
Newton
Quem se prepara, não para.

Análise de Sistemas

3º período

Professora: Michelle Hanne

Processo de Desenvolvimento de Software

Newton
Quem se prepara, não para.

As atividades vistas como fundamentais no processo de *software* são (SOMMERVILLE, 2011):

- **Especificação:** é o documento que define as funcionalidades e restrições do *software*.
- **Projeto e implementação:** é a fase onde são modelados e codificados os artefatos do *software*.
- **Validação de *software*:** é a fase que garante a qualidade do *software* que valida que ele está de acordo com a especificação.
- **Evolução:** na evolução, são adicionadas ao *software* já existente novas funcionalidades.

Processo de Desenvolvimento de Software

Newton
Quem se prepara, não para.

Atividades de arcabouço dos processos de Desenvolvimento de Software:

- **Comunicação:** é a atividade na qual são levantados os requisitos junto ao cliente (habilidades de Soft Skills e Sistêmicas)
- **Planejamento:** são elaborados o cronograma, as análises de riscos e os recursos necessários.
- **Modelagem:** traduz a especificação em formatos simbólicos, tanto em nível de requisito quanto em nível de projeto.
- **Construção:** são gerados os códigos e testes do *software*.
- **Implantação:** é quando se coloca o produto em produção, ou seja, é nesta atividade que o cliente recebe o *software*.

Processo de Desenvolvimento de Software

Os participantes dos projetos são classificados em três categorias (PFLEEGER, 2004:

- **Cliente:** é quem paga para que seja construído o sistema.
- **Usuário:** é quem usa o sistema, não necessariamente é o cliente.
- **Desenvolvedor:** é o responsável por desenvolver e manter o sistema.

Processo Unificado

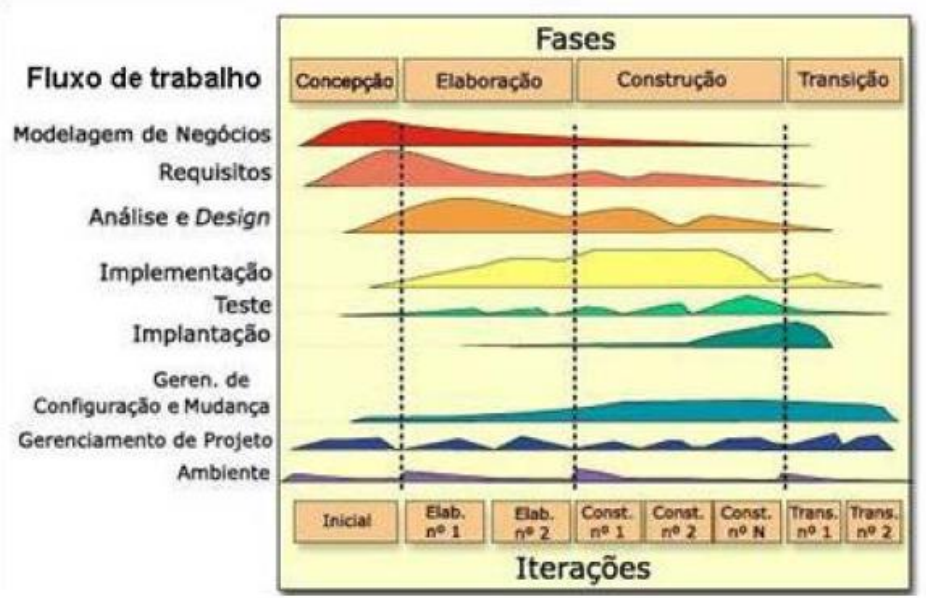
Newton
Quem se prepara, não para.

Processo Unificado (PU - Unified Process), o qual é chamado de framework, isto é, ele dispõe de processos e métodos para alcançar um objetivo que visa aumentar a qualidade do desenvolvimento de software.

Processo Unificado

Na fase de elaboração, é feito o detalhamento da análise de requisitos utilizando modelos definidos.
A construção consiste na geração de código e teste do sistema.
Na fase de transição, é colocado o sistema em uso no ambiente final, sendo necessários testes de aceitação e operação, treinamento do usuário.

Figura 1 – Processo Unificado (PU)



Fonte: KRUCHTEN, 2004.

Lembre-se de que ele não diz que cada atividade do fluxo é totalmente sequencial, mas que permeia em cada fase durante o ciclo de vida do projeto.

Analista de Sistemas

Newton
Quem se prepara, não para.

O Analista de sistemas é responsável por realizar a especificação do sistema, consiste em:

- Definição das funcionalidades do sistema.
- Documentação Geral

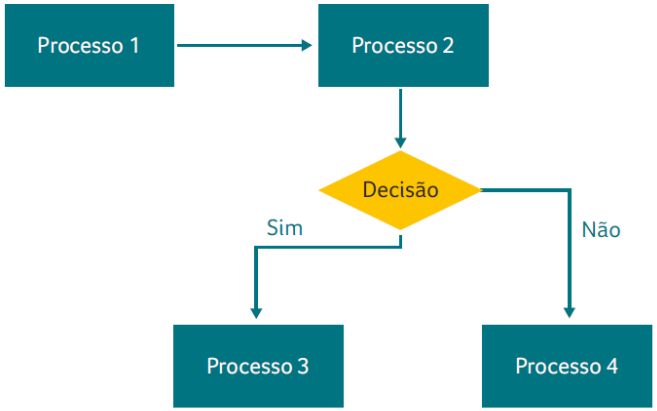
Analise Estruturada

Dados e processos são parte dos conceitos

A modelagem de requisitos que utiliza a análise estruturada é a modelagem clássica. Teve sua origem quando os sistemas computacionais mais complexos começaram a surgir.

