# 🧠 Herança + Construtores + Sobrecarga + Encapsulamento Solução

## 🧩Exercício 1 — Casa Inteligente (Smart Devices)

### 🎯Objetivo

Treinar os conceitos de **herança** (classe base e subclasses), **encapsulamento** (proteger atributos com private), e **sobrecarga** (usar o mesmo método com diferentes parâmetros).

O aluno vai criar um pequeno sistema que simula dispositivos inteligentes (lâmpadas, ventoinhas, termóstatos) controlados por uma classe "hub".

---

### ✅Solução possível

```
// ===== Classe base: Dispositivo =====
class Dispositivo {
    private final String nome;     // Encapsulado: imutável após construção
    private boolean ligado;

    public Dispositivo(String nome) {
        if (nome == null || nome.isBlank()) throw new IllegalArgumentException
("Nome inválido");
        this.nome = nome;
        this.ligado = false;
    }

    public void ligar()  { this.ligado = true; }
    public void desligar(){ this.ligado = false; }

    public boolean isLigado() { return ligado; }
```

```
    public String getNome()   { return nome; }

    @Override public String toString() {
        return getClass().getSimpleName() + "{" + "nome='" + nome + "', ligado
=" + ligado + "}";
    }
}

// ===== Subclasse: Lampada =====
class Lampada extends Dispositivo {
    private int intensidade; // 0..100

    public Lampada(String nome) {
        this(nome, 50); // sobrecarga: valor por defeito
    }
    public Lampada(String nome, int intensidade) {
        super(nome);
        setIntensidade(intensidade); // valida/clampa via setter centralizado
    }

    public void ajustarIntensidade(int nova) { setIntensidade(nova); }

    // Sobrecarga por String → mapeia modos para percentagens
    public void ajustarIntensidade(String modo) {
        if (modo == null) return;
        switch (modo.toLowerCase()) {
            case "baixo": setIntensidade(30); break;
            case "medio":
            case "médio": setIntensidade(60); break;
            case "alto":  setIntensidade(90); break;
            default: System.out.println("Modo de lâmpada inválido: " + modo);
        }
    }

    private void setIntensidade(int valor) {
        if (valor < 0)  valor = 0;
```

```
        if (valor > 100) valor = 100;
        this.intensidade = valor;
        if (!isLigado() && intensidade > 0) ligar(); // acende se > 0
    }

    @Override public String toString() {
        return "Lampada{nome='" + getNome() + "', ligado=" + isLigado() + ", int
ensidade=" + intensidade + "%}";
    }
}

// ===== Subclasse: Ventoinha =====
class Ventoinha extends Dispositivo {
    private int velocidade; // 0..3

    public Ventoinha(String nome) { this(nome, 0); }
    public Ventoinha(String nome, int velocidade) {
        super(nome);
        setVelocidade(velocidade);
    }

    public void setVelocidade(int valor) {
        if (valor < 0) valor = 0;
        if (valor > 3) valor = 3;
        this.velocidade = valor;
        if (valor == 0) desligar(); else ligar();
    }

    // Sobrecarga por String
    public void setVelocidade(String modo) {
        if (modo == null) return;
        switch (modo.toLowerCase()) {
            case "off":  setVelocidade(0); break;
            case "baixo":setVelocidade(1); break;
            case "medio":
            case "médio":setVelocidade(2); break;
```

```
            case "alto": setVelocidade(3); break;
            default: System.out.println("Modo de ventoinha inválido: " + modo);
        }
    }

    @Override public String toString() {
        return "Ventoinha{nome='" + getNome() + "', ligado=" + isLigado() + ", v
elocidade=" + velocidade + "}";
    }
}

// ===== Subclasse: Termostato =====
class Termostato extends Dispositivo {
    private double temperatura; // 10.0..30.0 °C

    public Termostato(String nome) { this(nome, 21.0); }
    public Termostato(String nome, double valor) {
        super(nome);
        setTemperatura(valor);
    }

    public void setTemperatura(double valor) {
        if (valor < 10.0) valor = 10.0;
        if (valor > 30.0) valor = 30.0;
        this.temperatura = Math.round(valor * 10.0) / 10.0; // 1 casa decimal
        ligar(); // ao definir um alvo, assume-se ligado
    }

    // Sobrecarga: aceita "21.5C" ou "70F"
    public void setTemperatura(String texto) {
        if (texto == null || texto.isBlank()) return;
        String s = texto.trim().toUpperCase();
        try {
            if (s.endsWith("C")) {
                setTemperatura(Double.parseDouble(s.substring(0, s.length() - 1)));
            } else if (s.endsWith("F")) {
```

```
            double f = Double.parseDouble(s.substring(0, s.length() - 1));
            double c = (f - 32) * 5.0 / 9.0;
            setTemperatura(c);
        } else {
            setTemperatura(Double.parseDouble(s)); // assume Celsius
        }
    } catch (NumberFormatException e) {
        System.out.println("Temperatura inválida: " + texto);
    }
  }

  @Override public String toString() {
    return "Termostato{nome='" + getNome() + "', ligado=" + isLigado() + ", t
emperatura=" + temperatura + "°C}";
  }
}

// ===== Controlador: RoomHub =====
class RoomHub {
  private final Lampada lampada;
  private final Ventoinha ventoinha;
  private final Termostato termostato;

  public RoomHub(Lampada l, Ventoinha v, Termostato t) {
    this.lampada = l;
    this.ventoinha = v;
    this.termostato = t;
  }

  public void aplicarCena(String nome) { aplicarCena(nome, 50); } // default

  public void aplicarCena(String nome, int intensidade) {
    String n = (nome == null) ? "" : nome.toLowerCase();
    switch (n) {
      case "filme":
        lampada.ajustarIntensidade(intensidade); // tipicamente 30
```

```
            ventoinha.setVelocidade(1);
            termostato.setTemperatura(21.0);
            break;
        case "foco":
            lampada.ajustarIntensidade(Math.max(60, intensidade));
            ventoinha.setVelocidade(2);
            termostato.setTemperatura(22.0);
            break;
        case "off":
        case "desligar":
            lampada.ajustarIntensidade(0);
            ventoinha.setVelocidade(0);
            termostato.desligar();
            break;
        default:
            System.out.println("Cena desconhecida: " + nome);
      }
      System.out.println("Cena \"" + nome + "\" aplicada com intensidade " + i
ntensidade);
   }

   @Override public String toString() {
      return lampada + "\n" + ventoinha + "\n" + termostato;
   }
}

// ===== Main: executa o cenário =====
public class SmartDevices {
   public static void main(String[] args) {
      Lampada l = new Lampada("Mesa", 120);      // fica 100
      Ventoinha v = new Ventoinha("Teto", "medio");
      Termostato t = new Termostato("Parede");
      t.setTemperatura("70F"); // ~21.1°C

      RoomHub hub = new RoomHub(l, v, t);
      hub.aplicarCena("Filme", 30);
```

```
        System.out.println(hub);
    }
}
```

