animal):

def som(self):

return "Miau"

def emitir\_sons(animais):

for animal in animais:

print(animal.som())

# Teste

animais = [Cachorro(), Gato()]

emitir\_sons(animais) # Latido \n Miau

#### **Java**

class Animal {

public void som() {

}

}

class Cachorro extends Animal {

@Override

public void som() {

System.out.println("Latido");

}

}

class Gato extends Animal {

@Override

public void som() {

System.out.println("Miau");

}

}

public class PolimorfismoTeste {

public static void emitirSons(Animal[] animais) {

for (Animal animal : animais) {

animal.som();

}

}

public static void main(String[] args) {

Animal[] animais = {new Cachorro(), new Gato()};

emitirSons(animais); // Latido \n Miau

}

}

#### **Golang**

package main

import "fmt"

type Animal interface {

Som() string

}

type Cachorro struct{}

func (c Cachorro) Som() string {

return "Latido"

}

type Gato struct{}

func (g Gato) Som() string {

return "Miau"

}

func emitirSons(animais []Animal) {

for \_, animal := range animais {

fmt.Println(animal.Som())

}

}

func main() {

animais := []Animal{Cachorro{}, Gato{}}

emitirSons(animais) // Latido \n Miau

}

### **6. Composição**

#### **Python**

class Motor:

def \_\_init\_\_(self, potencia):

self.potencia = potencia

class Carro:

def \_\_init\_\_(self, marca, modelo, motor):

self.marca = marca

self.modelo = modelo

self.motor = motor

def detalhes(self):

return f"{self.marca} {self.modelo} com motor de {self.motor.potencia} cavalos"

# Teste

motor = Motor(150)

carro = Carro("Honda", "Civic", motor)

print(carro.detalhes()) # Honda Civic com motor de 150 cavalos

#### **Java**

class Motor {

int potencia;

Motor(int potencia) {

this.potencia = potencia;

}

}

class Carro {

String marca;

String modelo;

Motor motor;

Carro(String marca, String modelo, Motor motor) {

this.marca = marca;

this.modelo = modelo;

this.motor = motor;

}

void detalhes() {

System.out.println(marca + " " + modelo + " com motor de " + motor.potencia + " cavalos");

}

public static void main(String[] args) {

Motor motor = new Motor(150);

Carro carro = new Carro("Honda", "Civic", motor);

carro.detalhes(); // Honda Civic com motor de 150 cavalos

}

}

#### **Golang**

package main

import "fmt"

type Motor struct {

Potencia int

}

type Carro struct {

Marca string

Modelo string

Motor Motor

}

func (c Carro) Detalhes() {

fmt.Printf("%s %s com motor de %d cavalos\n", c.Marca, c.Modelo, c.Motor.Potencia)

}

func main() {

motor := Motor{Potencia: 150}

carro := Carro{Marca: "Honda", Modelo: "Civic", Motor: motor}

carro.Detalhes() // Honda Civic com motor de 150 cavalos

}

### **7. Associação**

#### **Python**

class Professor:

def \_\_init\_\_(self, nome):

self.nome = nome

class Escola:

def \_\_init\_\_(self, nome):

self.nome = nome

self.professores = []

def adicionar\_professor(self, professor):

self.professores.append(professor)

def exibir\_professores(self):

return [professor.nome for professor in self.professores]

# Teste

professor1 = Professor("Maria")

professor2 = Professor("João")

escola = Escola("Escola ABC")

escola.adicionar\_professor(professor1)

escola.adicionar\_professor(professor2)

print(escola.exibir\_professores()) # ['Maria', 'João']

#### **Java**

import java.util.ArrayList;

class Professor {

String nome;

Professor(String nome) {

this.nome = nome;

}

}

class Escola {

String nome;

ArrayList<Professor> professores = new ArrayList<>();

Escola(String nome) {

this.nome = nome;

}

void adicionarProfessor(Professor professor) {

professores.add(professor);

}

void exibirProfessores() {

for (Professor professor : professores) {

System.out.println(professor.nome);

}

}

public static void main(String[] args) {

Professor professor1 = new Professor("Maria");

Professor professor2 = new Professor("João");

Escola escola = new Escola("Escola ABC");

escola.adicionarProfessor(professor1);

escola.adicionarProfessor(professor2);

escola.exibirProfessores(); // Maria \n João

}

}

#### **Golang**

package main

import "fmt"

type Professor struct {

Nome string

}

type Escola struct {

Nome string

Professores []Professor

}

func (e \*Escola) AdicionarProfessor(professor Professor) {

e.Professores = append(e.Professores, professor)

