Programação Avançada - AULA 05

Matheus Moresco
Engenharia de Software - 5º Período
2025/01

Modificadores de Acesso em Java

- Os **modificadores de acesso** em Java determinam a visibilidade e o escopo de classes, métodos e atributos dentro de um programa.
- Eles são essenciais para a aplicação dos princípios de **encapsulamento**, garantindo que partes do código possam ser acessadas ou restringidas conforme necessário.

Modificadores de Acesso em Java

- Os modificadores de acesso em Java são definidos pelos nomes:
  - public
  - private
  - protected
  - Sem Modificador - default

Public

- Permite acesso de qualquer lugar do código.
- Ideal para métodos e atributos que devem ser acessíveis globalmente.

```
public class MinhaClasse {
    public String nome = "Exemplo"; // Pode ser acessado de qualquer lugar

    public void mostrarMensagem() {
        System.out.println("Método público acessível de qualquer lugar!");
    }
}
```

Private

- Restringe o acesso ao próprio escopo da classe.
- Muito utilizado para encapsular dados e garantir que sejam acessados/modificados apenas por métodos da própria classe.

```
public class MinhaClasse {
    private String segredo = "Não pode ser acessado diretamente";

    private void metodoPrivado() {
        System.out.println("Só pode ser chamado dentro desta classe.");
    }
}
```

Private

- Para permitir acesso controlado, utilizamos métodos **getter** e **setter**:

```
public class Pessoa {
    private String nome;

    public String getNome() {  // Getter para acessar o atributo
        return nome;
    }

    public void setNome(String nome) {  // Setter para modificar o atributo
        this.nome = nome;
    }
}
```

Private

- A vantagem do uso dos métodos **getter** e **setter** é que podemos fazer validações adicionais antes de atribuir valores aos atributos

```
public class Pessoa {
    private String nome;

    public String getNome() {  // Getter para acessar o atributo
        return nome;
    }

    public void setNome(String nome) {  // Setter para modificar o atributo
        if (name.length() < 3){
            System.out.println("O nome precisa ter pelo menos 3 caracteres.");
        }else{
            this.nome = nome;
        }

    }
}
```

Protected

- Permite o acesso dentro do mesmo pacote e também por subclasses (mesmo que estejam em pacotes diferentes).
- Comumente usado em herança.

```
class Animal {
    protected String especie = "Felino";
}

class Gato extends Animal {
    public void mostrarEspecie() {
        System.out.println("O gato é um " + especie); // Pode acessar pois é protected
    }
}
```

Default

- O acesso é permitido apenas dentro do mesmo pacote.
- Se um atributo ou método não tiver um modificador explícito, ele terá esse nível de acesso.

```
class MinhaClasse {
    String mensagem = "Visível apenas dentro do mesmo pacote";
}
```

Default

- Caso o atributo seja chamado no mesmo pacote, será permitido

```java
package PacoteA;

class MinhaClasse { // Sem modificador (default)
    String mensagem = "Acessível apenas dentro do mesmo pacote!";

    void exibirMensagem() { // Método sem modificador (default)
        System.out.println(mensagem);
    }
}
```

```java
package PacoteA;

public class OutraClasse {
    public static void main(String[] args) {
        MinhaClasse obj = new MinhaClasse(); // OK, pois está no mesmo pacote
        obj.exibirMensagem(); // OK, método é package-private
    }
}
```

Default

- Caso o método ou o atributo sejam usados em outro pacote, a ação não será permitida.

```java
package PacoteA;

class MinhaClasse { // Sem modificador (default)
    String mensagem = "Acessível apenas dentro do mesmo pacote!";

    void exibirMensagem() { // Método sem modificador (default)
        System.out.println(mensagem);
    }
}
```

```java
package PacoteB;
import PacoteA.MinhaClasse; // ❌ Erro! MinhaClasse não é pública

public class Teste {
    public static void main(String[] args) {
        MinhaClasse obj = new MinhaClasse(); // ❌ Erro! Não pode ser acessado fora do pacote
        obj.exibirMensagem(); // ❌ Erro!
    }
}
```

Resumo

Modificador	Mesma Classe	Mesmo Pacote	Subclasse (herança)	Qualquer Classe
public	✓ Sim	✓ Sim	✓ Sim	✓ Sim
protected	✓ Sim	✓ Sim	✓ Sim	✗ Não
(sem modificador - "default")	✓ Sim	✓ Sim	✗ Não	✗ Não
private	✓ Sim	✗ Não	✗ Não	✗ Não

Resumo do Uso Ideal

public → Para funcionalidades que devem ser amplamente acessíveis.

private → Para ocultar detalhes da implementação e proteger os dados.

protected → Para permitir acesso controlado dentro da hierarquia de classes.

(**sem modificador - default**) → Quando só precisa de acesso dentro do mesmo pacote.

Atributos de Instância (Non-static Fields)

- São **declarados dentro da classe**, mas **fora dos métodos**.
- Pertencem a **cada objeto** da classe (cada instância tem sua própria cópia).
- Devem ser acessados através de um objeto.

Atributos de Classe (static)

- São declarados com static, pertencendo **à classe e não a objetos individuais**.
- Compartilhados entre todas as instâncias da classe.
- Podem ser acessados **diretamente pelo nome da classe**.

```
class Config {
    static String sistema = "Java System"; // Compartilhado por todos os objetos
}
```

Atributos Locais

- São declarados **dentro de um método, construtor ou bloco**.
- Só existem **durante a execução** do método e não têm modificadores como static ou final.
- Devem ser inicializados antes do uso.

```
class Teste {
    void metodo() {
        int numero = 10; // Variável local
        System.out.println(numero);
    }
}
```

Exemplo Prático

1. Criamos uma classe **Carro** para representar veículos
2. Modificadores de Acesso:
   - **private**: Restringe acesso direto a dados sensíveis (chassi).
   - **protected**: Permite que subclasses acessem tipoVeiculo.
   - **public**: Permite acesso global a métodos como acelerar(), frear().
   - (Sem modificador - default): Permite acesso apenas dentro do mesmo pacote.
3. Atributos e Métodos static:
   - **static final int velociadadeMaxima**: Indica a velocidade máxima permitida para criação de um carro
   - **Static int getVelocidadeMaxima()**: Retorna o valor da velocidade máxima para criação de um carro.

Exercício

1. Nas classes:
    1. ContaCorrente
        a) Adicionar o atributo tarifa, é um número real e é sempre o mesmo para todas as contas. Torna-lo estático.
    2. ContaPoupanca
        a) Adicionar o atributo rendimentoMensal, que informa a porcentagem que a conta rende no mês. Este valor é sempre o mesmo para todas as contas. Torna-lo estático.
2. Criar atributos nas ContaBancaria:
    1. Titular: Nome do Titular da conta.
    2. Agencia: Número da agencia
    3. Conta: núemro da conta.