

Programação Orientada a Objetos

Cap.3 - Introdução à Orientação a Objetos



Prof. Me. Renzo P. Mesquita

Objetivos

1. Dizer o que é e para que serve a Orientação a Objetos;
2. Conceituar Classes, Atributos e Comportamentos (Métodos);
3. Entender o significado de Variáveis e Objetos na memória;



Capítulo 3

Introdução a Orientação a Objetos

- 3.1. Comparativo entre as formas de Programação;*
- 3.2. Classes e Objetos;*
- 3.3. Uma Classe em Java;*
- 3.4. Criando e usando um Objeto;*
- 3.5. Métodos;*
- 3.6. UML e Diagrama de Classes;*
- 3.7. Objetos são acessados por referências;*
- 3.8. Classes dentro de Classes;*
- 3.9. Linguagens Orientadas a Objetos;*



3.1. Comparativo entre as formas de Programação Estruturada e Orientada a Objetos

Nas Universidades, a iniciação às linguagens de programação são, em grande maioria, realizadas com linguagens estruturadas ou procedurais (como C, Pascal e BASIC), por ser uma fase de adaptação à lógica de programação.

Mas o que caracteriza a programação Estruturada ou Procedural?

- Uma forma de escrever os códigos sem encapsular dados;
- Em qualquer parte do código é possível utilizar um dado de uma variável, sem necessidade de permissão;
- Variáveis tem seus valores alterados por meio de procedimentos (ou funções);
- Não há organização em camadas do código, ou seja, todos os tipos de códigos estão no mesmo arquivo;



3.1. Comparativo entre as formas de Programação Estruturada e Orientada a Objetos

Já a Orientação a Objetos é um modelo de Análise, Projeto e Programação de Sistemas de Software baseado na interação entre diversas unidades de software chamadas de OBJETOS.

Mas o que caracteriza a programação Orientada a Objetos?

- Traz OBJETOS do mundo real para se tornarem parte do código;
- Com o uso de CLASSES, podemos encapsular os objetos para que eles interajam entre si;
- Permite dividir todo o código em camadas (cada uma com suas responsabilidades);
- Implicitamente adiciona segurança à aplicação por meio do ENCAPSULAMENTO de dados;



3.1. Comparativo entre as formas de Programação Estruturada e Orientada a Objetos

3.1.1. Vantagens

PE

- Para programas simples, é de mais fácil compreensão, sendo amplamente usada em introdução à programação;
- Melhor visualização sobre o fluxo de execução do código;

POO

- Provê uma melhor organização do código;
- Contribui bastante para o reaproveitamento de código, além da facilidade de ser herdar atributos e comportamentos de outros Objetos;

**Programação
Procedural**



**Programação
Orientada a Objetos**



VS

3.1. Comparativo entre as formas de Programação Estruturada e Orientada a Objetos

3.1.2. Desvantagens

PE

- Tende a gerar códigos confusos, em que o tratamento dos dados são misturados com o comportamento do programa;
- Foco em como a tarefa deve ser feita e não em o que deve ser feito;

POO

- Pode não possuir o mesmo desempenho de códigos estruturados similares;
- Conceitos de mais difícil compreensão comparados aos conceitos da Programação Estruturada;

Programação
Procedural



Programação
Orientada a Objetos



VS

3.2. Classes e Objetos

Consideremos um programa para um banco. É bem fácil perceber que uma entidade extremamente importante para o nosso sistema é a CONTA.

O que toda CONTA tem?

- Número da conta;
- Dono da Conta;
- Saldo;
- Limite;



O que toda CONTA faz?

- Sacar;
- Depositar;
- Verificar saldo;
- Transferir;