Nesse tipo de análise, os diagramas de fluxo de dados (*data flow diagram*) são usados para representar para mostrar uma visão e o fluxo do sistema.

Figura 4 – Diagrama de fluxo de dados



Fonte: Elaborada pelo autor.

Orientação a Objetos

Newton
Quem se prepara, não para.

Orientação a objetos
Nessa abordagem, a ideia é representar os objetos do mundo real que são modelados e construídos no sistema.
Na orientação a objetos, um objeto é uma unidade autônoma que contém seus próprios dados que são manipulados para alcançar os objetivos do objeto (BEZERRA, 2015).

O ideal é um sistema que tenha baixo acoplamento e alta coesão.

Dependência
Pode ocasionar sobrecarga

Responsabilidade
Quando um componente possui responsabilidades específicas

Orientação a Objetos

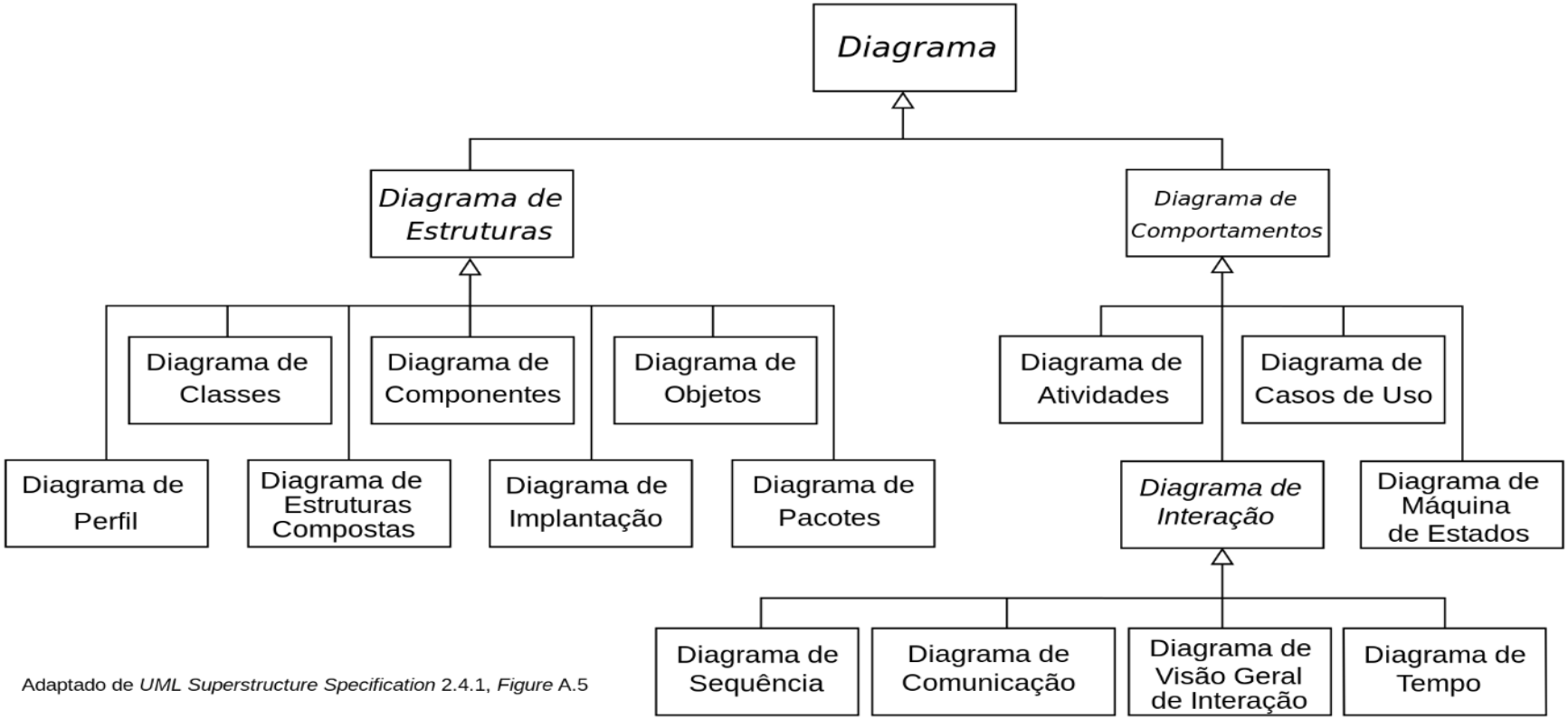
Newton
Quem se prepara, não para.

- O **polimorfismo** permite diferentes comportamentos para uma mesma classes ou métodos com a mesma assinatura. Para se fazer isso, é necessário utilizar a *herança e ou a interface*.
- A **generalização ou herança** possibilita a uma classe filha herdar comportamentos de uma classe pai, possibilitando o reúso.
- O **encapsulamento** é a forma como os atributos e métodos estão visíveis no sistema, como *private*, *public*, *protect*. Assim, o encapsulamento define a forma de acesso desses atributos e métodos.

Como principais vantagens são consideradas a reusabilidade e a manutenibilidade.

Modelagem de Sistemas em UML

Newton
Quem se prepara, não para.



Adaptado de *UML Superstructure Specification* 2.4.1, *Figure* A.5

Diagrama de Caso de Uso

Newton
Quem se prepara, não para.

