



*Técnico em Desenvolvimento de Sistemas Online*

# ANÁLISE DE SISTEMAS E PROJETOS

# GEEaD - Grupo de Estudo de Educação a Distância

## Centro de Educação Tecnológica Paula Souza

### Expediente

GEEaD – CETEC  
GOVERNO DO ESTADO DE SÃO PAULO  
EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO  
CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS  
FUNDAMENTOS DE INFORMÁTICA

*Autores:*  
*Eliana Cristina Nogueira Barion*  
*Marcelo Fernando Iguchi*

*Revisão Técnica:*  
*Lilian Aparecida Bertini*

*Revisão Gramatical:*  
*Juçara Maria Montenegro Simonsen Santos*

*Editoração e Diagramação:*  
*Flávio Biazim*

**São Paulo – SP, 2019**

# APRESENTAÇÃO

Este material didático do Curso Técnico em Desenvolvimento de Sistemas modalidade EaD foi elaborado especialmente por professores do Centro Paula Souza para as Escolas Técnicas Estaduais – ETECs.

O material foi elaborado para servir de apoio aos estudos dos discentes para que estes atinjam as competências e as habilidades profissionais necessárias para a sua plena formação como Técnicos em Desenvolvimento de Sistemas.

Esperamos que este livro possa contribuir para uma melhor formação e aperfeiçoamento dos futuros Técnicos.

---

# AGENDA 16

---

## MODELAGEM DE DADOS ORIENTADA A OBJETOS





## MERGULHANDO NO TEMA...

Como podemos identificar os objetos de dados no mundo real?

Devemos observar as situações onde se consiga extrair informações. Por exemplo, se pensarmos no objeto aluno, podemos extrair dele informações importantes que o identifique dentro de um sistema, como por exemplo, o nome do aluno, o curso que ele frequenta, as notas desse aluno etc.

Pense também nas características importantes para o professor dentro de um sistema acadêmico... Por exemplo, o nome do professor, a quantidade de aulas atribuídas a ele, as disciplinas que ministra etc. Então, a partir do momento em que conseguimos pensar num conjunto de dados e definir quais são pertinentes ao sistema, estamos definindo um modelo de dados e consequentemente, estamos definindo um objeto de dados dentro do sistema.

Portanto, a ideia de trabalhar com objetos é extrair do cenário real as informações que formarão um conjunto de dados para construir um espaço de armazenamento de informações dentro de um Banco de Dados e a partir disso, construir um sistema que utilizará essa base de dados.

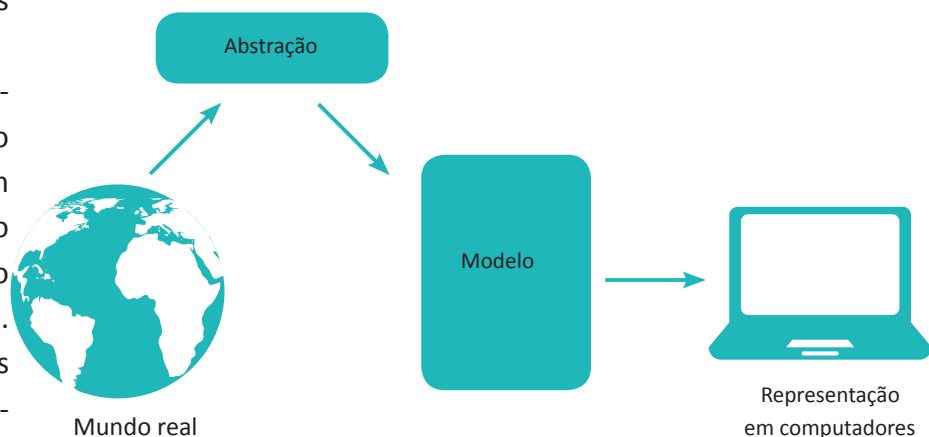
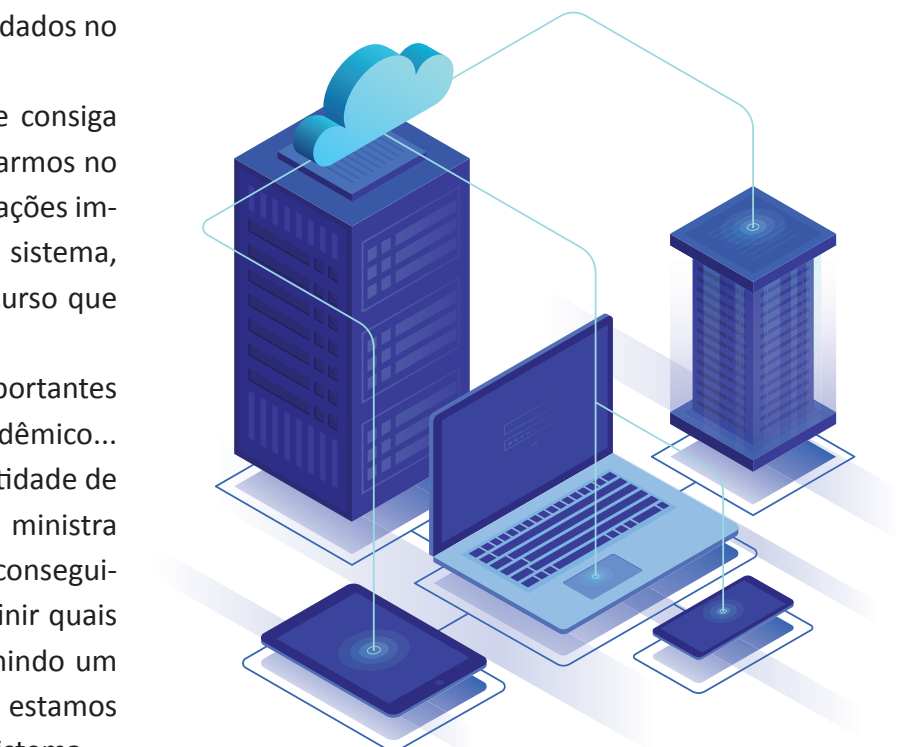
Além das informações sobre as características dos objetos de dados, é preciso coletar dados sobre as ações que esse objeto executará. Por exemplo, no caso do professor, é importante armazenar ações como lecionar aula, corrigir atividades etc. Essas ações são chamadas de métodos.

A modelagem Orientada a Objeto tem como propósito representar os objetos do mundo real dentro do sistema.

Para isso, vamos estudar os principais conceitos da modelagem orientada a objeto: **Abstração, Classe, Objeto, Métodos, Herança e Polimorfismo.**

A **abstração** consiste conseguir **identificar um item do mundo real e transformá-lo em uma classe**, preocupando-se apenas com suas principais propriedades.

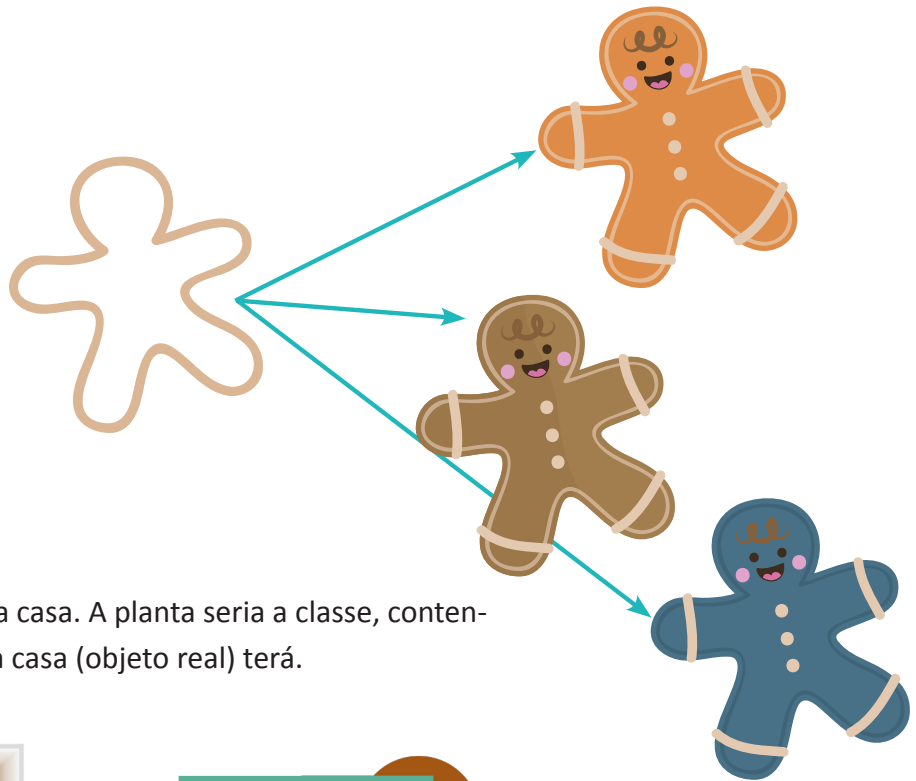
No desenvolvimento de um sistema acadêmico teremos os alunos que precisarão ser cadastrados. Esses alunos possuem características comuns, independente do seu tipo. Essas características comuns irão compor as propriedades da classe aluno. Pensando assim conseguimos abstrair os dados e transformá-lo em uma classe dentro do projeto.