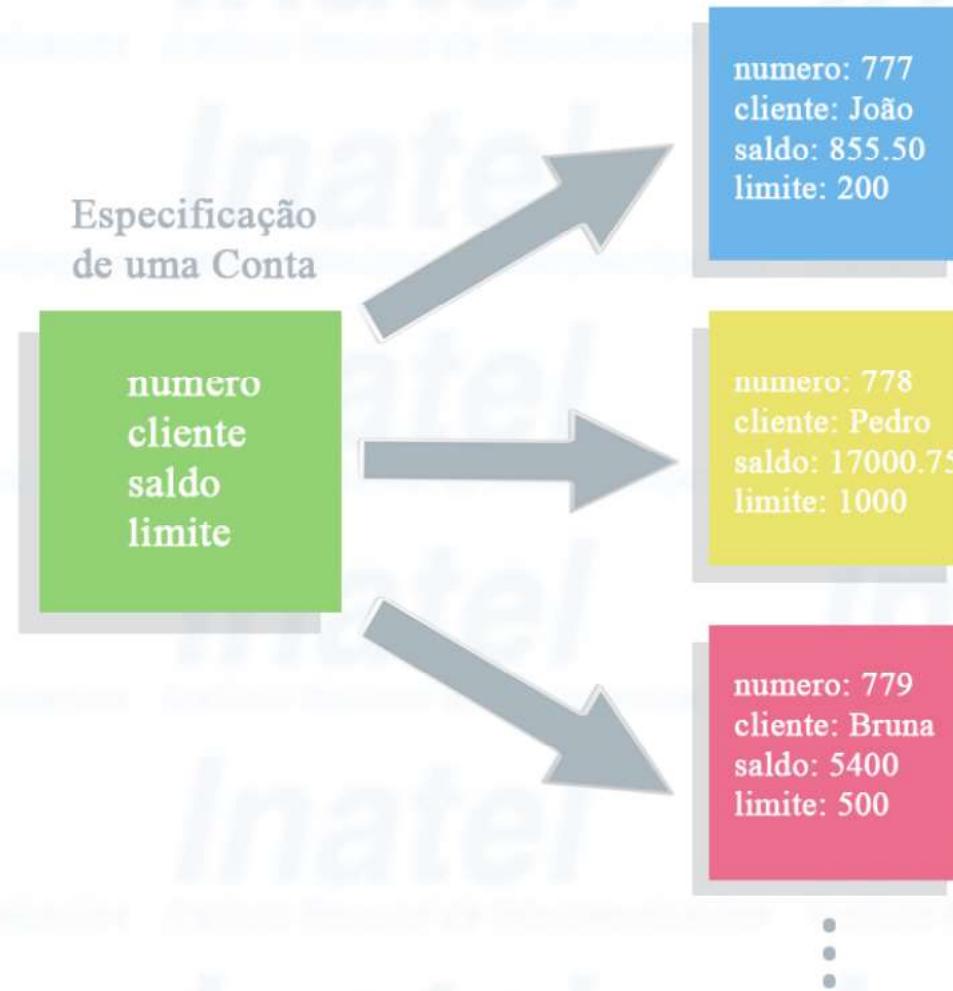
3.2. Classes e Objetos

Vejamos a Figura a seguir. Apesar do papel do lado esquerdo especificar uma conta, essa especificação é uma Conta? Podemos por exemplo depositar ou sacar dinheiro dessa Especificação? Não, pois este é apenas um projeto da Conta.

A partir de uma especificação podemos criar INSTÂNCIAS que realmente são contas, que aí sim poderemos realizar as operações que definimos.

Ao projeto da conta, isto é, a definição de como será uma conta, damos o nome de CLASSE;

Ao que podemos construir a partir de uma classe, neste caso, as contas de verdade, damos o nome de OBJETO;



3.2. Classes e Objetos

Outro exemplo: Uma Receita de Bolo

Podemos comer um receita de bolo? NÃO.

- Precisamos instanciá-la, criar um OBJETO bolo a partir dessa especificação (CLASSE).
- Podemos criar centenas de bolos a partir desta CLASSE (uma receita, no caso). Os bolos podem ser bem semelhantes e possuirem características iguais, mas são OBJETOS diferentes;

Classe



e



Objeto

3.3. Uma Classe em Java

Ainda utilizando do nosso exemplo da Conta, vamos começar apenas com que uma Conta tem, e não com o que ela faz (veremos em seguida).

Em Java, podemos começar escrevendo a Classe conta da seguinte maneira:

```
public class Conta {  
    int numero;  
    String donoDaConta;  
    float saldo;  
    float limite;  
  
    // Métodos...  
}
```

Estes são os atributos que toda conta, quando criada, deverá ter. Repare que as variáveis agora foram declaradas em um bloco separado, fora da Classe principal e método Main.

Quando uma variável é declarada diretamente dentro do escopo da Classe, é chamada de **Atributo** ou **Variável de Instância**;



3.4. Criando e usando um Objeto

Ok! Já temos uma Classe em Java que especifica o que todo objeto dessa Classe deve ter. Mas como usá-la?

Para criar (construir ou instanciar) uma Conta em outra Classe (como por exemplo na nossa Classe que possui o método main), basta usarmos a palavra *new*.

```
public class MeuPrograma {  
    public static void main(String[] args) {  
        new Conta();  
    }  
}
```

Jóia, o Objeto foi criado! Mas como acessá-lo? Precisamos de uma variável para referenciar este Objeto.

```
public class MeuPrograma {  
    public static void main(String[] args) {  
        Conta novaConta;  
        novaConta = new Conta();  
    }  
}
```

3.4. Criando e usando um Objeto

Agora, por meio da variável novaConta, podemos acessar o Objeto recém criado para alterar seu dono, saldo, etc.

