Semana 2 — Herança + Construtores

Revisão orientada a objetos e inicialização de objetos em Java

Programação II • ERSC • 2025/26

Objetivos da Sessão

- Relembrar classe, objeto, instância e herança (extends)
- Diferenciar "é um" (herança) e "tem um" (composição)
- Introduzir construtores e o seu propósito
- Preparar terreno para sobrecarga e encapsulamento

Nota: Aula focada em compreensão e exemplos curtos.

Revisão: Classe, Objeto, Instância e Herança

- **Classe** = molde (o que é + o que faz)
- **Objeto** = instância concreta (tem estado)
- **Instanciar** = criar com new
- **Herança** = reaproveitar código de outra classe (extends)

```
class Lampada {
    boolean ligada;

    void ligar() {
        ligada = true;
    }
}

class SmartLamp extends Lampada
{
    void ajustarIntensidade(int
v) {
        // ...
    }
}
```

Estado (ligada) + Comportamento (ligar) + Herança (SmartLamp)

O que é Herança?

- Reutiliza código comum entre classes
- Subclasse (filho) estende Superclasse (pai)
- Adiciona/Especializa comportamentos

"É um" vs "Tem um"

É um (Herança)

- Animal → Mamífero → Cão
- Dispositivo → Lampada
- Veículo → Carro

Tem um (Composição)

- Carro tem Motor
- Casa tem Lampada
- Computador tem Processador

Exemplo em Java (Herança)

```
class Dispositivo {
    void ligar() {
        System.out.println("Ligado!");
    }

    void desligar() {
        System.out.println("Desligado!");
    }
}

class Lampada extends Dispositivo {
    int intensidade;

    void ajustarIntensidade(int v) {
        intensidade = v;
    }
}
```

💡 Lampada herda ligar()/desligar() de Dispositivo

O que é Herdado? O que é Novo?

Herdado da superclasse

- Métodos públicos/protegidos
- Atributos acessíveis
- Comportamento base comum

Novo na subclasse

- Campos específicos (ex.: intensidade)
- Métodos específicos (ex.: ajustarIntensidade)
- Especialização do comportamento

"De onde vem ligar() quando chamo em Lampada?"

Benefícios e Cuidados com Herança

✓ Benefícios

- Menos duplicação de código
- Organização clara do código
- Manutenção facilitada

⚠ Cuidados

- Nem tudo precisa de extends
- Hierarquias profundas dificultam mudanças
- Prefere composição quando fizer mais sentido

Problema da Inicialização Manual

Criar e depois configurar, repetidamente:

```
Lampada l = new Lampada();
l.ligar();
l.ajustarIntensidade(70);
```

⚠️ Risco de esquecer passos ou criar estados inconsistentes

Construtores: Ideia e Sintaxe

- Método especial com o nome da classe
- Executa automaticamente no new
- Define o estado inicial do objeto

```
class Lampada extends Dispositivo {
    int intensidade;

    Lampada(int valorInicial) {
        intensidade = valorInicial;
    }
}
```

Exemplo Prático com Construtor

Antes

```
Lampada l = new Lampada();
l.ajustarIntensidade(70);
```

Depois

```
Lampada l = new
Lampada(70);
```

✓ Mais claro, menos passos; objetos "nascem prontos"

Regras e Boas Práticas de Construtores

- Mesmo nome da classe; não tem tipo de retorno
- Se não definires, o Java cria um construtor vazio
- Podes ter vários (SOBRECARGA) — próxima aula
- Usa `this` para distinguir parâmetros de campos

Quiz/Discussão Rápida

1. "Uma Casa é um Dispositivo?" (Porquê?)

2. "O construtor é herdado?" (Explica.)

3. "Prefiro composição ou herança neste caso: Carro e Motor?"