



+Devs2Blu

Linguagem de programação JAVA

Profª. Heloisa Moura

+Devs2Blu

Linguagem de programação JAVA

O que vamos ver:

- Atributos (de instância, de classe);
- Relacionamento entre classes;
- Acoplamento e Coesão

+Devs2Blu

Linguagem de programação JAVA

Atributo de instância

- São usados pelos objetos para armazenar seus estados. Existem antes dos métodos serem chamados , durante sua execução e após terminarem sua tarefa.
- Seus valores são específicos de cada instância(objeto) e não são compartilhados entre as instâncias.
- Podem usar qualquer nível de acesso – **public, private, protected** ou **default**, e serem manipulados por todos os métodos da classe.

+Devs2Blu

Linguagem de programação JAVA

Atributo de instância

Exemplo:

```
private String nomeTime;
```


+Devs2Blu

Linguagem de programação JAVA

Atributo de classe

- São campos que pertencem à classe em si, e não ao objeto instanciado. Eles são compartilhados por todos os objetos da classe.
- Diferença significativa em relação ao atributo de instância, o valor armazenado em um atributo de classe é o mesmo para todos os objetos e qualquer objeto poderá alterar esse valor, que será visível por todos os outros.

+Devs2Blu

Linguagem de programação JAVA

Atributo de classe

Exemplo:

```
private static String nomeSelecao;
```

+Devs2Blu

Linguagem de programação JAVA

Exemplo na pratica

+Devs2Blu

Linguagem de programação JAVA

Relacionamento entre classes: Tipos

Agregação: estabelecem um vínculo entre objetos.

Composição: relacionamento do tipo todo/parte.

Associação(Uso): um objeto usa a funcionalidade de outro sem estabelecer vínculo duradouro (referências).

+Devs2Blu

Linguagem de programação JAVA

Agregação: Forma de composição em que o objeto composto apenas usa ou tem conhecimento da existência do(s) objeto(s) componente(s). Os objetos componentes podem existir sem o agregado e vice-versa.

Composição: Forma de associação em que o objeto composto é responsável pela existência dos componentes. O componente não tem sentido fora da composição.

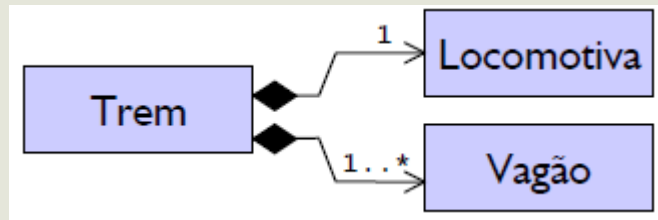
Uma classe pode ter referências a objetos de outras classes como membros. Isso é chamado de **composição** e, as vezes, é referido como um **relacionamento tem um**.

+Devs2Blu

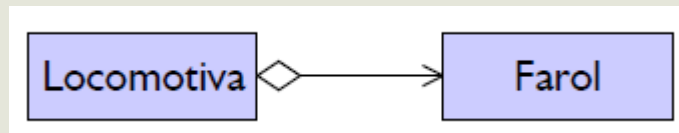
Linguagem de programação JAVA

Exemplo:

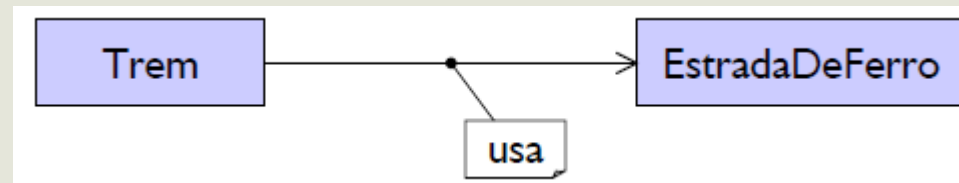
Composição: um trem é formado por locomotiva e vagões.



