

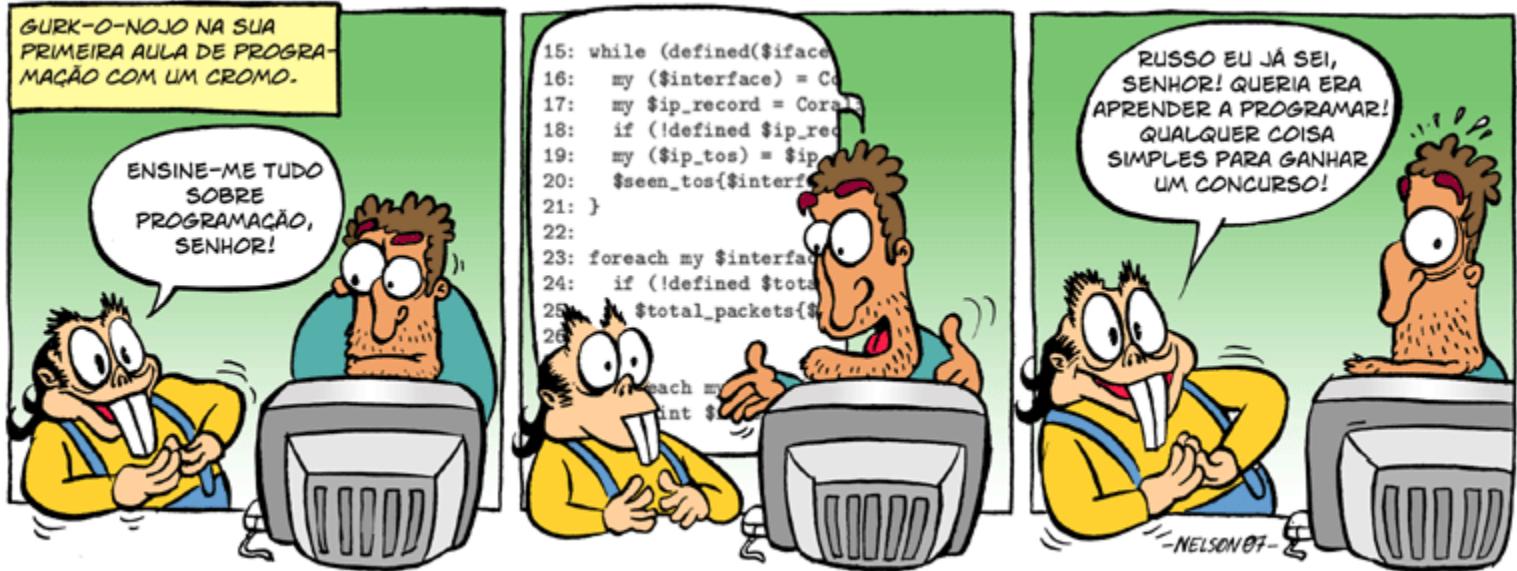


# CONCEITOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

**Professor:** Dr. Almir Rogério Camolesi  
[camolesi@fema.edu.br](mailto:camolesi@fema.edu.br)

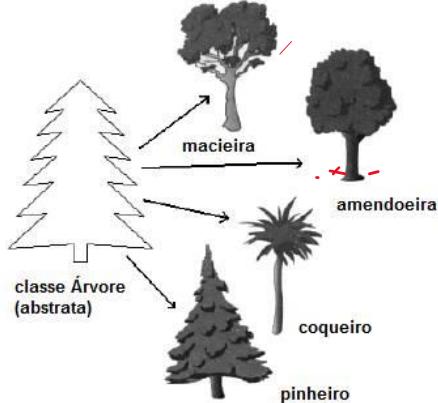
```
public class Castigo {  
    public static void main(String[] args) {  
        for(int i=1; i<=100; i++)  
            System.out.println  
            (" SEJAM BEM VINDOS");  
    }  
}
```





OS ESPECIALISTAS - [WWW.NITRODESIGN.COM/ESPECIALISTAS/](http://WWW.NITRODESIGN.COM/ESPECIALISTAS/)

©2007 - PEDRO COUTO E SANTOS E NELSON MARTINS

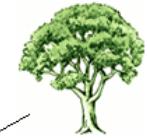


## Classe Árvore

???????

(Qualquer uma. Abstrata.)

Objeto árvore



## Classe

Pessoa

- Nome
- Endereço
- Telefone
- Idade
- Altura

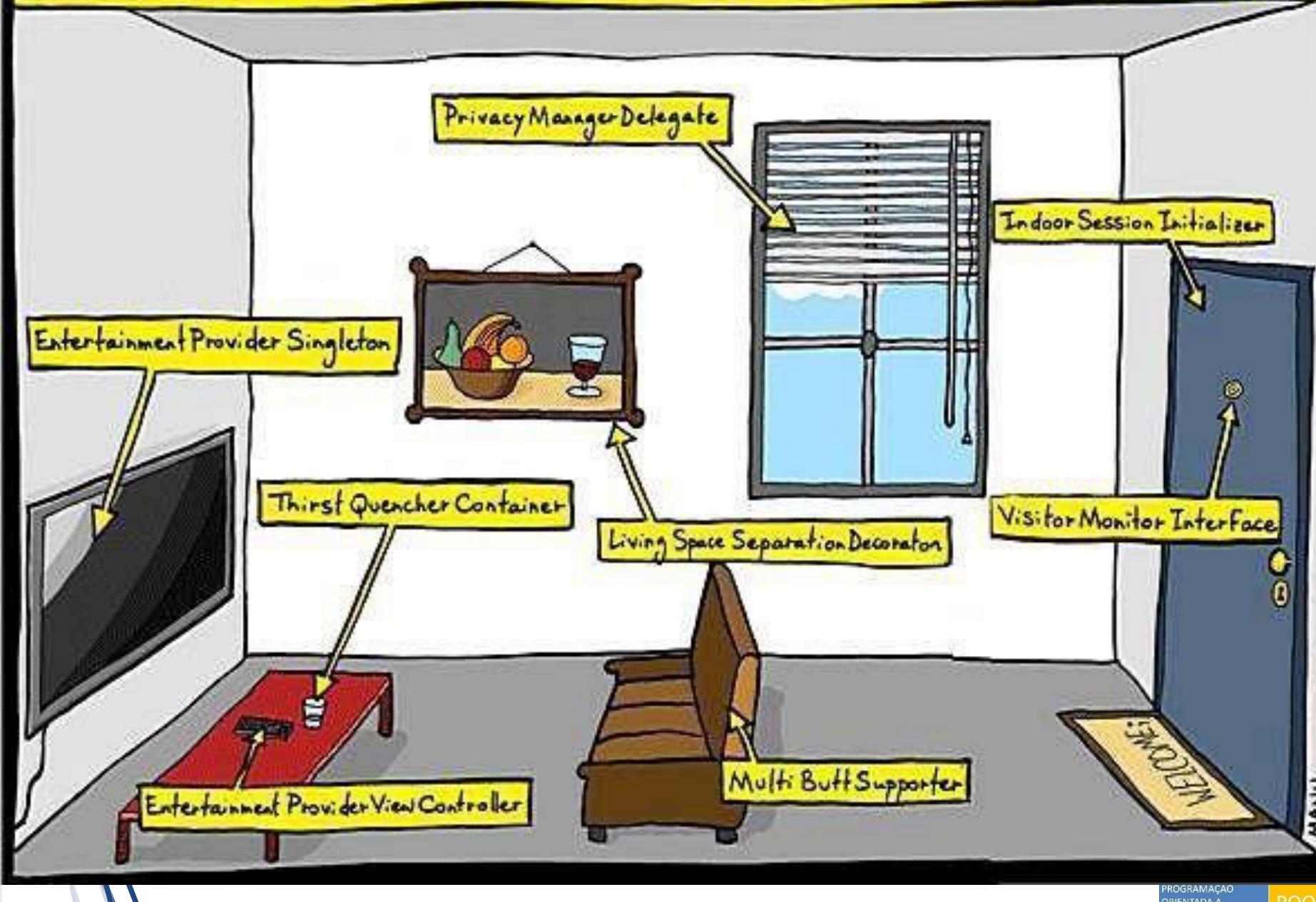
A T R I B U T O S M É T O D O S

Maria

Pedro

## Objetos

# THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.



## **DE ONDE VEM OS BEBES** - BY JULIOALVES



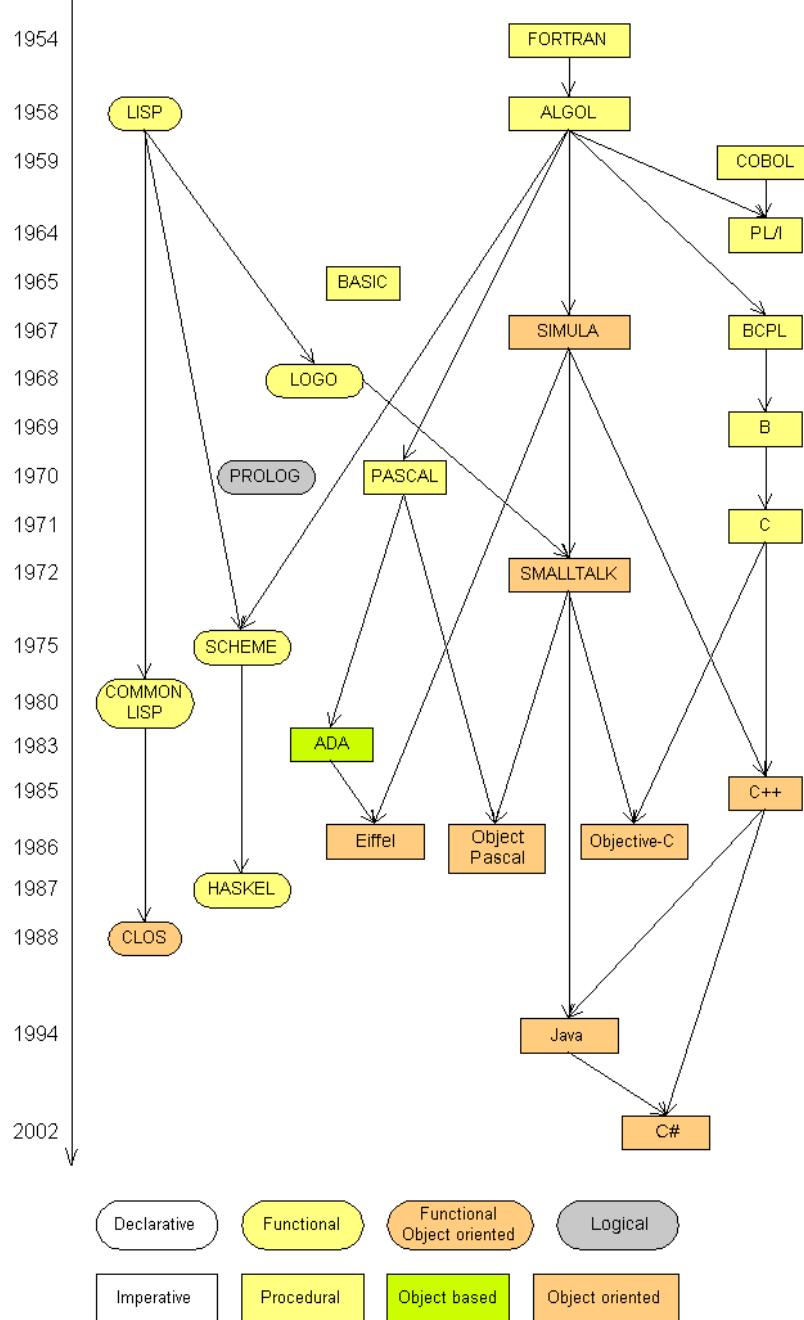
# O que vamos ver?



# Conceitos de Orientação a Objeto

## Histórico

- Os conceitos de orientação a objetos vêm sendo discutido há muito tempo, desde o lançamento da 1<sup>a</sup> linguagem orientada a objeto – SIMULA.
- Primeiros autores dessa metodologia foram Peter Coad, Edward Yourdon e Roger Pressman



# Conceitos de Orientação a Objeto

Os conceitos discutidos por **Coad, Yourdon e Pressman** são:

- | Orientação a objeto é uma tecnologia para a produção de modelos que especifique o domínio do problema de um sistema;
- | Quando construídos corretamente, sistemas orientados a objetos são flexíveis a mudanças, operando com componentes totalmente reutilizáveis;
- | Orientação a objeto não é só teoria, mas uma tecnologia de eficiência e qualidade comprovada usada em inumeros projetos de sistemas.

# Conceitos de Orientação a Objeto

As principais metodologias que deixaram a orientação objeto popular nos anos 90 são:

- | **Booch** – definiu que um sistema é analisado a partir de um número de visões, onde cada visão é descrita por vários diagramas – visão Macro e Micro de processos;
- | **OMT**- Técnica de Modelagem de Objetos, desenvolvida pela GE por Rumbaugh, opera sobre análise de **requisito e teste** dos modelos – apoia-se em **objetos, funções e use-case**;
- | **OOSE/Objectory** - baseia em Jacobson sendo orientado a objeto e utilizando **use-cases**, definindo os requisitos iniciais do sistema, **abordados como ator externo**.

# Conceitos de Orientação a Objeto

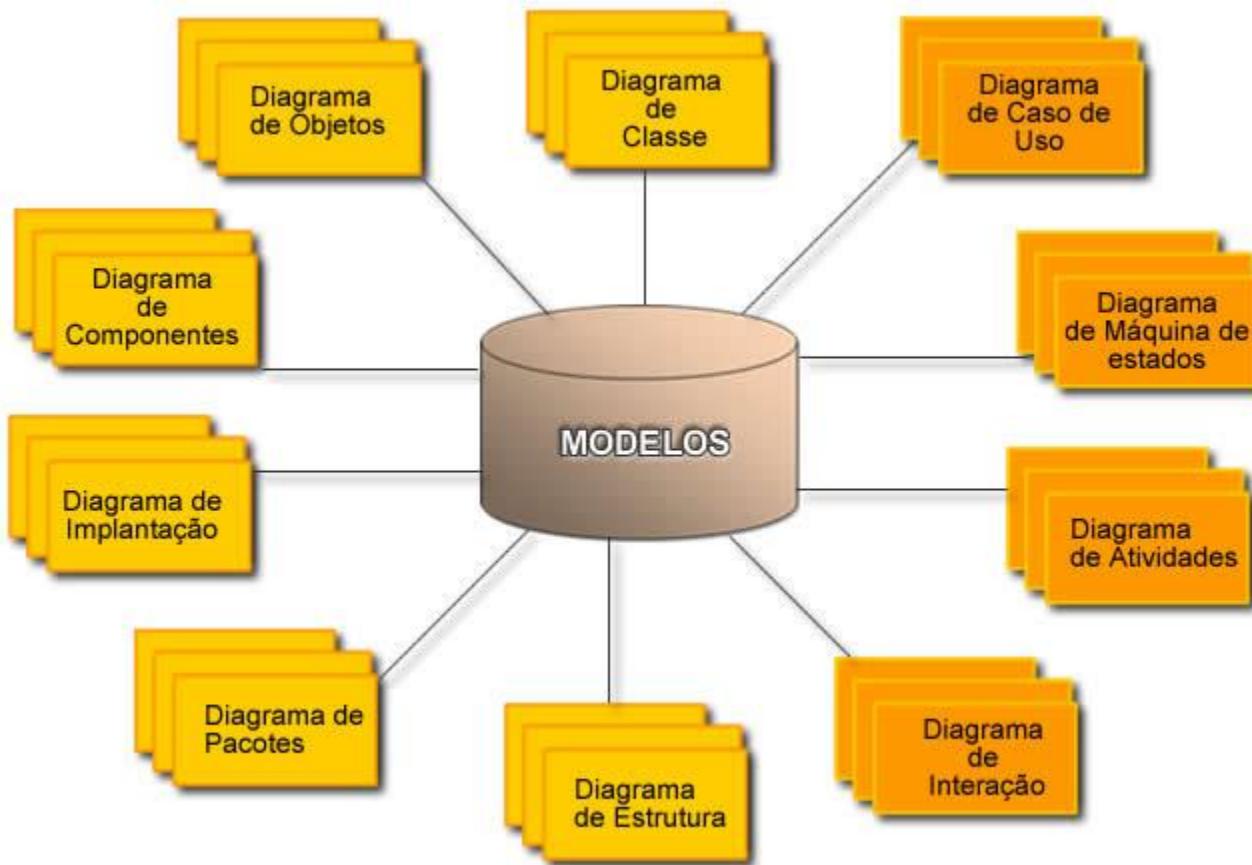
## UML – Unified Modeling Language

Diante desta diversidade de conceitos, "os três amigos", Grady Booch, James Rumbaugh e Ivar Jacobson decidiram criar uma **Linguagem de Modelagem Unificada - UML**. Eles disponibilizaram inúmeras versões preliminares da UML para a comunidade de desenvolvedores e a resposta incrementou muitas novas idéias que melhoraram ainda mais a linguagem.