Resumidamente, podemos dizer que **classe** é um conjunto de objetos distintos, porém com as mesmas características e comportamentos. A classe é uma abstração de entidades existentes no mundo real.

Algumas analogias ajudam a entender o conceito de Classe nos modelos Orientados a Objetos, como por exemplo:

Imagine um molde para criação de bonecos. O molde seria a classe, contendo as características básicas dos objetos bonecos.



Agora imagine uma planta de uma casa. A planta seria a classe, contendo todas as especificações que uma casa (objeto real) terá.



Planta casa



Casa objeto

Uma classe é composta por atributos (características) e por métodos (ações)

Pessoa	Nome da classe
<ul style="list-style-type: none"> <li>- nome: String</li> <li>- idade: int</li> <li>- peso: int</li> <li>- sexo: char</li> </ul>	Atributos
<ul style="list-style-type: none"> <li>+ andar() : void</li> <li>+ dormir() : void</li> <li>+ comer() : void</li> <li>+ andar() : void</li> <li>+ respirar() : void</li> </ul>	métodos



O objeto é uma instância ou um elemento derivado de uma classe. Assim, podemos dizer que o objeto é a representação do mundo real que irá ser manipulado ou armazenado pelo sistema.

Os **atributos** definem as propriedades de um objeto, conhecidos também como variáveis ou campos.

Os **métodos** são ações ou procedimentos dos objetos para interagirem e se comunicarem com os demais objetos afim de executarem uma tarefa qualquer, como por exemplo, a inclusão, exclusão ou alteração de dados. A execução dessas ações se dá por meio de mensagens, tendo como função o envio de uma solicitação ao objeto para que seja efetuada a rotina desejada.

Como boas práticas, é indicado sempre usar o nome dos métodos declarados como verbos, para que quando for efetuada alguma manutenção seja de fácil entendimento. Por exemplo: acaoVoltar, voltar, avançar, correr, resgatarValor, pesquisarNomes, cadastrarSalario, calcularJuros.

Adaptado de <https://www.devmedia.com.br/introducao-a-programacao-orientada-a-objetos-em-java/26452>

Observe os exemplos a seguir:

### Classe Pessoa

Pessoa
<ul style="list-style-type: none"> <li>- nome: String</li> <li>- idade: int</li> <li>- peso: int</li> <li>- sexo: char</li> </ul>
<ul style="list-style-type: none"> <li>+ andar() : void</li> <li>+ dormir() : void</li> <li>+ comer() : void</li> <li>+ andar() : void</li> <li>+ respirar() : void</li> </ul>

Objetos: João, Ana, José



### Classe Carro

Carro
<ul style="list-style-type: none"> <li>- modelo: String</li> <li>- velocidadeMaxima: int</li> <li>- velocidadeAtual: int</li> </ul>
<ul style="list-style-type: none"> <li>+ ligarCarro() : String</li> <li>+ desligarCarro() : String</li> <li>+ acelerar() : String</li> <li>+ reduzir() : String</li> <li>+ paraCarro() : String</li> <li>+ setModelo() : void</li> </ul>

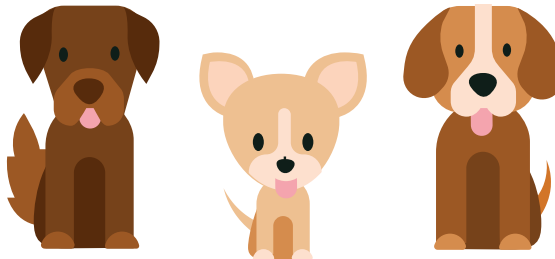
Objetos: Fusca, Gol, Jipe



## Classe Animal

Animal
<ul style="list-style-type: none"> <li>- tamanho: Float</li> <li>- cor: String</li> </ul>
<ul style="list-style-type: none"> <li>+comer() : void</li> </ul>

Objetos: Totó, Bidu, Pipoca



Assim, podemos afirmar que todo objeto é algo que existe, uma coisa concreta, já a classe é considerada como um modelo ou projeto de um objeto.

A herança é um dos principais conceitos da Orientação a Objetos por possibilitar que as classes compartilhem seus atributos, métodos e outros membros da classe entre si.

A herança é um conceito amplamente utilizado em linguagens orientadas a objetos. Além de possibilitar que as classes compartilhem seus atributos, métodos e outros membros da classe entre si, a herança é a base para outros conceitos, como a sobrescrita de métodos, classes e **polimorfismo**. Tais conceitos são fundamentais para a modelagem de sistemas mais robustos.

Durante a análise dos requisitos de um sistema (solicitações que o sistema deverá atender), podemos destacar os atributos ou os métodos comuns a um grupo de classes e concentrá-los em uma única classe (processo conhecido como generalização). Da mesma forma, é possível identificar o que é pertinente somente a determinada classe (conhecido como especificação). A primeira vantagem dessa organização é evitar a duplicidade de código (ter o mesmo trecho de código em lugares diferentes do sistema), o que traz maior agilidade e confiabilidade na manutenção e expansão do sistema.

*Adaptado de Manual de Informática Centro Paula Souza, V. 4. Fundação Padre Anchieta, 2010.*

Vamos entender os conceitos de Herança por meio de exemplos:

Imagine que você identifique a necessidade da criação das classes Professor e Aluno para armazenar e manipular os dados dessas classes em um Sistema Acadêmico:

Professor
<ul style="list-style-type: none"> <li>- nome: String</li> <li>- idade: int</li> <li>- formação: String</li> </ul>
<ul style="list-style-type: none"> <li>+ definirNome(nome: String) : void</li> <li>+ retornarNome(): String</li> <li>+ definirIdade(Idade: int) : void</li> <li>+ retornarIdade() : int</li> <li>+ definirformação(f: String): void</li> <li>+ retornarformação(): String</li> </ul>

Aluno
<ul style="list-style-type: none"> <li>- nome: String</li> <li>- idade: int</li> <li>- curso: String</li> </ul>
<ul style="list-style-type: none"> <li>+ definirNome(nome: String) : void</li> <li>+ retornarNome(): String</li> <li>+ definirIdade(Idade: int) : void</li> <li>+ retornarIdade() : int</li> <li>+ definircurso(f: String): void</li> <li>+ retornarcurso(): String</li> </ul>

Imagem adaptada - Fonte: <http://www2.ic.uff.br/~anselmo/cursos/TPA/apresentacoes/Heranca.pdf>

Perceba que as classes Professor e Aluno possuem atributos comuns (nome e idade) e vários métodos em comum também (definirNome, retornarNome, definirIdade, retornarIdade).



No entanto, alguns métodos e atributos são específicos de cada classe. A classe Professor tem o atributo formação que é específico dela e a classe Aluno tem o atributo curso que também é específico dessa classe.

Da mesma forma, os métodos definirFormação e retornarFormação pertencem apenas à classe Professor e os métodos definirCurso e retornarCurso pertencem apenas à classe Aluno.

Sendo assim, podemos destacar os atributos ou os métodos comuns a esse grupo de classes e concentrá-los em uma única classe (Generalização). Da mesma forma, é possível identificar o que é pertinente somente a determinada classe (conhecido como especificação).