Agregação: uma locomotiva tem um farol (mas não vai deixar de ser uma locomotiva se não o tiver)



Associação: um trem usa uma estrada de ferro (não faz parte do trem, mas ele depende dela)



+Devs2Blu

Linguagem de programação JAVA

Outro Exemplo:

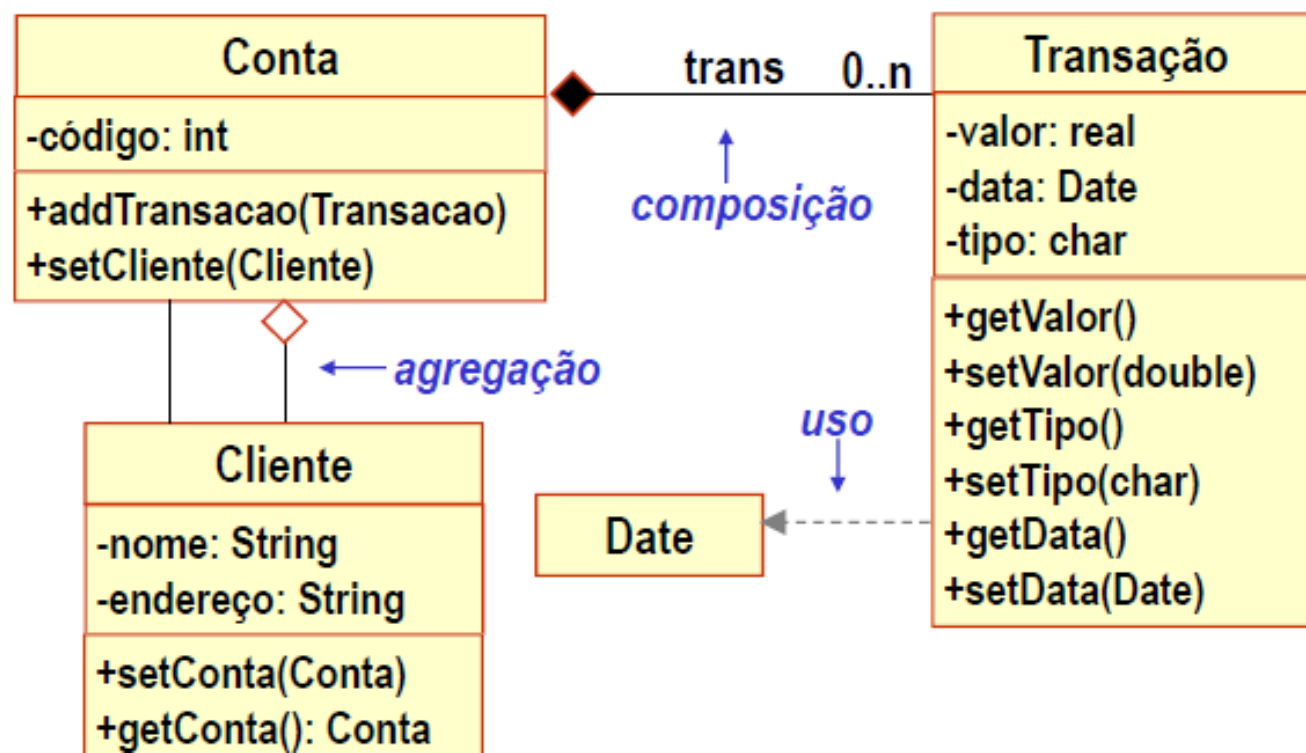
- Uma conta corrente é formada por várias transações de crédito e débito → composição
- Um cadastro de clientes é formado por vários clientes → agregação
- Um cliente tem uma conta-corrente → agregação
- Um documento possui um conjunto de parágrafos → composição
- Uma turma é um conjunto de alunos → composição

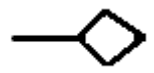
+Devs2Blu

Linguagem de programação JAVA

Relacionamento entre classes

Notação UML para definir composições, agregações e uso:



•  Se aberto, tem-se **Agregação**.

•  Se escuro, têm-se **Composição**.

