

ENCAPSULAMENTO E MÉTODOS DE INSTÂNCIA

Prof^a Josiane Lopes

OBJETIVO DA AULA

Ensinar o conceito de encapsulamento e a criação de métodos de instância para manipulação de dados dentro dos objetos.



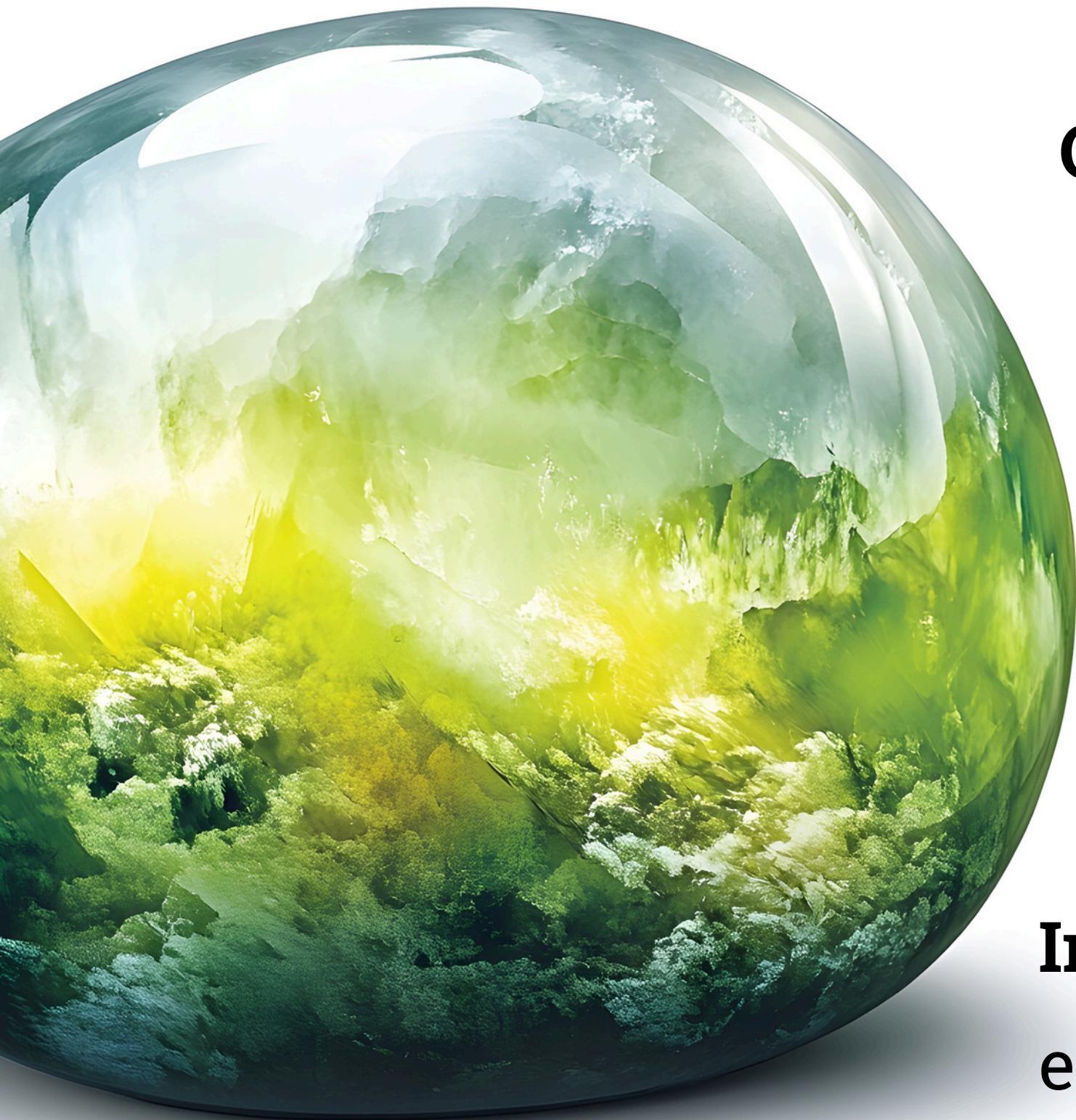
REVISANDO...

- Atributos e métodos definem Objetos
- Necessidade de proteger e controlar o acesso aos dados



**Encapsulamento é como
proteger as informações
de um objeto, permitindo
acesso controlado.**





O QUE É ENCAPSULAMENTO?

Esconder detalhes internos do objeto

Permite interação apenas por métodos

Importância: segurança, organização, evitar erros.