---

# 💰 Exercício 2 — Conta Bancária + Conta Poupança

## 🎯Objetivo

Compreender como a **herança** permite criar diferentes tipos de contas e como os **construtores sobrecarregados** e **métodos sobrecarregados** ajudam a inicializar e manipular objetos de formas flexíveis.

Também treina **encapsulamento**, impedindo o acesso direto ao saldo.

---

### ✅ Solução possível

```
// ===== Classe base: Conta =====
class Conta {
    private final String titular;
    private double saldo;

    public Conta(String titular) { this(titular, 0.0); }

    public Conta(String titular, double valor) {
        if (titular == null || titular.isBlank()) throw new IllegalArgumentException
("Titular inválido");
        if (valor < 0) throw new IllegalArgumentException("Saldo inicial não pode
ser negativo");
        this.titular = titular;
        this.saldo = valor;
    }

    public void depositar(double valor) {
        if (valor <= 0) { System.out.println("Depósito inválido"); return; }
        saldo += valor;
```

```
        System.out.printf("Depósito de %.2f€ efetuado.%n", valor);
    }

    // Sobrecarga por int → delega
    public void depositar(int valor) { depositar((double) valor); }

    public void levantar(double valor) {
        if (valor <= 0) { System.out.println("Levantamento inválido"); return; }
        if (valor > saldo) { System.out.println("Fundos insuficientes"); return; }
        saldo -= valor;
        System.out.printf("Levantamento de %.2f€ efetuado.%n", valor);
    }

    public double getSaldo()   { return saldo; }
    public String getTitular() { return titular; }

    @Override public String toString() {
        return "Conta{titular='" + titular + "', saldo=" + String.format("%.2f", sald
o) + "}";
    }
}

// ===== Subclasse: ContaPoupanca =====
class ContaPoupanca extends Conta {
    private double taxaJuro; // ex: 0.03 = 3%/ano

    public ContaPoupanca(String titular) { this(titular, 0.0, 0.02); }

    public ContaPoupanca(String titular, double valor, double taxaJuro) {
        super(titular, valor);
        setTaxaJuro(taxaJuro);
    }

    public void setTaxaJuro(double taxa) {
        if (taxa < 0 || taxa > 0.20) throw new IllegalArgumentException("Taxa for
a de [0..0.20]");
```

```
        this.taxaJuro = taxa;
    }

    public void renderJuros() {
        double acrescimo = getSaldo() * taxaJuro;
        super.depositar(acrescimo); // reutiliza validação
        System.out.printf("Juros aplicados (%.2f%% ano).%n", taxaJuro * 100);
    }

    public void renderJuros(int meses) {
        if (meses <= 0) { System.out.println("Meses inválidos"); return; }
        double acrescimo = getSaldo() * (taxaJuro * (meses / 12.0));
        super.depositar(acrescimo);
        System.out.printf("Juros aplicados (%d meses, %.2f%% anual).%n", mes
es, taxaJuro * 100);
    }

    @Override public String toString() {
        return "ContaPoupanca{titular='" + getTitular() + "', saldo=" +
                String.format("%.2f", getSaldo()) + ", taxaJuro=" + taxaJuro + "}";
    }
}

// ===== Main: executa o cenário =====
public class Banco {
    public static void main(String[] args) {
        ContaPoupanca c = new ContaPoupanca("Ana", 100, 0.03);
        c.depositar(50);
        c.levantar(30);
        c.renderJuros(12);
        System.out.println(c);
    }
}
```

# 🎬 Exercício 3 — Bilhetes de Cinema

## 🎯 Objetivo

Compreender **sobrecarga de métodos** e **herança** criando diferentes formas de calcular o preço de um bilhete com base na idade, estatuto de estudante ou tipo de bilhete.