Caso de Uso descritivo

Quadro 4 – Caso de uso

Ator: professor

Precondição: está no período de lançamento de notas

Cenário principal:

1 – O professor acessou o caso de uso de lançamento de notas

2 – O professor escolhe a turma

3 – O sistema exibe os alunos com os campos de notas habilitados

4 – O professor lança as notas dos alunos

5 – O sistema soma todas as notas de cada aluno e preenche o campo total para cada aluno

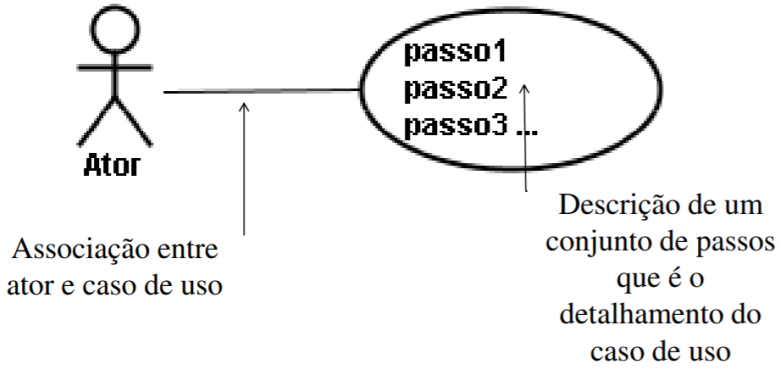
6 – O professor salva o lançamento de nota

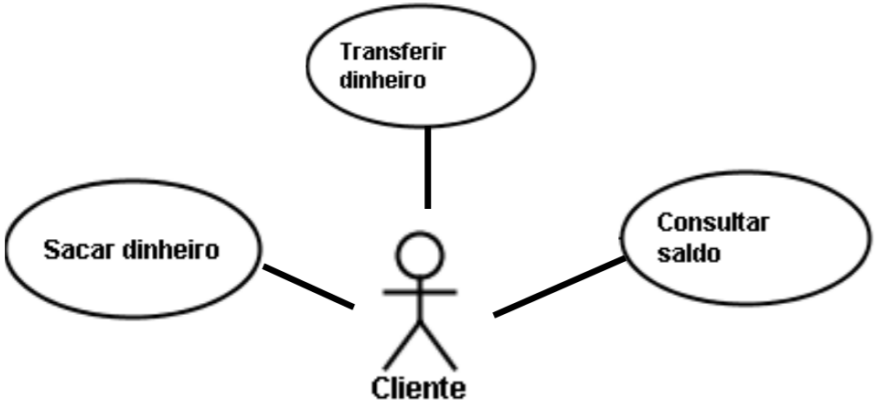
Extensão:

5a – O professor cancela o lançamento de notas

Fonte: Elaborado pelo autor.

Diagrama de Caso de Uso - Exemplo





Descreve o que o sistema faz, mas não especifica como deve ser feito.

Diagrama de Caso de Uso - Relacionamentos

Newton
Quem se prepara, não para.

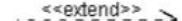
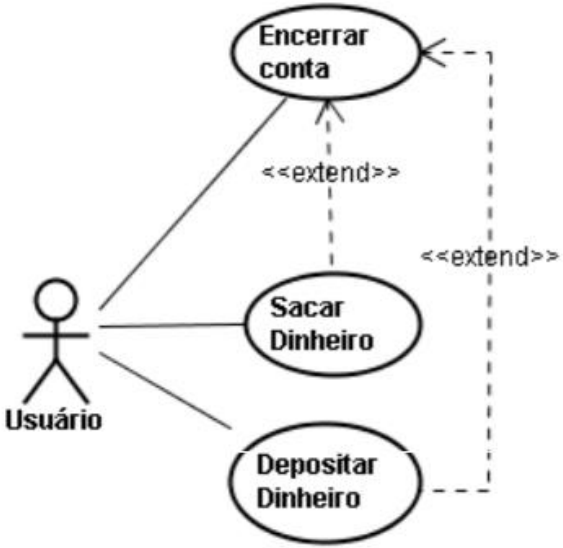
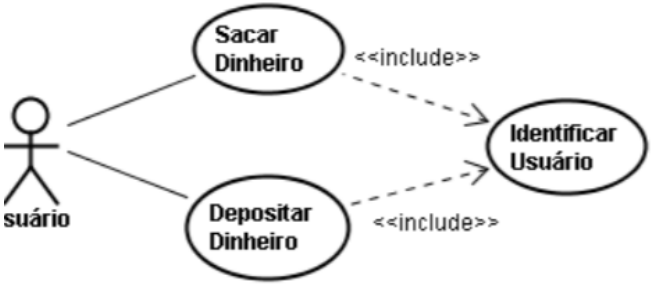
Relação	Função	Notação
Associação	O caminho de comunicação entre um ator e o(s) caso(s) de uso em que participa	
Inclusão	A inserção de um comportamento adicional em um caso de uso base que explicitamente descreve a inserção	
Generalização	Um relacionamento entre um caso de uso geral e um mais específico que herda e adiciona propriedades à aquele	
Extensão	A inserção de um comportamento adicional em um caso de uso base que não sabe sobre o comportamento adicional	

Diagrama de Caso de Uso





Projeto de Software

Newton
Quem se prepara, não para.

Durante os requisitos, são gerados os **artefatos necessários para o entendimento do projeto, que pode conter o modelo de domínio, também chamado de modelo conceitual**.

