

Escopo de Classe, Controle de Acesso e Construtores

Onde as variáveis vivem e como controlar a criação de objetos em Java

1. Introdução

Escrever uma classe em Java envolve três decisões fundamentais que costumam aparecer juntas: **onde** cada variável pode ser vista e utilizada (seu **escopo**), **quem** pode acessar cada atributo ou método (o **controle de acesso**) e **como** os objetos dessa classe serão criados e inicializados (os **construtores**). Este texto aborda os três temas em conjunto, com exemplos práticos em Java.

2. Escopo de Classe

O **escopo** de uma variável define a região do código em que ela existe e pode ser utilizada. Em Java, os escopos mais comuns são: **escopo de classe** (atributos), **escopo de método** (variáveis locais) e **escopo de bloco** (variáveis declaradas dentro de { }, como em um for ou if).

2.1. Atributos (Escopo de Classe/Instância)

Atributos são declarados diretamente no corpo da classe, fora de qualquer método. Eles existem enquanto o objeto existir e podem ser acessados por qualquer método da própria classe.

```
public class ContaBancaria {  
    private double saldo; // atributo: escopo de classe/instância  
  
    public void depositar(double valor) {  
        saldo += valor; // acessível em qualquer método da classe  
    }  
  
    public void exibirSaldo() {  
        System.out.println("Saldo: " + saldo); // também acessível aqui  
    }  
}
```

2.2. Variáveis Locais (Escopo de Método)

Variáveis declaradas dentro de um método existem apenas durante a execução desse método e não podem ser acessadas de fora dele — nem mesmo por outros métodos da mesma classe.

```
public class Calculadora {  
    public int somar(int a, int b) {  
        int resultado = a + b; // variável local: só existe dentro de somar()  
        return resultado;  
    }  
  
    public void exibir() {  
        // System.out.println(resultado); // ERRO: resultado não existe aqui  
    }  
}
```

2.3. Variáveis de Bloco (Escopo de Bloco)

Variáveis declaradas dentro de um bloco delimitado por chaves (como o corpo de um `for`, `while` ou `if`) só existem dentro desse bloco específico.

```
public class Exemplo {
    public void processar() {
        for (int i = 0; i < 3; i++) {
            int quadrado = i * i; // escopo de bloco: só existe dentro do for
            System.out.println(quadrado);
        }
        // System.out.println(i);          // ERRO: i não existe fora do for
        // System.out.println(quadrado);    // ERRO: quadrado não existe fora do for
    }
}
```

2.4. Ocultação de Variáveis (Shadowing)

Quando uma variável local (ou parâmetro) tem o **mesmo nome** de um atributo da classe, ela "esconde" o atributo dentro daquele escopo — fenômeno chamado de *shadowing*. Para acessar o atributo da classe nesse caso, é necessário usar `this`.

```
public class Produto {
    private String nome;

    public void setNome(String nome) { // parâmetro "nome" oculta o atributo "nome"
        this.nome = nome; // this.nome refere-se ao atributo da classe
        // nome, sem "this", refere-se ao parâmetro
    }
}
```

2.5. Escopo static vs Escopo de Instância

Atributos e métodos `static` pertencem à **classe** como um todo, sendo compartilhados por todos os objetos. Já atributos e métodos de instância pertencem a cada **objeto individualmente**, cada um com sua própria cópia dos dados.

```
public class Funcionario {
    static int totalFuncionarios = 0; // compartilhado por todos os objetos
    private String nome;              // exclusivo de cada objeto

    public Funcionario(String nome) {
        this.nome = nome;
        totalFuncionarios++; // acessa o atributo static da classe
    }
}

new Funcionario("Ana");
new Funcionario("Bruno");
System.out.println(Funcionario.totalFuncionarios); // 2
```

3. Controle de Acesso

Além de definir onde uma variável existe, é preciso controlar **quem** pode enxergá-la ou utilizá-la fora da classe. Isso é feito por meio dos **modificadores de acesso**: `private`, `default` (sem

modificador), protected e public.