}

func (e Escola) ExibirProfessores() {

for \_, professor := range e.Professores {

fmt.Println(professor.Nome)

}

}

func main() {

professor1 := Professor{Nome: "Maria"}

professor2 := Professor{Nome: "João"}

escola := Escola{Nome: "Escola ABC"}

escola.AdicionarProfessor(professor1)

escola.AdicionarProfessor(professor2)

escola.ExibirProfessores() // Maria \n João

}

### **8. Agregação**

#### **Python**

class Empregado:

def \_\_init\_\_(self, nome, cargo, salario):

self.nome = nome

self.cargo = cargo

self.salario = salario

class Empresa:

def \_\_init\_\_(self, nome):

self.nome = nome

self.empregados = []

def adicionar\_empregado(self, empregado):

self.empregados.append(empregado)

def exibir\_empregados(self):

for empregado in self.empregados:

print(f"Nome: {empregado.nome}, Cargo: {empregado.cargo}, Salário: {empregado.salario}")

# Teste

empregado1 = Empregado("Alice", "Desenvolvedora", 5000)

empregado2 = Empregado("Bob", "Designer", 4000)

empresa = Empresa("Tech Corp")

empresa.adicionar\_empregado(empregado1)

empresa.adicionar\_empregado(empregado2)

empresa.exibir\_empregados()

#### **Java**

import java.util.ArrayList;

class Empregado {

String nome;

String cargo;

double salario;

Empregado(String nome, String cargo, double salario) {

this.nome = nome;

this.cargo = cargo;

this.salario = salario;

}

}

class Empresa {

String nome;

ArrayList<Empregado> empregados = new ArrayList<>();

Empresa(String nome) {

this.nome = nome;

}

void adicionarEmpregado(Empregado empregado) {

empregados.add(empregado);

}

void exibirEmpregados() {

for (Empregado empregado : empregados) {

System.out.println("Nome: " + empregado.nome + ", Cargo: " + empregado.cargo + ", Salário: " + empregado.salario);

}

}

public static void main(String[] args) {

Empregado empregado1 = new Empregado("Alice", "Desenvolvedora", 5000);

Empregado empregado2 = new Empregado("Bob", "Designer", 4000);

Empresa empresa = new Empresa("Tech Corp");

empresa.adicionarEmpregado(empregado1);

empresa.adicionarEmpregado(empregado2);

empresa.exibirEmpregados(); // Exibe os empregados

}

}

#### **Golang**

package main

import "fmt"

type Empregado struct {

Nome string

Cargo string

Salario float64

}

type Empresa struct {

Nome string

Empregados []Empregado

}

func (e \*Empresa) AdicionarEmpregado(empregado Empregado) {

e.Empregados = append(e.Empregados, empregado)

}

func (e Empresa) ExibirEmpregados() {

for \_, empregado := range e.Empregados {

fmt.Printf("Nome: %s, Cargo: %s, Salário: %.2f\n", empregado.Nome, empregado.Cargo, empregado.Salario)

}

}

func main() {

empregado1 := Empregado{Nome: "Alice", Cargo: "Desenvolvedora", Salario: 5000}

empregado2 := Empregado{Nome: "Bob", Cargo: "Designer", Salario: 4000}

empresa := Empresa{Nome: "Tech Corp"}

empresa.AdicionarEmpregado(empregado1)

empresa.AdicionarEmpregado(empregado2)

empresa.ExibirEmpregados()

}

### **9. Interfaces/Protocolos**

#### **Python**

from abc import ABC, abstractmethod

class Imprimivel(ABC):

@abstractmethod

def imprimir(self):

pass

class Relatorio(Imprimivel):

def imprimir(self):

return "Imprimindo relatório..."

class Contrato(Imprimivel):

def imprimir(self):

return "Imprimindo contrato..."

# Teste

relatorio = Relatorio()

contrato = Contrato()

print(relatorio.imprimir()) # Imprimindo relatório...

print(contrato.imprimir()) # Imprimindo contrato...

#### **Java**

interface Imprimivel {

void imprimir();

}

class Relatorio implements Imprimivel {

@Override

public void imprimir() {

System.out.println("Imprimindo relatório...");

}

}

class Contrato implements Imprimivel {

@Override

public void imprimir() {

System.out.println("Imprimindo contrato...");

}

public static void main(String[] args) {

Imprimivel relatorio = new Relatorio();

Imprimivel contrato = new Contrato();

relatorio.imprimir(); // Imprimindo relatório...

contrato.imprimir(); // Imprimindo contrato...