A classe de *software* representa uma perspectiva de especificação ou implementação de um elemento de *software*, independentemente do processo ou método.

A classe de implementação é executada em uma linguagem de programação como Java.

A classe conceitual identifica os termos de negócio do cliente para modelar o sistema. A classe conceitual representa conceitos do mundo real.

Classes

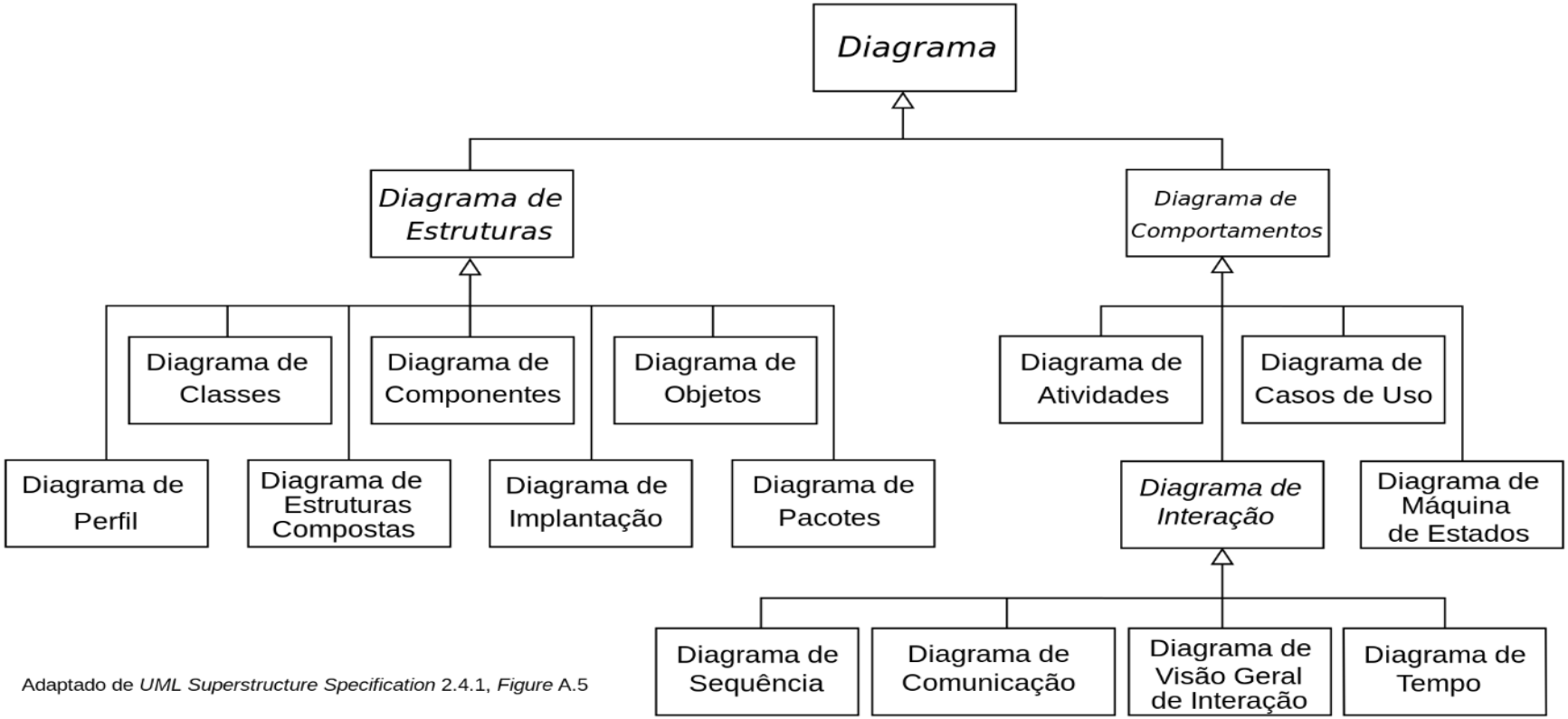
Newton
Quem se prepara, não para.

As classes de análise podem ser categorizadas da seguinte maneira (PRESSMAN; MAXIM, 2016):

- classes de entidade ou classes de modelo ou negócio, que representam o que deve ser armazenado no banco de dados;
- classes de fronteira são usadas para criar uma interface exibida ou interagida com o usuário, como uma tela ou relatório;
- classes de controle são usadas para gerenciar a criação ou atualização de objetos, instância de objetos de fronteira, comunicação entre conjuntos de objetos e validação dos dados

Modelagem de Sistemas em UML

Newton
Quem se prepara, não para.



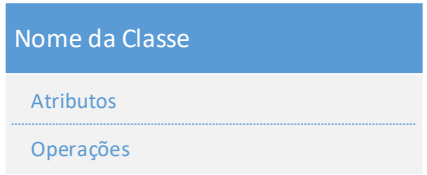
Adaptado de *UML Superstructure Specification* 2.4.1, *Figure* A.5

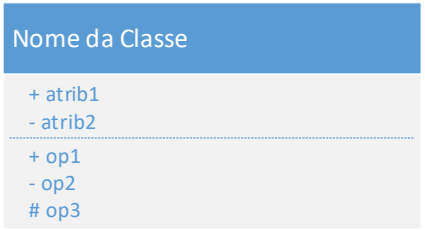
Diagrama de Classe

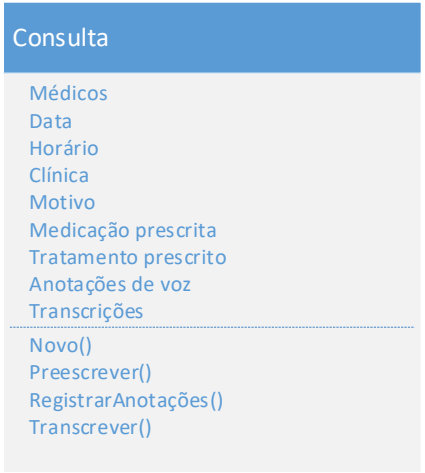
Newton
Quem se prepara, não para.

