Encapsulamento

O que é Encapsulamento?

Encapsulamento é um dos pilares fundamentais da Programação Orientada a Objetos (POO). Ele se refere ao ato de **proteger os dados** de uma classe e controlar o acesso a eles. Isso é feito utilizando **modificadores de acesso** como `private`, `protected` e `public`.

O encapsulamento ajuda a manter a integridade dos dados, permitindo que sejam acessados e modificados apenas por meio de métodos específicos, como **getters** e **setters**.

Benefícios do Encapsulamento

1. **Segurança**: Restringe o acesso direto aos atributos, evitando alterações indevidas.
2. **Manutenção**: Torna o código mais fácil de atualizar, pois a lógica pode ser centralizada nos métodos.
3. **Controle**: Permite validar os dados antes de alterá-los.

Como Funciona o Encapsulamento?

Para encapsular dados em Java, siga estas etapas:

1. **Declare os atributos como `private`.**
2. **Crie métodos `public` para acessar e modificar os atributos.** Esses métodos são conhecidos como **getters** e **setters**.

Exemplo:

```java
public class Pessoa {
    // Atributos privados
    private String nome;
    private int idade;

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

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

    // Getter para a idade
    public int getIdade() {
        return idade;
    }

    // Setter para a idade (com validação)
    public void setIdade(int idade) {
        if (idade >= 0) {
            this.idade = idade;
        } else {
            System.out.println("Idade inválida!");
        }
    }
}
```

Detalhando o Exemplo

No exemplo acima:

- Os atributos `nome` e `idade` são **privados**.
- Os métodos **getter** (`getNome`, `getIdade`) fornecem acesso às informações.
- Os métodos **setter** (`setNome`, `setIdade`) permitem modificar os valores, com validação no setter de `idade`.

Assim, garantimos que:

- Ninguém pode alterar diretamente os valores de `nome` e `idade`.
- A idade só é alterada se for um valor válido.

Modificadores de Acesso

Os modificadores de acesso controlam a visibilidade dos atributos e métodos.

Modificador	Descrição
`private`	Acesso restrito à própria classe.
`default`	Acesso permitido apenas dentro do mesmo pacote.
`protected`	Acesso permitido dentro do mesmo pacote e por subclasses.
`public`	Acesso permitido de qualquer lugar.

Exemplos Visuais e Teste

Exemplo Prático

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

        // Usando os setters para definir os valores
        pessoa.setNome("Maria");
        pessoa.setIdade(25);

        // Usando os getters para acessar os valores
        System.out.println("Nome: " + pessoa.getNome());
        System.out.println("Idade: " + pessoa.getIdade());

        // Tentando definir uma idade inválida
        pessoa.setIdade(-5); // Exibe: "Idade inválida!"
    }
}
```

Saída Esperada:

```
Nome: Maria
Idade: 25
Idade inválida!
```

Observação Visual

Os códigos acima podem ser divididos em:

- **Declaração de atributos**: Destacados em azul.
- **Getter e Setter**: Destacados em verde.
- **Validação no setter**: Destacada em laranja.

Exercício Proposto

Implemente uma classe chamada `ContaBancaria` com os seguintes requisitos:

- Atributos privados:
  - `numeroConta` (String)
  - `saldo` (double)
- Getters e Setters para os atributos.
- O setter de `saldo` deve impedir valores negativos.

Teste a classe com o seguinte código:

```java
public class BancoTest {
    public static void main(String[] args) {
        ContaBancaria conta = new ContaBancaria();

        conta.setNumeroConta("12345-6");
        conta.setSaldo(1000);

        System.out.println("Conta: " + conta.getNumeroConta());
        System.out.println("Saldo: R$" + conta.getSaldo());

        // Tentando definir um saldo inválido
        conta.setSaldo(-500); // Deve impedir a alteração
    }
}
```

Conclusão

O encapsulamento é essencial para proteger os dados de uma classe e controlar seu acesso. Pratique criando classes com atributos privados e usando getters e setters para gerenciar os valores de forma segura.