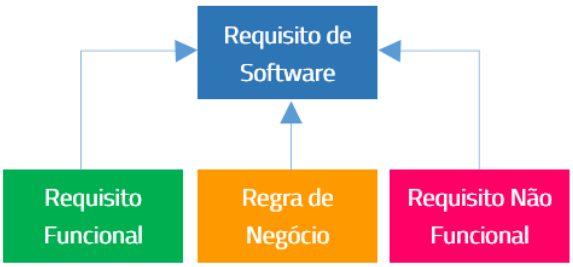
Relembrando.....

O que é um Requisito Funcional? Vamos primeiro ao que é Requisito. **Requisito é uma** exigência**, solicitação, desejo,** necessidade**.**

Quando falamos de um Requisito Funcional estamos nos referindo à requisição de uma função que um software deverá atender/realizar. Ou seja, **exigência, solicitação, desejo, necessidade, que um software deverá materializar.**

Um Requisito Funcional é um Requisito de Software.



É comum os profissionais de engenharia de software associarem a ideia de um requisito funcional a uma tela, uma rotina, que no fim serão as funcionalidades de fato de um sistema.

Mas é necessário entender que **uma funcionalidade não necessariamente realizará apenas um Requisito Funcional.**

Para entender melhor isso vamos a um exemplo mais básico. Imaginemos um sistema que possui uma tela para **"Manter Cliente"**, que mantém os dados cadastrais de um cliente no sistema.

Estamos falando de uma **única funcionalidade.** Nesta tela é possível **incluir/alterar/consultar/excluir** clientes dos tipos **pessoa física** e **pessoa jurídica**.

Mas quantos requisitos são realizados (atendidos) por esta funcionalidade? **Oito requisitos.**

Vejamos a lista a seguir:

Requisitos Funcionais (Identificador e Nome)
RF01 – Incluir cliente pessoa física
RF02 – Alterar cliente pessoa física
RF03 – Consultar cliente pessoa física
RF04 – Excluir cliente pessoa física
RF05 – Incluir cliente pessoa jurídica
RF06 – Alterar cliente pessoa jurídica
RF07 – Consultar cliente pessoa jurídica
RF08 – Excluir cliente pessoa jurídica

Esse exercício já foi desenvolvido nas primeiras atividades em aula (em 06/04)!

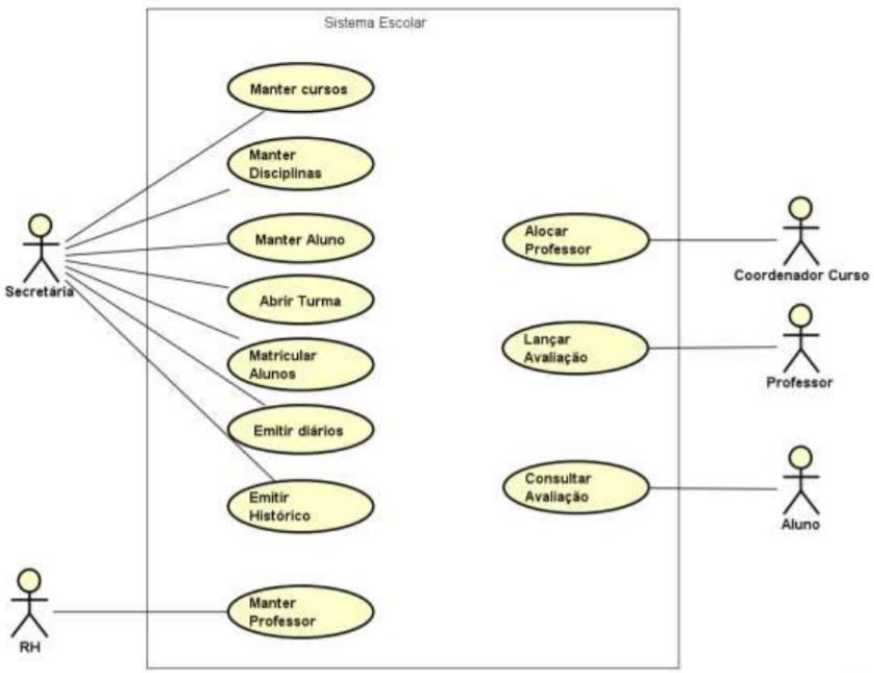
Vamos continuar esse desenvolvimento...