Exemplo:

```
public class MeuPrograma {  
    public static void main(String[] args) {  
        Conta novaConta;  
        novaConta = new Conta();  
  
        novaConta.donoDaConta = "Duke";  
        novaConta.saldo = 1000;  
  
        System.out.println(novaConta.donoDaConta  
        + " seu saldo é: R$" +novaConta.saldo);  
    }  
}
```

Observações:

- Porque devemos utilizar os parênteses em Conta()?
- Porque escrevemos Conta duas vezes? Na declaração e no new?



Saberemos o motivo em breve! ;)

3.5. Métodos

Muitas vezes também temos que descrever os comportamentos que cada classe tem, ou seja, o que elas fazem. Fazemos isso por meio da criação de Métodos.

No caso da nossa conta, como podemos, por exemplo, sacar ou depositar dinheiro? Criando os métodos necessários para isso.

```
public class Conta {  
    //...Outros atributos  
    float saldo;  
  
    void saca(float quantia)  
    {  
        float novosaldo = saldo - quantia;  
        saldo = novosaldo;  
    }  
  
    void deposita(float quantia)  
    {  
        saldo += quantia;  
    }  
}
```

Observações:

- Neste caso nossos métodos não possuem retorno (void);
- Nosso métodos possuem parâmetros (float quantia);
- O que acontecem com as variáveis locais e parâmetros após a execução dos métodos?
MORREM!

3.5. Métodos

Para mandar uma mensagem ao objeto e pedir que ele execute um método, também usamos o ponto (.). O termo usado para isso é "Invocação de Método".

Exemplo:

```
// Sacando uma quantia da Conta  
novaConta.saca(300);  
  
// Depositando uma quantia na Conta  
novaConta.deposita(700);
```

Lembrando que um método também pode retornar um valor para um código que o chamou. Atenção com o tipo de retorno!

Exemplo:

```
boolean saca(float quantia)  
{  
    if(saldo > quantia)  
    {  
        saldo -= quantia;  
        return true;  
    }  
    else return false;  
}
```



3.6. UML e Diagrama de Classes

A *UML (Unified Modeling Language)* é uma linguagem visual para especificação, visualização e documentação de sistemas de software.

O **DIAGRAMA DE CLASSES** é um dos vários tipos de diagramas oferecidos pela UML. Ele tem como função descrever os vários tipos de objetos em um sistema e o relacionamento entre eles.

Por exemplo, para representar a nossa Classe Conta vista anteriormente por meio do Diagrama de Classes podemos utilizar do seguinte padrão:

Nome da Classe

Atributos

Métodos

Conta
+numero: int
+donoDaConta: String
+saldo: float
+limite: float
+saca(quantia : float): boolean
+deposita(quantia: float)

Obs: Conforme avançamos na disciplina aprenderemos outras propriedades deste diagrama.

3.6. UML e Diagrama de Classes

Exercício - JavaPets

Crie uma Classe chamada "AnimalPet" em Java que permitirá ao usuário criar novos animais quando necessário. Esta Classe deverá conter a seguinte especificação:

Depois de criada a Classe, crie pelo menos 3 pet's diferentes (Objetos) e faça com que cada um tome pelo menos 2 ações.

AnimalPet

+nome: String
+especie: String
+som: String
+comida: String
+idade: int
+comer()
+dormir(horas: int)
+movimentar(metros: int)
+fazerBarulho()



3.7. Objetos são acessados por referência

Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma maneira de acessá-lo, chamada de REFERÊNCIA.

É por esse motivo que, diferente dos tipos primitivos como int e long, precisamos dar "new" depois de declarada a variável.

Exemplo:

```
public class MeuPrograma {  
    public static void main(String[] args) {  
        // c1 se refere a um Objeto Conta  
        Conta c1 = new Conta();  
        // c2 se refere a OUTRO Objeto Conta  
        Conta c2 = new Conta();  
    }  
}
```

Não é correto dizer que c1 e c2 são Objetos, mas sim, que são VARIÁVEIS QUE SE REFEREM A UM OBJETO. Em Java uma variável NUNCA é um Objeto.