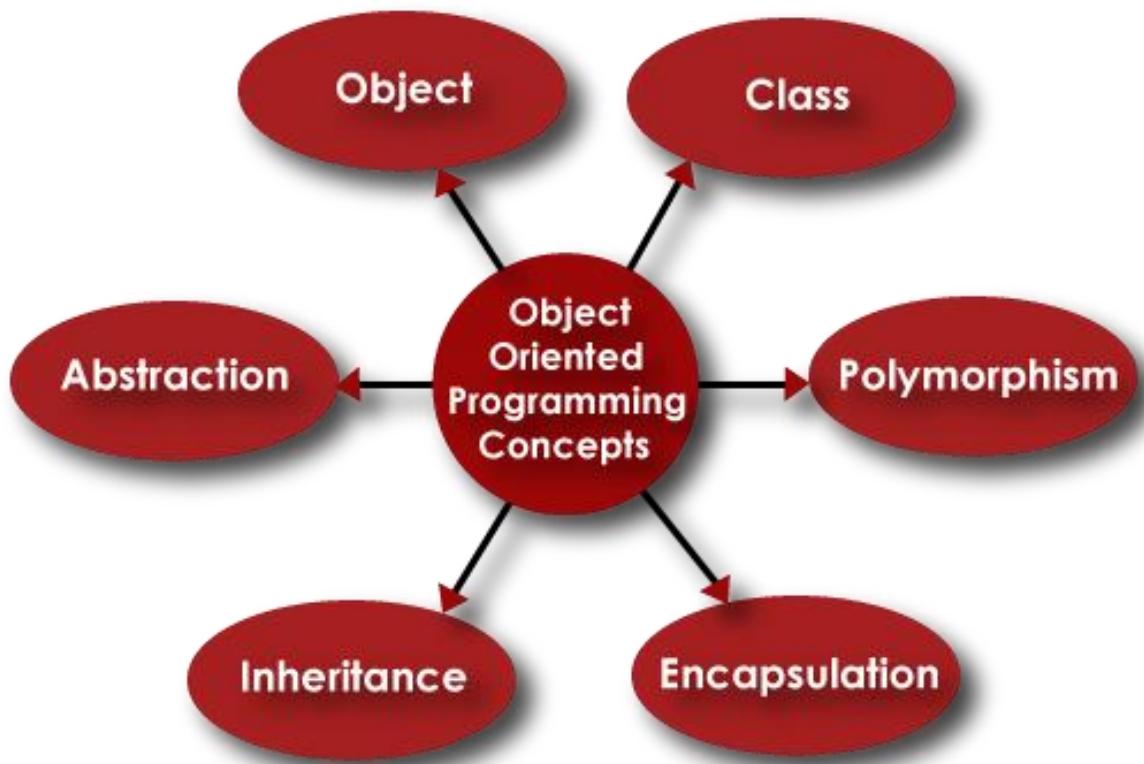
O objetivo da UML é descrever qualquer tipo de sistema, em termos de diagramas orientados a objetos.

*Obs: este não é o foco desta etapa, modelagem orientada a objetos vocês discutirão modelagem na disciplina de Engenharia de Software.*

# Diagramas de Modelagem Orientação a Objetos



# Conceitos de Programação Orientada a Objetos

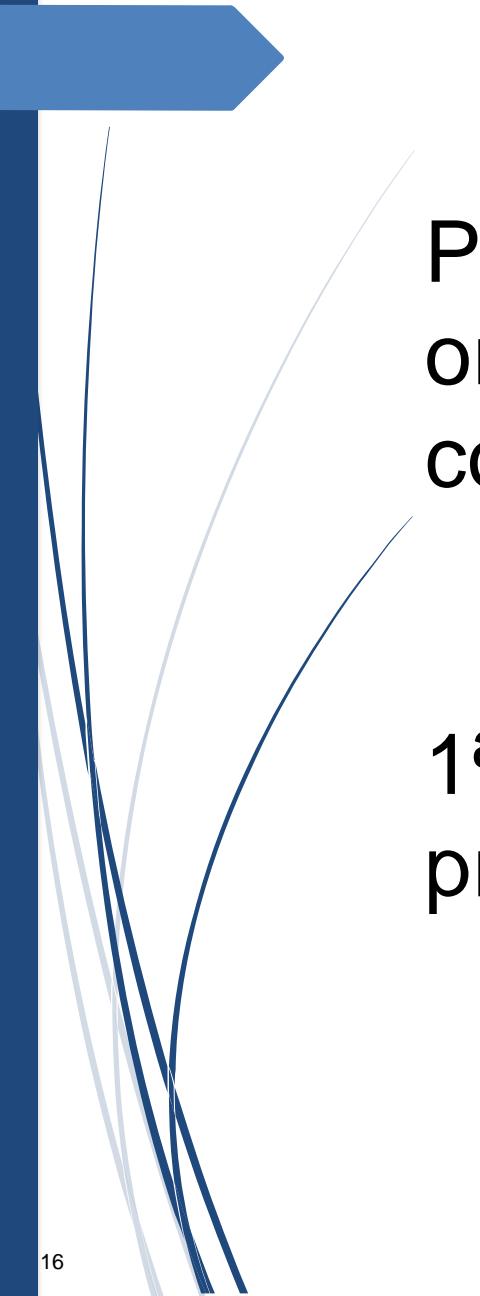




# CONCEITOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

**Professor:** Dr. Almir Rogério Camolesi  
[camolesi@fema.edu.br](mailto:camolesi@fema.edu.br)

# Conceitos de Orientação a Objeto



Para entendermos os conceitos de orientação a objetos, vamos começar com duas perguntas fáceis:

- 1<sup>a)</sup>) Olhe para os automóveis das próximas páginas.

# Conceitos de Orientação a Objeto



# Conceitos de Orientação a Objeto



Artwork by FreakOne

# Conceitos de Orientação a Objeto



# Conceitos de Orientação a Objeto



# Conceitos de Orientação a Objeto

Após observar as figuras....

Quais características similares você identificaria nesses veículos?

# Conceitos de Orientação a Objeto

2<sup>a)</sup>) Como se calcula um seguro total para cada um desses veículos?

# Conceitos de Orientação a Objeto

- Pois bem, até hoje o foco da modelagem concentrou-se nas funcionalidades de um sistema.
- Módulos distintos atuavam diretamente sobre as bases de dados, às vezes de forma desordenada.

# Conceitos de Orientação a Objeto

- Quando qualquer alteração era feita nessas bases, diversos módulos eram afetados.
- Erros advindos dessas alterações, por muitas vezes, só eram descobertos semanas mais tarde.

# Conceitos de Orientação a Objeto

- A grande vantagem que obtemos com a orientação a objetos é o fato de podermos abstrair de uma maneira mais fiel as situações do dia-a-dia.
- Para alcançarmos esta facilidade precisamos mudar a forma de pensar sobre sistemas.

# Conceitos de Orientação a Objeto

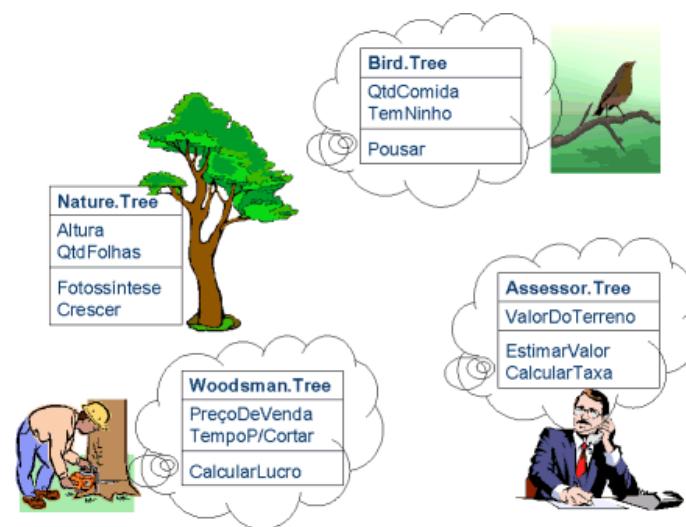
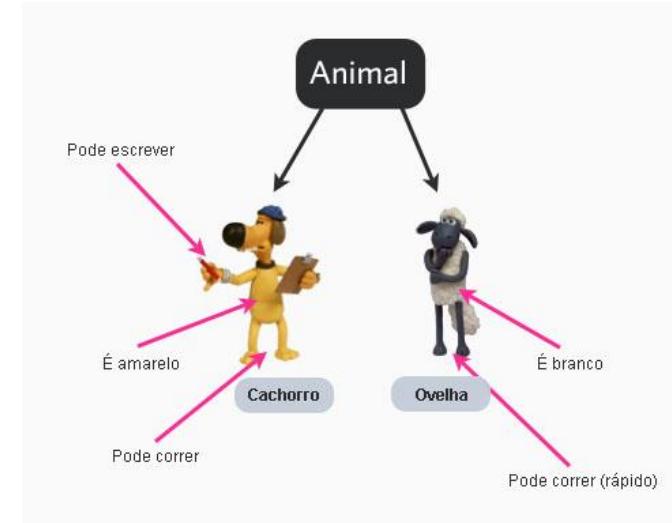
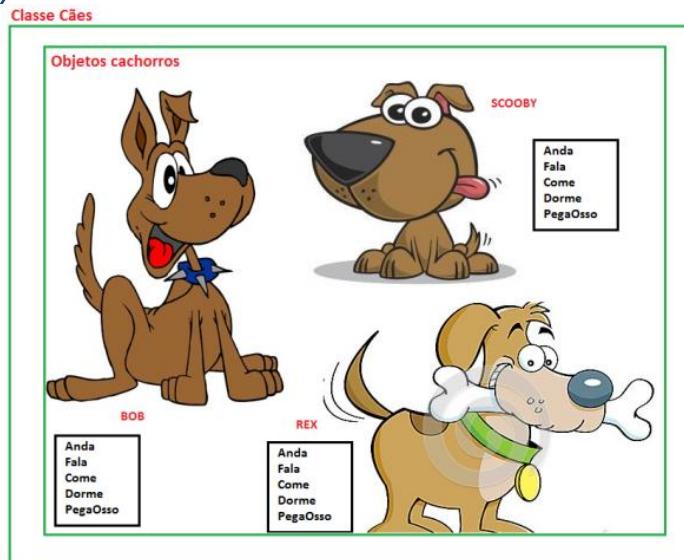
- Esta abstração é feita por representações do mundo real, chamadas de objetos.
- Só precisamos voltar a praticar o conhecimento que possuíamos desde a nossa infância:
  - identificar os objetos e seus comportamentos, o que possibilita que eles sejam categorizados.

# Programação Orientada a Objetos



<https://youtu.be/QY0Kdg83orY>

# Objeto



# Conceitos de Orientação a Objeto

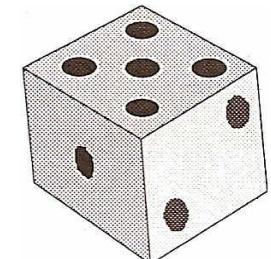
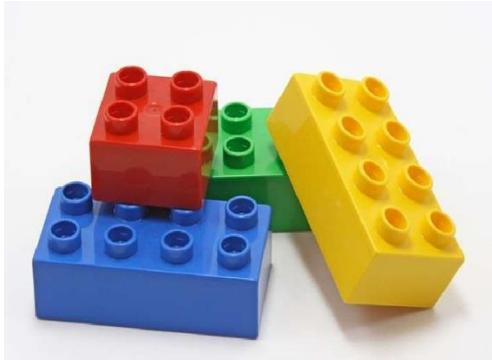
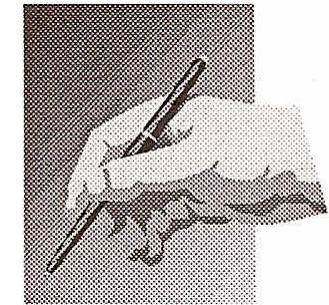
## A orientação a objeto:

é uma abordagem para o desenvolvimento de software que organiza os problemas e suas soluções como um conjunto de objetos distintos.

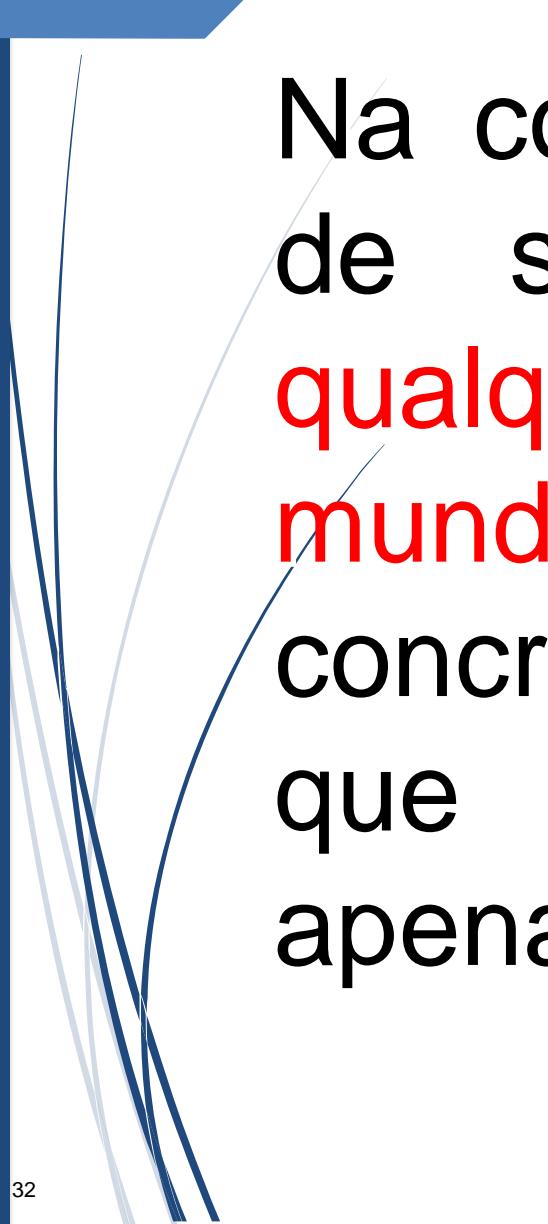
# Conceitos de Orientação a Objeto

- Passamos a identificar os objetos por suas características e comportamentos.
- Ao olharmos uma caneta, independentemente de seu formato ou cor de sua tinta, conseguimos identificá-la

# Conceitos de Orientação a Objeto



# Conceitos de Orientação a Objeto



Na concepção de modelagem de sistemas, um **objeto** é **qualquer coisa existente no mundo real**, em formato concreto ou abstrato, ou seja, que exista fisicamente ou apenas conceitualmente.

# Conceitos de Orientação a Objeto

São exemplos de objetos:

- aluno, professor, mesa, cadeira, caneta, automóvel, disciplina, estoque, avaliação, seguro, janela do Windows, botão, caixa de diálogo, etc.

Isto significa que ao modelarmos um sistema baseado no paradigma da orientação a objetos, nada mais estamos fazendo do que modelar os conceitos existentes em nosso cotidiano.

# Conceitos de Orientação a Objeto

- Os objetos possuem características ou propriedades que são seus atributos.
- Esses atributos identificam o estado de um objeto.
- Um atributo é uma abstração do tipo de dados ou estado que os objetos da classe possuem.

# Conceitos de Orientação a Objeto



## Atributos do objeto Joana:

Nome:	<b>Joana Souza</b>
Endereço:	<b>Rua Santa Cecília, 1000</b>
Sexo:	<b>Feminino</b>
Altura:	<b>1,75 m</b>
Peso:	<b>60 Kg</b>
Estado Civil:	<b>Solteira</b>
Cor dos olhos:	<b>verdes</b>
Cor dos cabelos:	<b>castanhos</b>
etc.	

# Conceitos de Orientação a Objeto

- Tipicamente, identificamos e diferenciamos objetos por seus atributos.
- Podemos fazer uma brincadeira: a cada atributo que eu relacionar, tentem identificar possíveis objetos.
- Vocês vão perceber que ao obtermos um conjunto maior de atributos, podemos mais precisamente identificar o objeto.

# Conceitos de Orientação a Objeto



Vamos considerar a lista de atributos: nome, endereço, data de nascimento (até agora posso estar falando de um funcionário, aluno ou até mesmo um animal de estimação), especialização, CRM.

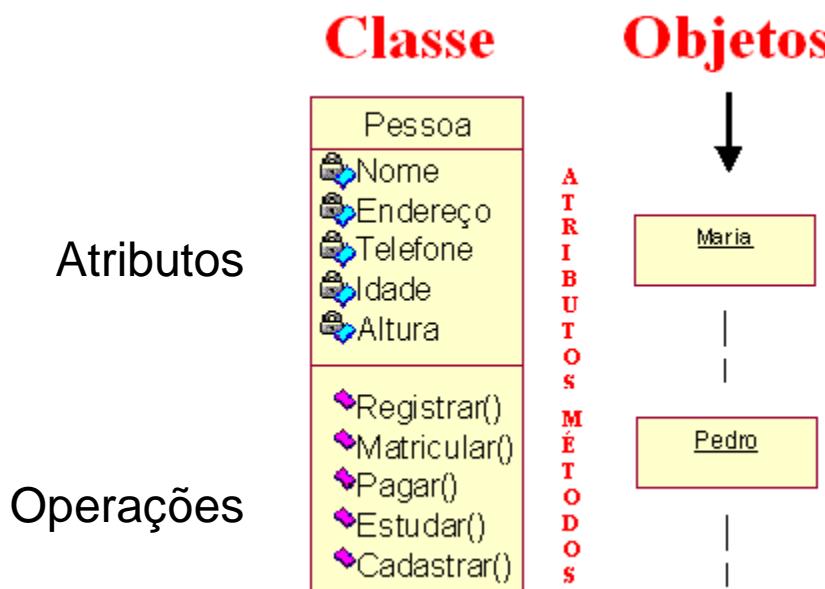
# Conceitos de Orientação a Objeto

- Além dos atributos, os objetos possuem comportamentos que modificam seu estado ou prestam serviços a outros objetos.
- Nesse caso, estamos falando de suas operações.
- Se um funcionário possui o atributo Salário, este deve ser atualizado por operações do tipo Reajustar Salário.

