ESTRUTURA DE DADOS

Orientação a objetos básica

prof.pedrofreitas@redebatista.edu.br

Prof. Pedro Freitas

O método transfere()

- E se quisermos ter um método que **transfere** dinheiro entre duas contas?
- Será necessário criar um método que recebe dois parâmetros: conta1 e conta2 do tipo Conta?

O método transfere()

- Quando chamarmos o método **transfere**, já teremos um objeto do tipo Conta, já que o método transfere pertence a classe Conta.
- Assim o método irá receber apenas um parâmetro do tipo Conta, a Conta destino mais o valor a ser transferido.

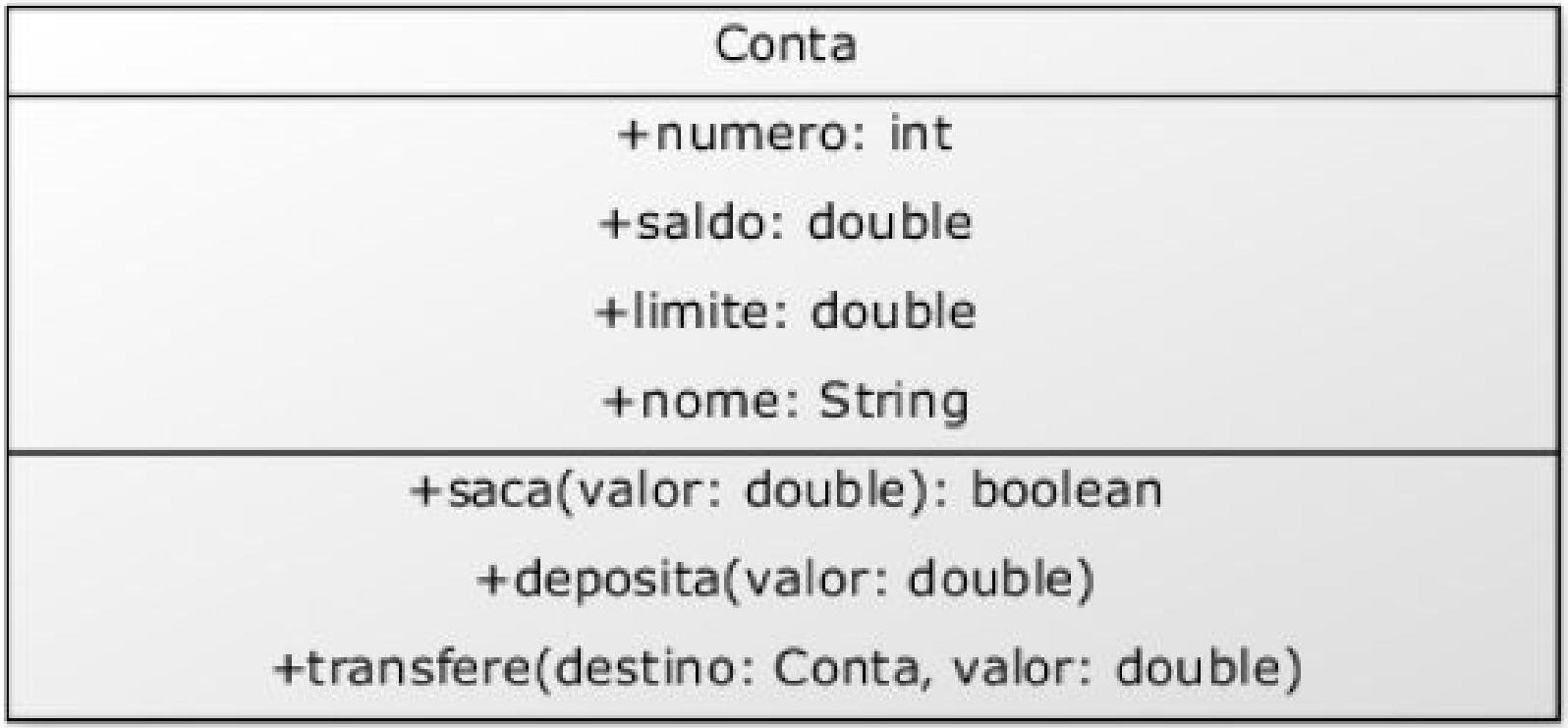
O método transfere()

```
class Conta {

    // atributos e métodos...

    void transfere(Conta destino, double valor) {
        this.saldo = this.saldo - valor;
        destino.saldo = destino.saldo + valor;
    }
}
```