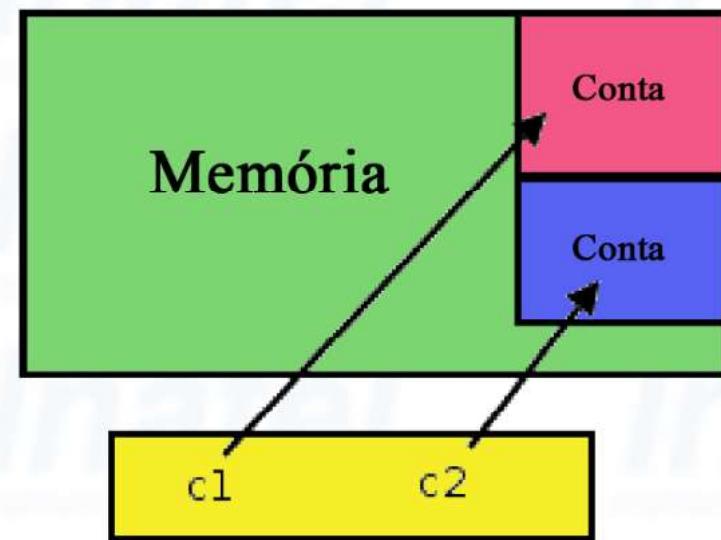
3.7. Objetos são acessados por referência

Internamente, c1 e c2 vão guardar um número que identifica em que posição de memória aquela Conta se encontra. Dessa maneira, ao utilizarmos o "." para navegar, o Java vai acessar a conta que se encontra naquela posição de memória, e não uma outra.

```
Conta c1 = new Conta();
```

```
Conta c2 = new Conta();
```

====



Estas variáveis funcionam como ponteiros, porém, não podemos manipulá-las como números e nem utilizá-las para aritmética, pois elas são tipadas.



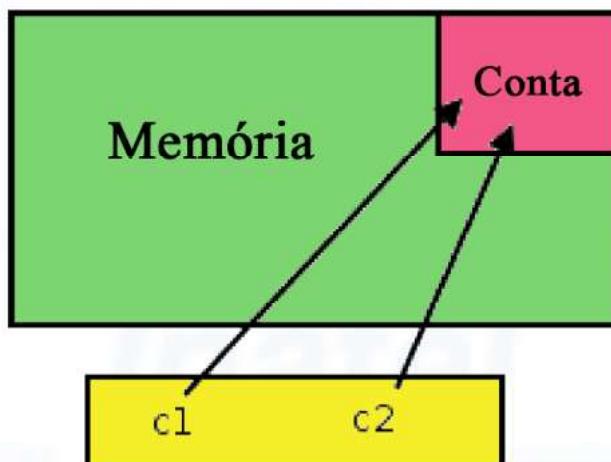
3.7. Objetos são acessados por referência

Outro ponto importante. Na memória, o que aconteceria neste caso abaixo?

Exemplo: Conta c1 = new Conta();

```
Conta c2 = c1;
```

Quando fizemos `c2 = c1`, `c2` passa a fazer referência para o mesmo objeto que `c1` referencia neste instante. Logo, `c1` e `c2` agora estão referenciando o mesmo Objeto.



Observação:

- Neste caso como demos apenas um "new", então só haverá um Objeto Conta na memória.

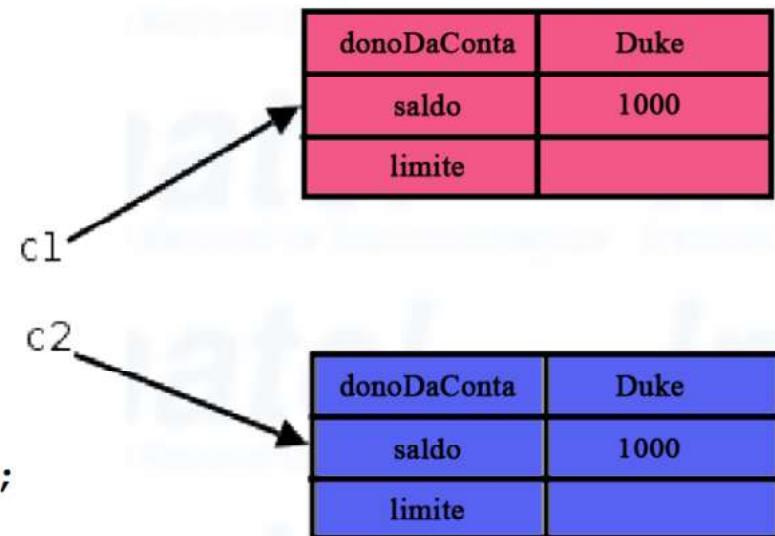
3.7. Objetos são acessados por referência

Outra situação importante. No exemplo abaixo, os Objetos são iguais ou não? Como em cada uma das variáveis (c1 e c2) guardamos duas contas criadas diferentemente, elas estão em espaços de memória diferentes. Logo, os objetos não são iguais.

```
Conta c1 = new Conta();
c1.donoDaConta = "Duke";
c1.saldo = 1000;

Conta c2 = new Conta();
c2.donoDaConta = "Duke";
c2.saldo = 1000;

if(c1 == c2)
{
    System.out.println("Contas iguais!");
}
else
{
    System.out.println("Contas diferentes!");
}
```



3.7. Objetos são acessados por referência

E se quisermos criar um método que transfere dinheiro entre duas contas?