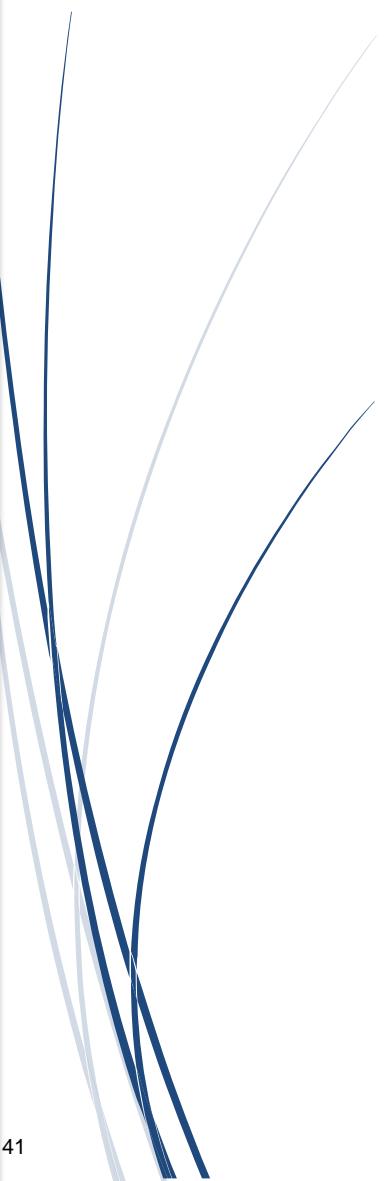
# Conceitos de Orientação a Objeto

- Utilizamos de forma conjunta o conceito de métodos.
- Um método é a implementação de uma operação, ou seja, sua representação em código.

# Conceitos de Orientação a Objeto



# Conceitos de Orientação a Objeto



Os métodos de uma classe manipulam somente as estruturas de dados daquela classe, ou seja, não podem acessar diretamente os dados de outra classe.

# Conceitos de Orientação a Objeto

- Um objeto possui limites nítidos com relação ao problema em estudo.
- Na modelagem, quando pensamos em um objeto, devemos fazê-lo dentro de um determinado contexto.

# Conceitos de Orientação a Objeto

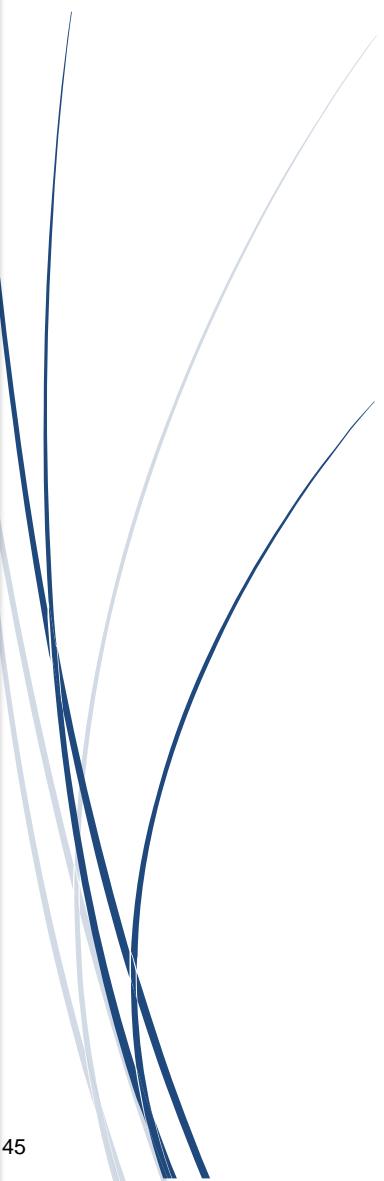


Devemos abstrair as informações de um objeto dentro do papel que ele exercerá num determinado sistema, ou seja, seu ambiente determinará o limite da modelagem de cada objeto.

# Conceitos de Orientação a Objeto

- Se pensarmos em todos os atributos que identificam uma pessoa, chegaremos a um número considerável.
- Todavia se modelarmos um objeto Pessoa no papel de um aluno, não precisaremos incluir atributos do tipo: altura, peso, cor dos olhos, cor da pele e muitos outros.

# Conceitos de Orientação a Objeto



Todavia, se estivermos falando do aluno de uma escola para modelos e manequins, esses atributos serão relevantes.

# Conceitos de Orientação a Objeto

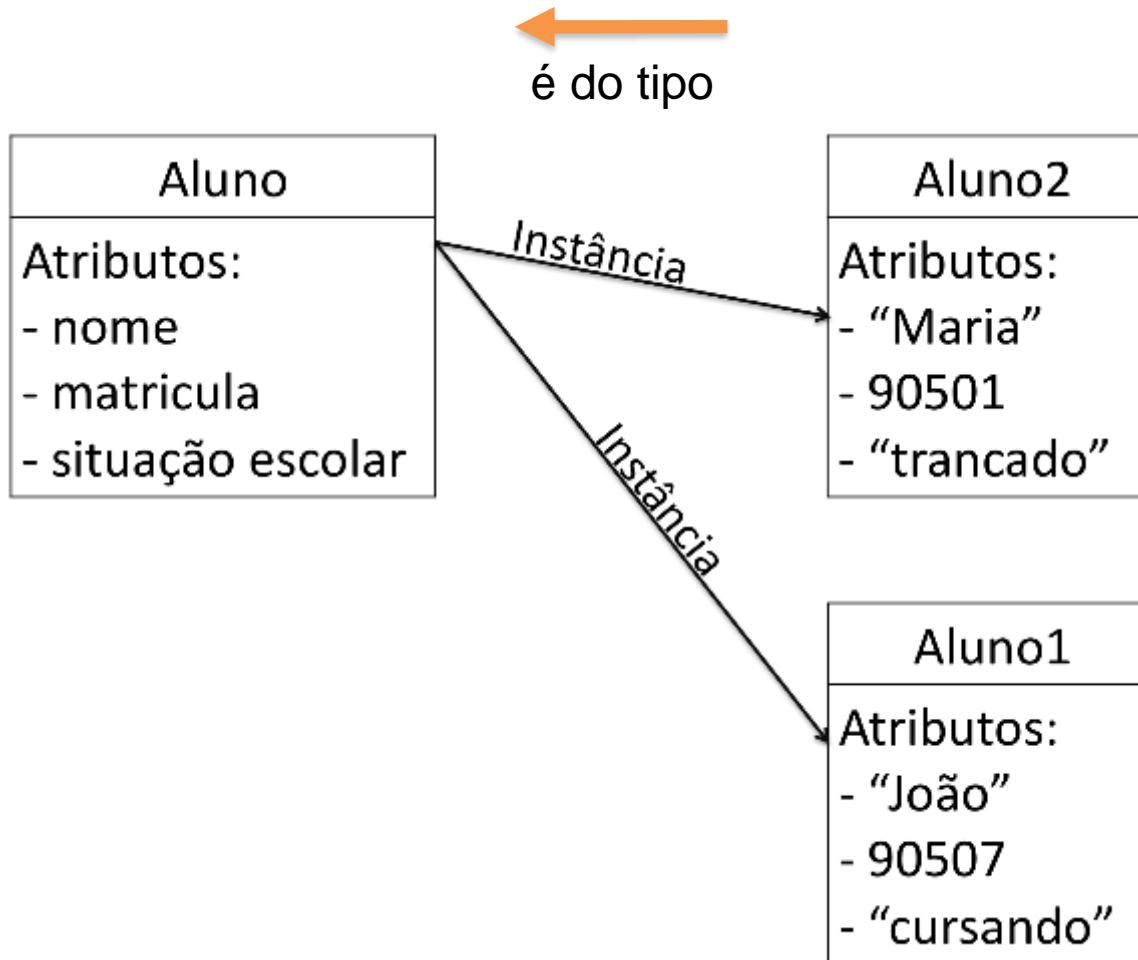
## Classes

- Quando identificamos características e operações similares em objetos distintos, estamos realizando sua classificação, ou seja, identificando classes.
- Uma classe é a representação de um conjunto de objetos que compartilham a mesma estrutura de atributos, operações e relacionamentos, dentro de um mesmo contexto (semântica).

# Conceitos de Orientação a Objeto

- Assim, uma classe especifica a estrutura de um objeto sem informar quais serão seus valores.
- Em contrapartida, um objeto corresponde à ocorrência (instância) de uma classe num determinado momento.

# Conceitos de Orientação a Objeto



# Conceitos de Orientação a Objeto

- Num sistema, trabalhamos com as instâncias de uma classe (os objetos criados a partir desta classe).
- Os dados são carregados nas instâncias.
- Para exemplificar, pense num formulário de inscrição.

# Conceitos de Orientação a Objeto



A secretária de uma empresa cuidadosamente desenha esse formulário para ser preenchido pelos clientes, criando uma matriz.

# Conceitos de Orientação a Objeto

- Todavia, o que lhes é entregue é uma xerox dessa matriz e não a original.
- O mesmo acontece com as classes e objetos.
- As classes funcionam como matrizes.
- Os objetos (suas instâncias) é que possuem vida própria.

# Conceitos de Orientação a Objeto

CLASSE	OBJETOS	
Funcionario	func1	func2
Nome	Ana Cristina	Gustavo
Empresa	Casa de Festas ABC	Software Ltda
Veículo	Vectra	Corolla

Atributos  
↑

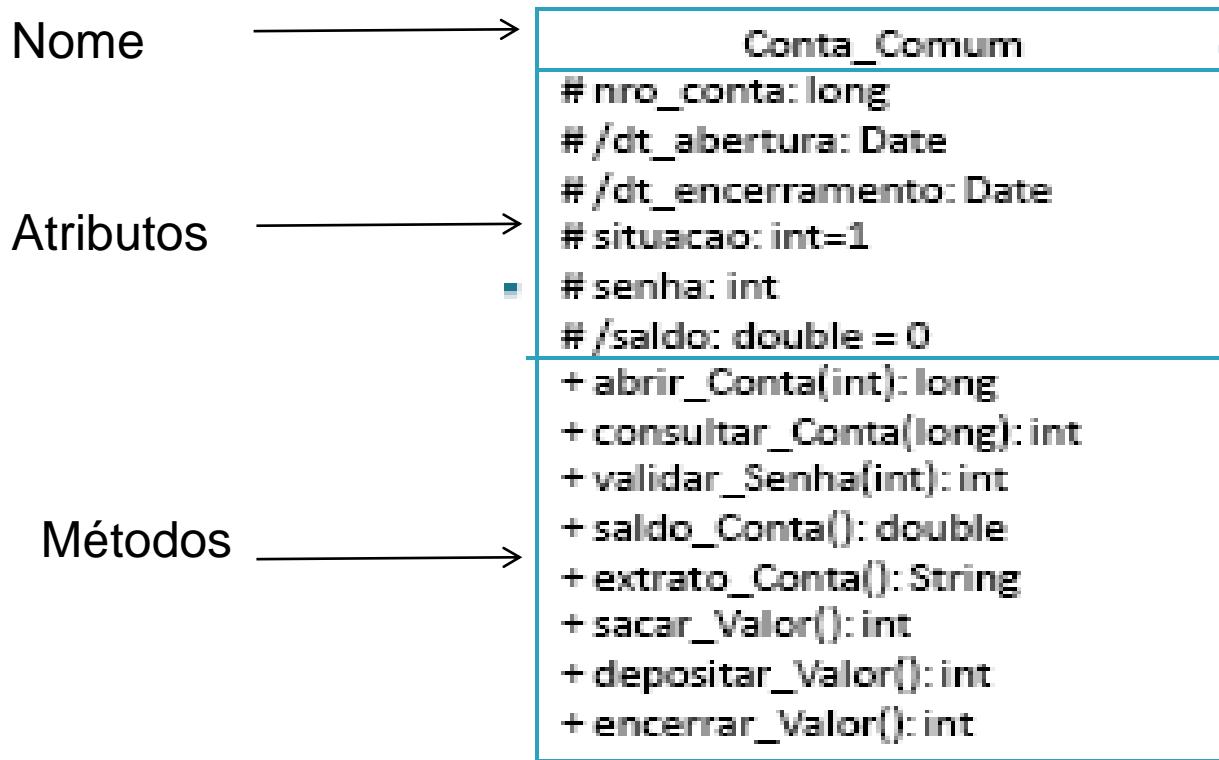
Valores  
↑↑

# Conceitos de Orientação a Objeto

- Uma classe pode ter qualquer número de atributos ou mesmo nenhum atributo.
- Da mesma forma, pode ter qualquer número de operações ou mesmo nenhuma operação.

# Conceitos de Orientação a Objeto

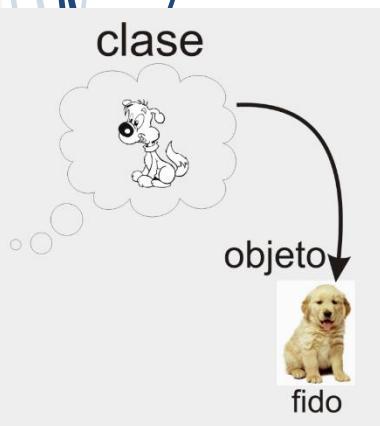
## Classe Conta em um sistema bancário



# Conceitos de Orientação a Objeto

## Resumindo Classe e Objetos

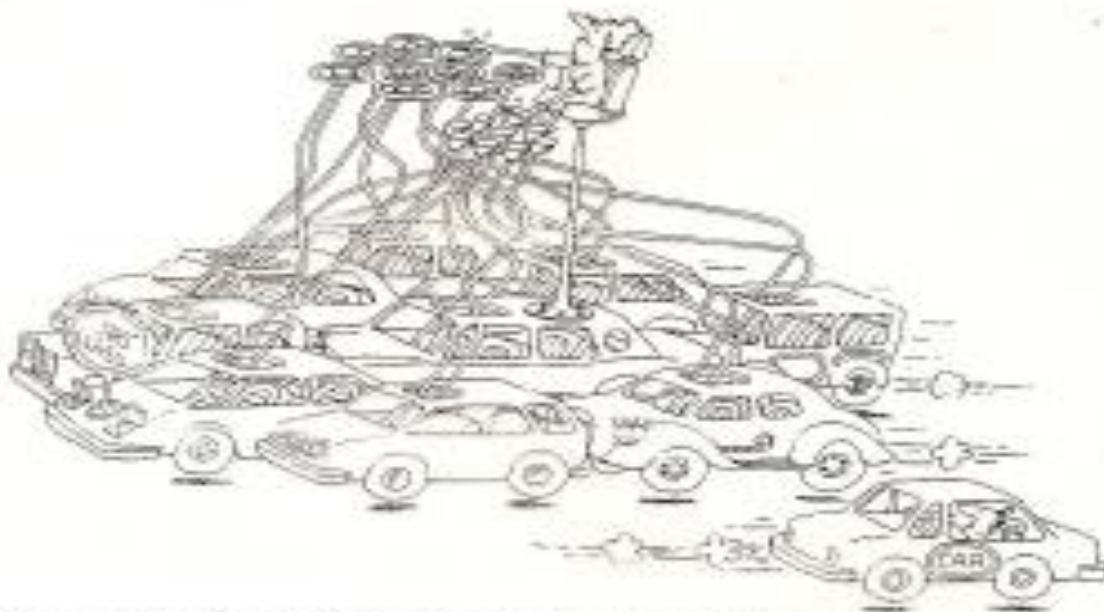
- Objetos com a mesma estrutura de dados (atributos) e o mesmo comportamento (operações) são agrupados em uma classe.
- Cada classe descreve um conjunto de objetos individuais. Cada objeto é instância de uma classe.
- Cada objeto armazena o seu próprio valor para cada atributo, mas compartilha os nomes de atributos e operações com outros objetos da mesma classe.



# Conceitos de Orientação a Objeto

## Classe

- Concepts



A class represents a set of objects that share a common structure and a common behavior.