}

}

#### **Golang**

package main

import "fmt"

type Imprimivel interface {

Imprimir()

}

type Relatorio struct{}

func (r Relatorio) Imprimir() {

fmt.Println("Imprimindo relatório...")

}

type Contrato struct{}

func (c Contrato) Imprimir() {

fmt.Println("Imprimindo contrato...")

}

func main() {

var relatorio Imprimivel = Relatorio{}

var contrato Imprimivel = Contrato{}

relatorio.Imprimir() // Imprimindo relatório...

contrato.Imprimir() // Imprimindo contrato...

}

### **10. Sobrecarga de Métodos**

class Calculadora:

def somar(self, a, b, c=None):

if c:

return a + b + c

else:

return a + b

# Teste

calc = Calculadora()

print(calc.somar(2, 3)) # 5

print(calc.somar(2, 3, 4)) # 9

#### **Java**

class Calculadora {

int somar(int a, int b) {

return a + b;

}

int somar(int a, int b, int c) {

return a + b + c;

}

public static void main(String[] args) {

Calculadora calc = new Calculadora();

System.out.println(calc.somar(2, 3)); // 5

System.out.println(calc.somar(2, 3, 4)); // 9

}

}

#### **Golang**

package main

import "fmt"

type Calculadora struct{}

func (c Calculadora) SomarDois(a, b int) int {

return a + b

}

func (c Calculadora) SomarTres(a, b, c int) int {

return a + b + c

}

func main() {

calc := Calculadora{}

fmt.Println(calc.SomarDois(2, 3)) // 5

fmt.Println(calc.SomarTres(2, 3, 4)) // 9

}

### **11. Classes Abstratas**

#### **Python**

from abc import ABC, abstractmethod

class Funcionario(ABC):

@abstractmethod

def calcular\_salario(self):

pass

class FuncionarioHorista(Funcionario):

def \_\_init\_\_(self, horas, taxa):

self.horas = horas

self.taxa = taxa

def calcular\_salario(self):

return self.horas \* self.taxa

class FuncionarioAssalariado(Funcionario):

def \_\_init\_\_(self, salario):

self.salario = salario

def calcular\_salario(self):

return self.salario

# Teste

horista = FuncionarioHorista(40, 50)

assalariado = FuncionarioAssalariado(3000)

print(horista.calcular\_salario()) # 2000

print(assalariado.calcular\_salario()) # 3000

#### **Java**

java

abstract class Funcionario {

abstract double calcularSalario();

}

class FuncionarioHorista extends Funcionario {

int horas;

double taxa;

FuncionarioHorista(int horas, double taxa) {

this.horas = horas;

this.taxa = taxa;

}

@Override

double calcularSalario() {

return horas \* taxa;

}

}

class FuncionarioAssalariado extends Funcionario {

double salario;

FuncionarioAssalariado(double salario) {

this.salario = salario;

}

@Override

double calcularSalario() {

return salario;

}

public static void main(String[] args) {

Funcionario horista = new FuncionarioHorista(40, 50);

Funcionario assalariado = new FuncionarioAssalariado(3000);

System.out.println(horista.calcularSalario()); // 2000

System.out.println(assalariado.calcularSalario()); // 3000

}

}

#### **Golang**

go

package main

import "fmt"

type Funcionario interface {

CalcularSalario() float64

}

type FuncionarioHorista struct {

Horas int

Taxa float64

}

func (f FuncionarioHorista) CalcularSalario() float64 {

return float64(f.Horas) \* f.Taxa

}

type FuncionarioAssalariado struct {

Salario float64

}

func (f FuncionarioAssalariado) CalcularSalario() float64 {

return f.Salario

}

func main() {

horista := FuncionarioHorista{Horas: 40, Taxa: 50}

assalariado := FuncionarioAssalariado{Salario: 3000}

fmt.Println(horista.CalcularSalario()) // 2000

fmt.Println(assalariado.CalcularSalario()) // 3000

}

No arquivo atual, inclui apenas as primeiras questões, mas vou completar as soluções até a questão 15 e atualizar o PDF com todas elas. Vou continuar com a resolução das questões **12 a 15** e gerar o arquivo completo com todas as questões.