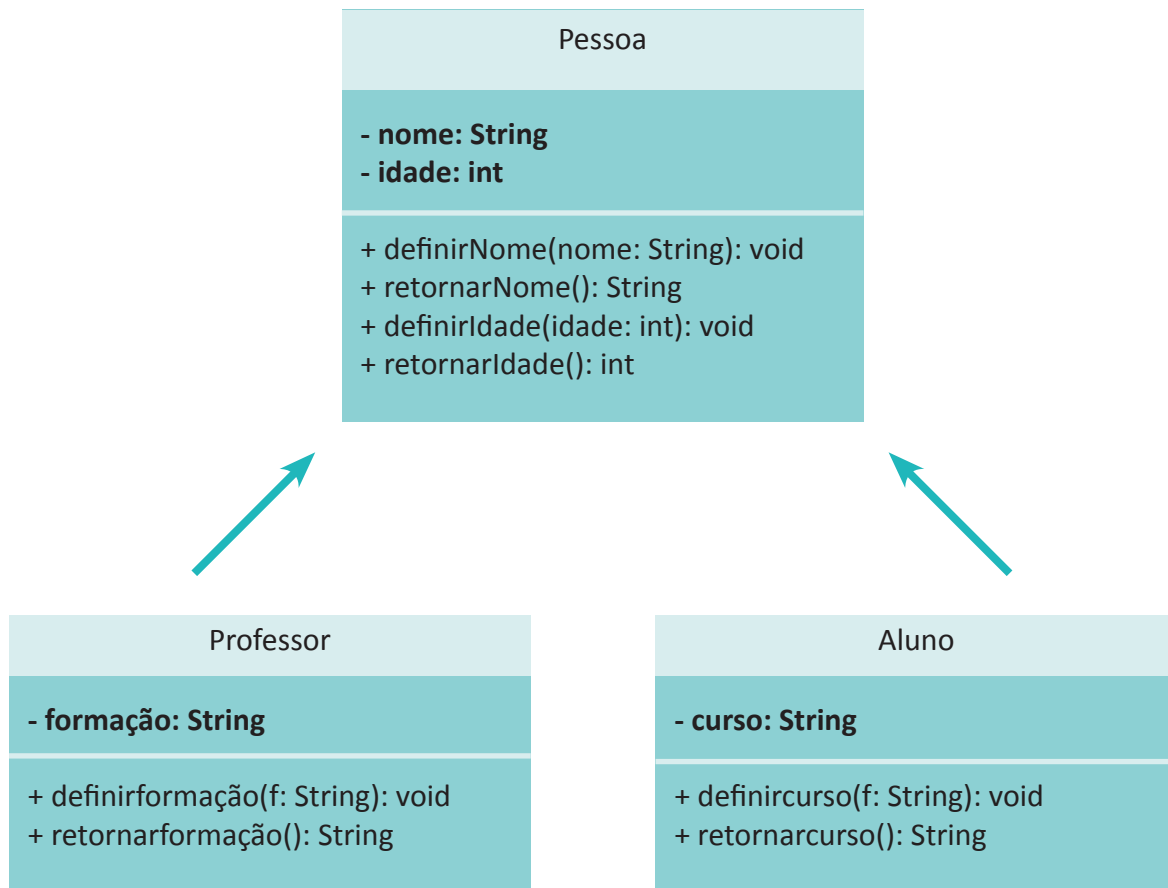


Imagem adaptada. Fonte imagem: <http://www2.ic.uff.br/~anselmo/cursos/TPA/apresentacoes/Heranca.pdf>

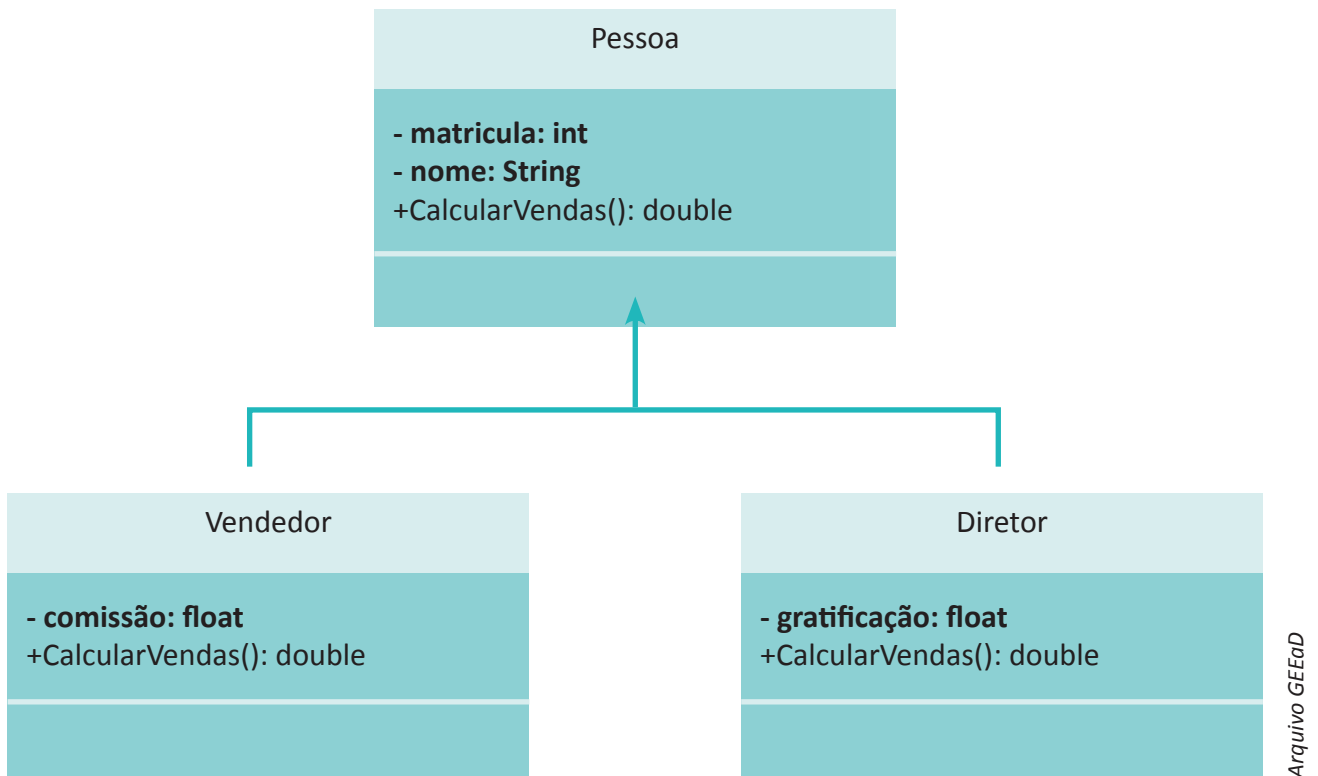
## Polimorfismo

Polimorfismo é uma palavra grega que significa múltiplas formas.

Na modelagem Orientada a Objeto, o Polimorfismo tem como princípio a criação de uma hierarquia de objetos com métodos e nomes comuns para operações conceitualmente similares, porém implementados de forma diferente para cada classe da hierarquia. Como resultado, a mesma mensagem, recebida por objetos diferentes podem causar ações complementares diferentes.

Vamos aos exemplos para entender melhor esse conceito!

Podemos dizer que uma classe chamada Vendedor e outra chamada Diretor podem ter como base uma classe chamada Pessoa, com um método chamado CalcularVendas. Se este método (definido na classe base) se comportar de maneira diferente para as chamadas feitas a partir de uma instância de Vendedor e para as chamadas feitas a partir de uma instância de Diretor, ele será considerado um método polimórfico, ou seja, um método que assume várias formas.



Observe a função do método `CalcularVendas` na classe base `Pessoa`, implementada em Java:

```

public decimal CalcularVendas()
{
    decimal valorUnitario = decimal.MinValue;
    decimal produtosVendidos = decimal.MinValue;

    return valorUnitario * produtosVendidos;
}
    
```

Na classe `Vendedor` temos o mesmo método, mas com a codificação diferente:

```

public decimal CalcularVendas()
{
    decimal valorUnitario = 50;
    decimal produtosVendidos = 1500;

    return valorUnitario * produtosVendidos;
}
    
```

Agora perceba que na classe Diretor, o método CalcularVendas() assume outra codificação, também com resultados distintos dos demais.

```
public decimal CalcularVendas()
{
    decimal valorUnitario = 150;
    decimal produtosVendidos = 3800;
    decimal taxaAdicional = 100;

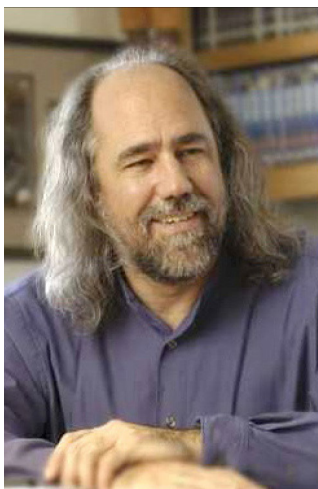
    return taxaAdicional + (valorUnitario * produtosVendidos);
}
```

Assim temos um mesmo método, mas com diferentes formas. Portanto, o método CalcularVendas() é um método polimórfico.

## Análise Orientada a Objetos

No Product Backlog fazemos o levantamento e análise de requisitos. Nesse momento, é preciso identificar quais classes deverão ser implementadas e quais serão seus principais atributos e métodos para que os requisitos sejam satisfeitos. Para isso, geralmente são utilizados os Diagramas de Casos de Uso e o Diagrama de Classe da UML (Unified Modeling Language) - Linguagem Unificada de Modelagem, que possibilita visualização, especificação, construção e documentação de artefatos de um sistema complexo de software orientado a objetos.