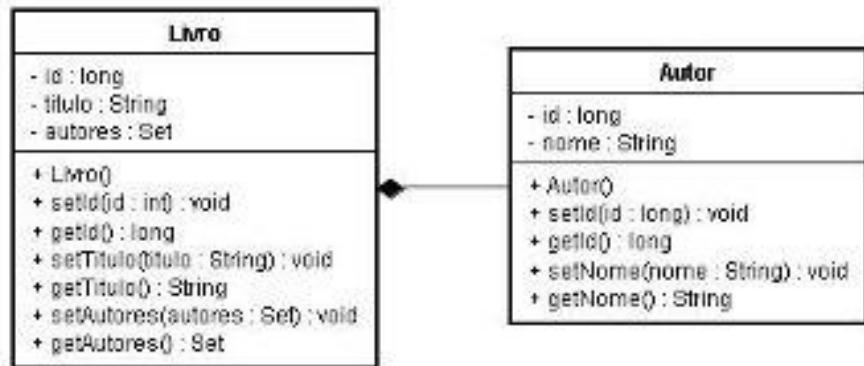


# Encapsulamento

# Conceitos de Orientação a Objeto

## Encapsulamento

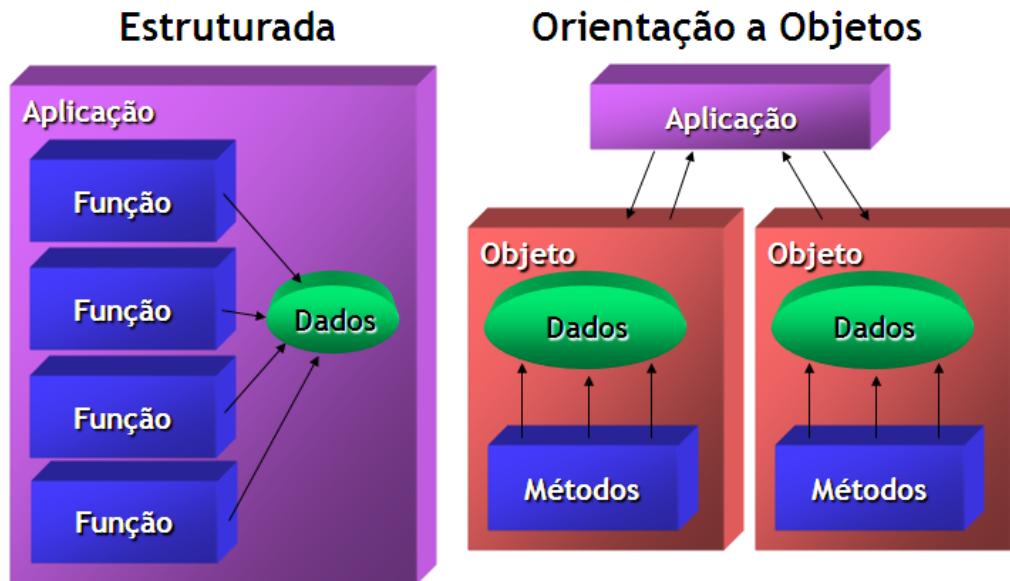
- **Encapsulamento** (ou ocultamento da informação) significa esconder os dados armazenados nos objetos, fazendo com que os mesmos se mantenham íntegros, isento de alterações não previstas ou acidentais.
- Somente as funções disponíveis na interface do objeto podem acessar tais dados. É necessário que atributos sejam privados e métodos sejam públicos



# Conceitos de Orientação a Objeto

## Encapsulamento

**Exemplo:** atributo CPF deve ter sempre 11 dígitos. Se este atributo não for privado, ele perde essa regra, alguém pode colocar mais ou menos dígitos, tornando-o inválido.



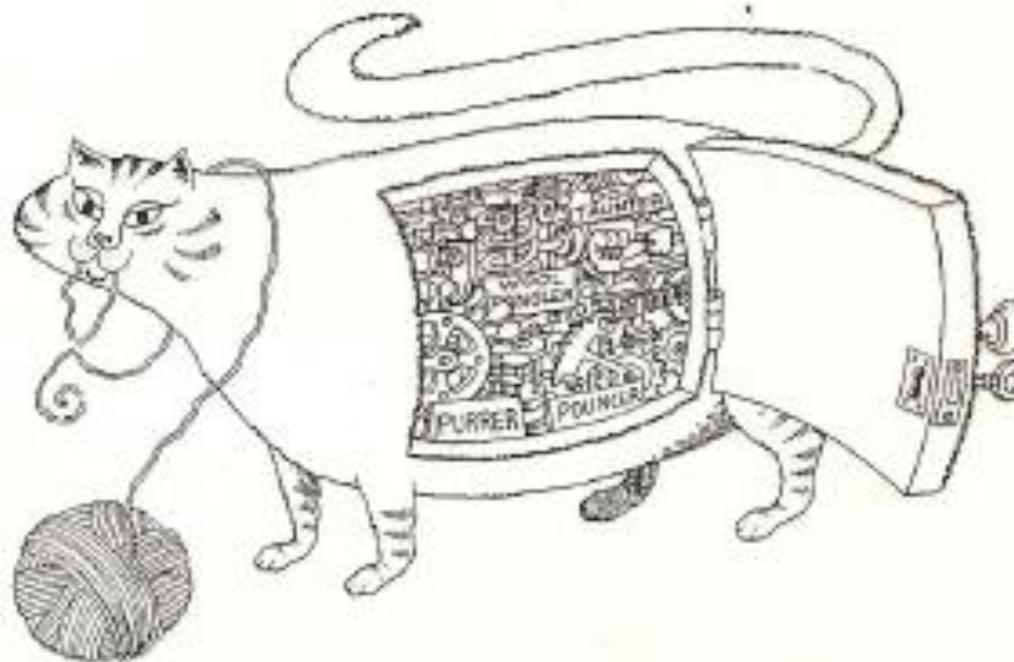
# Conceitos de Orientação a Objeto

- A interface serve como intermediária entre a classe e o mundo externo, protegendo os usuários dessa classe de quaisquer alterações futuras.
- As alterações na classe são feitas de modo transparente para o usuário.

# Conceitos de Orientação a Objeto

## Encapsulamento

Concepts



Encapsulation hides the details of the implementation of an object.

# Conceitos de Orientação a Objeto

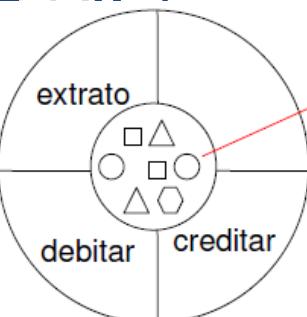
- O conceito de encapsulamento é visto também de outro ângulo.
- Determina que os atributos só podem ser acessados e atualizados pelas operações do objeto.
- Todavia, mais importante que o conceito é levar em conta que atendemos o encapsulamento quando preservamos informações e operações de cunho privado, do conhecimento externo.

# Conceitos de Orientação a Objeto

- Na OO, **encapsulamento** é o mecanismo utilizado para disponibilizar métodos que operam sobre os dados e que protegem o acesso direto indevido aos atributos de uma instância fora da classe onde estes foram declarados.
- Esta proteção consiste em se usar modificadores de acesso mais restritivos sobre os atributos definidos na classe e fornecendo métodos que alteram os valores destes atributos de alguma forma. São conhecidos com métodos “set”
- O encapsulamento ajuda a prevenir o problema de interferência externa indevida sobre os dados de um objeto, como objetos que possam alterar os dados de outros objetos indevidamente.

# Conceitos de Orientação a Objeto

- Um exemplo deste problema pode ser o saldo da conta bancária.
- O saldo certamente não pode ser alterado ou manipulado diretamente, mas sim através de métodos adequados para isso, como métodos que fazem lançamentos de débitos e créditos.
- A alteração direta do saldo causaria um problema de cálculos e inconsistência de dados.
- Justamente por isso devemos criar classes bem encapsuladas, que fornecem métodos adequados para operar sobre os dados dos objetos daquela classe.



The diagram illustrates a class structure for a bank account. At the center is a circle containing geometric shapes: a square, a triangle, a circle, and a hexagon. Surrounding this central circle are four quadrants labeled: 'debitar' (bottom-left), 'creditar' (bottom-right), 'extrato' (top-left), and 'saldo' (top-right). A red line connects the 'saldo' label to the top-right quadrant of the class diagram.

O uso de encapsulamento também evita que um programa torne-se tão interdependente que uma pequena mudança tenha grandes efeitos colaterais.

# Construtor

# Conceitos de Orientação a Objeto

Construtor é um método chamado sempre que a classe é instanciada.

- Não tem tipo de retorno.
- É executado após a alocação de memória para o objeto.

```
Conta::Conta (unsigned int numero, string titular)
{
    this->numero = numero;
    this->saldo = 0;
    this->titular = titular;
}
```

- É sempre definido implicitamente um construtor sem parâmetros, que é usado por padrão e com alocação dinâmica, mas esse pode ser definido explicitamente. Se QUALQUER construtor for definido, esse construtor padrão é perdido, e então você precisará declará-lo mesmo vazio.
- É muito útil para inicializar certos atributos de todos as instâncias da classe ( saldo de uma conta inicia com 0, pilha sempre começa vazia, etc. )

# Conceitos de Orientação a Objeto

- Analogamente ao Construtor, Um Destruitor é um método chamado quando a instância sai de escopo ou é desalocada.
- Um destrutor não pode aceitar parâmetros.

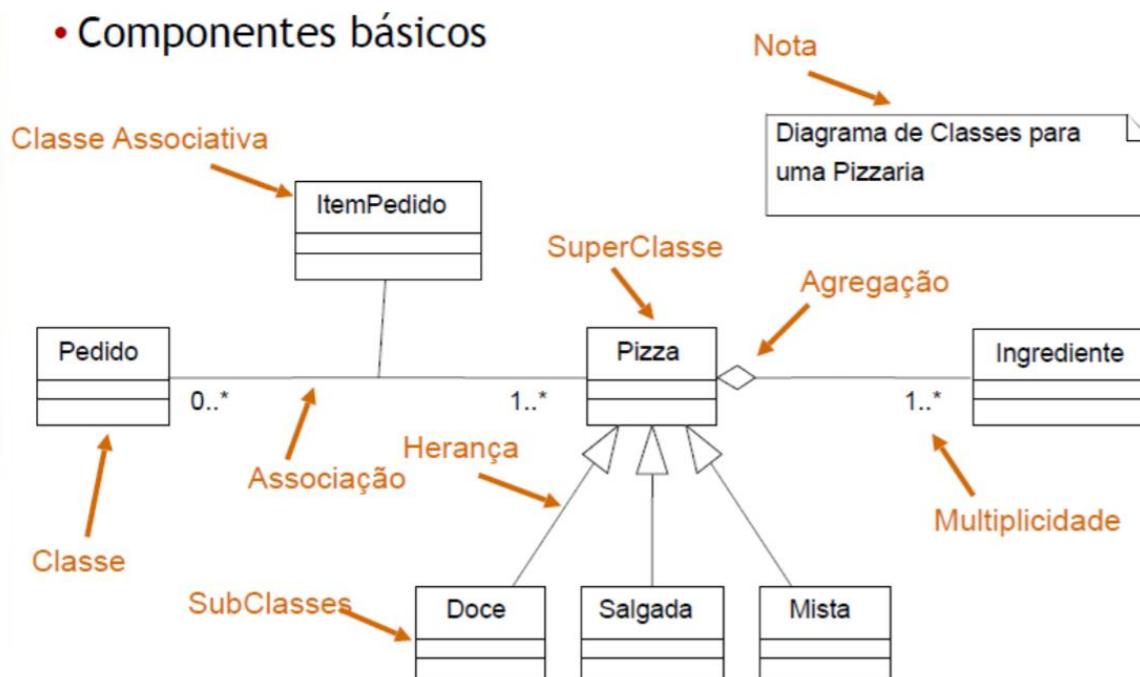
```
Conta :: ~Conta ()  
{  
}
```

- Um destrutor padrão é implementado, que não faz nada.
- É útil implementar um destrutor quando há objetos alocados dinamicamente dentro da classe, já que é preciso desalocá-los antes de destruir o objeto.
- Não é usual ( nem recomendado ) chamar destrutores de objetos alocados estaticamente, pois esses são chamados automaticamente. Chamar um destrutor duas vezes causa erro de execução.

# HERANÇA OU GENERALIZAÇÃO

X

## ASSOCIAÇÃO: AGREGAÇÃO e COMPOSIÇÃO



# Conceitos de Orientação a Objeto

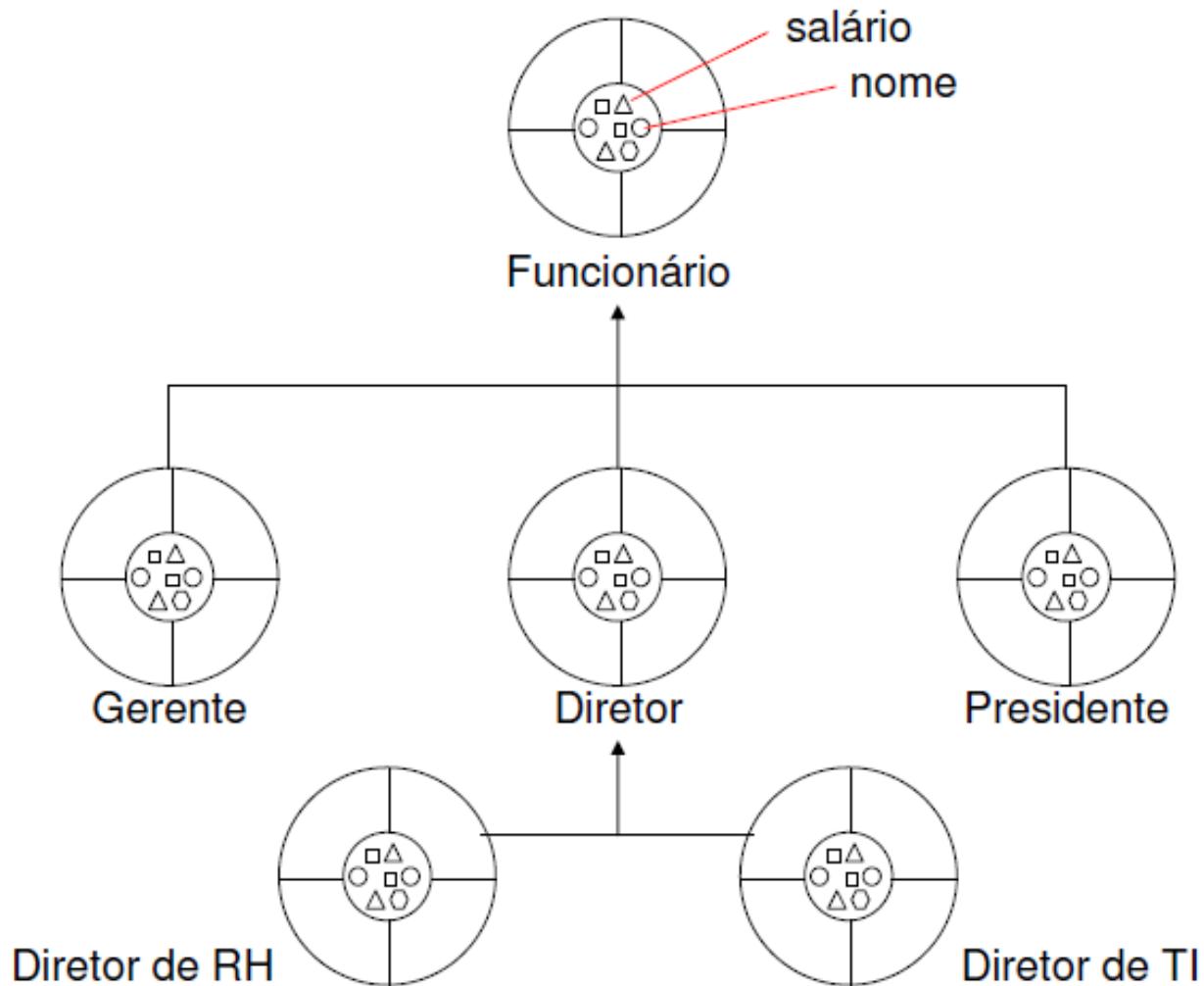
- **Herança** é um mecanismo da OO que permite criar novas classes a partir de classes já existentes, aproveitando-se das características existentes na classe a ser estendida.
- Este mecanismo é muito interessante pois promove um grande reuso e reaproveitamento de código existente.
- Com a herança é possível criar classes derivadas (subclasses) a partir de classes bases (superclasses). As subclasses são mais especializadas do que as suas superclasses, mais genéricas.
- As subclasses herdam todas as características de suas superclasses, como suas variáveis e métodos.

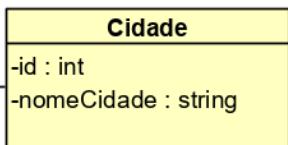
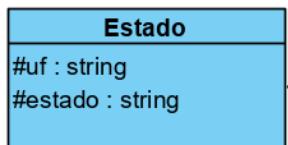
# Conceitos de Orientação a Objeto

- Imagine que dentro de uma organização empresarial, o sistema de RH tenha que trabalhar com os diferentes níveis hierárquicos da empresa, desde o funcionário de baixo escalão até o seu presidente.
- Todos são funcionários da empresa, porém cada um com um cargo diferente. Mesmo a secretária, o pessoal da limpeza, o diretor e o presidente possuem um número de identificação, além de salário e outras características em comum.
- Essas características em comum podem ser reunidas em um tipo de classe em comum, e cada nível da hierarquia ser tratado como um novo tipo, mas aproveitando-se dos tipos já criados, através da herança.