Modificador	Mesma classe	Mesmo pacote	Subclasse (outro pacote)	Qualquer lugar
private	Sim	Não	Não	Não
default	Sim	Sim	Não	Não
protected	Sim	Sim	Sim	Não
public	Sim	Sim	Sim	Sim

```
public class ContaBancaria {  
    private double saldo;           // só a própria classe acessa  
  
    public double getSaldo() {      // qualquer classe pode chamar  
        return saldo;  
    }  
  
    protected void aplicarTaxa(double taxa) { // subclasses podem usar  
        saldo -= taxa;  
    }  
}
```

Uma boa prática é manter os atributos como private (protegendo o **escopo de dados** da classe) e expor apenas os métodos necessários como public, reforçando o encapsulamento.

4. Construtores

O **construtor** é o método especial responsável por inicializar um objeto assim que ele é criado com new. Ele tem o mesmo nome da classe e não possui tipo de retorno. Assim como atributos e métodos, os construtores também podem receber modificadores de acesso, controlando quem pode criar objetos daquela classe.

4.1. Construtor Padrão e Construtor com Parâmetros

```
public class Produto {  
    private String nome;  
    private double preco;  
  
    // Construtor padrão (sem parâmetros)  
    public Produto() {  
        this.nome = "Sem nome";  
        this.preco = 0.0;  
    }  
  
    // Construtor com parâmetros  
    public Produto(String nome, double preco) {  
        this.nome = nome;  
        this.preco = preco;  
    }  
}  
  
Produto p1 = new Produto();  
Produto p2 = new Produto("Caderno", 15.90);
```

4.2. Encadeamento com this() e Controle de Acesso do Construtor

Construtores podem ser encadeados com `this(...)` para reaproveitar lógica de inicialização. Além disso, é possível declarar um construtor como `private`, impedindo que objetos sejam criados diretamente fora da classe — uma técnica usada, por exemplo, no padrão de projeto *Singleton*.

```
public class Configuracao {
    private static Configuracao instancia;
    private String ambiente;

    private Configuracao(String ambiente) { // construtor privado
        this.ambiente = ambiente;
    }

    public static Configuracao getInstancia() {
        if (instancia == null) {
            instancia = new Configuracao("producao");
        }
        return instancia;
    }
}

// Configuracao c = new Configuracao("teste"); // ERRO: construtor é private
Configuracao c = Configuracao.getInstancia(); // forma correta de obter o objeto
```

5. Exemplo Integrado

O exemplo a seguir combina escopo de classe, controle de acesso e construtores em uma única classe `Funcionario`, com um contador estático, atributos privados e um construtor que valida os dados recebidos.

```

public class Funcionario {
    private static int totalContratados = 0; // escopo de classe (compartilhado)

    private String nome; // escopo de instância, acesso privado
    private double salario;

    public Funcionario(String nome, double salario) {
        if (salario < 0) {
            throw new IllegalArgumentException("Salário não pode ser negativo.");
        }
        this.nome = nome;
        this.salario = salario;
        totalContratados++; // atualiza o contador compartilhado
    }

    public String getNome() {
        return nome; // acesso controlado ao atributo privado
    }

    public static int getTotalContratados() {
        return totalContratados;
    }
}

public class Main {
    public static void main(String[] args) {
        new Funcionario("Ana", 3500.0);
        new Funcionario("Bruno", 4200.0);

        System.out.println("Total de funcionários: " + Funcionario.getTotalContratados());
    }
}

```

6. Conclusão

Escopo, controle de acesso e construtores atuam juntos na definição de uma classe bem projetada: o escopo determina onde cada variável pode ser usada; os modificadores de acesso decidem quem pode enxergar atributos e métodos de fora da classe; e os construtores garantem que cada objeto nasça em um estado válido, já respeitando essas regras de visibilidade. Compreender como esses três conceitos se relacionam é essencial para escrever classes Java seguras, previsíveis e fáceis de manter.