

# Métodos de Acesso: Set e Get

*Encapsulamento e controle de atributos por meio de getters e setters em Java*

---

## 1. Introdução

Em Java, é uma boa prática declarar os atributos de uma classe como `private`, impedindo que sejam acessados diretamente por outras classes. Para permitir, ainda assim, que esses atributos sejam consultados ou alterados de forma controlada, utilizam-se os chamados **métodos de acesso**: o **get** (também chamado de *getter*), usado para **ler** o valor de um atributo, e o **set** (também chamado de *setter*), usado para **alterar** esse valor.

Esses métodos são a principal ferramenta para colocar em prática o pilar do **encapsulamento**, pois permitem que a classe controle exatamente como seus dados internos podem ser acessados e modificados, aplicando validações quando necessário.

## 2. Por que não deixar os atributos públicos?

Se um atributo é declarado como `public`, qualquer classe pode alterá-lo livremente, sem qualquer tipo de validação ou controle, o que pode colocar o objeto em um estado inválido ou inconsistente.

```
public class Produto {  
    public double preco; // atributo público, sem proteção  
}  
  
// Em outra classe:  
Produto p = new Produto();  
p.preco = -50.0; // valor inválido, mas o compilador permite!
```

*Ao declarar o atributo como `private` e controlar seu acesso por meio de um método `set`, é possível validar o valor antes de aceitá-lo.*

## 3. O Método get

O método `get` é responsável por retornar o valor atual de um atributo, sem permitir que ele seja alterado diretamente. Por convenção, seu nome começa com o prefixo *get*, seguido do nome do atributo com a primeira letra maiúscula (por exemplo, `getNome()` para o atributo `nome`). Para atributos do tipo `boolean`, é comum usar o prefixo *is* (como em `isAtivo()`).

```

public class Produto {
    private String nome;
    private double preco;

    public String getNome() {
        return nome;
    }

    public double getPreco() {
        return preco;
    }
}

// Uso:
Produto p = new Produto();
System.out.println(p.getNome()); // lê o valor de nome
System.out.println(p.getPreco()); // lê o valor de preco

```

## 4. O Método set

O método set é responsável por atribuir (ou alterar) o valor de um atributo. Por convenção, seu nome começa com o prefixo set, seguido do nome do atributo, e recebe um parâmetro com o novo valor. É dentro do set que se aplicam validações, garantindo que o objeto nunca fique em um estado inválido.

```

public class Produto {
    private String nome;
    private double preco;

    public void setNome(String nome) {
        this.nome = nome;
    }

    public void setPreco(double preco) {
        if (preco >= 0) {
            this.preco = preco;
        } else {
            System.out.println("Erro: o preço não pode ser negativo.");
        }
    }
}

// Uso:
Produto p = new Produto();
p.setNome("Caderno");
p.setPreco(15.90);
p.setPreco(-10.0); // rejeitado pela validação dentro do setter

```

*Note que a palavra-chave this é usada para diferenciar o atributo da classe (this.preco) do parâmetro recebido pelo método (preco), que possuem o mesmo nome.*

## 5. Exemplo Completo

A seguir, um exemplo completo de uma classe Pessoa que utiliza atributos privados e métodos get e set para controlar o acesso a esses dados, incluindo uma validação simples no setter da idade.

```

public class Pessoa {
    private String nome;
    private int idade;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        if (idade >= 0 && idade <= 130) {
            this.idade = idade;
        } else {
            System.out.println("Idade inválida: " + idade);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa();

        pessoa.setNome("Carla");
        pessoa.setIdade(28);

        System.out.println("Nome: " + pessoa.getNome());
        System.out.println("Idade: " + pessoa.getIdade());

        pessoa.setIdade(-5); // valor rejeitado pela validação
    }
}

```

*Saída do programa: Nome: Carla, Idade: 28 e, em seguida, Idade inválida: -5.*

## 6. Vantagens do Uso de Getters e Setters

Utilizar métodos de acesso em vez de expor atributos diretamente traz diversos benefícios para o projeto de um sistema orientado a objetos:

- **Validação de dados:** o setter pode recusar valores inválidos antes de alterá-los.
- **Controle de leitura e escrita:** é possível criar apenas o getter (atributo somente leitura) ou apenas o setter, conforme a necessidade.
- **Encapsulamento:** os detalhes internos da classe ficam escondidos, e a forma como o dado é armazenado pode mudar sem afetar quem usa a classe.
- **Facilidade de manutenção:** alterações na lógica de acesso ao atributo são feitas em um único lugar, sem impactar o restante do código.

É possível, por exemplo, criar um atributo somente com o método `get`, tornando-o "somente leitura" para outras classes, como no exemplo abaixo.

```
public class Usuario {  
    private final String cpf; // não deve ser alterado após a criação  
  
    public Usuario(String cpf) {  
        this.cpf = cpf;  
    }  
  
    public String getCpf() { // apenas leitura, sem setCpf()  
        return cpf;  
    }  
}
```

## 7. Conclusão

Os métodos `get` e `set` são a forma padrão, em Java, de acessar e modificar atributos privados de uma classe de maneira segura e controlada. Ao adotar essa prática, o desenvolvedor reforça o encapsulamento, evita estados inválidos nos objetos e mantém o sistema mais organizado e fácil de manter, já que toda a lógica de validação e acesso aos dados fica centralizada dentro da própria classe.