O método transfere()



O método transfere()

- Fazendo melhoria no código:
  - Verificando se a conta possui a quantidade a ser transferida disponível.
  - Será utilizado os métodos existentes **deposita** e **saca**.

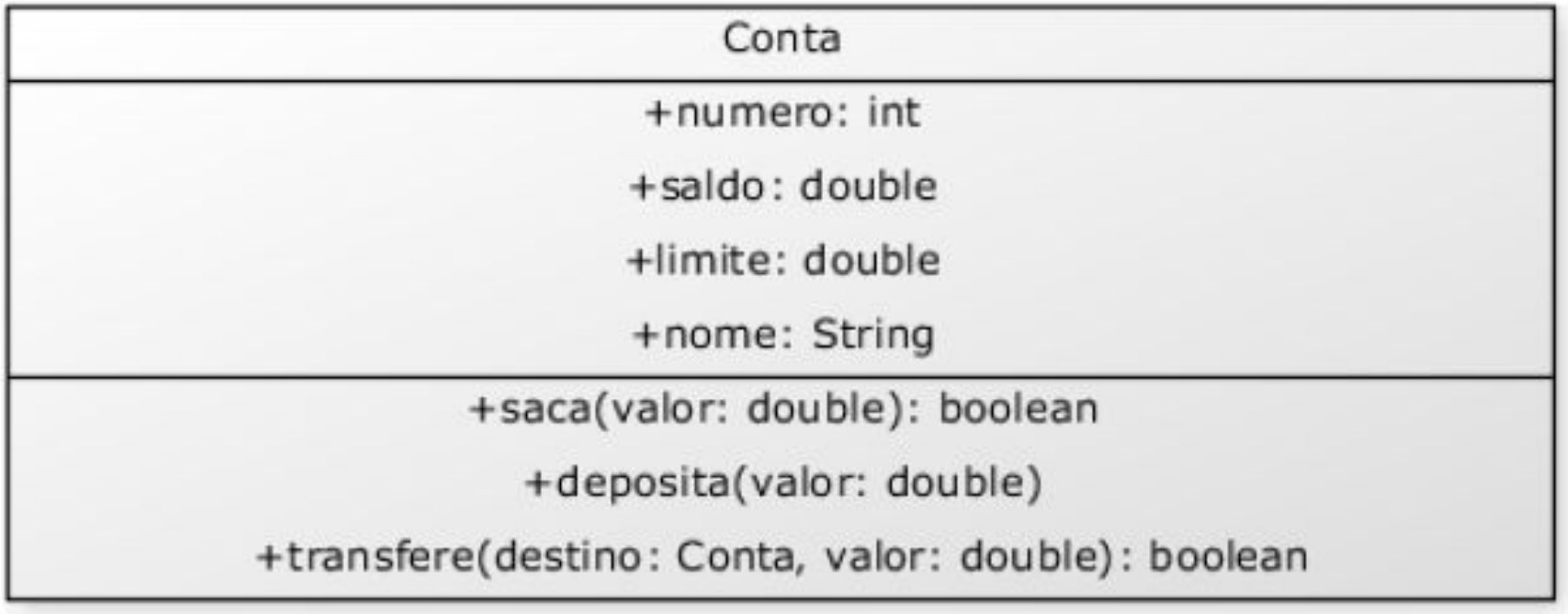
O método transfere()

```
class Conta {

    // atributos e métodos...

    boolean transfere(Conta destino, double valor) {
        boolean retirou = this.saca(valor);
        if (retirou == false) {
            // não deu pra sacar!
            return false;
        }
        else {
            destino.deposita(valor);
            return true;
        }
    }
}
```