A UML foi criada pelos norte-americanos Grady Booch e James Rumbaugh e pelo suíço Ivar Jacobson, que em 1995 lançaram a UML 0.8, unificando o Método de Booch, o método OOSE (Object-Oriented Software Engineering), de Jacobson e o método de OMT (Object Modeling Technique), de Rumbaugh.



Grady Booch



Ivar Jacobson



James Rumbaugh

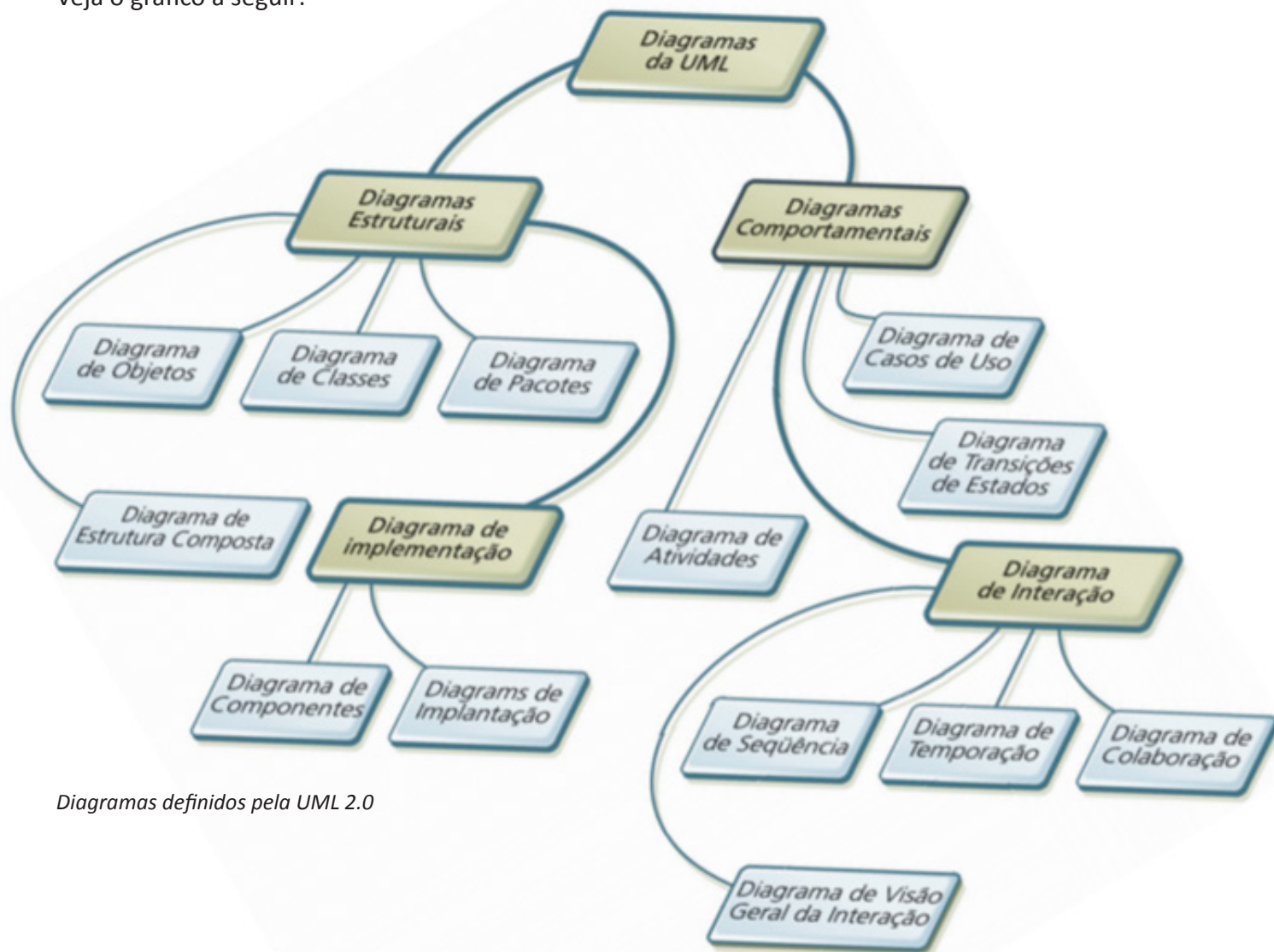
Criadores da UML. Fonte imagens:

<http://www.fabiobmed.com.br/booch-rumbaugh-jacobson-e-a-uml-unified-modeling-language/>

Por oferecer ferramentas que podem ser utilizadas em todas as fases do desenvolvimento de software, a UML acabou se tornando padrão de desenvolvimento de software orientado a objetos.

Existem 13 diagramas na UML 2.0, os quais são divididos em quatro grupos, de acordo com o tipo de análise que os modelos gerados por sua utilização possibilitam. São esses os grupos: diagramas estruturais, diagramas comportamentais, diagramas de interação e diagramas de implementação.

Veja o gráfico a seguir:



Diagramas definidos pela UML 2.0

Assista ao vídeo “UML é importante para você?” do Canal DevMedia.



Para conhecer os diagramas da UML, acesse os slides do Prof. Eduardo Figueiredo, da UFMG.

Disponível em [https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagramas\\_v01-1.pdf](https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagramas_v01-1.pdf)

Acessado em 10/07/2018.

Nessa agenda, vamos nos atentar ao Diagrama de Caso de Uso e ao Diagrama de Classe.

## Diagrama de Caso de Uso

Os diagramas de Caso de Uso são uma excelente técnica para auxiliar no levantamento dos requisitos de um sistema

É composto por um conjunto de casos de uso, atores e seus relacionamentos - cada caso de uso é um de seus requisitos funcionais. Você pode criar diagramas de caso de uso para avaliar alguma situação que não ficou muito clara durante as entrevistas ou para definir como será a relação dos diversos agentes de software no sistema, ou ainda para verificar que funcionalidades este deverá implementar. O que faremos é mapear os requisitos funcionais do sistema, sua análise e também as relações que tais requisitos terão com os demais componentes, internos ou externos ao sistema.

Principais componentes do Diagrama de Caso de Uso são: o cenário, os atores, os casos de uso e os relacionamentos.

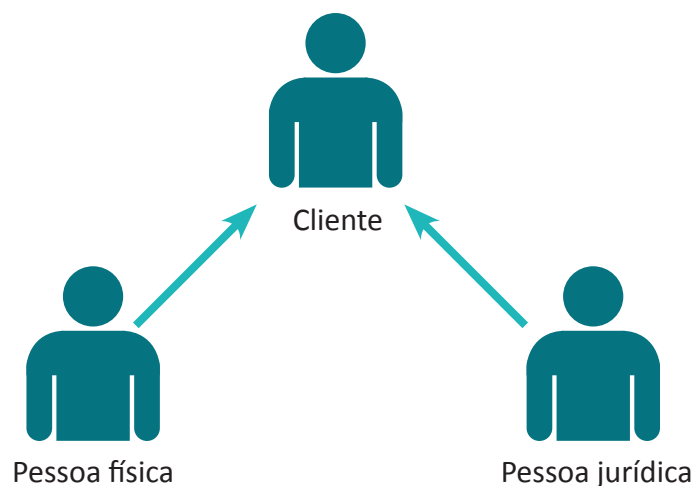
Um Ator em um Caso de Uso representa uma entidade externa ao sistema que interage de alguma forma com um Caso de Uso. Normalmente, um ator é uma pessoa, um dispositivo de hardware ou um outro sistema.

No Diagrama de Caso de Uso, o ator é representado como na imagem:



Representação do Ator

Da mesma forma como estudamos no conceito de herança, temos também a Generalização de Atores:



Vamos criar um exemplo de um cenário para entender melhor a notação de um Diagrama de Caso de Uso:

“O Cliente chega ao caixa eletrônico de um banco, efetua um saque em sua conta corrente, emite o extrato da conta e efetua um pagamento de um boleto. O sistema bancário controla as operações de saque, de emissão de extrato e do pagamento do boleto. O administrador do sistema alimenta o caixa com as cédulas monetárias (dinheiro).