MODIFICADORES DE ACESSO

Público
(public)

Acessível de
qualquer lugar

this.titular

Protegido
(protected)

Convenção: não acessa
diretamente fora da
classe

this._saldo

Privado
(private)

o acesso externo é
realmente bloqueado

#senha

MODIFICADORES DE ACESSO

Público
(public)

Campos públicos são
sempre acessíveis
diretamente

this.titular

Protegido
(protected)

O sublinhado _ é usado
como convenção de
protegido.

this._saldo

Privado
(private)

O uso de # torna o
campo realmente
privado (N
acessível
e nem herdável)

#senha

Exemplo prático: Classe ContaBancária (JS)

```
class Conta {
    #senha;

    constructor(titular, saldo) {
        this.titular = titular;      // público
        this._saldo = saldo;        // protegido (por convenção)
        this.#senha = "1234";       // privado real
    }

    exibirSaldo() {
        console.log(`Saldo: R$ ${this._saldo}`);
    }

    autenticar(senha) {
        if (senha === this.#senha) {
            console.log("Acesso permitido!");
        } else {
            console.log("Senha incorreta!");
        }
    }
}

const conta1 = new Conta("Maria", 1000);
console.log(conta1.titular);    // público
console.log(conta1._saldo);    // protegido (acesso não recomendado)
// console.log(conta1.#senha); // ERRO!
conta1.exibirSaldo();
conta1.autenticar("1234");
```

MÉTODOS DE INSTÂNCIA PARA MANIPULAÇÃO DE DADOS

Métodos para ler e alterar os atributos com
segurança

getters

método usado para
ler o valor de um
atributo

setters

método usado para
alterar o valor de um
atributo

SINTAXE:

```
get nomeDoAtributo() {  
    // código para retornar o valor  
}  
  
set nomeDoAtributo(valor) {  
    // código para definir o valor  
}
```

EXEMPLO 1:

```
class Pessoa {  
    #idade; // atributo privado  
  
    constructor(nome, idade) {  
        this.nome = nome;  
        this.#idade = idade;  
    }  
  
    get idade() {  
        return this.#idade;  
    }  
  
    set idade(novaIdade) {  
        if (novaIdade > 0) {  
            this.#idade = novaIdade;  
        } else {  
            console.log("Idade inválida!");  
        }  
    }  
}
```

```
apresentar() {  
    console.log(`Olá, eu sou ${this.nome} e tenho ${this.#idade} anos.`);  
}  
}  
  
const pessoa1 = new Pessoa("Lucas", 25);  
pessoa1.apresentar();  
console.log(pessoa1.idade); // usa o getter  
pessoa1.idade = 30; // usa o setter  
pessoa1.apresentar();  
pessoa1.idade = -5; // setter bloqueia valor inválido
```

- **#idade** é um campo privado

EXEMPLO 2:

```
class ContaBancaria {  
    #saldo;  
  
    constructor(titular, saldoInicial) {  
        this.titular = titular;  
        this.#saldo = saldoInicial;  
    }  
  
    get saldo() {  
        return this.#saldo;  
    }  
  
    set saldo(novoSaldo) {  
        if (novoSaldo >= 0) {  
            this.#saldo = novoSaldo;  
        } else {  
            console.log("Saldo não pode ser negativo!");  
        }  
    }  
  
    depositar(valor) {  
        this.#saldo += valor;  
    }  
  
    sacar(valor) {  
        if (valor <= this.#saldo) {  
            this.#saldo -= valor;  
        } else {  
            console.log("Saldo insuficiente!");  
        }  
    }  
  
    const conta = new ContaBancaria("Maria", 500);  
    console.log("Saldo inicial:", conta.saldo);  
    conta.depositar(200);  
    conta.sacar(100);  
    conta.saldo = -50; // bloqueado pelo setter  
    console.log("Saldo final:", conta.saldo);
```

VANTAGENS DO USO DE GETTERS E SETTERS

- Protege os dados internos da classe
- Permite aplicar validações e regras de negócio
- Facilita manutenção do código
- Permite controle sobre o comportamento interno dos objetos

EXERCÍCIOS (GETTER E SETTER)

Crie códigos em JavaScript, com as seguintes características:

CLASSE PRODUTO

- Atributos: #nome, #preco
- Método: getter e setter para preco, que nao permita valores negativos
- Exiba as informações (nome e preço do produto)

CLASSE RETÂNGULO

- Atributos: #largura e #altura
- Métodos: getter e setter para ambos (valores > 0)
- Exiba o valor da área

EXERCÍCIOS

Crie 2 códigos em JavaScript, com as seguintes características:

CLASSE USUÁRIO

- Atributos: nome, senha
- Método: autenticar e alterar
- Crie dois usuários e exiba autenticação e a alteração de senha

CLASSE CARTÃOCRÉDITO

- Atributos: titular, limite
- Métodos: realizar compra
- crie um objeto e teste o metodo de realizar a compra

RESUMINDO...

- Encapsulamento protege os dados e organiza o código
- Modificadores de acesso: público, protegido e privado
- Getters e Setters → controle no acesso e alteração dos atributos



PRÁTICA RECOMENDADA

**Manter
consistência e
segurança no
acesso a dados.**





OBRIGADA

Prof^a Josiane Lopes