O método transfere()

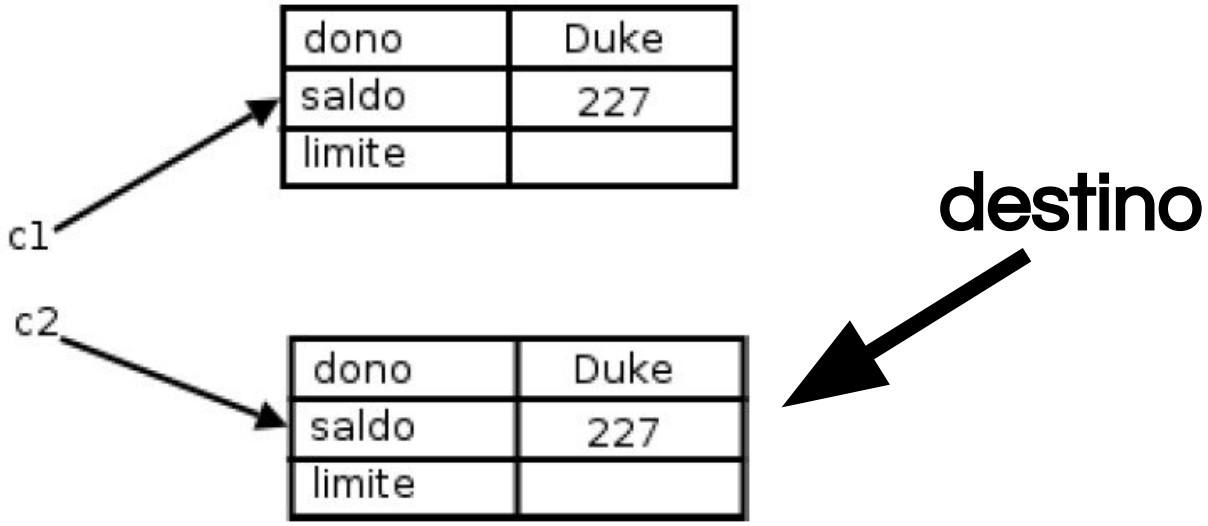


O método transfere()

- Quando uma Conta é passada como argumento, o que será que acontece na memória? O objeto é clonado?
- No Java, a passagem de parâmetro funciona como uma atribuição "=".
  - Chamada da função
    - c1.transfere(c2, 300);
    - destino = c2;
    - valor = 300;

O método transfere()

- O parâmetro vai receber o valor da variável do tipo Conta que for passado como argumento.
- E qual é o valor de uma variável dessas?
  - Seu valor é um endereço, uma referência, nunca um objeto. Por isso não há cópia de objetos aqui.



Continuando com atributos

- Os atributos da classe recebem um valor padrão. No caso numérico, valem 0, no caso de boolean, valem false.
- Mas é possível dar valores default (padrão).

```
class Conta {
    int numero = 1234;
    String dono = "Duke";
    String cpf = "123.456.789-10";
    double saldo = 1000;
    double limite = 1000;
}
```

Continuando com atributos

- Os atributos da classe recebem um valor padrão. No caso numérico, valem 0, no caso de boolean, valem false.
- Mas é possível dar valores default (padrão).
- Quando for criado uma conta, seus atributos já estão "populados" com esses valores colocados.

```
class Conta {
    int numero = 1234;
    String dono = "Duke";
    String cpf = "123.456.789-10";
    double saldo = 1000;
    double limite = 1000;
}
```

Continuando com atributos

- Imagine que o número de atributos da classe comece a aumentar, adicionando atributo referente ao cliente da conta:
  - nome;
  - sobrenome;
  - cpf.
- Começaríamos a ter muitos atributos, mas se for feito uma analise mais detalhada veremos que, uma Conta não tem **nome**, **sobrenome** e **cpf**.
- E atributos são de um cliente cliente.
- **Qual a solução para isso?**