Com esse cenário simples podemos começar a criar nosso diagrama. Inicialmente vamos definir nossos atores:

- Cliente
- Sistema Bancário
- Funcionário do Banco


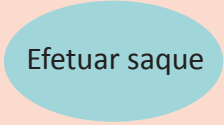

Analisando a descrição do cenário, podemos identificar as ações dos atores:

**Cliente:** Efetua saque; Emite extrato; Efetua Pagamento.

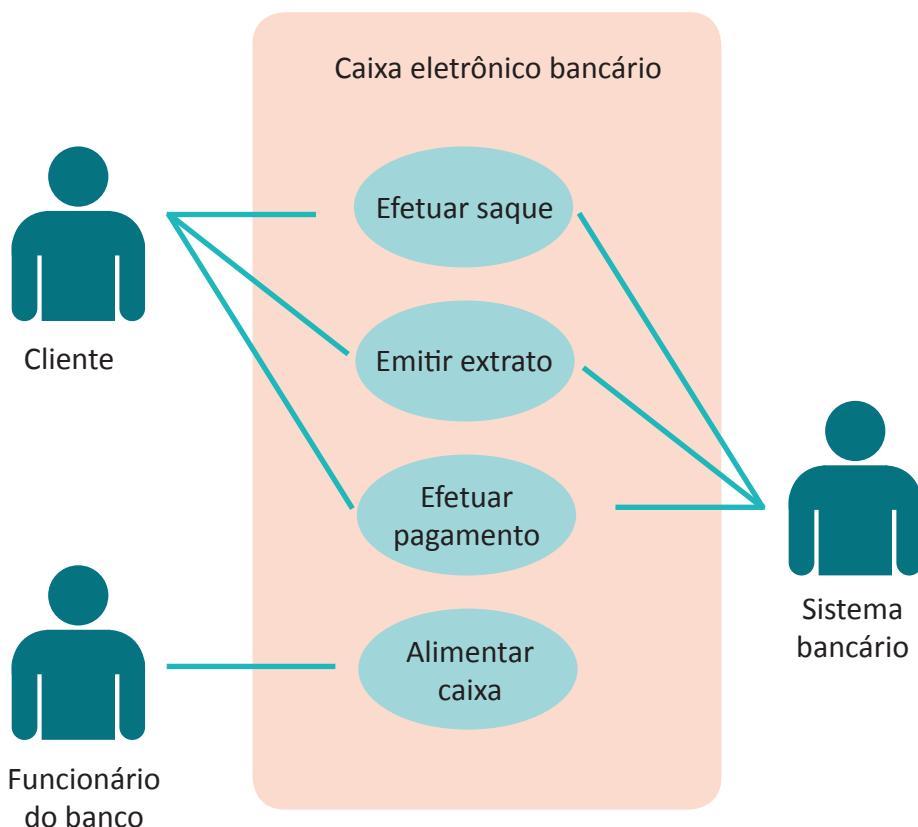
**Sistema Bancário:** Controla operação de efetuar saque; Controla operação de emitir extrato; Controla operação de efetuar pagamento.

**Funcionário do Banco:** Alimenta o caixa.

Veja as representações de cada elemento no quadro “Elementos do Diagrama de Caso de Uso”:

Ator	Caso de Uso	Comunicação
		

Agora vamos construir o diagrama de caso de uso:



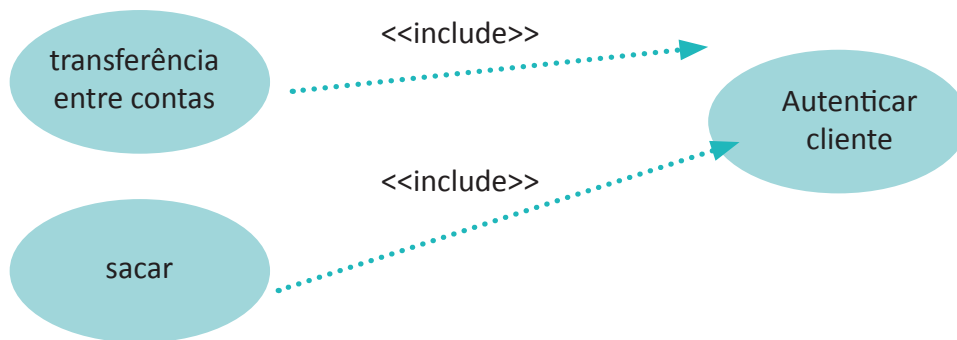


Como se pode observar, o diagrama de caso de uso é composto por desenhos simples que descrevem as funcionalidades do sistema e as interações dos atores com elas.

Existem ainda os conceitos de <>Include e <>Extend que podem melhorar ainda mais as especificações das ações no diagrama.

### <>Include

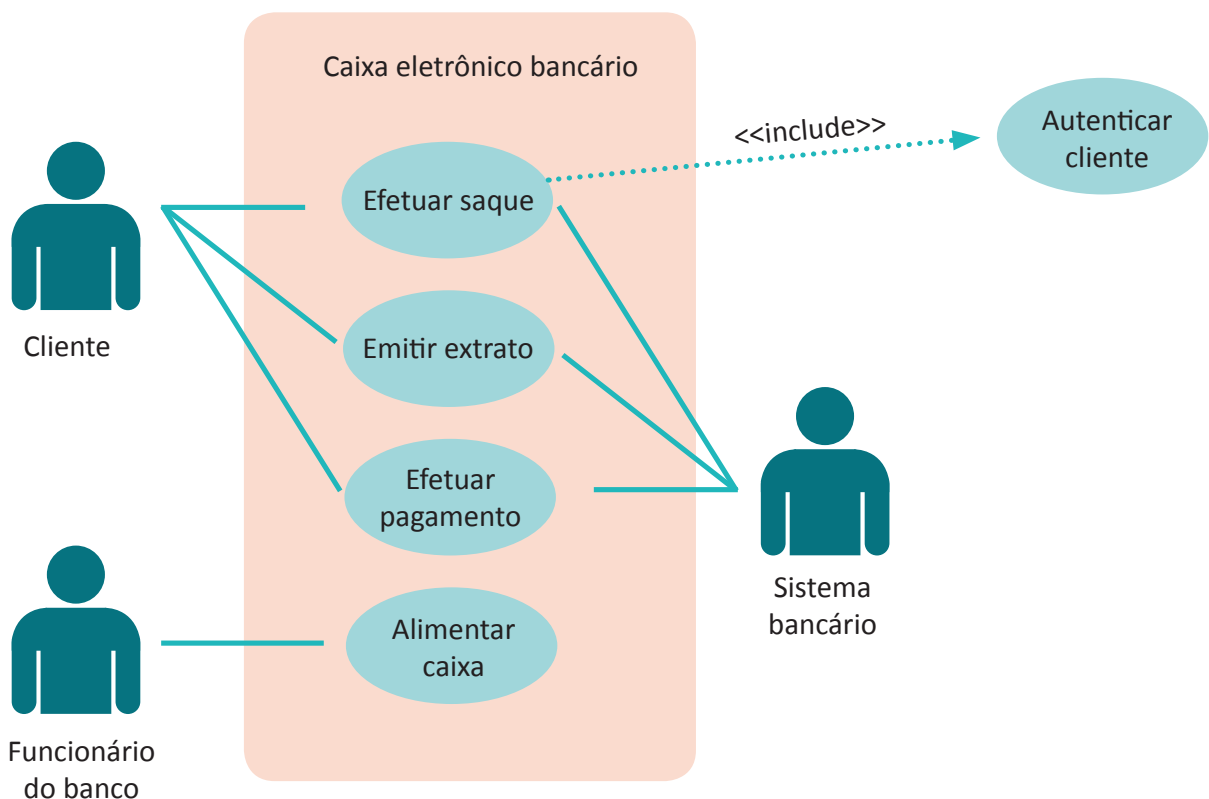
Uma relação de Inclusão de um **Caso de Uso A** com um **Caso de Uso B** indica que uma instância do **Caso de Uso A** deve incluir o comportamento especificado para o **Caso de Uso B**.



### <>Extend

Uma relação de Extensão de um **Caso de Uso A** com um **Caso de Uso B** indica que uma instância do **Caso de Uso B** pode incluir - sujeito ao atendimento de certas condições - o comportamento especificado para o **Caso de Uso A**.

Com isso, podemos aprimorar o diagrama inserindo um novo caso de uso “Autenticar Cliente, que será utilizado no caso de uso “Efetuar saque”. Por questão de segurança, o sistema precisa confirmar a autenticidade do cliente.



O diagrama de Caso de Uso é muito importante para **auxiliar na definição dos Requisitos**, na Comunicação com os Clientes por não exigir conhecimentos técnicos e para **Delimitar o Contexto do Sistema**.



Tente criar um diagrama de Caso de Uso para um Sistema de Biblioteca Escolar com o seguinte contexto:

- A biblioteca de uma escola possui muitos usuários, dentre eles: professores e alunos que realizam empréstimos e devoluções de exemplares com o atendente, funcionário da biblioteca. Se o usuário não estiver cadastrado, ele deve informar seus dados pessoais como: nome completo, telefone e endereço. No caso do professor, deve ser informada também a sua formação e no caso do aluno, deve ser informado o nome do curso no qual está matriculado.
- No empréstimo e/ou devolução os usuários podem realizar a retirada ou devolução dos livros desejados, e o atendente registra a operação com os seguintes dados: data do empréstimo, data da devolução e os exemplares desejados.
- A biblioteca possui muitos exemplares de um livro, cujos atributos são: código do livro, editora e título.
- É responsabilidade do atendente manter os cadastros dos professores e alunos atualizado.