+Devs2Blu

Linguagem de programação JAVA

Relacionamento entre classes

Implementação em Java:

```
public class Conta {  
    private int codigo;  
    private Cliente cliente;  
    private Transacao[] trans;  
    private int qTransacoes;  
    //...  
}
```

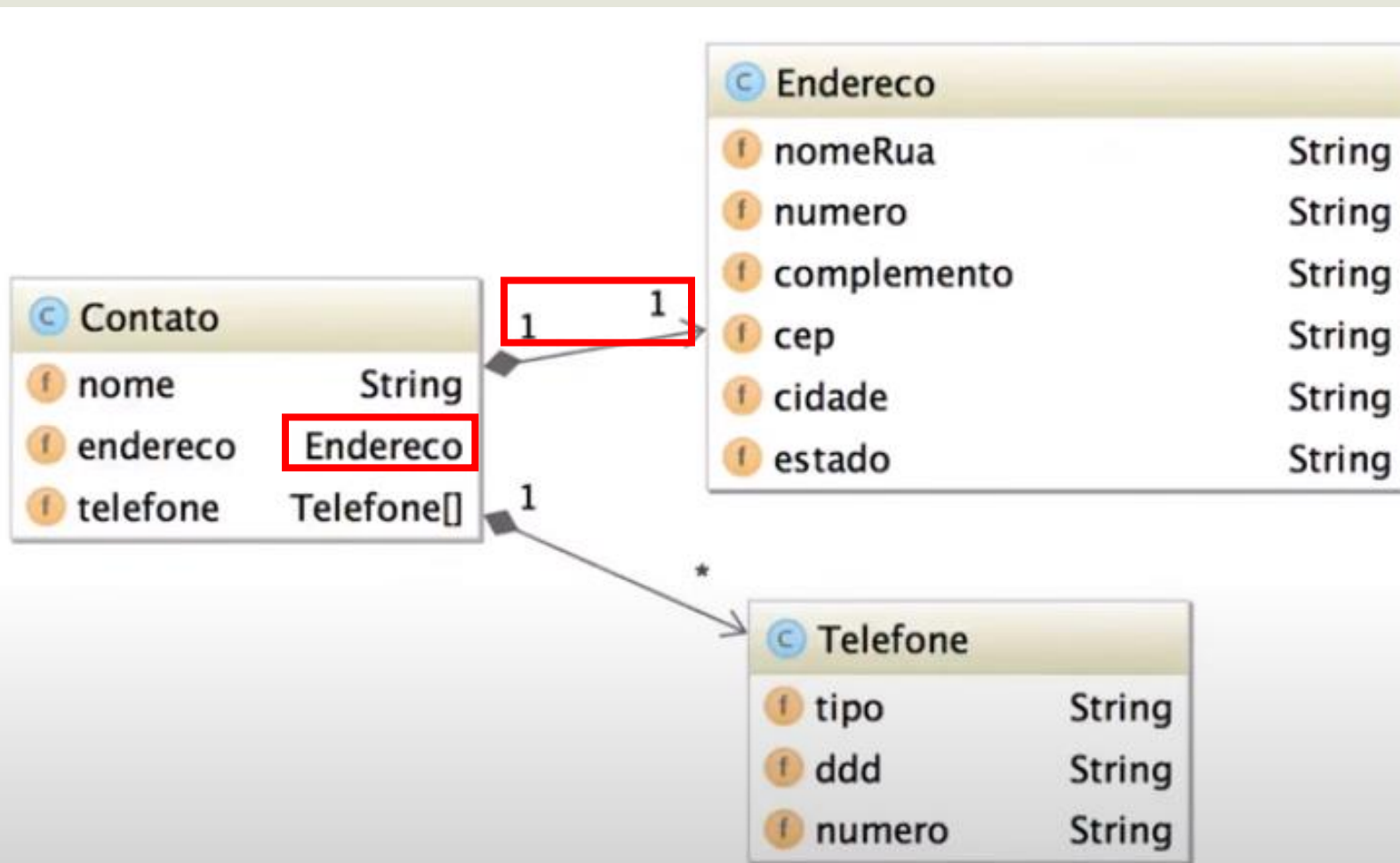
```
public class Cliente {  
    private String nome;  
    private Conta conta;  
    private String endereço;  
    //...  
}
```

```
public class Transacao {  
    private double valor;  
    private char tipo;  
    private Date data;  
    //...  
}
```