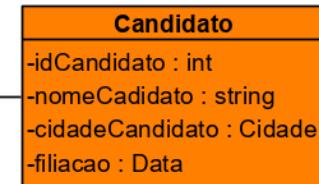
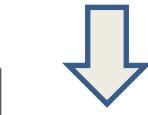
# Conceitos de Orientação a Objeto

## Herança

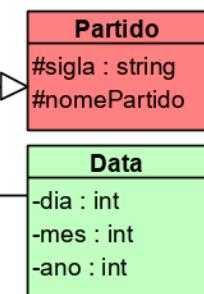




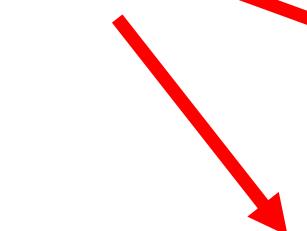
Associação



Herança  
Ou  
Generalização



Herança  
Ou  
Generalização



```
class Estado{
protected:
    string uf;
    string estado;
```

```
class Cidade: public Estado{
private:
    int id;
    string cidade;
```

```
class Candidato: public Partido{
private:
    int id;
    string nome;
    Cidade cidCand;
    Data filiacao;
```

```
class Partido {
protected:
    string sigla;
    string partido;
```

```
class Data {
private:
    int dia;
    int mes;
    int ano;
```

Associação



# Conceitos de Orientação a Objeto

- Os subtipos, além de herdarem todas as características de seus supertipos, também podem adicionar mais características, seja na forma de variáveis e/ou métodos adicionais, bem como reescrever métodos já existentes na superclasse (polimorfismo).
- A herança permite vários níveis na hierarquia de classes, podendo criar tantos subtipos quanto necessário, até se chegar no nível de especialização desejado.
- Podemos tratar subtipos como se fossem seus supertipos, por exemplo o sistema de RH pode tratar uma instância de Presidente como se fosse um objeto do tipo Funcionário, em determinada funcionalidade.
- Porém não é possível tratar um supertipo como se fosse um subtipo, a não ser que o objeto em questão seja realmente do subtipo desejado e a linguagem suporte este tipo de tratamento, seja por meio de conversão de tipos ou outro mecanismo.

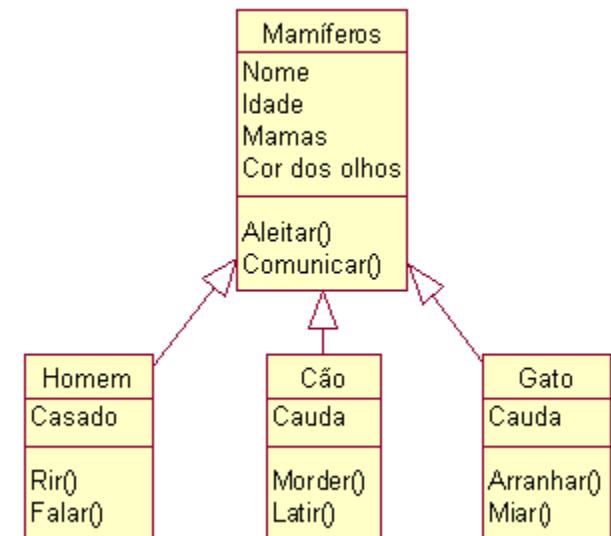
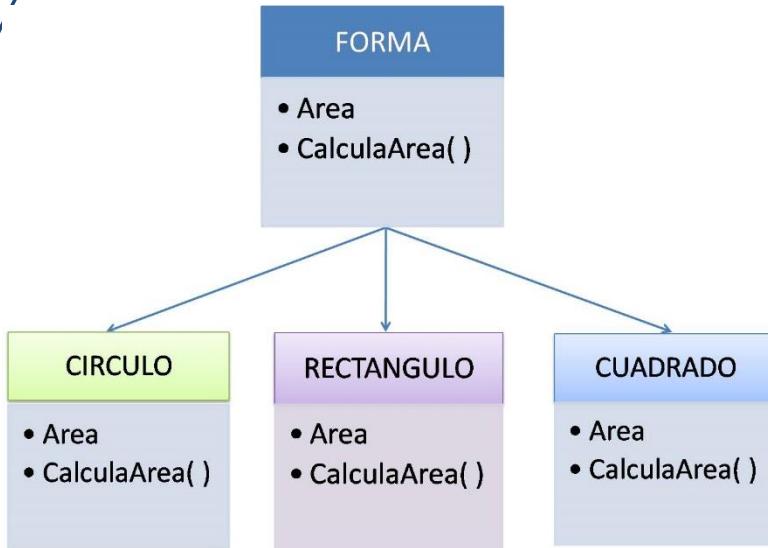
# Conceitos de Orientação a Objeto

- Algumas linguagens de programação permitem herança múltipla, ou seja, uma classe pode estender características de várias classes ao mesmo tempo. É o caso do C++.
- Outras linguagens não permitem herança múltipla, por se tratar de algo perigo se não usada corretamente. É o caso do Java.
- Na Orientação a Objetos as palavras classe base, supertipo, superclasse, classe pai e classe mãe são sinônimos, bem como as palavras classe derivada, subtípo, subclasse e classe filha também são sinônimos

# Conceitos de Orientação a Objeto

**Herança:** é o compartilhamento de atributos e operações entre classes com base em um relacionamento hierárquico.

- Uma subclasse herda todas as características de sua superclasse e acrescenta suas características exclusivas.
- **Vantagem:** reduz repetições.



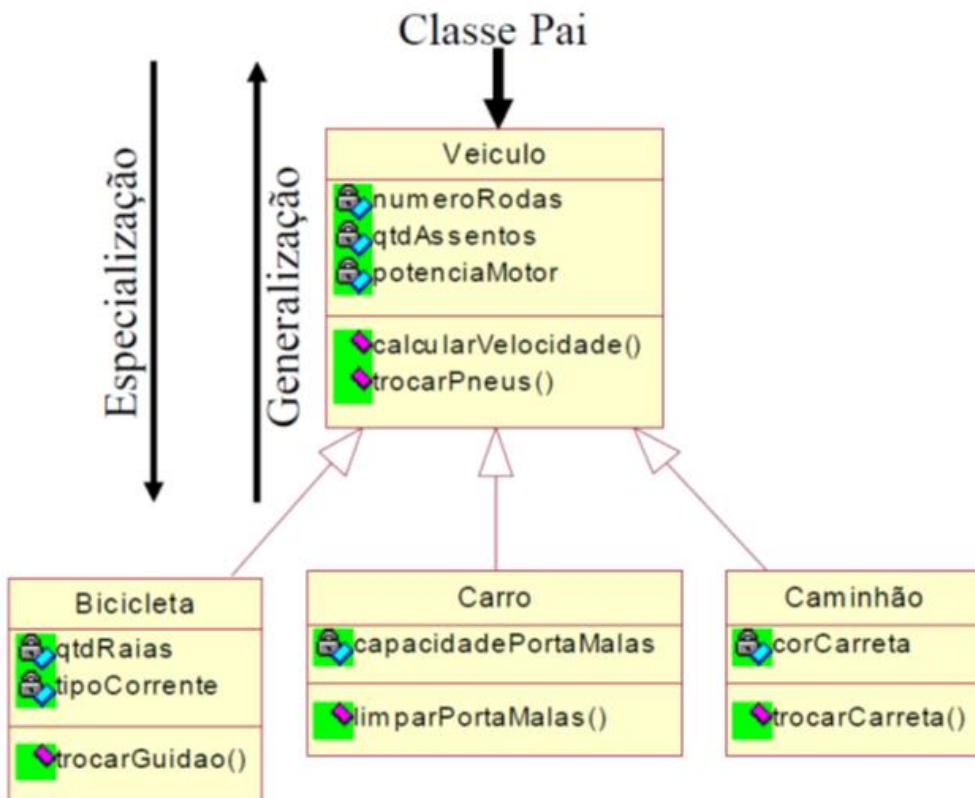
# Herança

Bicicleta, Carro e Caminhão são veículos.

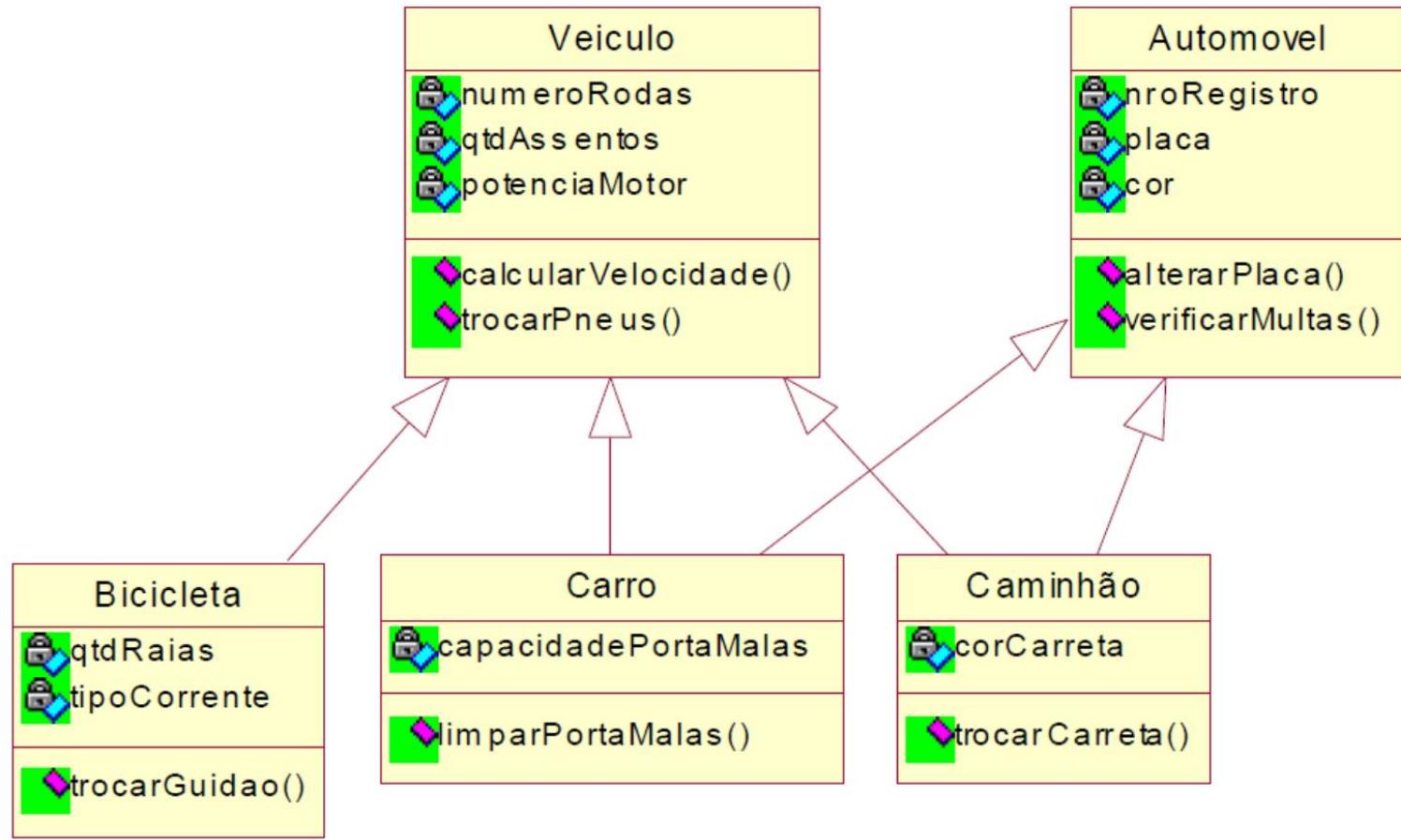
Sendo assim, eles possuem seu comportamento (atributos + métodos) específico mais o comportamento herdado.

Esse tipo de relacionamento entre classes também é conhecido como generalização/especialização.

Pois, analisando-se as classes filhas pode-se generalizar o comportamento na Pai. Do mesmo modo analisando-se a classe Pai pode-se especializar classes filhas.

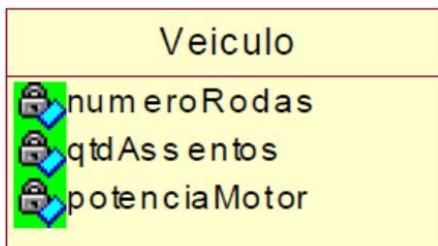


# Herança Multipla



Carro e Caminhão, além de serem Veículos também são automóveis.

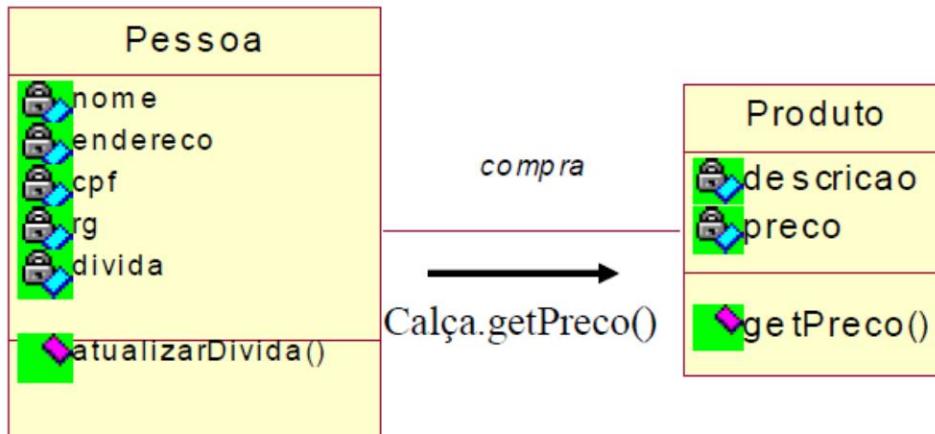
Sendo assim, herdam o comportamento de ambas as classes



# Mensagem

- Objetos se comunicam por meio de mensagens
- Um mensagem é um sinal enviado à um objeto requisitando a execução de um serviço através da execução de uma operação
- Essa operação é executada dentro do objeto que recebe a mensagem com base nos dados de seus alcance na hierarquia de classes
- Sender e receiver
- As mais conhecidas são: Agregação e Associação

# Exemplo Mensagem



É enviada uma Mensagem ao Objeto “calça” Pedindo o seu preço.

Ex. Um objeto “João” necessita atualizar seu atributo “divida”.

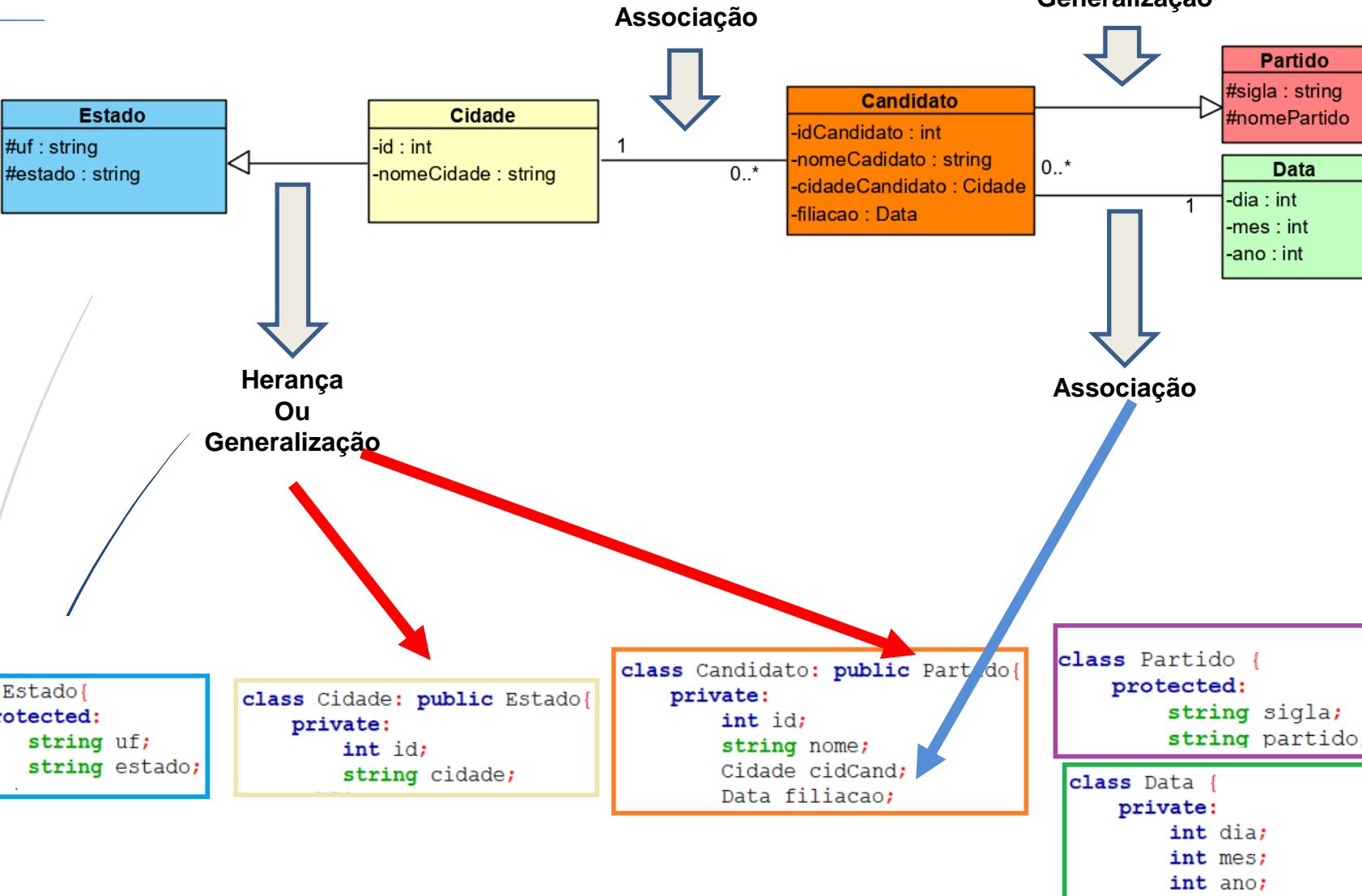
Para isso há necessidade de saber o preço do produto que o “João” comprou. Sendo assim, o método **getPreco()** da classe **Produto** deve ser invocado (mensagem) para obter o preço do produto.