---

## ✅ Solução possível

```
// ===== Classe base: Bilhete =====
class Bilhete {
    private final double base; // preço base (mínimo 3.0)

    public Bilhete(double base) {
        if (base < 3.0) throw new IllegalArgumentException("Preço base mínimo
é 3.0€");
        this.base = base;
    }

    public double getBase() { return base; }

    // Sobrecarga 1: só idade
    public double preco(int idade) {
        return max3(calcularBase(idade, false));
    }

    // Sobrecarga 2: só estudante
    public double preco(boolean estudante) {
        return max3(calcularBase(-1, estudante)); // idade desconhecida → ignor
a descontos de idade
    }

    // Sobrecarga 3: idade + estudante
    public double preco(int idade, boolean estudante) {
        return max3(calcularBase(idade, estudante));
```

```
    }

    // Lógica comum centralizada
    protected double calcularBase(int idade, boolean estudante) {
        double preco = base;

        if (idade >= 0) {
            if (idade < 12)       preco *= 0.5; // -50%
            else if (idade >= 65) preco *= 0.7; // -30%
        }

        if (estudante) preco *= 0.8; // -20%

        return preco;
    }

    protected double max3(double valor) { return (valor < 3.0) ? 3.0 : round2(va
lor); }

    protected double round2(double v) { return Math.round(v * 100.0) / 100.0; }

    @Override public String toString() {
        return "Bilhete{base=" + base + "}";
    }
}

// ===== Subclasse: BilheteEstudante =====
class BilheteEstudante extends Bilhete {
    private final double descontoExtra = 0.5; // -0.5€ adicional (mínimo 3€)

    public BilheteEstudante(double base) { super(base); }

    @Override
    protected double calcularBase(int idade, boolean estudante) {
        double p = super.calcularBase(idade, true); // força estudante=true
        p = p - descontoExtra;
```

```
        return (p < 3.0) ? 3.0 : round2(p);
    }
}

// ===== Main: executa o cenário =====
public class Cinema {
    public static void main(String[] args) {
        Bilhete b = new Bilhete(8.0);
        System.out.println(b.preco(10));          // criança: 4.0
        System.out.println(b.preco(true));        // estudante: 6.4
        System.out.println(b.preco(70, true));    // sénior + estudante: 4.48

        BilheteEstudante be = new BilheteEstudante(8.0);
        System.out.println(be.preco(20, true));   // desconto extra (mínimo 3.0)
    }
}// ===== Classe base: Conta =====
class Conta {
    private final String titular;
    private double saldo;

    public Conta(String titular) { this(titular, 0.0); }

    public Conta(String titular, double valor) {
        if (titular == null || titular.isBlank()) throw new IllegalArgumentException
("Titular inválido");
        if (valor < 0) throw new IllegalArgumentException("Saldo inicial não pode
ser negativo");
        this.titular = titular;
        this.saldo = valor;
    }

    public void depositar(double valor) {
        if (valor <= 0) { System.out.println("Depósito inválido"); return; }
        saldo += valor;
        System.out.printf("Depósito de %.2f€ efetuado.%n", valor);
    }
```

```
    // Sobrecarga por int → delega
    public void depositar(int valor) { depositar((double) valor); }

    public void levantar(double valor) {
        if (valor <= 0) { System.out.println("Levantamento inválido"); return; }
        if (valor > saldo) { System.out.println("Fundos insuficientes"); return; }
        saldo -= valor;
        System.out.printf("Levantamento de %.2f€ efetuado.%n", valor);
    }

    public double getSaldo()   { return saldo; }
    public String getTitular() { return titular; }

    @Override public String toString() {
        return "Conta{titular='" + titular + "', saldo=" + String.format("%.2f", sald
o) + "}";
    }
}

// ===== Subclasse: ContaPoupanca =====
class ContaPoupanca extends Conta {
    private double taxaJuro; // ex: 0.03 = 3%/ano

    public ContaPoupanca(String titular) { this(titular, 0.0, 0.02); }

    public ContaPoupanca(String titular, double valor, double taxaJuro) {
        super(titular, valor);
        setTaxaJuro(taxaJuro);
    }

    public void setTaxaJuro(double taxa) {