- Mostra a **estrutura do sistema**, subsistema ou componente projetado como **classes e interfaces** relacionadas, com seus **recursos**, **restrições** e **relacionamentos** - associações, generalizações, dependências, etc. (Sommerville, 2011).

Notação da Classe



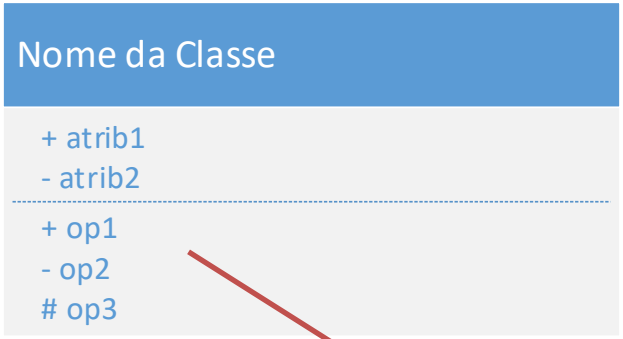




Fonte: Sommerville (2011), p. 91.

Classe é uma abstração de um conjunto de objetos com características similares.

Notação da Classe



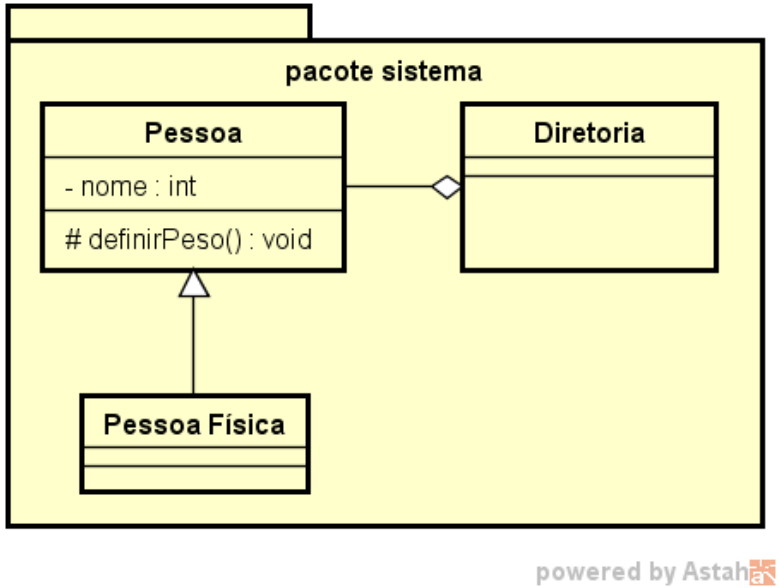
Indica o nível de acessibilidade de um atributo ou método:
+ Público
- Privado
 # Protegido

Notação da Classe

Newton
Quem se prepara, não para.

- (private): Atributos e métodos declarados como *private* são acessíveis somente pela classe que os declara. Métodos e atributos com o modificador *private* **não são herdados.**

 # (Protected): Atributos e métodos declarados como protected são acessíveis pela classe que os declara, suas subclasses em outros pacotes e outras classes dentro do mesmo pacote.



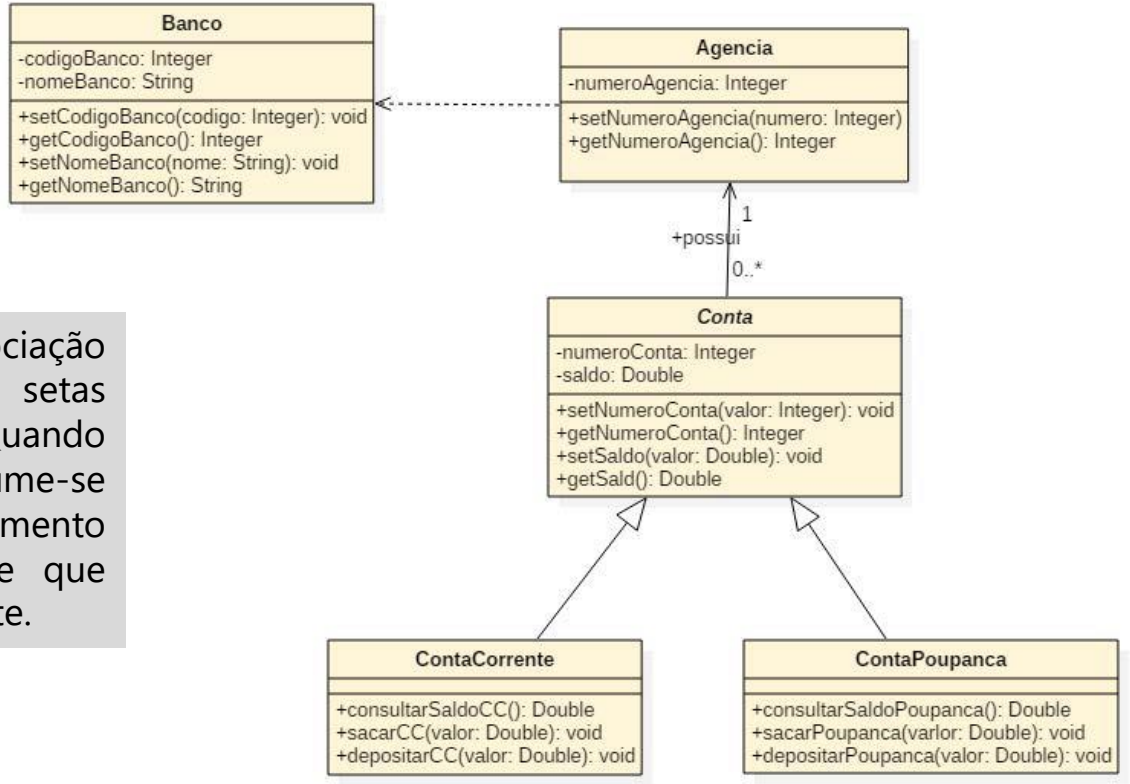
Neste exemplo o método definirPeso( )é visível na subclasse "Pessoa Física" e na classe "Diretoria" que está no mesmo pacote.