Sistema de Controle Acadêmico

Requisitos Funcionais

RF01 – o sistema deve permitir à secretaria cadastrar cursos contendo código, descrição e coordenador.
RF02 – o sistema deve permitir à secretaria cadastrar disciplinas de cursos, contendo código, descrição, carga horária, ementa, bibliografia e pré-requisitos.
RF03 – o sistema deve permitir à secretaria cadastrar alunos, contendo matrícula, nome, endereço, telefone e curso para o qual foi aprovado.
RF04 – o sistema deve permitir ao departamento de recursos humanos (RH) cadastrar professores, contendo nome, endereço, telefone e titulação máxima (graduação, especialização, mestrado, doutorado) e cursos que esteja vinculado
RF05 – o sistema deve permitir à secretaria abrir turmas de disciplinas de cursos, informando ano e semestre, dias da semana e horários de realização.
RF06 – o sistema deve permitir aos coordenadores de curso alocar professores a determinadas turmas.
RF07 – o sistema deve permitir à secretaria matricular alunos em turmas.
RF08 – o sistema deve permitir aos professores lançar avaliações (duas notas parciais, nota da prova final e frequência) dos alunos das turmas que estejam sob sua responsabilidade.
RF09 – o sistema deve permitir aos alunos consultar suas avaliações.
RF10 – o sistema deve permitir à secretaria emitir diários de classe das turmas.
RF11 – o sistema deve permitir à secretaria emitir históricos escolares dos alunos.
RF12 – o sistema deve efetuar o cálculo da aprovação de alunos em turmas, sendo que, para ser aprovado, deve-se ter frequência mínima de 75%. Além disso, para aprovação sem prova final, a média das notas parciais deve ser maior ou igual a 70. para reprovação direta, esta média deve ser menor que 30. médias entre 30 (inclusive) e 70 (exclusive) colocam o aluno em prova final. Se a média da prova final com a média anterior for menor que 50, o aluno está reprovado, caso contrário, aprovado.
RF13 – o sistema deve controlar a situação de um aluno, podendo estar matriculado, trancado, formado ou evadido.

Diagrama de Caso de Uso



Relembrando....

O que são Classes? Na lista de uma das várias definições que o Google nos dá achei uma interessante: *"grupo ou coleção de coisas que se distinguem das outras pela natureza, uso etc."*.

Considerando a realidade onde o conceito de Classes surgiu, no contexto de produção software, podemos entender que é uma Classe é uma **abstração de um objeto da vida real** (vida real que será tratada via software), que agrupa **dados** (atributos) e **procedimentos** (operações) relacionados ao seu contexto.

Classes x Objetos

Uma classe num Diagrama de Classes (ou até mesmo no código fonte) é apenas um **conceito**. Um conceito em forma de desenho (se num diagrama) ou texto (se em código fonte).

O diagrama de classes ilustra **graficamente** como será a **estrutura** do software (em nível micro ou macro – veremos adiante sobre as possibilidades de uso do diagrama), e como cada um dos componentes da sua estrutura **estarão interligados**.

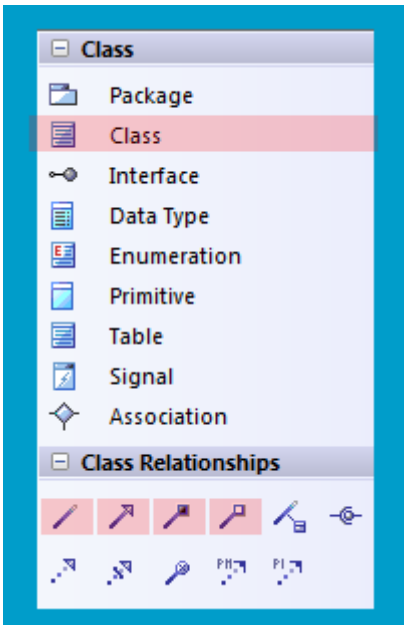
Como usar?