        if (taxa < 0 || taxa > 0.20) throw new IllegalArgumentException("Taxa for
a de [0..0.20]");
        this.taxaJuro = taxa;
    }
```

```
    public void renderJuros() {
        double acrescimo = getSaldo() * taxaJuro;
        super.depositar(acrescimo); // reutiliza validação
        System.out.printf("Juros aplicados (%.2f%% ano).%n", taxaJuro * 100);
    }

    public void renderJuros(int meses) {
        if (meses <= 0) { System.out.println("Meses inválidos"); return; }
        double acrescimo = getSaldo() * (taxaJuro * (meses / 12.0));
        super.depositar(acrescimo);
        System.out.printf("Juros aplicados (%d meses, %.2f%% anual).%n", mes
es, taxaJuro * 100);
    }

    @Override public String toString() {
        return "ContaPoupanca{titular='" + getTitular() + "', saldo=" +
                String.format("%.2f", getSaldo()) + ", taxaJuro=" + taxaJuro + "}";
    }
}

// ===== Main: executa o cenário =====
public class Banco {
    public static void main(String[] args) {
        ContaPoupanca c = new ContaPoupanca("Ana", 100, 0.03);
        c.depositar(50);
        c.levantar(30);
        c.renderJuros(12);
        System.out.println(c);
    }
}// ===== Classe base: Conta =====
class Conta {
    private final String titular;
    private double saldo;

    public Conta(String titular) { this(titular, 0.0); }
```

```
    public Conta(String titular, double valor) {
        if (titular == null || titular.isBlank()) throw new IllegalArgumentException
("Titular inválido");
        if (valor < 0) throw new IllegalArgumentException("Saldo inicial não pode
ser negativo");
        this.titular = titular;
        this.saldo = valor;
    }

    public void depositar(double valor) {
        if (valor <= 0) { System.out.println("Depósito inválido"); return; }
        saldo += valor;
        System.out.printf("Depósito de %.2f€ efetuado.%n", valor);
    }

    // Sobrecarga por int → delega
    public void depositar(int valor) { depositar((double) valor); }

    public void levantar(double valor) {
        if (valor <= 0) { System.out.println("Levantamento inválido"); return; }
        if (valor > saldo) { System.out.println("Fundos insuficientes"); return; }
        saldo -= valor;
        System.out.printf("Levantamento de %.2f€ efetuado.%n", valor);
    }

    public double getSaldo()   { return saldo; }
    public String getTitular() { return titular; }

    @Override public String toString() {
        return "Conta{titular='" + titular + "', saldo=" + String.format("%.2f", sald
o) + "}";
    }
}

// ===== Subclasse: ContaPoupanca =====
```

```
class ContaPoupanca extends Conta {
    private double taxaJuro; // ex: 0.03 = 3%/ano

    public ContaPoupanca(String titular) { this(titular, 0.0, 0.02); }

    public ContaPoupanca(String titular, double valor, double taxaJuro) {
        super(titular, valor);
        setTaxaJuro(taxaJuro);
    }

    public void setTaxaJuro(double taxa) {
        if (taxa < 0 || taxa > 0.20) throw new IllegalArgumentException("Taxa for
a de [0..0.20]");
        this.taxaJuro = taxa;
    }

    public void renderJuros() {
        double acrescimo = getSaldo() * taxaJuro;
        super.depositar(acrescimo); // reutiliza validação
        System.out.printf("Juros aplicados (%.2f%% ano).%n", taxaJuro * 100);
    }

    public void renderJuros(int meses) {
        if (meses <= 0) { System.out.println("Meses inválidos"); return; }
        double acrescimo = getSaldo() * (taxaJuro * (meses / 12.0));
        super.depositar(acrescimo);
        System.out.printf("Juros aplicados (%d meses, %.2f%% anual).%n", mes
es, taxaJuro * 100);
    }

    @Override public String toString() {
        return "ContaPoupanca{titular='" + getTitular() + "', saldo=" +
                String.format("%.2f", getSaldo()) + ", taxaJuro=" + taxaJuro + "}";
    }
}
```

```
// ===== Main: executa o cenário =====
public class Banco {
    public static void main(String[] args) {
        ContaPoupanca c = new ContaPoupanca("Ana", 100, 0.03);
        c.depositar(50);
        c.levantar(30);
        c.renderJuros(12);
        System.out.println(c);
    }
}
```