+Devs2Blu

Linguagem de programação JAVA

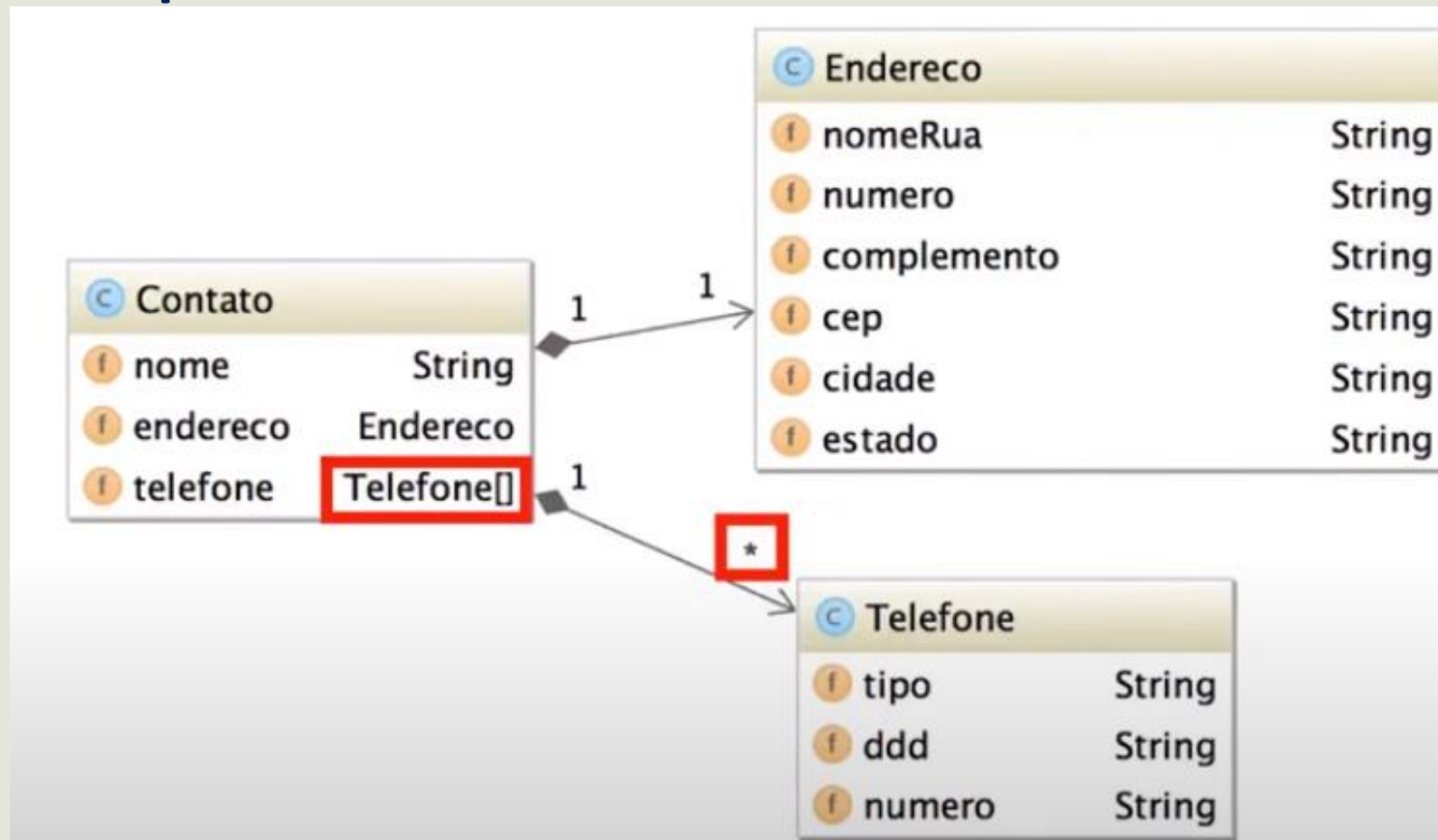
Exemplo na pratica: Relacionamento tem um



+Devs2Blu

Linguagem de programação JAVA

Exemplo na pratica: Relacionamento tem muitos



+Devs2Blu

Linguagem de programação JAVA

Exemplo na pratica

Exercício 1:

Escreva uma classe Agenda, que contém vários contatos do tipo Contato. Cada contato possui nome, telefone e email. A Agenda também possui um nome. Crie um programa teste que peça para o usuário entrar com o nome da Agenda e em seguida 3 contatos. Crie métodos que retornem uma String com a informação de cada contato e também de todos os contatos da agenda.