Podemos criar um método que receba como parâmetros um parâmetro do tipo Conta (neste caso, a conta destino) e um valor a ser transferido.

```
public class Conta {  
    //..Outros atributos e métodos  
  
    void transfere(Conta contaDestino, float valor)  
    {  
        this.saldo = this.saldo - valor;  
        contaDestino.saldo += valor;  
    }  
}
```

Mas e quando passamos uma variável de Objeto como parâmetro, por acaso ele é clonado?

Não. Lembre-se que o valor dessa variável é um endereço de memória, uma REFERÊNCIA, logo, não existe cópia de objetos aqui.

3.7. Objetos são acessados por referência

Exercício - TrabalhadorBrasileiro

Crie uma Classe chamada "Trabalhador" em Java que permitirá ao usuário criar novos trabalhadores quando necessário. Esta Classe deverá conter a seguinte especificação:

Depois de criada a Classe, crie pelo menos 2 trabalhadores e teste seus métodos.

Dicas:

- O método calculaGanhoAnual() deverá retornar o total que o trabalhador recebe no ano;
- O método mostraInfosFuncionario() mostra de uma só vez todas as informações de um determinado funcionário;

Trabalhador
+nome: String +profissao: String +salario: float +rg: String +dataNascimento: String
+recebeAumento(valor: float) +calculaGanhoAnual(): float +mostraInfosFuncionario()



3.7. Objetos são acessados por referência

Exercício - TrabalhadorBrasileiro (Pt.2)

- 1) Construa dois funcionários, com os mesmos valores de atributos, mas cada um com o seu "new". Compare-os com "==".
O que acontece?
- 2) Agora crie duas referências para o mesmo funcionário e também compare-as com "==".
O que acontece?
- 3) O que acontece se tentarmos acessar um atributo ou método diretamente na Classe?

Dicas:

```
if (t1 == t2) System.out.println("Objetos Iguais!");  
else System.out.println("Objetos Diferentes!");
```

```
System.out.println(Funcionario.salario); //Funciona?
```



3.8. Classes dentro de Classes

As Classes em muitos casos não trabalham sozinhas, mas se comunicam umas com as outras. A essa comunicação entre elas damos o nome de ASSOCIAÇÃO.

Uma associação indica que um objeto contém ou está conectado a outro objeto. Devemos utilizá-la quando um objeto contiver outro objeto (o relacionamento tem um) ou quando um objeto toma ação sobre outro objeto.

Exemplos:



A seguir, veremos que a associação ainda pode se quebrar em dois outros tipos distintos, denominadas associações Todo/Parte. São elas:

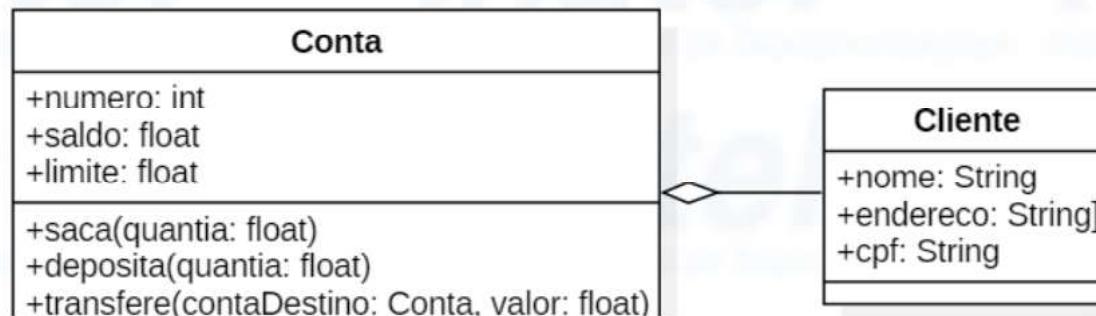


3.8. Classes dentro de Classes

Imagine que começemos a aumentar nossa classe Conta adicionando nome, sobrenome e cpf do cliente dono da conta. Começaríamos a ter muitos atributos, não?