---

## 1️⃣ Herança (extends)

- Permite **reutilizar código** de uma classe base.
- Cria uma relação **"é um"** → uma Lampada **é um** Dispositivo.
- A subclasse herda métodos e atributos e pode **acrescentar ou adaptar** comportamentos.

💡 *Exemplo:*

```
class ContaPoupanca extends Conta {
    // herda depositar(), levantar() e adiciona renderJuros()
}
```

---

## 2️⃣ Encapsulamento (private, get, set)

- **Protege os dados** internos do objeto.
- Só métodos controlados podem alterar os atributos.
- Evita estados inválidos (ex: saldo negativo, intensidade > 100).

💡 *Exemplo:*

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

---

## 3️⃣ Construtores sobrecarregados

- Permitem criar objetos com **diferentes formas de inicialização**.
- Todos devem garantir que o objeto começa num **estado válido**.

💡 *Exemplo:*

```
public Lampada(String nome) { this(nome, 50); }
public Lampada(String nome, int intensidade) { ... }
```

---

## 4️⃣ Sobrecarga de métodos

- Mesmo nome, **diferentes parâmetros**.
- Torna a API mais flexível e legível.
- Evita repetir lógica — as versões devem **delegar** a uma implementação central.

💡 *Exemplo:*

```
public void setVelocidade(int v);
public void setVelocidade(String modo); // "baixo", "medio", "alto"
```