Notação da Classe

Newton
Quem se prepara, não para.

+ (Public): Atributos, métodos e classes declarados como public são acessíveis por qualquer classe do Java. Todos os métodos e atributos declarados como public são herdados pelas subclasses. Métodos e atributos declarados como public devem se manter public em todas as subclasses.
~ (Default): Modificador de acesso padrão, usado quando nenhum for definido. Neste caso os atributos, métodos e classes são visíveis por todas as classes dentro do mesmo pacote.

Exemplo

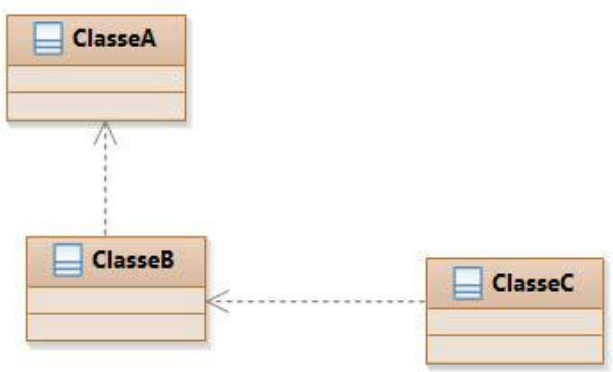


A navegabilidade de uma associação pode ser indicada por setas direcionais ou bidirecionais. Quando não expostas as setas, assume-se que é um relacionamento bidirecional ou simplesmente que essa informação não é relevante.

Relacionamentos

São conexões entre classes:

1. **Dependência** - uma classe usa a outra.

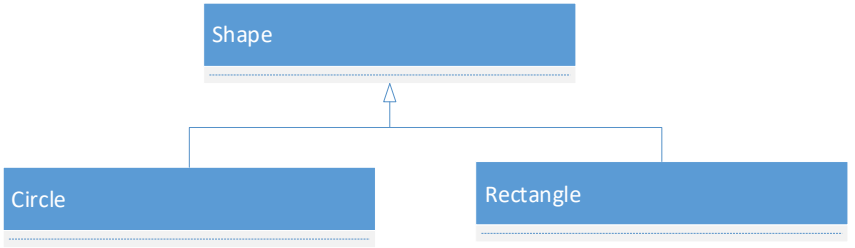


A dependência caminha em única direção, ou seja, não é um relacionamento transitivo. As alterações feitas na ClasseA refletem na ClasseC e ClasseB. O mesmo não ocorre se a alteração for feita na ClasseB, em que apenas a ClasseC sofre com as mudanças. Se as mudanças ocorrerem na ClasseC, nenhuma das outras sofre com a alteração.

Relacionamentos

São conexões entre classes:

2. **Generalização** - geral (superclasse) e uma coisa mais específica (subclasse)

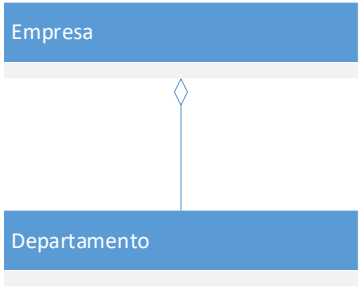


3. **Associação** - classes ou objetos estão interconectados.

Ornamentos para Associações

- **Multiplicidade**

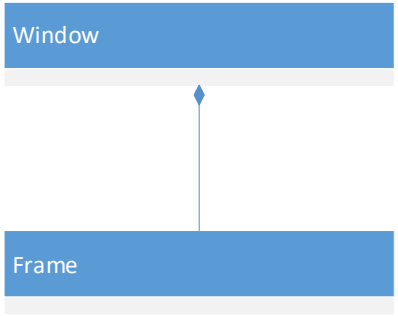
- **Agregação**: É uma relação do tipo “todo/parte” ou “possui um”



Ornamentos para Associações

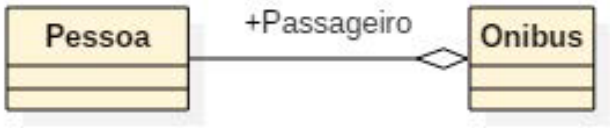
Newton
Quem se prepara, não para.

- **Composição**: Um tipo de agregação na qual as partes são inseparáveis do todo.