Continuando com atributos

- Criar uma classe chamada cliente com seus respectivos atributos.
- E como ficaria a classe Conta?

```
class Cliente {
    String nome;
    String sobrenome;
    String cpf;
}
```

Continuando com atributos

- Atributos de uma classe também podem ser referências para outras classes.
- Temos uma referencia da classe **Cliente** entre os atributos da classe **Conta**.

```
class Conta {
    int numero;
    double saldo;
    double limite;
    Cliente titular;
    // ..
}
```

Continuando com atributos

- Na **main** ficaria assim:
  - Atribuiu o valor da variável **c** ao atributo **titular** do objeto **minhaConta**.
  - **minhaConta** tem uma referência ao mesmo Cliente que **c** se refere, e pode ser acessado através de **minhaConta.titular**.

```
class Teste {
    public static void main(String[] args) {
        Conta minhaConta = new Conta();
        Cliente c = new Cliente();
        minhaConta.titular = c;
        // ...
    }
}
```

Continuando com atributos

- Usando o ponto é possível navegar sobre toda essa estrutura de informação:

```
Cliente clienteDaMinhaConta = minhaConta.titular;
clienteDaMinhaConta.nome = "Duke";

minhaConta.titular.nome = "Duke";
```

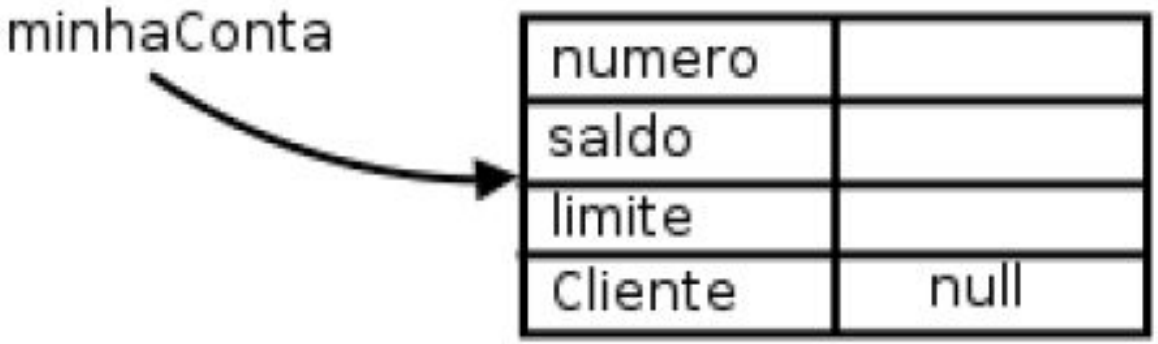
Continuando com atributos

- Se tentar acessar algum atributo do Cliente dentro da classe Conta sem dar um new em cliente irá gerar um erro.

```
class Teste {
    public static void main(String[] args) {
        Conta minhaConta = new Conta();
        minhaConta.titular.nome = "Manoel";
        // ...
    }
}
```

Continuando com atributos

- Quando damos new em um objeto, ele o inicializa com seus valores default, 0 para números, false para boolean e null para referências.
- null é uma palavra chave em java, que indica uma referência para nenhum objeto.
- O new não traz um efeito cascata