Se pensarmos direito, uma conta não tem nome, nem sobrenome ou cpf. Quem tem esses atributos é um cliente. Então podemos criar uma nova classe (Cliente) e fazer uma AGREGAÇÃO entre elas.

Exemplo:



```
public class Conta {
    int numero;
    float saldo;
    float limite;
    Cliente titular;
    // resto da classe..
}

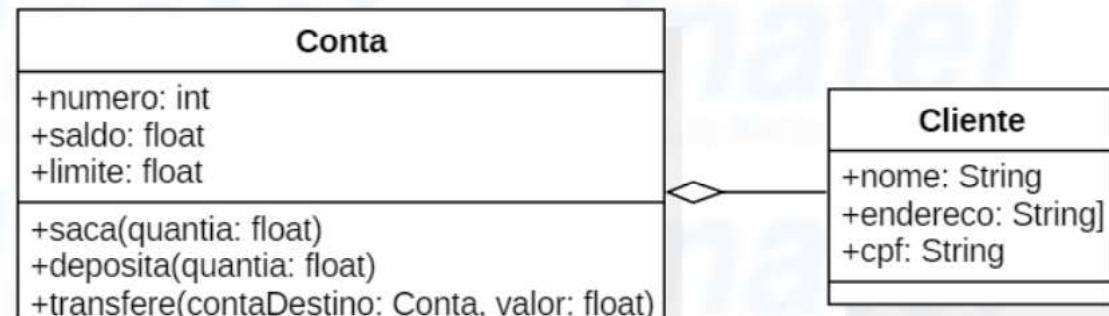
public class Cliente {
    String nome;
    String sobrenome;
    String cpf;
    //...resto da classe
}
```

3.8. Classes dentro de Classes

Mas afinal de contas, como podemos conceituar a AGREGAÇÃO?

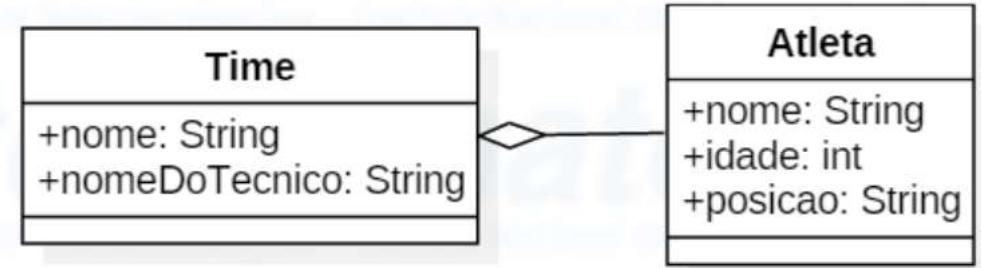
É um tipo de associação Todo/Parte. Na Agregação, a existência do Objeto-Parte faz sentido, mesmo não existindo o Objeto-Todo.

No nosso exemplo anterior, um Cliente pode existir independentemente de uma Conta existir.



Outro Exemplo: Um Time de Futebol

Um time é formado por atletas, ou seja, os atletas são parte integrante de um time, mas os atletas existem independentemente de um time existir.



3.8. Classes dentro de Classes

Agora suponha que tenhamos a seguinte situação na classe main:

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

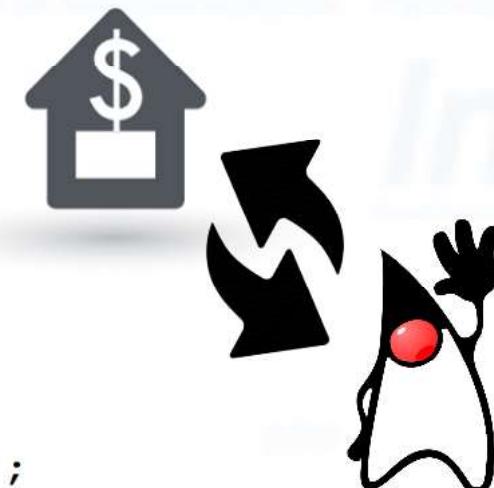
Aqui, simplesmente houve uma atribuição. O valor da variável c é copiado para o atributo titular do objeto ao qual minhaConta se refere. Agora, minhaConta guarda uma referência ao mesmo cliente que c se refere, e pode ser acessado através de minhaConta.titular .

3.8. Classes dentro de Classes

Podemos realmente navegar sobre toda a estrutura de classes, sempre usando o ponto.