# Herança e Tipos de Associações

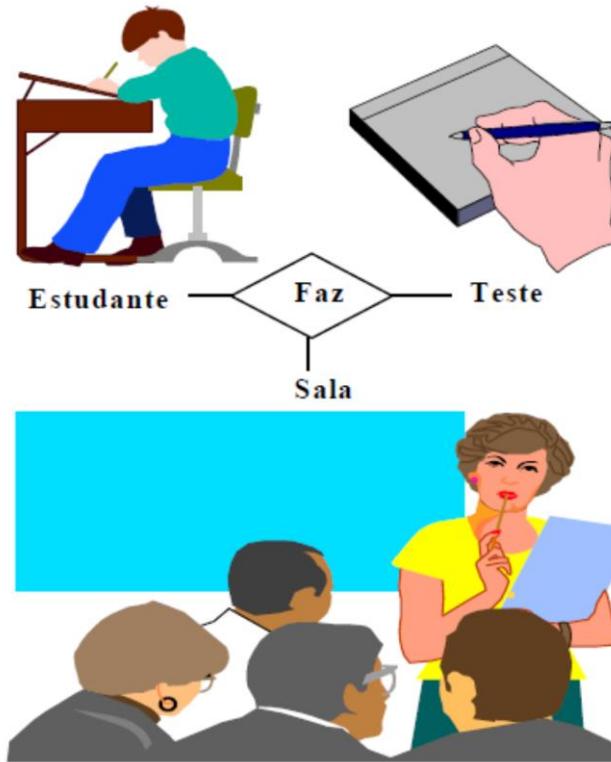
Você deve estar se perguntando: e os atributos da superclasse, como serão associados à subclasse?

- É simples: pelo mecanismo de herança, presente nas linguagens orientadas a objeto.

Dizemos que a classe-filha tem como origem outra classe.



# Associação



# Associação

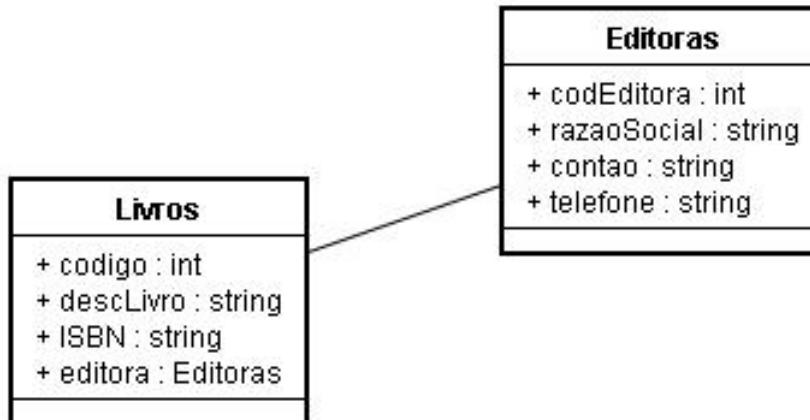
- Usada para agrupar objetos que ocorrem sob algumas circunstâncias similares ou um ponto específico no tempo
- Associação é um relacionamento estrutural que ocorre entre classes;
- Esse relacionamento existe porque um objeto necessita de outros para cumprir certas responsabilidades;

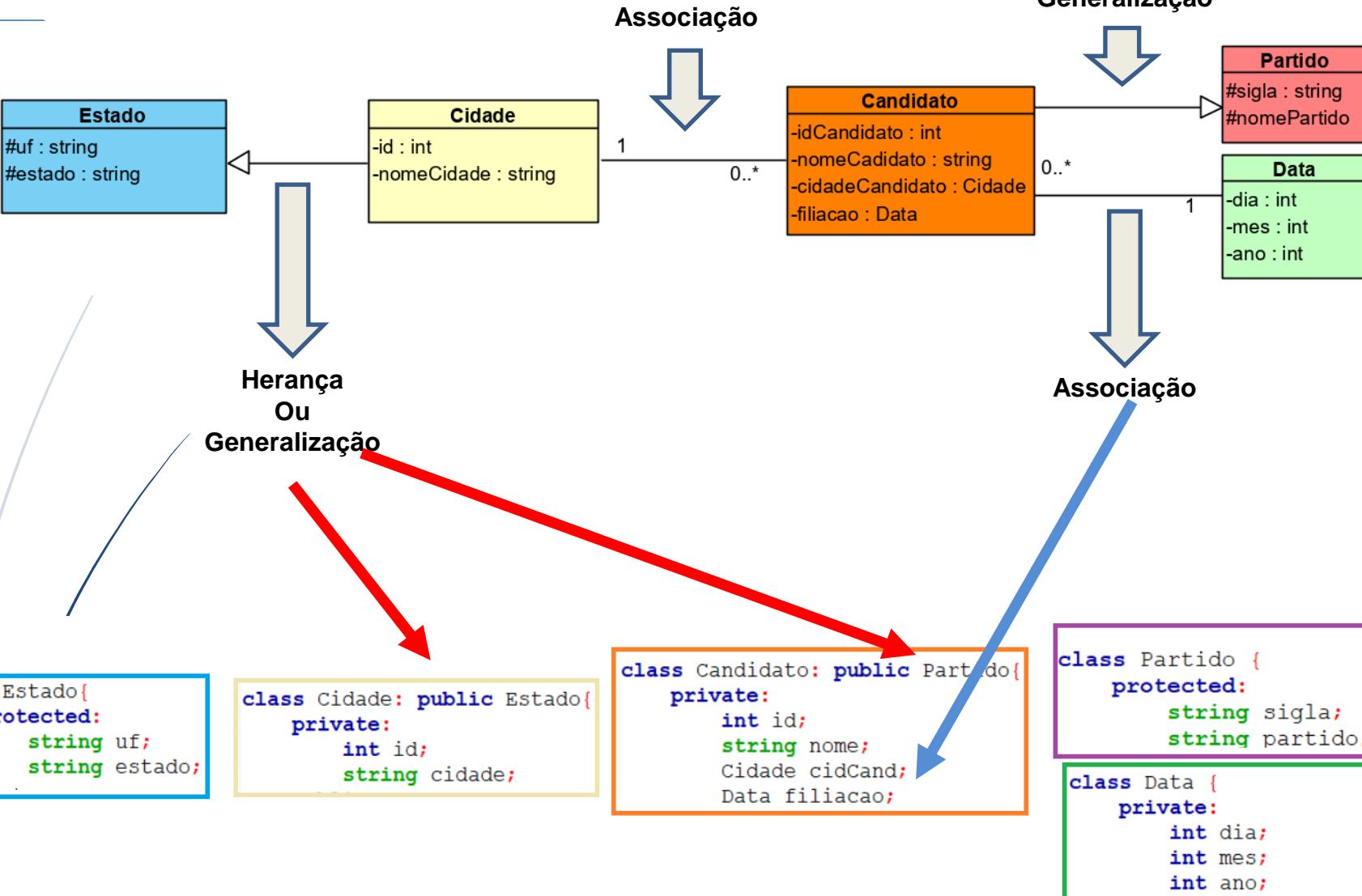
# Associação

- descreve um vínculo que ocorre entre classes - associação binária -
- possível até mesmo que uma classe esteja vinculada a si própria, - associação unária
- ou que uma associação seja compartilhada por mais de uma classe - associação ternária ou N-ária

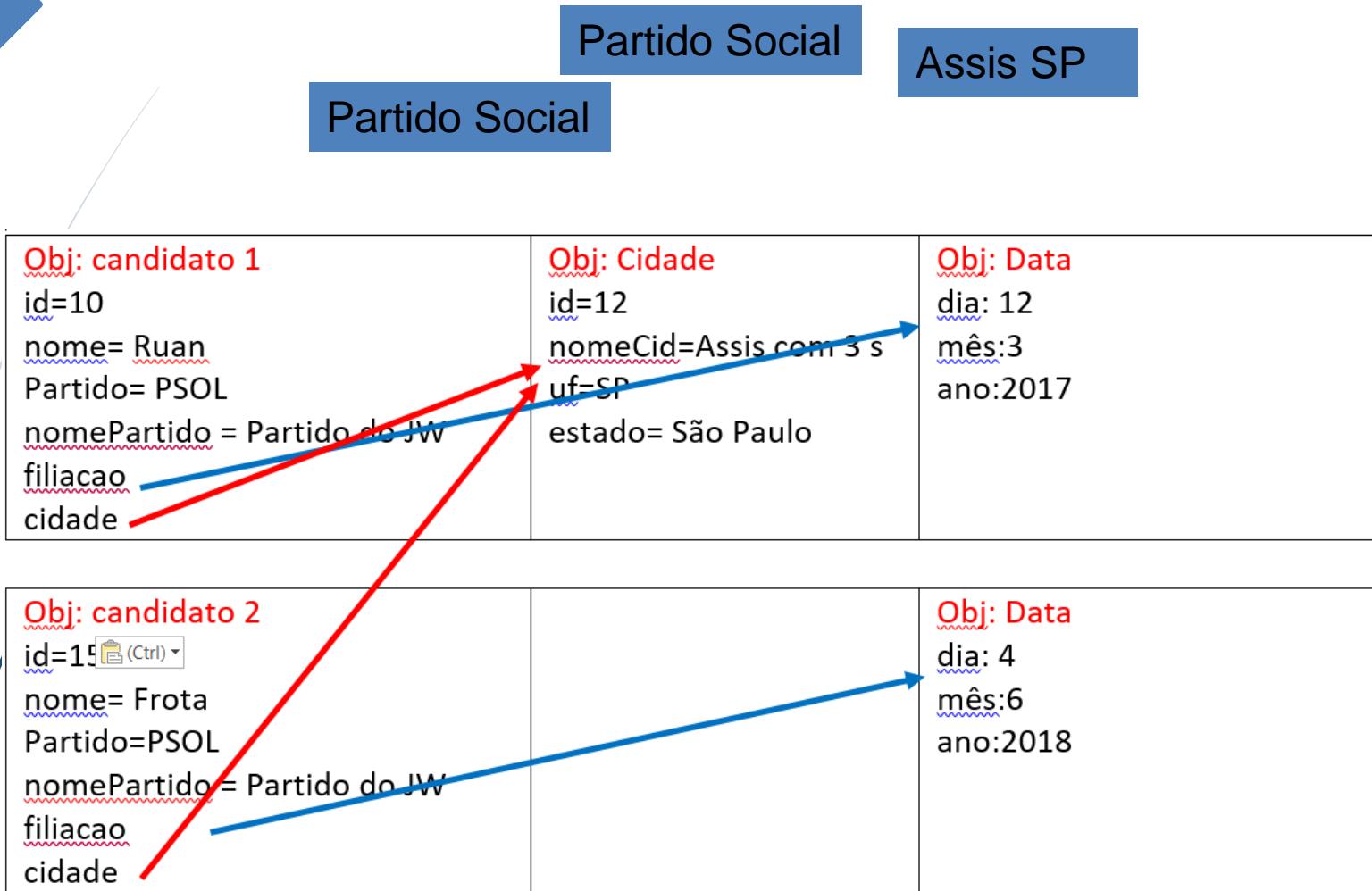
# Exemplo Associação

- A forma mais comum de implementar associação é ter um objeto como atributo de outro.
- No exemplo, abaixo temos uma associação entre a Classe Livros e a classe Editoras.
- No código cria-se um objeto do tipo Livro e outro do tipo Editora. Um dos atributos do Livro é a Editora.





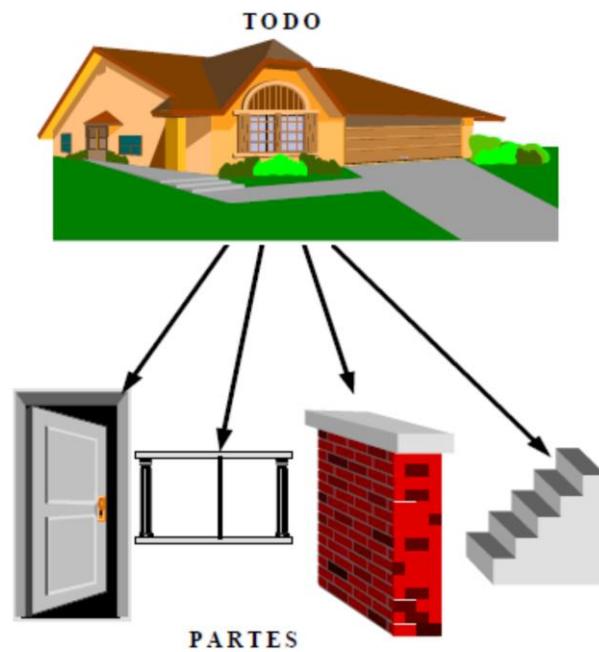
# Comparação Herança x Associação (Memória)



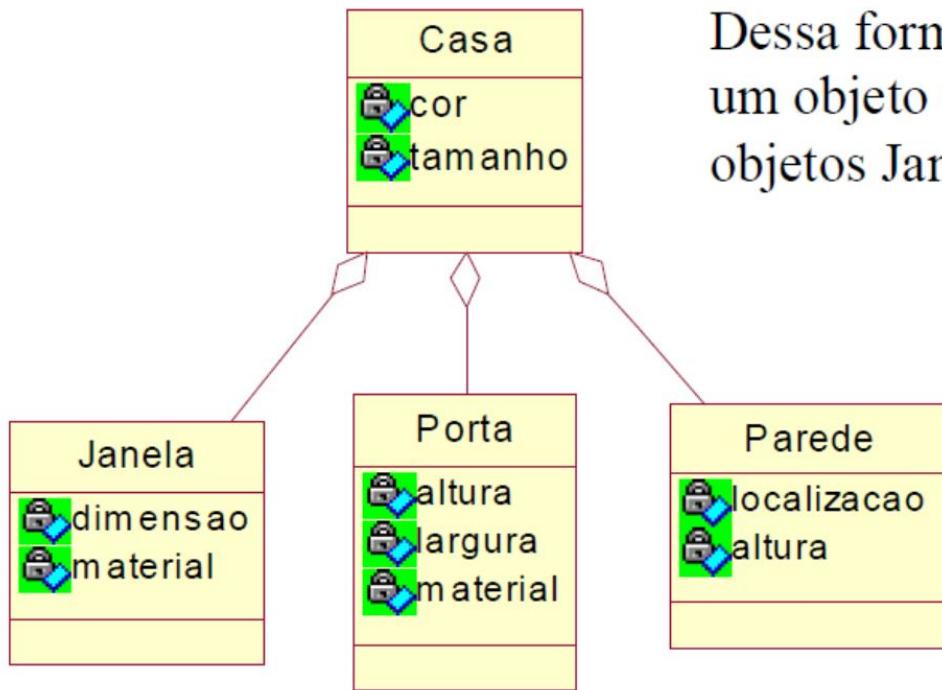
# Todo Parte (Agregação)

- Esse conceito permite a construção de uma classe agregada a partir de outras classes componentes.
- Usa-se dizer que um objeto da classe Agregada (Todo) tem objetos da classe componente (Parte)
- Por exemplo: Pode-se imaginar esse tipo de relacionamento como uma casa, que é composta por portas, janelas, paredes, etc.
- A pergunta a ser feita para identificar um relacionamento de agregação é: “é parte de ?”

# Todo Parte (Agregação)



# Todo Parte (Agregação)



Dessa forma representa-se que um objeto Casa é composto pelos objetos Janela, Porta e Parede

# Agregação

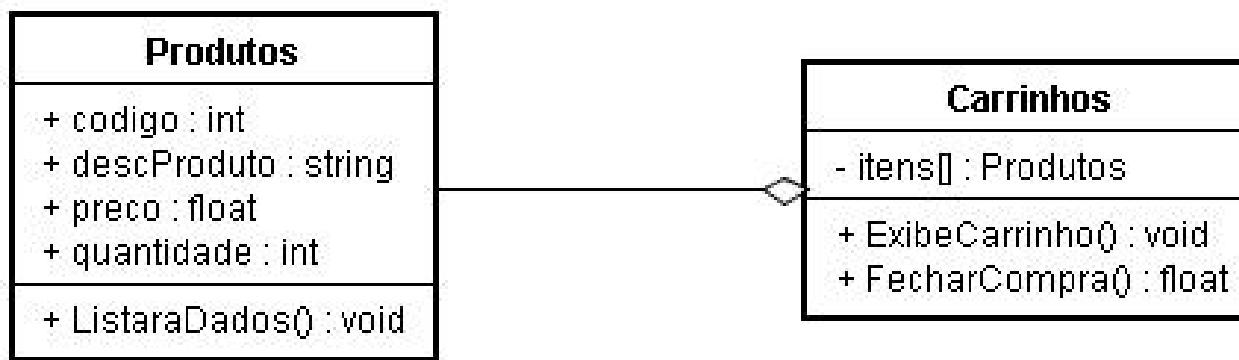
É um tipo especial de associação onde tenta-se demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementados pelas informações contidas em um ou mais objetos de outra classe (chamados objetos-parte); conhecemos como todo/parte.