Se você fez o rascunho do seu Diagrama de Caso de Uso, é hora de conferir se você acertou.

### Veja:

Para iniciar o levantamento das informações devemos começar identificando os atores do caso de uso, neste contexto temos:

Neste diagrama, o atendente, o professor e o aluno são identificados como atores porque são externos à situação do cenário e não representam nenhuma funcionalidade e também não se comportam como um requisito do sistema.

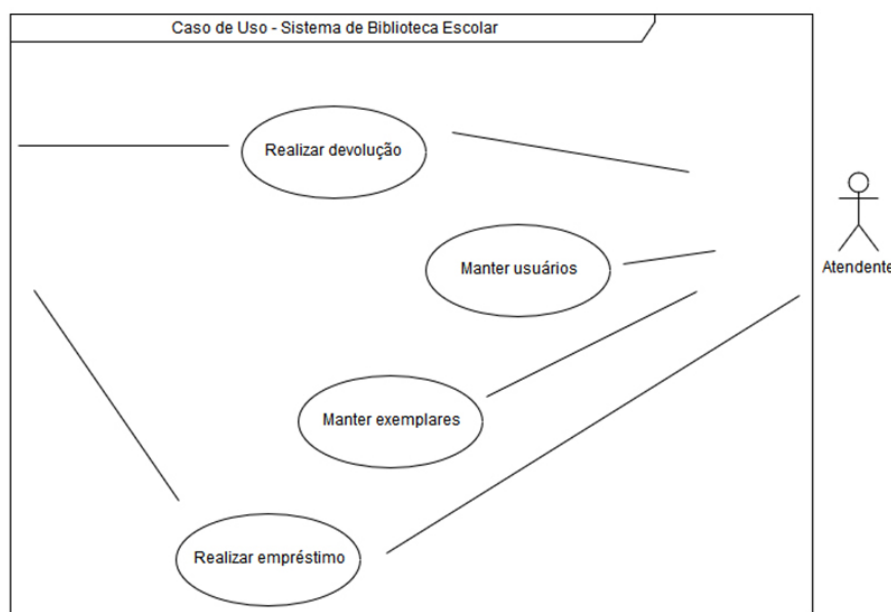
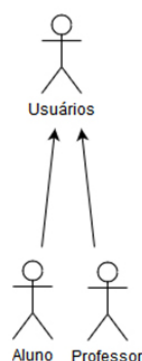


Diagrama de caso de uso – Sistema de Biblioteca Escolar

### As ações realizadas pelos atores são:

- alunos, professores e atendente: realizar devolução e realizar empréstimo
- atendente: manter professores e alunos
- atendente: manter exemplares

De acordo com os conceitos vistos podemos aplicar a generalização de atores entre alunos e professores e criar um ator usuário pois, ambos têm atributos em comum como: nome completo, telefone e endereço.

Para representar o caso de uso, lembre-se de que os bonecos são os atores, as elipses são os casos de uso e as linhas representam a comunicação entre ator e caso de uso.

A imagem apresenta o Diagrama de Caso de Uso, de acordo com análise do contexto da Biblioteca Escolar

## Diagrama de Classes

O Diagrama de Classes é o mais utilizado da UML. Esse diagrama fornece uma visão estática do modelo a ser criado. Como as classes são um dos componentes mais importantes da orientação a objetos, esse diagrama deve constar de todo projeto orientado a objetos.

O Diagrama de classe não deve faltar em projetos orientados a objetos. Devemos prestar muita atenção ao criar um diagrama de classes, que será a base da nossa solução

Principais componentes: classes, interfaces, relacionamentos.

Identificar uma classe não é tarefa das mais simples, mas o caminho é procurar itens que têm as mesmas informações e comportamentos. Nem sempre uma classe tem atributos e métodos. Pode ter apenas métodos ou apenas atributos.

O caminho é analisar o cenário do projeto com base no diagrama de Caso de Uso, fazer uma lista do que você identificou como classes. Acrescente os atores, que geralmente são também classes de seu sistema. Analise as candidatas a classes e tente achar atributos para elas e, se possível, alguma funcionalidade. Coloque as classes em um diagrama e comece a analisar as relações entre elas, de acordo com os tipos de relacionamentos que esse diagrama propõe. Veja:

Associação

Agregação

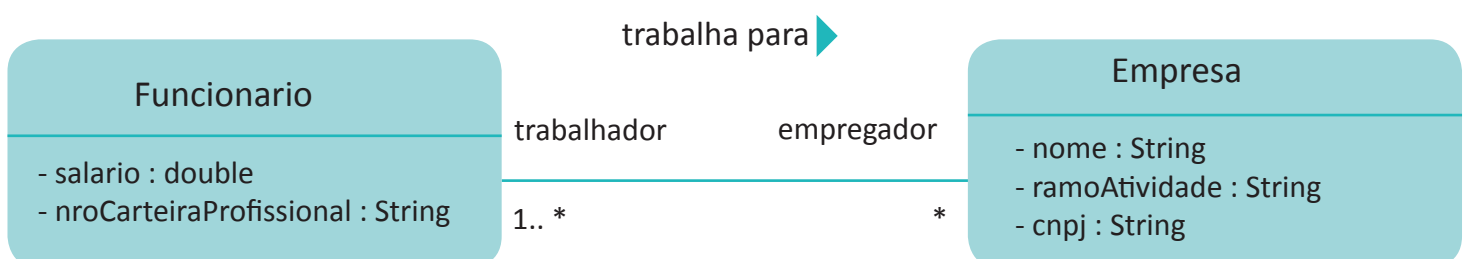
Composição

Herança: Especialização/Generalização

Dependência

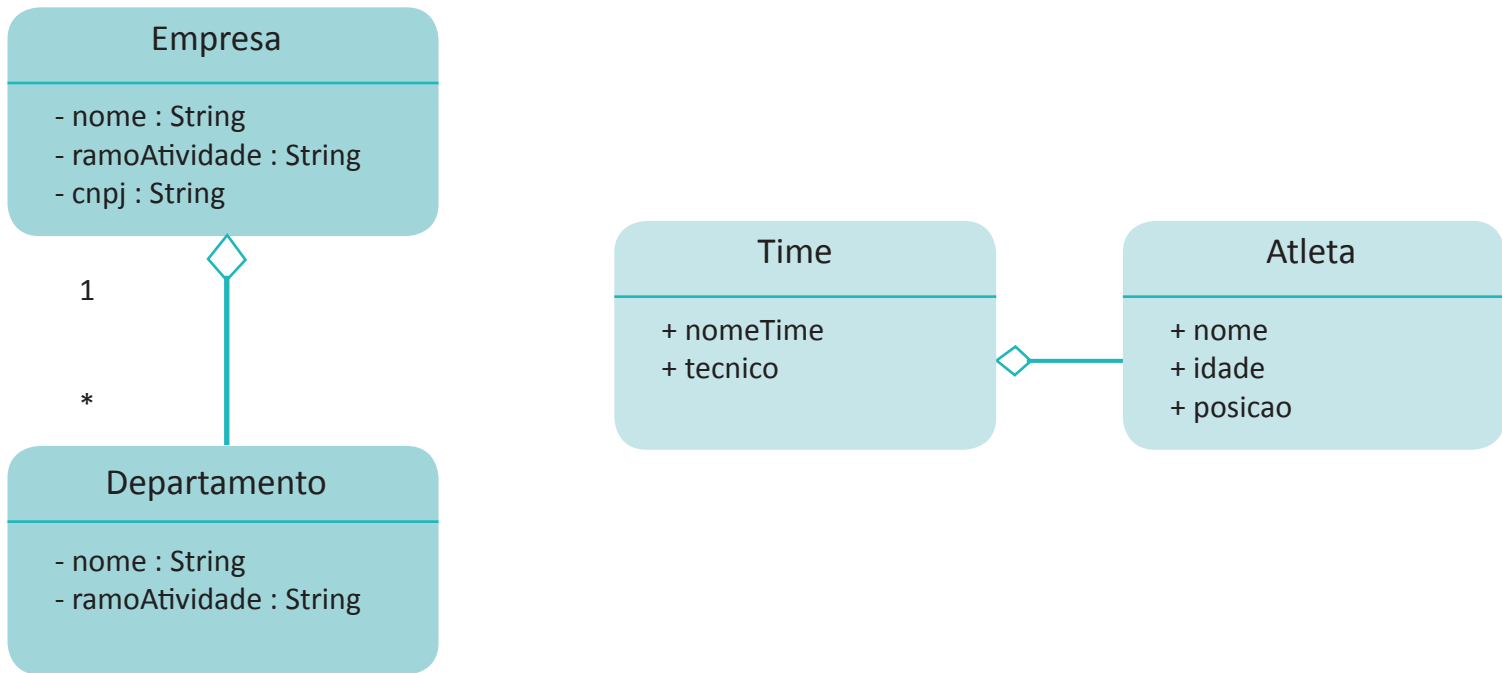
**Associação:** é um relacionamento estrutural que especifica objetos de uma classe conectados a objetos de outra classe. A partir de uma associação, conectando duas classes, você é capaz de navegar do objeto de uma classe até o objeto de outra classe e vice-versa (BOOCH, RUMBAUGH e JACOBSON, 2005).