Exemplo:

```
public class MeuPrograma {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
        Cliente c = new Cliente();  
        c.nome = "Duke";  
        c.sobrenome = "Java";  
        minhaConta.titular = c;  
        System.out.print(minhaConta.titular.nome);  
        System.out.print(" "+minhaConta.titular.sobrenome);  
    }  
}
```



Um sistema Orientado a Objetos é um grande conjunto de classes que se comunicam, delegando responsabilidades para quem for mais apto a realizar determinada tarefa. Dizemos que esses objetos colaboram, TROCANDO MENSAGENS ENTRE SI.

3.8. Classes dentro de Classes

Um ponto importante! E se dentro do meu código eu não desse "new" em Cliente e tentasse acessá-lo diretamente?

Exemplo:

```
public class MeuPrograma {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
        System.out.print(minhaConta.titular.nome);  
    }  
}
```

Neste caso daria um erro de NullPointerException pois o objeto não foi instanciado, ou seja, não possui uma referência de memória para trabalhar!

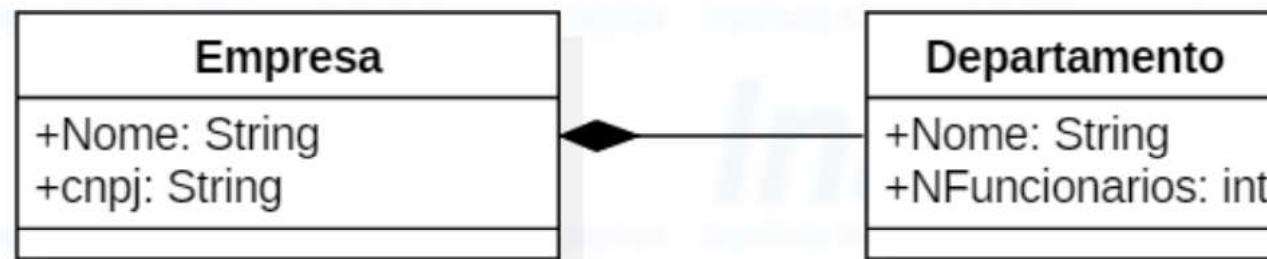


3.8. Classes dentro de Classes

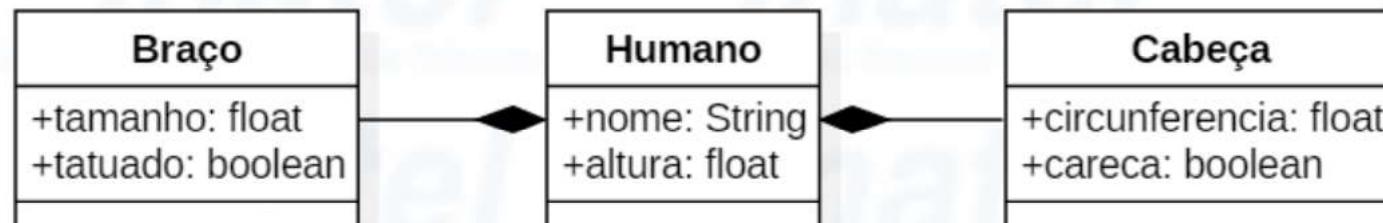
Nos exemplos anteriores, realizamos nossas implementações por meio de Agregações. Outro conceito importante é o da **COMPOSIÇÃO**.

A **Composição** também é uma associação Todo/Parte e funciona como uma agregação mais forte. Nela, a existência do Objeto-Parte NÃO faz sentido se o Objeto-Todo não existir.

Exemplo: Uma Empresa com seus Departamentos

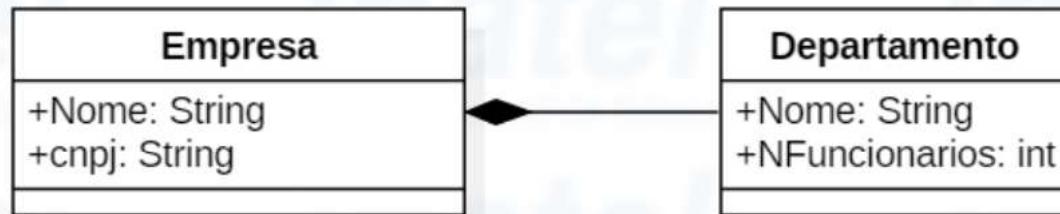


Outro exemplo: Um Humano com seus membros



3.8. Classes dentro de Classes

Um detalhe importante na COMPOSIÇÃO é que aqui o "todo" é responsável por suas "partes", ou seja, o "todo" também é responsável pela criação e destruição das suas "partes".



Mas afinal, o que é este Construtor?

Um construtor é um "pseudo"-método que determina as ações que devem ser executadas quando um objeto é criado.