O objeto-pai poderá usar as informações do objeto agregado. Segundo Dall'Oglio (2007) "Nesta relação, um objeto poderá agregar uma ou mais instâncias de um outro objeto. Para agregar muitas instâncias, a forma mais simples é utilizando arrays. Criamos um array como atributo da classe, sendo que o papel deste array é armazenar inúmeras instâncias de uma outra class". (DALL'OGLIO, 2007, p. 118)

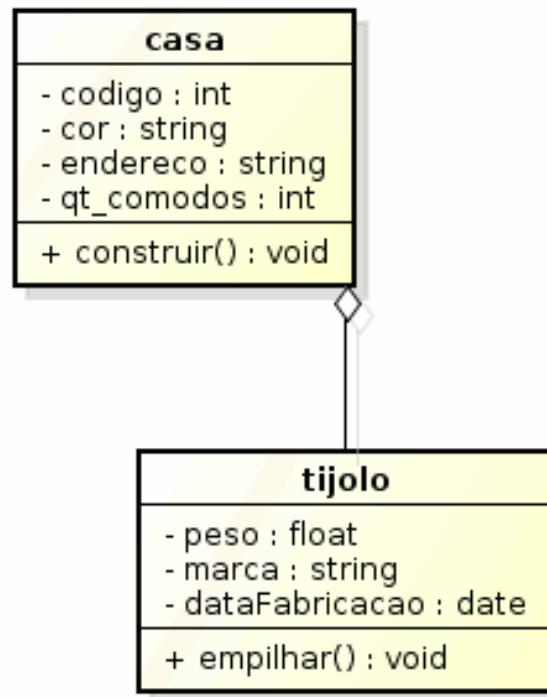
# Exemplo Agregação

- Pense em um ambiente Web, onde teríamos o carrinho de compras (classe Carrinhos) com vários itens do tipo produtos (classe Produtos).
- Para agregar os produtos ao carrinho, usa-se o método IncluirItem( )
- na classe Carrinhos, que contém outro método chama ExibeCarrinho() responsável por listar todos os itens pedidos, por meio da listagem dos dados do produto - método ListarDados() da classe Produtos-,
- e um método FechaCompra( ) responsável por efetuar a soma dos itens adicionados no carrinho apresentando ao final o preço a ser pago pelo cliente.

# Exemplo Agregação

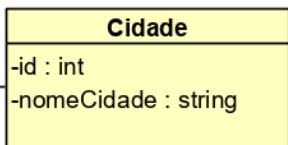
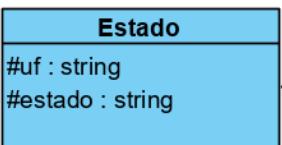


# Exemplo Agregação

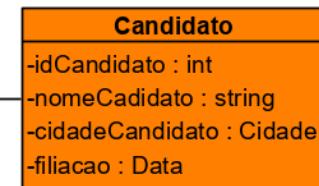
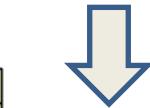


# Composição

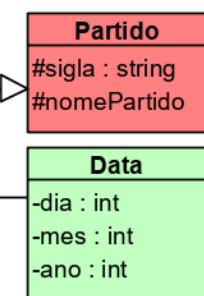
- Pode-se dizer que composição é uma variação da agregação.
- Uma composição tenta representar também uma relação todo - parte.
- No entanto, na composição o objeto-pai (todo) é responsável por criar e destruir suas partes.
- Em uma composição um mesmo objeto-parte não pode se associar a mais de um objeto-pai.



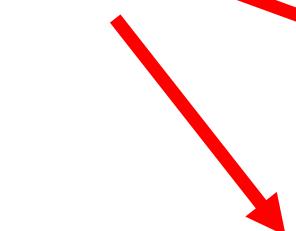
Associação



Herança  
Ou  
Generalização



Herança  
Ou  
Generalização



```
class Estado{
protected:
    string uf;
    string estado;
```

```
class Cidade: public Estado{
private:
    int id;
    string cidade;
```

```
class Candidato: public Partido{
private:
    int id;
    string nome;
    Cidade cidCand;
    Data filiacao;
```

```
class Partido {
protected:
    string sigla;
    string partido;
```

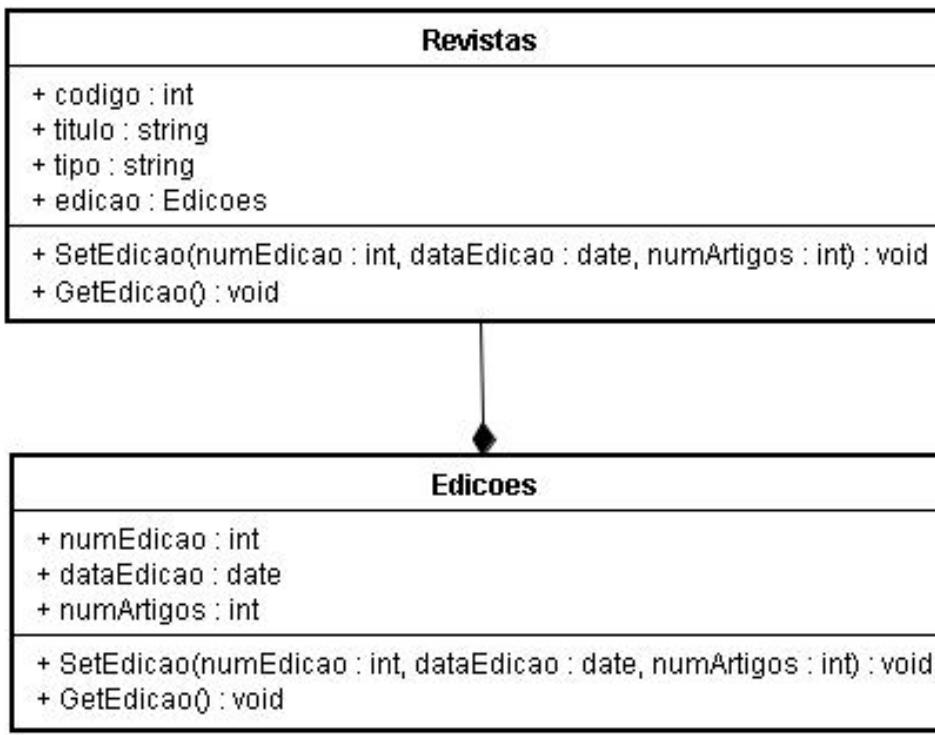
```
class Data {
private:
    int dia;
    int mes;
    int ano;
```

Associação



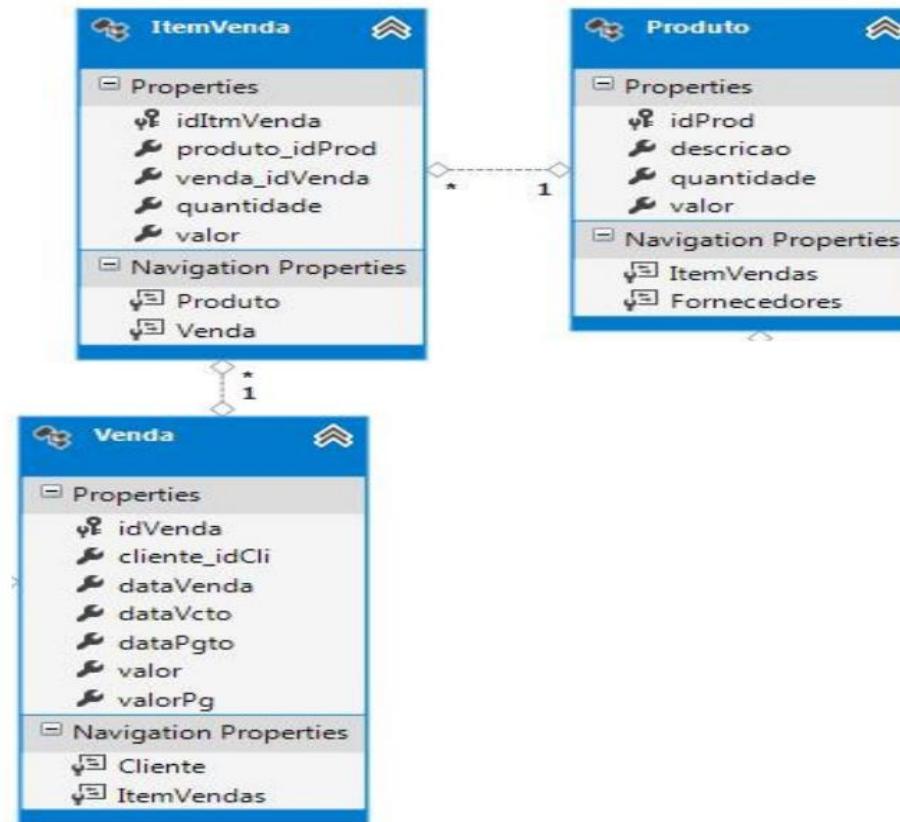
# Exemplo Composição

- Nela temos o objeto-todo Revistas e objeto-parte Edicoes.



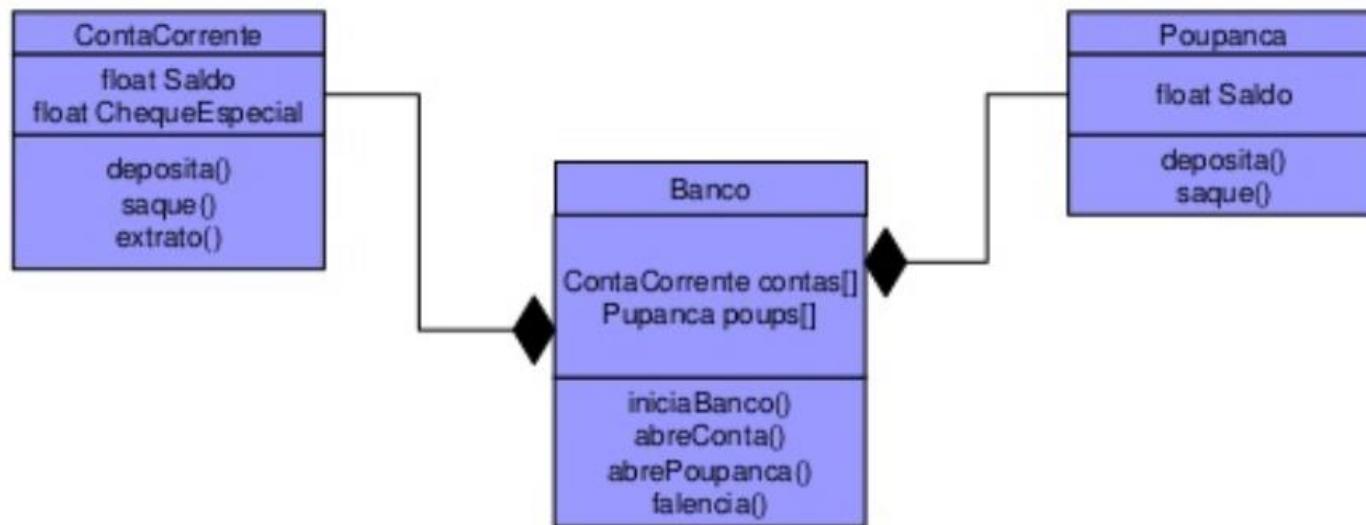
# Exemplo Agregação

- Pedido e item pedido



# Exemplo Composição

- Ex: Banco e contas bancárias

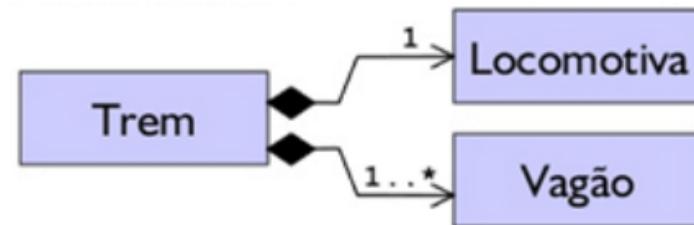


# Resumindo

## Composição, Agregação e Associação

- **Composição**

- Um trem **é formado por 1** locomotiva e n vagões;
- Relacionamento **tem-um**;



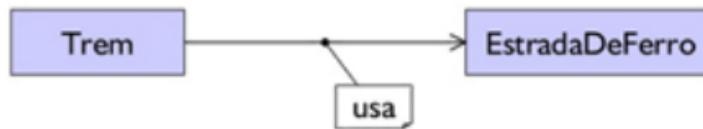
- **Agregação**

- Uma locomotiva (todo) **tem** um farol (parte), mas não deixa de existir se não o tiver;



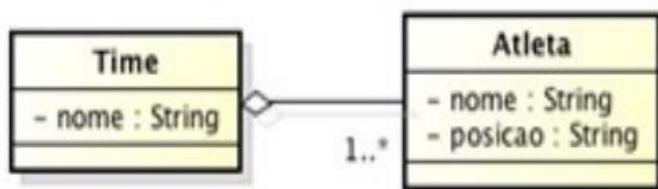
- **Associação**

- Um trem **usa** uma estrada de ferro (a estrada não faz parte do trem, mas ele depende dela).

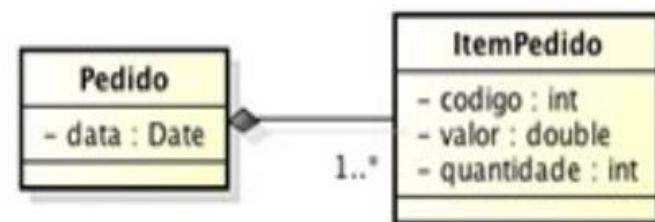


# Resumindo Agregação/Composição

- A **Agregação** e a **Composição** são tipos específicos de **associação** entre classes

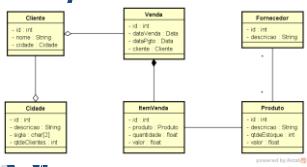


Time é uma agregação de (mínimo 1, máximo N) Atletas. Objetos Atleta existem independente do Time.

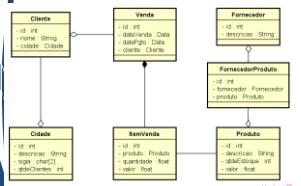


Pedido é uma composição de (mínimo 1, máximo N) ItensPedido. Objetos ItemPedido não existem se o Pedido deixar de existir.

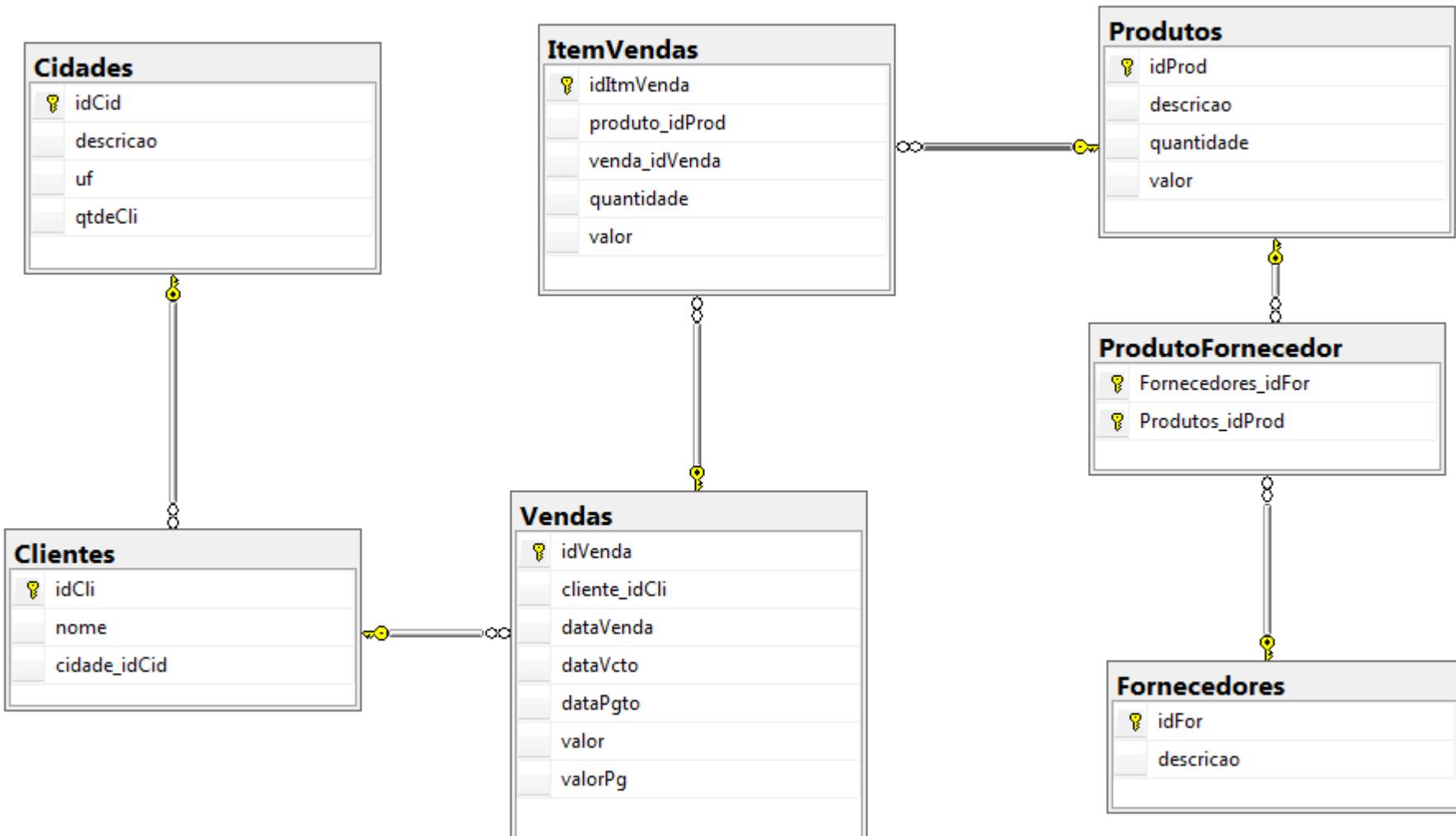
# Exemplo Associação, Agregação e Composição



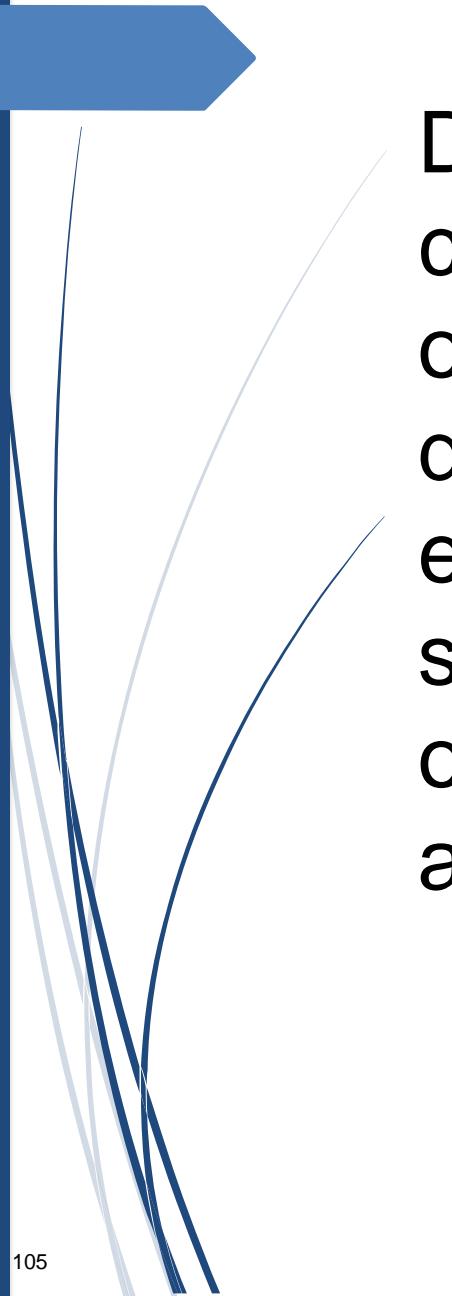
# Exemplo Associação, Agregação e Composição



# Exemplo Associação, Agregação e Composição



# Conceitos de Orientação a Objeto



Desta forma, pela herança, codificamos apenas os atributos e operações específicos da classe descendente, e fazemos com que esta receba, automaticamente, e sem qualquer esforço, os atributos e operações de todas as classes ancestrais.