Acoplamento

É o nível de dependência entre duas ou mais classes de um programa.

- Quando uma classe depende muito de outra, dizemos que elas têm um **alto acoplamento**.
- Quando essa dependência é menor, temos um **baixo acoplamento**.

+Devs2Blu

Linguagem de programação JAVA

Por que isso importa?

- Sistema com **alto acoplamento** é mais difícil de manter e modificar
- Alterar uma classe pode forçar a alteração de várias outras.
- ✓ Sistema com **baixo acoplamento** é mais fácil de manter e modificar
- ✓ Classes são mais independentes.

+Devs2Blu

Linguagem de programação JAVA

Exemplo acoplamento forte(alto acoplamento)

Imagine que temos duas classes: **Motor** e **Carro**. Se a classe **Carro** usa diretamente as funcionalidades da classe **Motor**, o **Carro** depende fortemente de como o **Motor** está implementado.

Exemplo acoplamento forte(alto acoplamento)

Desvantagem:

Se quisermos mudar a classe **Motor** (por exemplo, adicionando um novo tipo de motor), vamos ter que alterar a classe **Carro**, porque ela está fortemente acoplada a **Motor**.

```
class Motor {  
    public void ligarMotor() {  
        System.out.println("Motor ligado");  
    }  
}  
  
class Carro {  
    private Motor motor = new Motor();  
  
    public void ligarCarro() {  
        motor.ligarMotor();  
        System.out.println("Carro ligado");  
    }  
}
```

+Devs2Blu

Linguagem de programação JAVA

Acoplamento Fraco (Baixo Acoplamento)

Quando as classes estão menos dependentes umas das outras, dizemos que elas têm **acoplamento fraco**. Isso significa que as mudanças em uma classe não afetam diretamente as outras.

+Devs2Blu

Linguagem de programação JAVA

Exemplo Acoplamento Fraco (Baixo Acoplamento)

Podemos reduzir o acoplamento usando interfaces. Isso faz com que o Carro dependa apenas da interface, e não da implementação específica do Motor.

Exemplo Acoplamento Fraco (Baixo Acoplamento)

Vantagem: Se quisermos mudar o tipo de motor (por exemplo, para um MotorV12) não precisamos alterar a classe `Carro`, porque ela depende apenas da interface `Motor`, não da implementação específica.

```
interface Motor {  
    void ligarMotor();  
}  
  
class MotorV8 implements Motor {  
    public void ligarMotor() {  
        System.out.println("Motor V8 ligado");  
    }  
}  
  
class Carro {  
    private Motor motor;  
  
    public Carro(Motor motor) {  
        this.motor = motor;  
    }  
  
    public void ligarCarro() {  
        motor.ligarMotor();  
        System.out.println("Carro ligado");  
    }  
}
```


+Devs2Blu

Linguagem de programação JAVA

Quando Usar Cada Um?

Acoplamento Forte:

Quando as classes são muito específicas e sempre vão trabalhar juntas.
Exemplo: Um sistema simples, com poucos componentes, onde as mudanças são raras.

Acoplamento Fraco:

Quando queremos tornar o código mais flexível e fácil de modificar.
Exemplo: Em sistemas grandes e complexos, onde várias partes do código podem mudar com o tempo.

+Devs2Blu

Linguagem de programação JAVA

Resumo

- O **acoplamento** refere-se ao nível de dependência entre classes.
- **Acoplamento Forte** é quando as classes dependem muito umas das outras, o que torna o código difícil de modificar.
- **Acoplamento Fraco** é quando as classes são mais independentes, facilitando a manutenção e atualização do código.

Exercício 1: Cenário

Imagine que você está desenvolvendo um sistema simples para uma padaria, com as classes Produto e Pedido. O pedido precisa de informações sobre o produto para ser processado. Você vai implementar as abordagens de acoplamento forte além de criar uma classe de teste para verificar o comportamento do código.