É representada por uma linha interligando as duas classes, uma associação pode definir papéis das classes relacionadas, assim como a multiplicidade de sua associação, além de ter um nome.



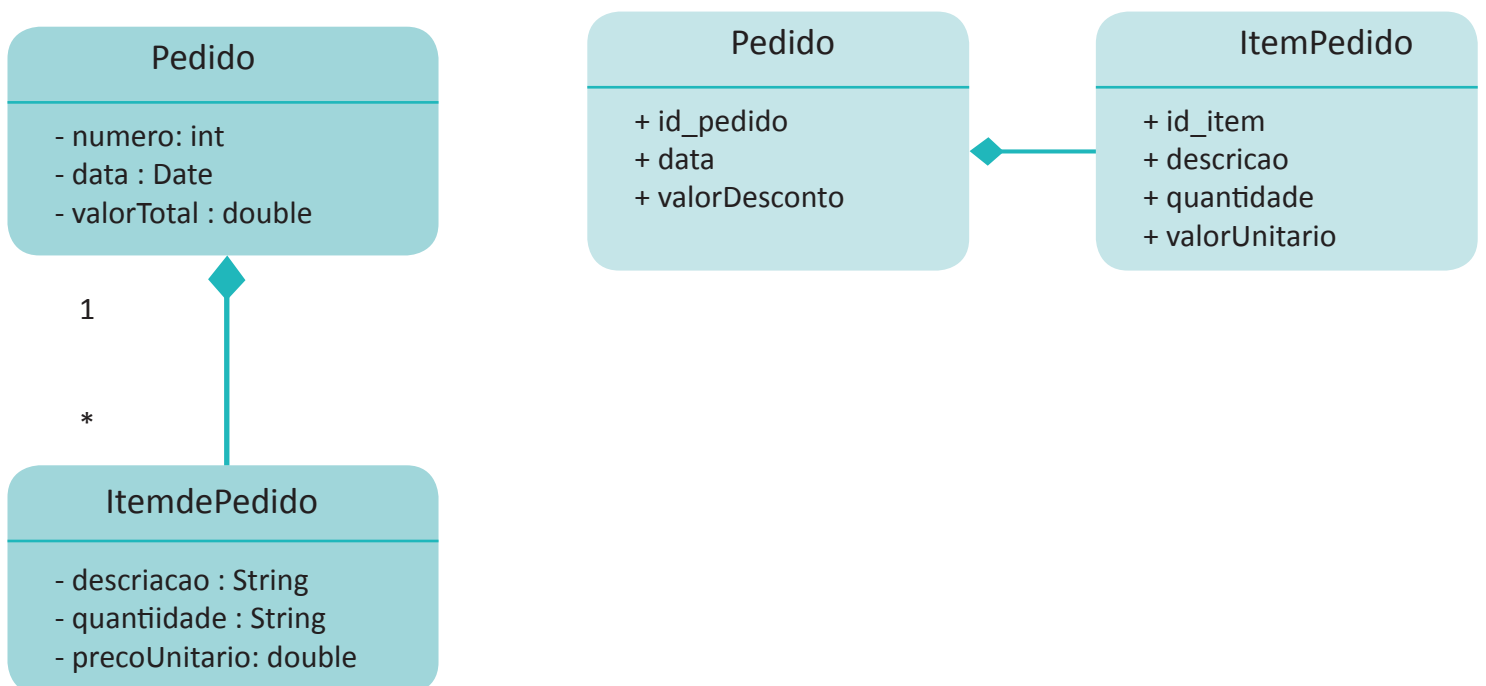
**Agregação:** é um tipo especial de associação que tenta demonstrar que as informações de um objeto-todo precisam ser complementadas pelas informações contidas em um (ou mais) objetos-parte.

Exemplos de agregação:



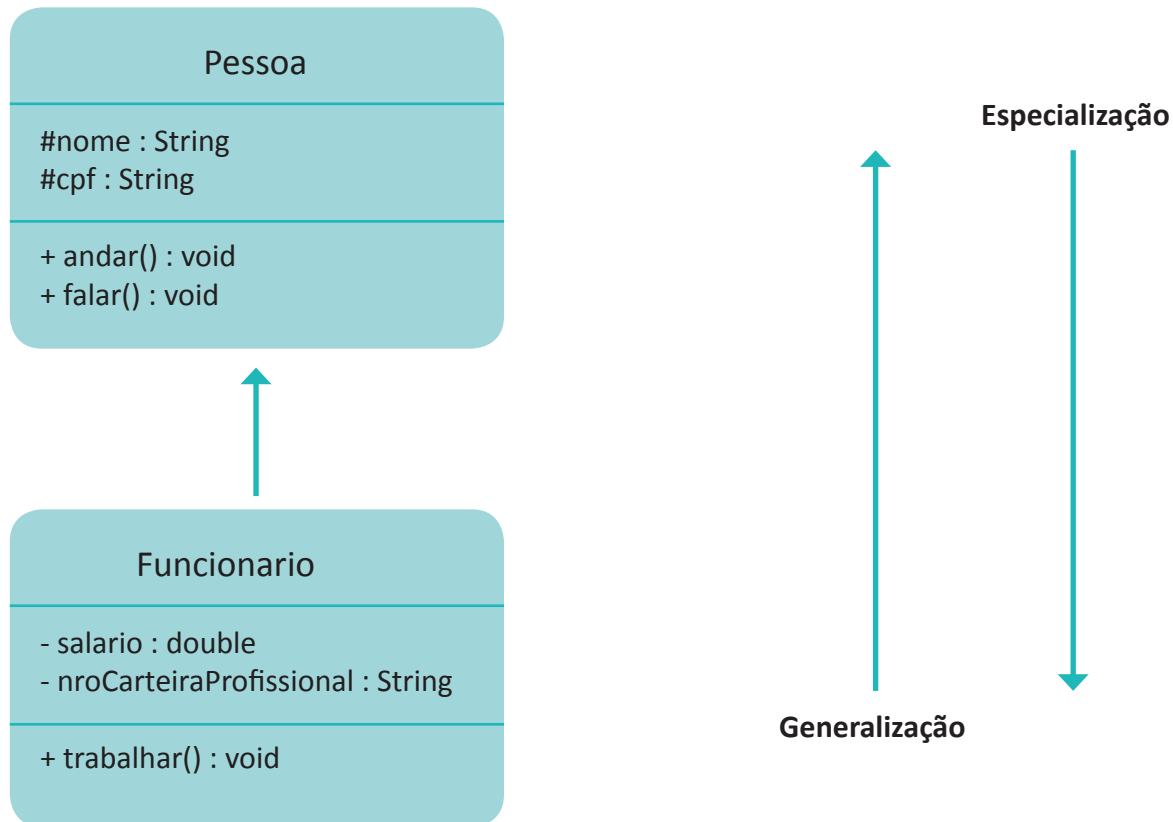
**Composição:** é uma variação da agregação e também representa uma relação todo - parte. No entanto, na composição o objeto-pai (todo) é responsável por criar e destruir suas partes. Em uma composição um mesmo objeto-parte não pode se associar a mais de um objeto-pai.