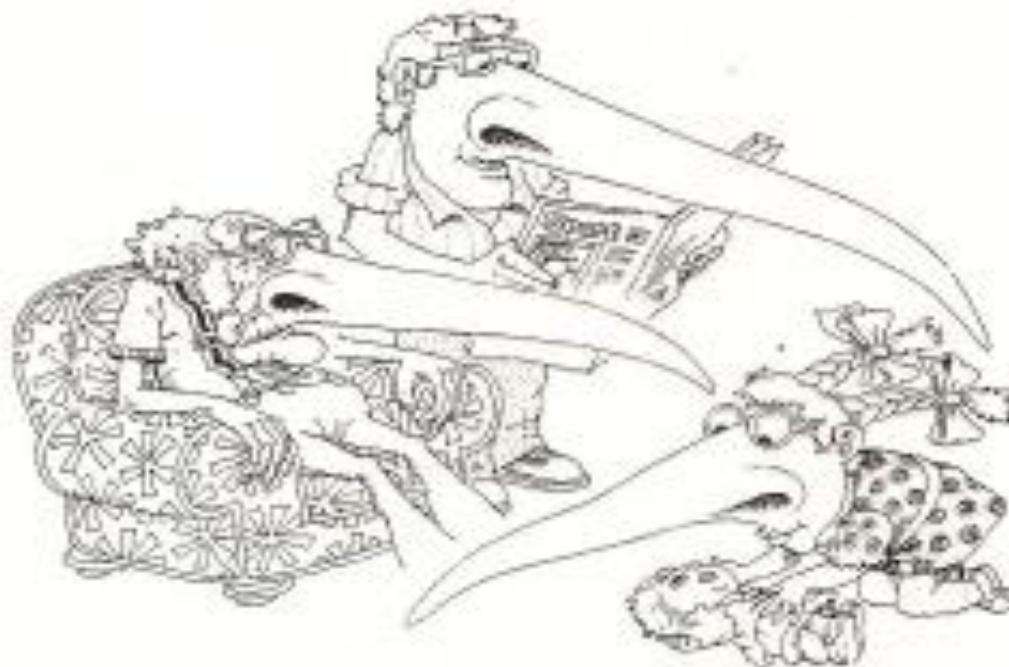
# Conceitos de Orientação a Objeto

- Pela herança podemos estabelecer relações entre classes, permitindo o compartilhamento de atributos e operações semelhantes.
- Assim, temos a flexibilidade de criar uma nova classe, incluindo somente as diferenças com relação à classe mais genérica.

# Conceitos de Orientação a Objeto

## Herança

Concepts



A subclass may inherit the structure and behavior of its superclass.

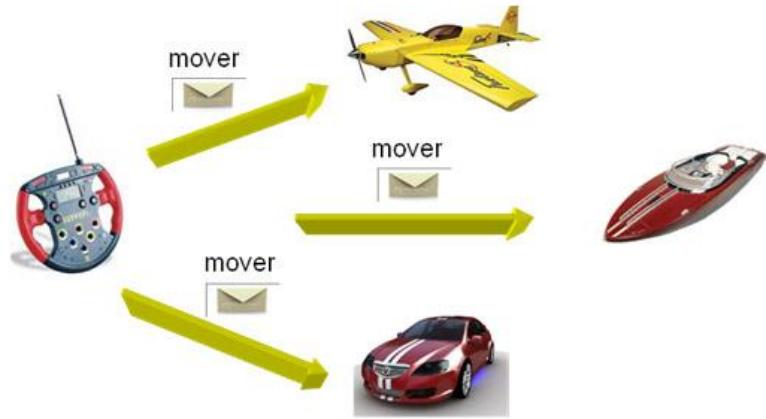


# Polimorfismo

# Conceitos de Orientação a Objeto

## Polimorfismo

- **Definição:** a mesma operação pode atuar de modos diversos em classes diferentes. Diferentes objetos respondem de maneira diferente à mesma mensagem.
- **Método:** uma implementação específica de uma operação por uma determinada classe.



# Conceitos de Orientação a Objeto

## Polimorfismo

### **Exemplo 1:** Operações “Mover” e “Desenhar”

- I “Mover” pode ter um código diferente para cada classe “Janela” e “PeçaXadrez”.
- “Desenhar” pode ter um código diferente para cada classe “Quadrado” e “Elipse”.

### **Exemplo 2:** Calcular aumento

- Aumentar(): método não recebe parâmetros, aumenta por padrão 20%
- Aumentar(int perc): recebe um parâmetro contendo o valor do percentual de aumento. Aumenta o valor conforme o percentual recebido.

Obs: a implementação do método é diferente em cada caso

# Sobrecarga (Overload)

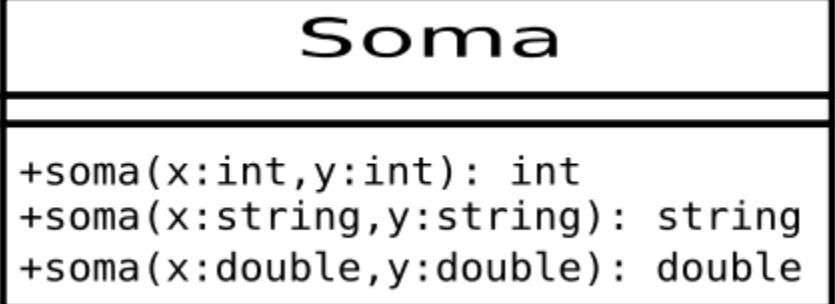
A sobrecarga de métodos (overload) é um conceito do polimorfismo que consiste basicamente em criar variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes totalmente iguais em uma classe. A Sobrecarga permite que utilizemos o mesmo nome em mais de um método contanto que suas listas de argumentos sejam diferentes para que seja feita a separação dos mesmos.

# Sobrecarga (Overload)

```
class Cat{  
    public void Sound(){  
        System.out.println("meow");  
    }  
  
    //overloading method  
    public void Sound(int num){  
        for(int i=0; i<2;i++){  
            System.out.println("meow");  
        }  
    }  
}
```

## OVERLOADING

Same method  
name but  
different  
parameters



# Sobrecarga (Overload)

```
10 public class Classe {           Sobrecarga (Overload) de Construtores - O primeiro  
11  
12     private int numa;  
13     private int numb;  
14  
15     public Classe() {           não possui parâmetro, já o segundo possui.  
16         this.numa = 2;  
17         this.numb = 2;  
18     }  
19  
20     public Classe(int numa, int numb) {  
21         this.numa = numa;  
22         this.numb = numb;  
23     }  
24  
25     public int somaValores() {  
26         return numa+numb;  
27     }  
28  
29     public void somaValores(int a, int b) {  
30         this.numa = a;  
31         this.numb = b;  
32         int total = a+b;  
33     }  
34  
35     public int somaValores(double b, double a){  
36         double total = a+b;  
37         int contotal = (int)total;  
38         return contotal;  
39     }
```

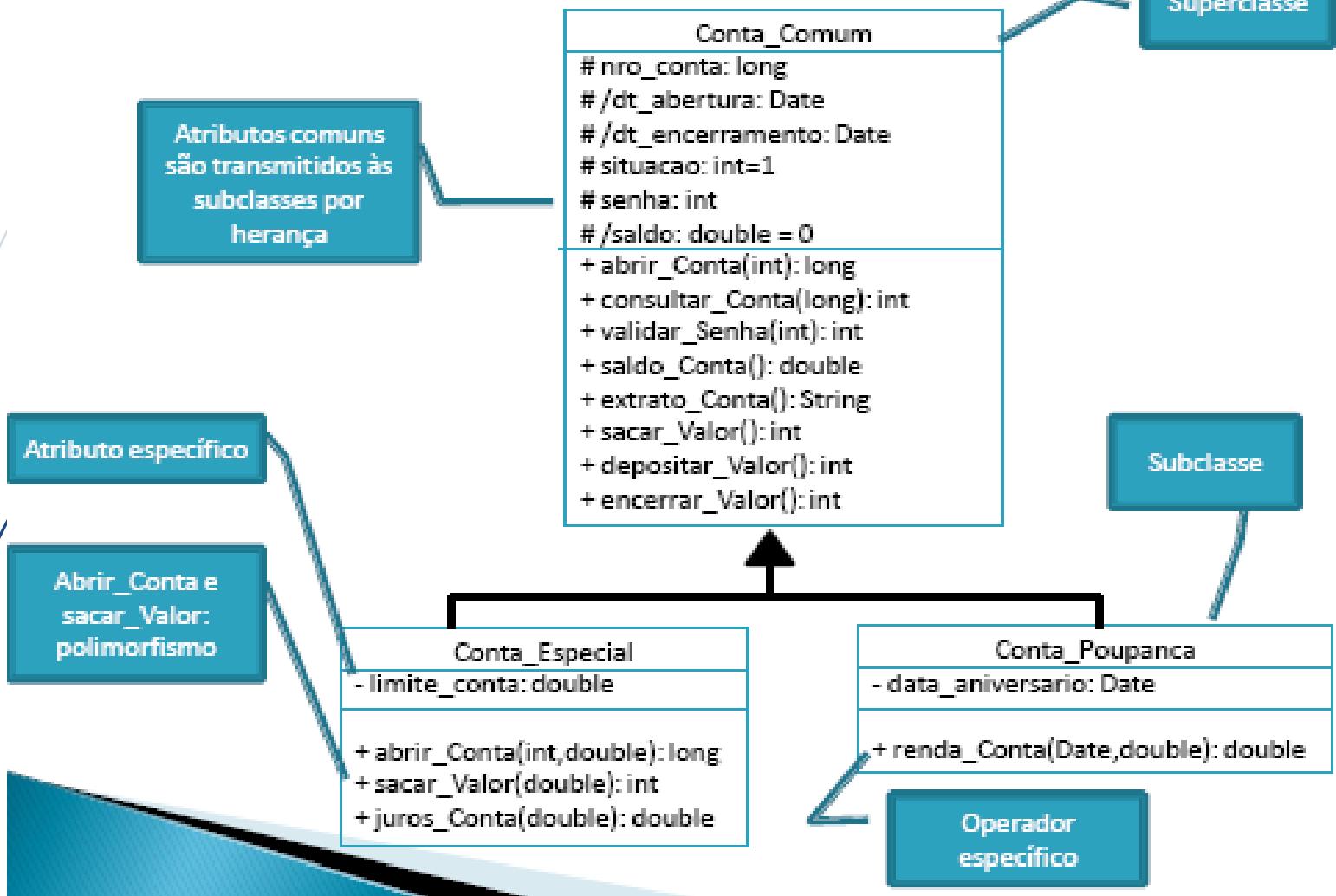
Sobrecarga de Métodos, todos eles tem a mesma função, mas recebem parâmetros diferentes, e tem retornos diferentes, ou nenhum retorno.

# Herança e Polimorfismo

- Refere-se à diferentes formas de um objeto
- Polimorfismo refere-se a capacidade de uma mesma operação realizar funções diferentes dependendo do objeto que a recebe e dos parâmetros que lhes são passados.
- Por exemplo, pode-se ter em uma classe uma operação denominada “calcularDivida()”. Caso essa operação seja invocada sem parâmetros ela realizará algo, caso seja invocada passando um determinado parâmetro realizará algo diferente.

# Conceitos de Orientação a Objeto

## Exemplo: herança e polimorfismo



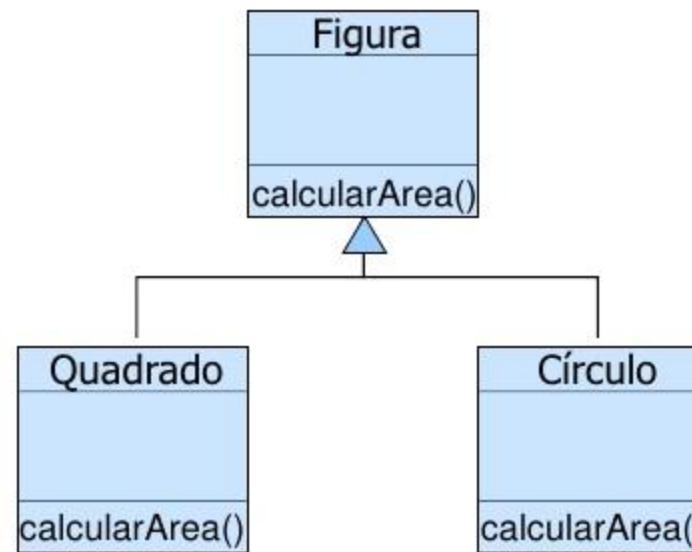
# Conceitos de Orientação a Objeto

- Na Figura anterior temos uma classe mais genérica (**Conta\_Comum**), chamada de classe-mãe ou superclasse, contendo atributos que servem a qualquer situação.
- A partir dessa classe criamos outras mais específicas (**Conta\_Especial** e **Conta\_Poupança**), chamadas classes-filhas ou subclasses.
- Estamos diante dos conceitos de generalização e especialização.

# Sobreposição (Override)

A Sobreposição de métodos (override) é um conceito do polimorfismo que nos permite reescrever um método, ou seja, podemos reescrever nas classes filhas métodos criados inicialmente na classe pai, os métodos que serão sobrepostos, diferentemente dos sobrecarregados, devem possuir o mesmo nome, tipo de retorno e quantidade de parâmetros do método inicial, porém o mesmo será implementado com especificações da classe atual, podendo adicionar um algo a mais ou não.

# Sobreposição (Override)



# Programação Estruturada X POO

## Programação Estruturada

- Processo
- Revela Dados
- Projeto Monolítico
- Uso Único
- Algoritmo Ordenado

## Orientação a Objeto

- Objeto
- Oculta Dados
- Projeto Modular
- Reutilização
- Algoritmo Desordenado

# Conceitos de Orientação a Objeto

- O paradigma da Orientação a Objetos traz um ganho significativo na qualidade da produção de software, porém grandes benefícios são alcançados quando as técnicas de programação OO são colocadas em prática com o uso de uma tecnologia que nos permita usar todas as características da OO; além de agregar à programação o uso de boas práticas de programação e padrões de projeto (*design patterns*).
- Esse é um dos motivos do sucesso da tecnologia Java, que suporta a OO completamente e também fornece mecanismos para se usar os *design patterns* conhecidos.
- Além do conhecimento da Orientação a Objetos, o conhecimento da UML (*Unified Modelling Language*) ajuda muito no desenho e planejamento de sistemas na sua concepção.



**Área:** Informática

**Curso:** ADS/BCC

**Disciplina:** Linguagem de Programação

**Avaliação:** POO - II - 1º Semestre

**Data:** ADS: 16/04/2021 e BCC: 20/04/2021

## Conteúdo:

- Classe
- Objeto
- Interface
- Atributos e Métodos
- Construtor Padrão e Parametrizado
- Métodos de acesso (gets)
- Métodos modificadores (sets)
- Métodos constantes
- Métodos inline
- Polimorfismo (Overload e Override)
- Herança (Generalização, Associação: (Composição e Agregação))



Material inspirado no material dos professores

***Professor Msc Rafael Soares  
Professor Msc Sérgio Sigrist  
Msc Wladimir da Costa***