Exemplo de Agregação

- É possível ver como o relacionamento entre as classes é fraco.



Exemplo de agregação em UML

- Um ônibus pode ter várias pessoas, no caso, passageiros, porém os passageiros existem sem o ônibus e o ônibus existe sem os passageiros.

Exemplo: https://cursos.alura.com.br/forum/topico-ajuda-no-entendimento-de-composicao-agregacao-e-associacao-52193

Exemplo de Composição

Newton
Quem se prepara, não para.

- Forte relacionamento entre uma nota fiscal e os itens da nota fiscal.

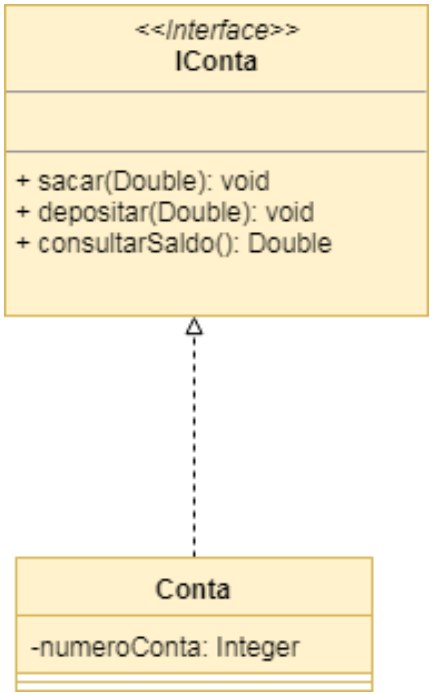
Exemplo composição em UML

- Nesse exemplo, os itens da nota só fazem sentido estando na nota fiscal, e a nota fiscal precisa de itens para ser gerada.

Exemplo: https://cursos.alura.com.br/forum/topico-ajuda-no-entendimento-de-composicao-agregacao-e-associacao-52193

Interface

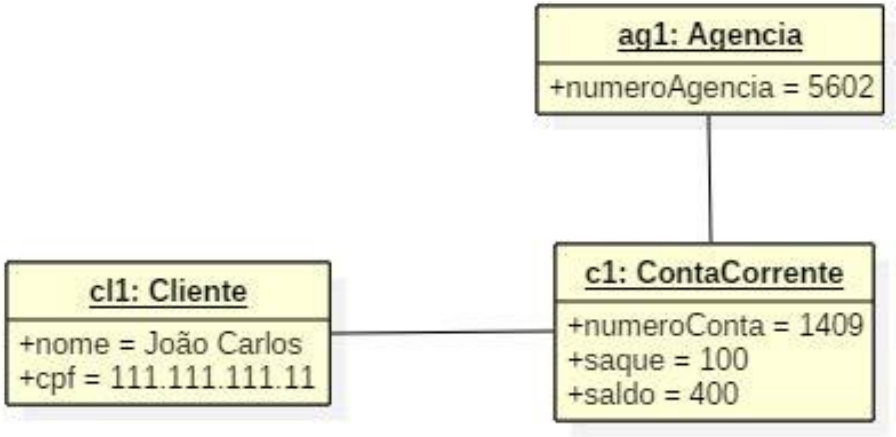
- Uma interface é uma classe que possui apenas métodos públicos, porém esses métodos apenas possuem a assinatura, não possuem corpo.
- No diagrama de classes, ela é definida com a palavra <<interface>>.
- Gera um contrato entre a classe que a declara, no qual essa classe precisa implementar os métodos da interface. Esse contrato estabelece um padrão que deve ser obedecido e ajuda a estruturar a arquitetura do sistema.



Exemplo de interface em UML

Diagrama de Objetos

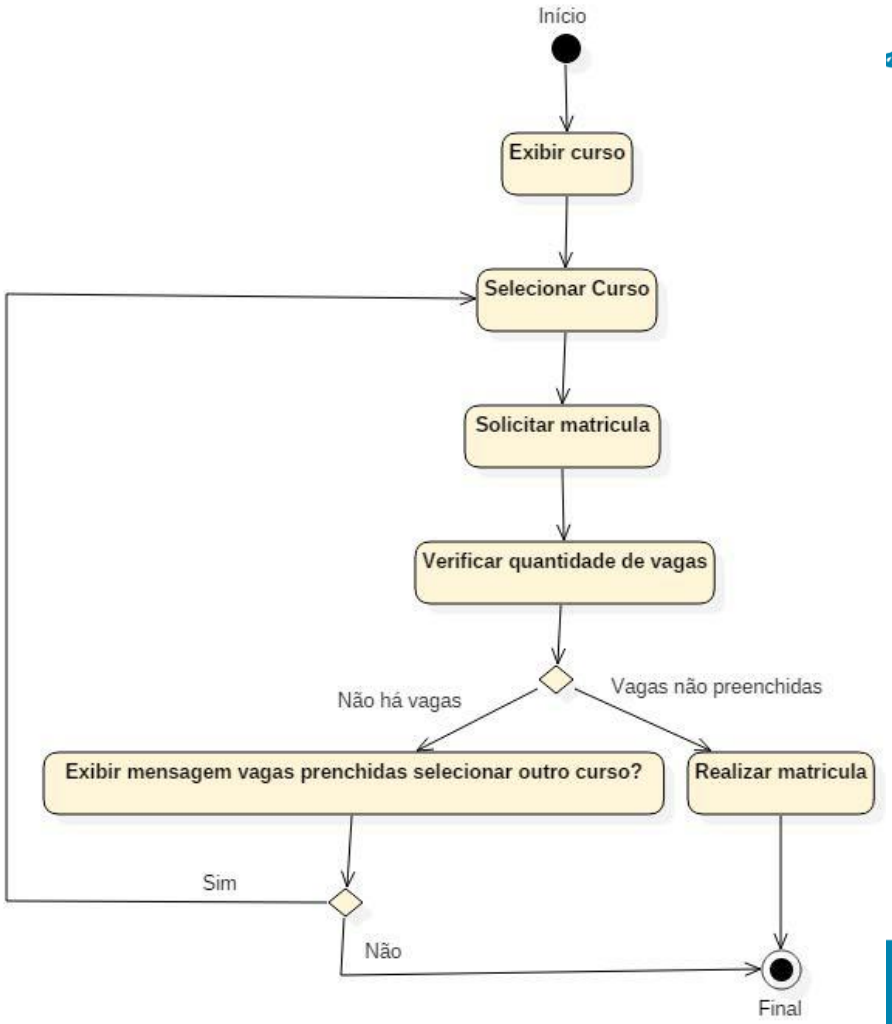
- O objeto é representado em um retângulo com o nome em minúsculo, seguido de dois pontos e o nome da classe. Nele, se informa apenas o valor dos atributos e seu vínculo apenas com uma linha cheia ligando um objeto a outro.