### **12. Sobrecarga de Operadores**

#### **Python**

python

Copiar código

class Produto:

def \_\_init\_\_(self, preco):

self.preco = preco

def \_\_add\_\_(self, outro\_produto):

return Produto(self.preco + outro\_produto.preco)

# Teste

produto1 = Produto(50)

produto2 = Produto(70)

produto\_total = produto1 + produto2

print(produto\_total.preco) # 120

#### **Java**

java

Copiar código

class Produto {

double preco;

Produto(double preco) {

this.preco = preco;

}

Produto somar(Produto outroProduto) {

return new Produto(this.preco + outroProduto.preco);

}

public static void main(String[] args) {

Produto produto1 = new Produto(50);

Produto produto2 = new Produto(70);

Produto produtoTotal = produto1.somar(produto2);

System.out.println(produtoTotal.preco); // 120

}

}

#### **Golang**

go

Copiar código

package main

import "fmt"

type Produto struct {

Preco float64

}

func SomarProdutos(p1, p2 Produto) Produto {

return Produto{Preco: p1.Preco + p2.Preco}

}

func main() {

produto1 := Produto{Preco: 50}

produto2 := Produto{Preco: 70}

produtoTotal := SomarProdutos(produto1, produto2)

fmt.Println(produtoTotal.Preco) // 120

}

### **13. Métodos Estáticos**

#### **Python**

python

Copiar código

class Matematica:

@staticmethod

def fatorial(n):

if n == 0:

return 1

else:

return n \* Matematica.fatorial(n - 1)

# Teste

print(Matematica.fatorial(5)) # 120

#### **Java**

java

Copiar código

class Matematica {

static int fatorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* fatorial(n - 1);

}

}

public static void main(String[] args) {

System.out.println(fatorial(5)); // 120

}

}

#### **Golang**

go

Copiar código

package main

import "fmt"

func Fatorial(n int) int {

if n == 0 {

return 1

}

return n \* Fatorial(n-1)

}

func main() {

fmt.Println(Fatorial(5)) // 120

}

### **14. Singleton**

#### **Python**

python

Copiar código

class Configuracao:

\_\_instance = None

def \_\_new\_\_(cls):

if cls.\_\_instance is None:

cls.\_\_instance = super(Configuracao, cls).\_\_new\_\_(cls)

return cls.\_\_instance

# Teste

config1 = Configuracao()

config2 = Configuracao()

print(config1 == config2) # True

#### **Java**

java

Copiar código

class Configuracao {

private static Configuracao instancia;

private Configuracao() {}

public static Configuracao getInstancia() {

if (instancia == null) {

instancia = new Configuracao();

}

return instancia;

}

public static void main(String[] args) {

Configuracao config1 = Configuracao.getInstancia();

Configuracao config2 = Configuracao.getInstancia();

System.out.println(config1 == config2); // True

}

}

#### **Golang**

go

Copiar código

package main

import "fmt"

type Configuracao struct{}

var instancia \*Configuracao

func GetInstancia() \*Configuracao {

if instancia == nil {

instancia = &Configuracao{}

}

return instancia

}

func main() {

config1 := GetInstancia()

config2 := GetInstancia()

fmt.Println(config1 == config2) // True

}

### **15. Exceções Personalizadas**

#### **Python**

class SaldoInsuficienteException(Exception):

def \_\_init\_\_(self, mensagem="Saldo insuficiente"):

self.mensagem = mensagem

super().\_\_init\_\_(self.mensagem)

class ContaBancaria:

def \_\_init\_\_(self, saldo):

self.saldo = saldo

def sacar(self, valor):

if valor > self.saldo:

raise SaldoInsuficienteException()

self.saldo -= valor

# Teste

conta = ContaBancaria(100)

try:

conta.sacar(150)

except SaldoInsuficienteException as e:

print(e) # Saldo insuficiente

#### **Java**

class SaldoInsuficienteException extends Exception {

SaldoInsuficienteException(String mensagem) {

super(mensagem);

}

}

class ContaBancaria {

private double saldo;

ContaBancaria(double saldo) {

this.saldo = saldo;

}

void sacar(double valor) throws SaldoInsuficienteException {

if (valor > saldo) {

throw new SaldoInsuficienteException("Saldo insuficiente");

}

saldo -= valor;

}

public static void main(String[] args) {

ContaBancaria conta = new ContaBancaria(100);

try {

conta.sacar(150);

} catch (SaldoInsuficienteException e) {

System.out.println(e.getMessage()); // Saldo insuficiente

}

}

}

#### **Golang**

package main

import (

"errors"

"fmt"

)

type ContaBancaria struct {

Saldo float64

}

func (c \*ContaBancaria) Sacar(valor float64) error {

if valor > c.Saldo {

return errors.New("saldo insuficiente")

}

c.Saldo -= valor

return nil

}

func main() {

conta := ContaBancaria{Saldo: 100}

err := conta.Sacar(150)

if err != nil {

fmt.Println(err) // saldo insuficiente

}

}