Obs: mais adiante vamos explorar outras características importantes dos construtores das Classes.

```
public class Empresa {
    String nome;
    String cnpj;
    Departamento dpt;

    // Construtor da Classe
    public Empresa() {
        dpt = new Departamento();
    }
    //...resto da Classe
}

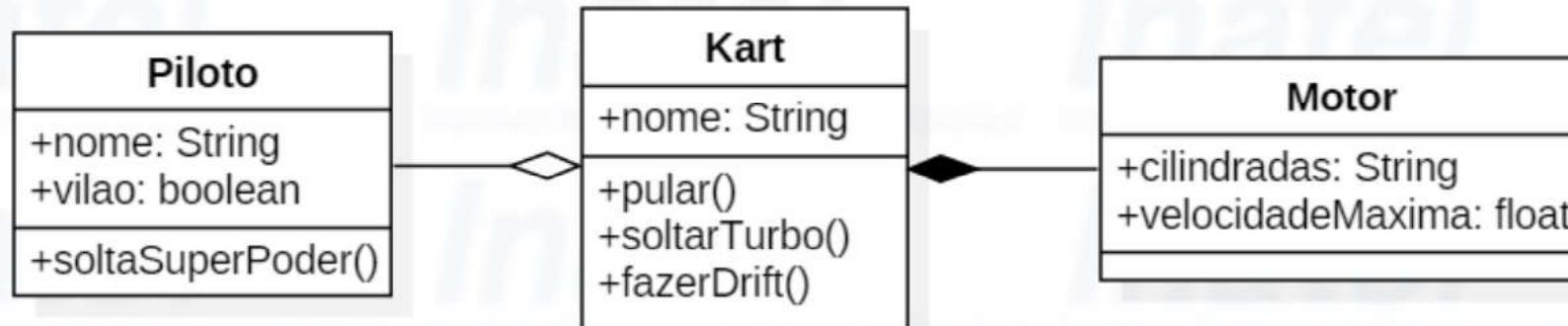
public class Departamento {

    String nome;
    int nFuncionarios;
    //...resto da classe
}
```

3.8. Classes dentro de Classes

Exercício 4 - MarioKart

Crie Classes em Java que atendam a seguinte especificação:



Depois de criada as Classes, crie pelo menos 2 objetos de cada, os relacionem e teste seus métodos.

Dicas:

- Todos os métodos propostos devem simplesmente imprimir as ações que eles realizam;
- As cilindradas podem assumir três valores: 50, 100 e 150;
- A velocidade pode assumir valores de 0 a 150 Km/h;
- O atributo vilao indica se o piloto é do tipo vilão ou não;
- Alguns nomes de Kart's famosos:

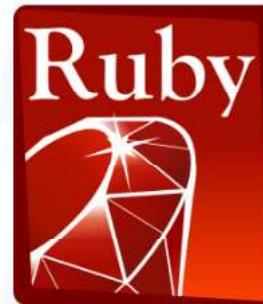
Standard Egg1 Blue Falcon
Mushmellow Sprinter Piranha P



3.9. Linguagens Orientadas a Objetos

Aprendendo os conceitos de Orientação a Objetos, poderei programar só em Java?

De jeito nenhum! Outras linguagens que também utilizam do paradigma Orientado a Objetos e possuem grande destaque no mercado de trabalho:

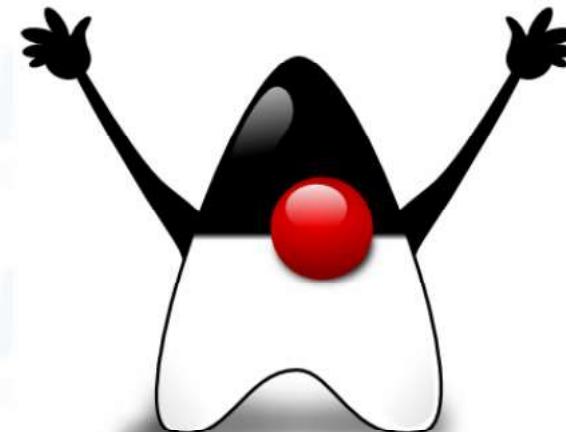


e muitas outras..

O que muda entre elas é a sintaxe na hora de criar e manipular objetos, mas os conceitos são os mesmos! :)

Obs: Lembrando que o que vimos até agora é apenas uma rápida Introdução a Orientação a Objetos. Muitos conceitos importantes ainda serão explorados nesta disciplina.

**FIM
DO
CAPÍTULO 3**



Próximo Capítulo
Arrays