Continuando com atributos

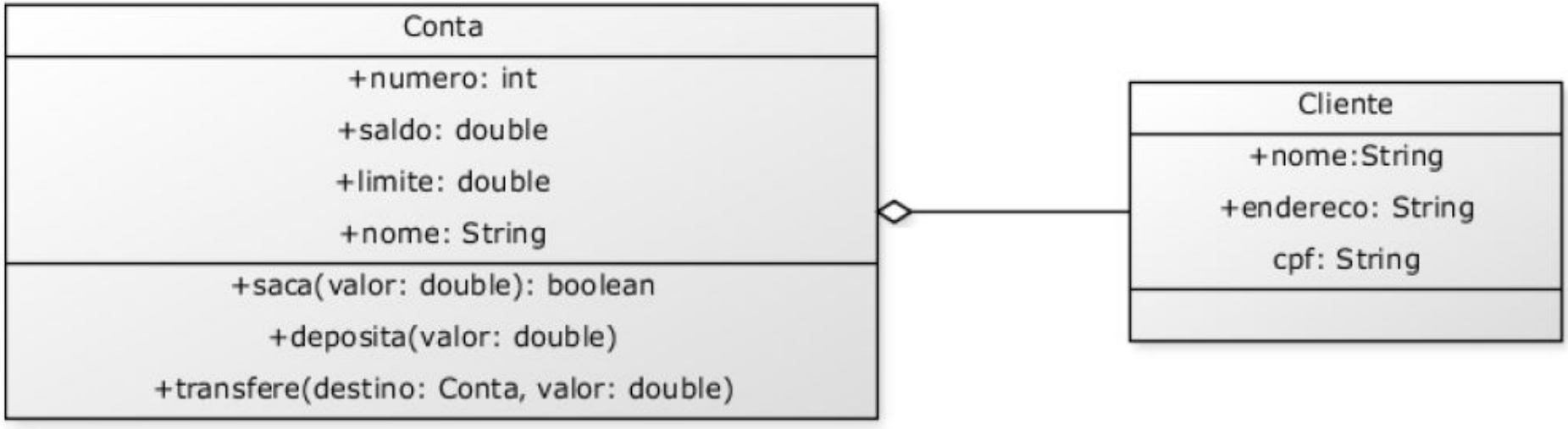
- Nesse código, toda nova Conta criada já terá um novo Cliente associado, sem necessidade de instanciálo logo em seguida da instanciação de uma Conta.

```
class Conta {
    int numero;
    double saldo;
    double limite;
    Cliente titular = new Cliente();

                                    // quando chamarem new Conta,
}                                   //havera um new Cliente para ele.
```

Continuando com atributos

- Qual alternativa você deve usar?
- Depende do caso: para toda nova Conta você precisa de um novo Cliente?
- É essa pergunta que deve ser respondida.
- Nesse nosso caso a resposta é não, mas depende do nosso problema.



Para saber mais

- Veremos como cariar certas classes relacionadas a uma fábrica de carros.
- Vamos criar uma classe Carro, com certos atributos, que descrevem suas características, e com certos métodos, que descrevem seu comportamento.

Fábrica de Carros

- Atributos:

```
class Carro {
    String cor;
    String modelo;
    double velocidadeAtual;
    double velocidadeMaxima;
}
```

Fábrica de Carros

- Métodos:

```
//liga o carro
void liga() {
    System.out.println("O carro está ligado");
}

//acelera uma certa quantidade
void acelera(double quantidade) {
    double velocidadeNova = this.velocidadeAtual + quantidade;
    this.velocidadeAtual = velocidadeNova;
}
```

Fábrica de Carros

- Métodos:

```
//devolve a marcha do carro
int pegaMarcha() {
    if (this.velocidadeAtual < 0) {
        return -1;
    }
    if (this.velocidadeAtual >= 0 && this.velocidadeAtual < 40) {
        return 1;
    }
    if (this.velocidadeAtual >= 40 && this.velocidadeAtual < 80) {
        return 2;
    }
    return 3;
}
```

Fábrica de Carros

- Testando o Carro:

```
class TestaCarro {
    public static void main(String[] args) {
        Carro meuCarro;
        meuCarro = new Carro();
        meuCarro.cor = "Verde";
        meuCarro.modelo = "Fusca";
        meuCarro.velocidadeAtual = 0;
        meuCarro.velocidadeMaxima = 80;

        // liga o carro
        meuCarro.liga();

        // acelera o carro
        meuCarro.acelera(20);
        System.out.println(meuCarro.velocidadeAtual);
    }
}
```

Fábrica de Carros

- O carro pode ter um Motor:

```
class Motor {
    int potencia;
    String tipo;
}
```

```
class Carro {
    String cor;
    String modelo;
    double velocidadeAtual;
    double velocidadeMaxima;
    Motor motor;

    // ..
}
```