Na UML temos três conceitos necessários de entender: **diagramas**, **elementos** e **relacionamentos**.

As formas gráficas que compõe cada diagrama são chamadas "**elementos**". Estes elementos são "o grande lance" da UML, é o que sustenta a ideia de "notação", é a sintaxe contida nos diagramas.

Cada elemento possui um objetivo específico, e a combinação destes elementos torna-se o diagrama, que gera a semântica do respectivo modelo.

Como tudo na vida, na UML também aplica-se o Princípio de Pareto. Com 20% dos elementos fazemos 80% dos diagramas. Então, vou focar nos elementos mais utilizados do diagrama de classes.



- **Class** (Classe) – É a classe propriamente dita. Usamos este **elemento** quando queremos demostrar visualmente a classe no diagrama (exemplos mais à frente).

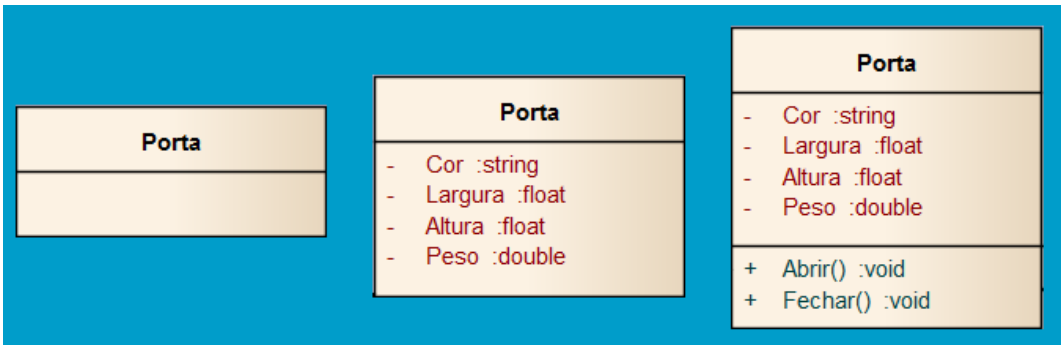
- **Association** (Associação – conector sem pontas) – É um tipo de relacionamento usado entre classes. Aplicável a classes que **são independentes** (vivem sem dependência umas das outras), mas que em algum momento no ciclo de vida do software (enquanto ele está em execução) podem ter alguma relação conceitual.

- **Generalization** (Herança – conector com seta em uma das pontas) – É um tipo de relacionamento onde a classe generalizada (onde a "ponta da seta" do conector fica) fornece recursos para a classe especializada (herdeira). Excetuando conceitos mais avançados (como padrões de projeto, interfaces, visibilidades específicas etc.), **tudo que a classe mãe (generalizada) tem, a filha (especializada) terá**.

- **Compose** (Composição – conector com um "diamante" hachurado na ponta) – É um tipo de relacionamento onde a classe composta **depende de outras classes para "existir"**. Por exemplo, a classe "CorpoHumano" possui um composição com a classe "Coracao". Sem a classe "Coracao", a classe "CorpoHumano" não pode existir.

- **Aggregate** (Agregação – conector com um "diamante" vazado na ponta) – É um tipo de relacionamento onde a classe agregada **usa outra classes para "existir", mas pode viver sem ela**. Por exemplo, a classe "CorpoHumano" possui uma agregação com a classe "Mao". Sem a "Mao" a classe "CorpoHumano" pode existir.