A Classe Pedido dependerá diretamente da classe Produto

- 1 - Crie uma classe Produto que tenha um nome e um preço.
- 2 - Crie uma classe Pedido que contenha uma instância de Produto e um método para calcular o preço total de uma certa quantidade do produto.
- 3 - Crie uma classe de teste chamada TestePedido para validar o funcionamento da classe Pedido

Coesão

É o grau em que os métodos e atributos de uma classe estão relacionados e trabalham juntos para cumprir um propósito específico.

- Uma classe com alta coesão realiza uma única tarefa ou conjunto de tarefas relacionadas.
- Uma classe com baixa coesão tenta fazer muitas coisas diferentes, o que pode dificultar a manutenção do código.

+Devs2Blu

Linguagem de programação JAVA

Por que isso importa?

- **Alta coesão** resulta em código mais fácil de entender, modificar e testar.
- ✓ **Baixa coesão** pode levar a classes confusas, difíceis de manter e que quebram facilmente quando precisam ser modificadas.

+Devs2Blu

Linguagem de programação JAVA

Coesão Alta:

- Uma classe faz **uma coisa bem feita** e todas as suas funcionalidades estão relacionadas.

Exemplo

Uma classe **Cliente** que só cuida das informações e comportamentos relacionados ao cliente, como obter o nome, e-mail, etc.

+Devs2Blu

Linguagem de programação JAVA

Vantagens da Alta Coesão:

A classe Cliente é fácil de entender e modificar, pois todas as suas funcionalidades estão diretamente ligadas a um Cliente.

```
class Cliente {  
    private String nome;  
    private String email;  
  
    public Cliente(String nome, String email) {  
        this.nome = nome;  
        this.email = email;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
}
```

+Devs2Blu

Linguagem de programação JAVA

Coesão Baixa:

- Uma classe faz muitas coisas não relacionadas, tornando o código mais complexo.

Exemplo

Imagine uma classe **Cliente** que também cuida de questões de pagamento e envio de produtos. Isso confunde o propósito da classe e diminui a coesão.

+Devs2Blu

Linguagem de programação JAVA

Desvantagens da Baixa Coesão:

A classe Cliente está misturando responsabilidades de cliente e pagamento, o que complica a manutenção e dificulta futuras modificações no sistema.

```
class Cliente {  
    private String nome;  
    private String email;  
    private double saldo;  
  
    public Cliente(String nome, String email, double saldo) {  
        this.nome = nome;  
        this.email = email;  
        this.saldo = saldo;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    // Baixa coesão: Responsabilidade adicional de pagamento  
    public void adicionarSaldo(double valor) {  
        saldo += valor;  
    }  
  
    public void pagar(double valor) {  
        if (saldo >= valor) {  
            saldo -= valor;  
        } else {  
            System.out.println("Saldo insuficiente");  
        }  
    }  
}
```

+Devs2Blu

Linguagem de programação JAVA

Resumo

- **Alta coesão** = Classes focadas em uma tarefa específica.
- **Baixa coesão** = Classes que tentam fazer muitas coisas não relacionadas.
- **Boa prática**: Sempre tente dividir responsabilidades e focar o propósito de cada classe.

Exercício 1: Cenário

Você foi encarregado de criar uma classe em Java para representar uma **Calculadora de Operações Simples**. Esta classe deve ter alta coesão, ou seja, cada método da classe deve ser responsável por uma única tarefa, e a classe, como um todo, deve ter um foco bem definido.

- A calculadora deverá ser capaz de realizar as seguintes operações: **Soma, Subtração, Multiplicação e Divisão**: (lembre-se de verificar se o divisor não é zero)
- Crie uma classe chamada Calculadora
- A classe deve ter métodos separados para cada uma das operações mencionadas.
- Cada método deve receber dois números como parâmetros e retornar o resultado.
- A classe deve ter coesão, ou seja, apenas lidar com operações matemáticas.
- Crie a Classe TesteCalculadora

+Devs2Blu

Linguagem de programação JAVA

Exercícios fixação