Exemplos de Composição:



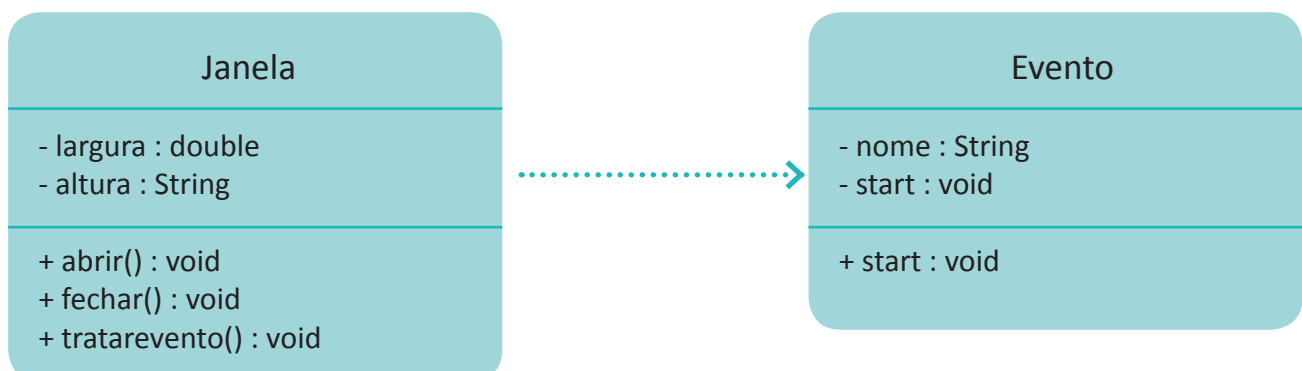
**Herança: Especialização e Generalização:** tem como objetivo identificar classes-mãe, denominadas de gerais, e classes-filha chamadas de especializadas. São chamados de relacionamentos “é um tipo de”.

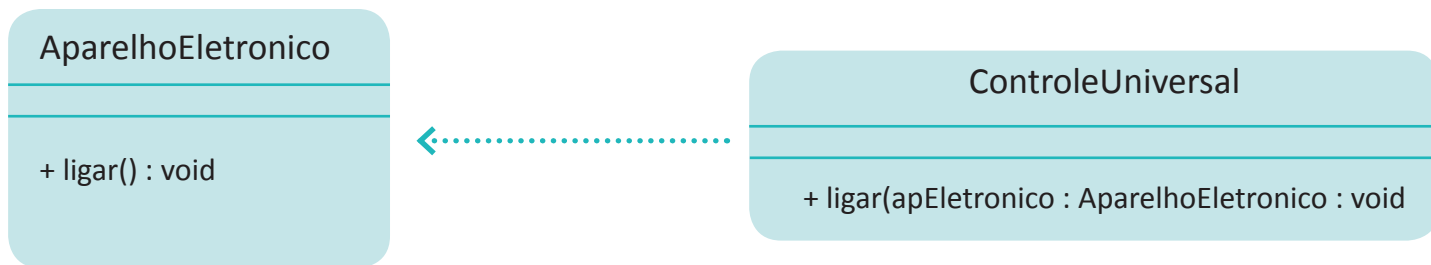
Podemos verificar, na imagem a seguir, que a classe Pessoa possui os atributos nome e CPF e pode executar os métodos de andar e falar. Já a classe Funcionario, por herdar os atributos e métodos da classe Pessoa, possui nome, CPF, salário e nro Carteira Profissional, podendo executar os métodos andar, falar e trabalhar. Dizemos que a classe Pessoa é a superclasse da classe Funcionario, que é sua subclasse, ou que Pessoa é a classe pai de Funcionario e que Funcionario é classe filha de Pessoa.



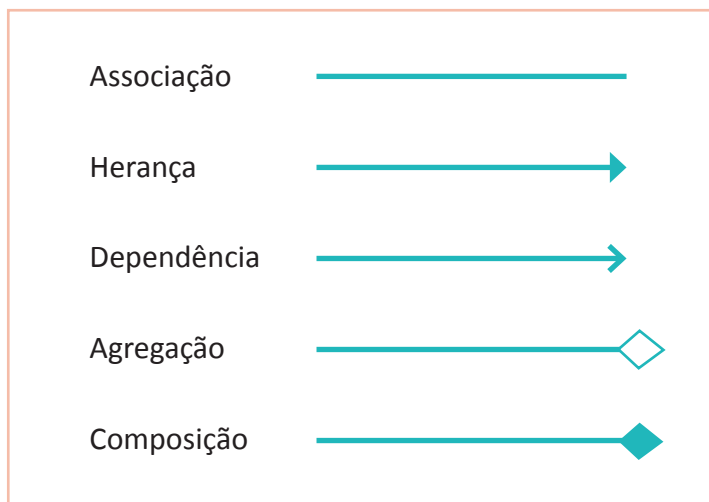
**Dependência:** como o nome sugere, indica um grau de dependência entre uma classe e outra. Uma dependência difere de uma associação porque a conexão entre as classes é temporária. É representada por uma seta tracejada entre duas classes.

Exemplos de Dependência:

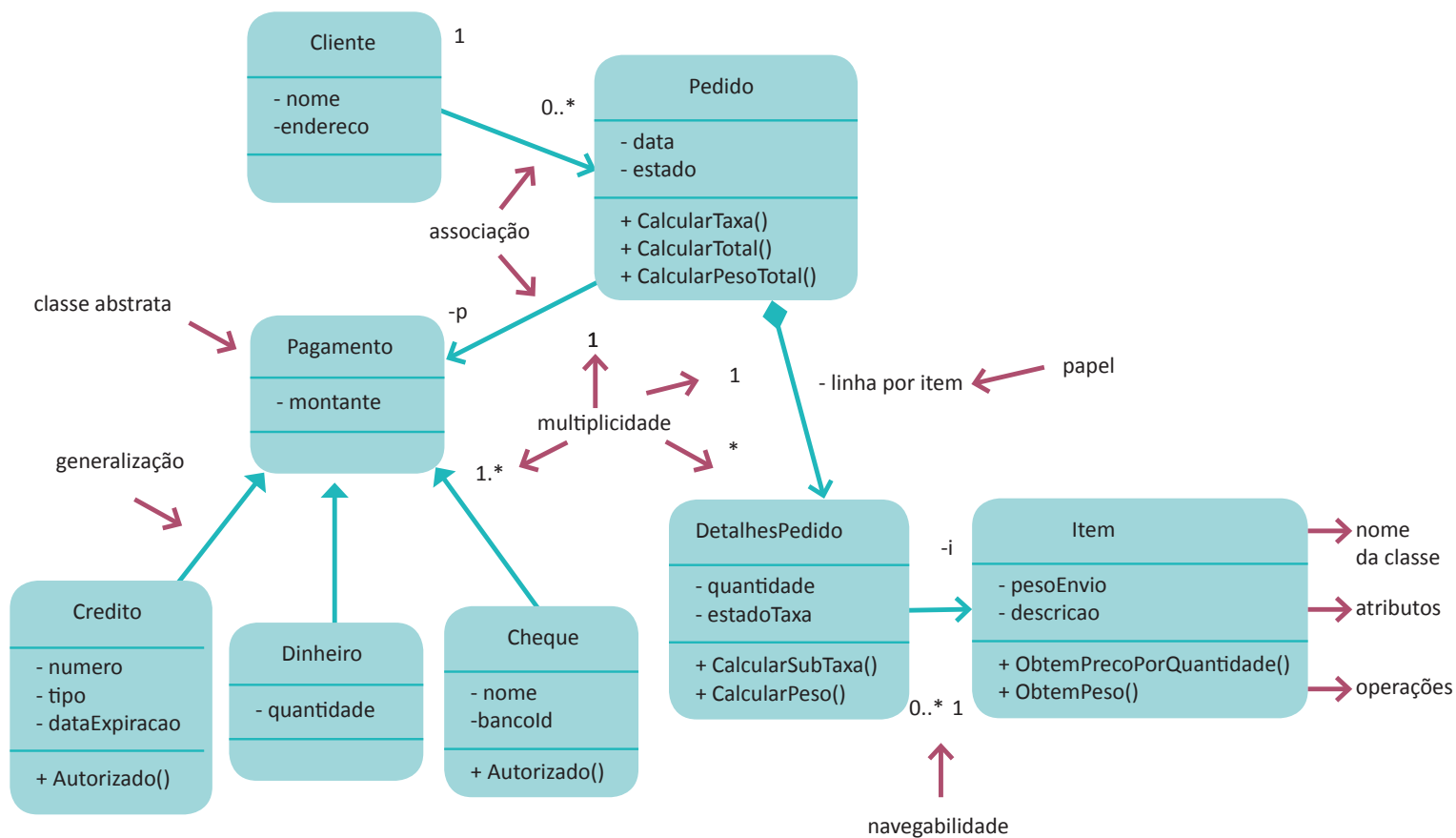




Representação dos tipos de relacionamentos da UML



Exemplo de Diagrama de Classe e seus relacionamentos:



Exemplo de Diagrama de Classe (adaptado). Fonte: Tecnologia Local, 2009.

Disponível em <http://tecnologia.local.blogspot.com/2009/>. Acessado em 12/11/2018.



O diagrama de classe modela um pedido de um cliente para um catálogo de itens. A classe central é o Pedido. Associada a ela estão: o Cliente fazendo a compra e o Pagamento. Um Pagamento pode ser de três tipos: Dinheiro, Cheque ou Crédito. O pedido contém DetalhesPedido, cada um associado a um Item.