Exemplo de diagrama de objetos em UML

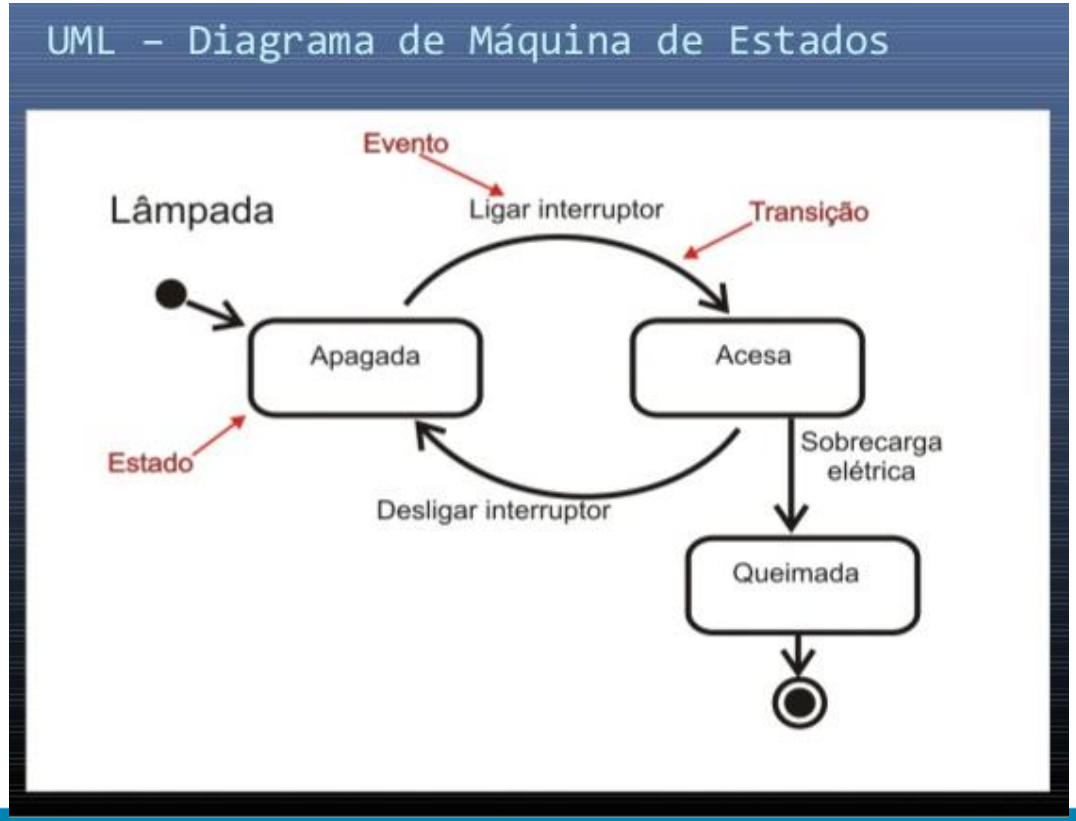
Exemplo Diagrama de Atividade

Bifurcação (*fork*) e União (*join*): é usado quando se têm fluxos concorrentes, a barra de sincronização
é empregada para especificar a bifurcação, também chamada de *fork*, e a união desses fluxos paralelos,
também chamada de *join*.



Exemplo Diagrama de máquina de estado

O estado de um objeto é a condição ou situação que o objeto assume durante a sua vida, a qual ele satisfaz a alguma condição, realiza alguma atividade ou aguarda algum evento (BOOCH; JACOBSON; RUMBAUGH, 2016).



Referências

M. Michell, Complexity: **A guided tour**, Oxford University Press, 2009.
PFLEEGER, S. L. **Engenharia de software**: teoria e prática. Belo Horizonte: Prentice Hall, 2004.
PRESSMAN, Roger S. MAXIM, Bruce R. **Engenharia de Software - Uma Abordagem Profissional**. 8.ed. Porto Alegre: Amgh Editora, 2016. 968p. ISBN 9788580555332.
SOMMERVILLE, Ian. **Engenharia de software**. 8.ed. São Paulo: A. Wesley publishing company, 2010.