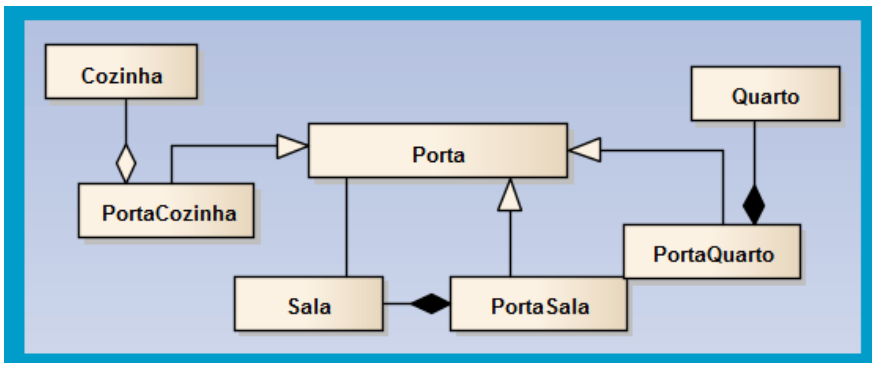
Exemplo de Uso



Uma classe, na UML (e na Orientação a Objetos também) possui **três compartimentos**, sendo para: **Nome** (primeiro), **Atributos** (segundo) e **Operações** (terceiro).

Quando fazemos o uso de um diagrama de classes no dia a dia da produção de software, nem sempre é necessário/relevante representar cada classe no menor nível de detalhe , ou seja, com os três compartimentos, e com todo rigor nas especificações dos atributos e operações.

Abaixo temos três exemplos de um mesmo diagrama, em níveis de detalhes diferentes. No último exemplo vamos ver detalhes de cada classe e seus relacionamentos.

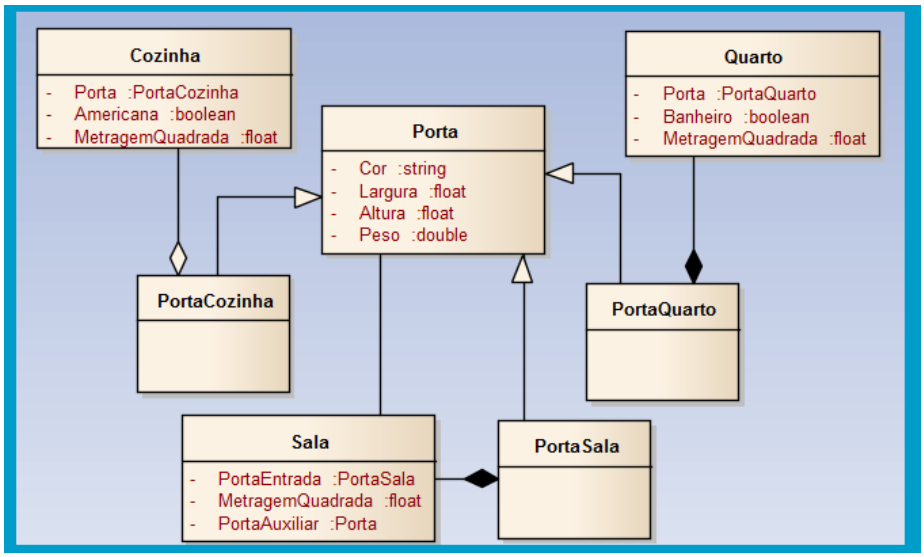


No diagrama cima temos relacionamentos de Associação, Agregação, Composição e Generalização (Herança). **A explicação a seguir aplica-se a todos os três exemplos**, pois foca apenas nos relacionamentos:

- Cozinha **pode ter ou não** uma PortaCozinha, **podendo existir** se não tiver. (Agregação)
- PortaCozinha generaliza Porta, **possuindo todas** as características que Porta têm, além das suas específicas. (Generalização)
- Quarto **deve ter** PortaQuarto, **não podendo existir** se não tiver. (Composição)
- PortaQuarto generaliza Porta, **que tem** todas as características que Porta têm, além das suas específicas. (Generalização)
- Sala **deve ter** PortaSala, **não podendo existir** se não tiver. (Composição)
- PortaSala generaliza Porta, **que tem** todas as características que Porta têm, além das suas específicas. (Generalização)
- Sala **pode ter ou não** uma Porta que não seja uma PortaSala, mas se tiver ou não isso não fará diferença, pois Porta **pode existir sem** Sala, e Sala **pode existir sem** Porta. (Associação).

(no contexto do diagrama anterior) este tipo de representação acima é muito usado/recomendado quando:

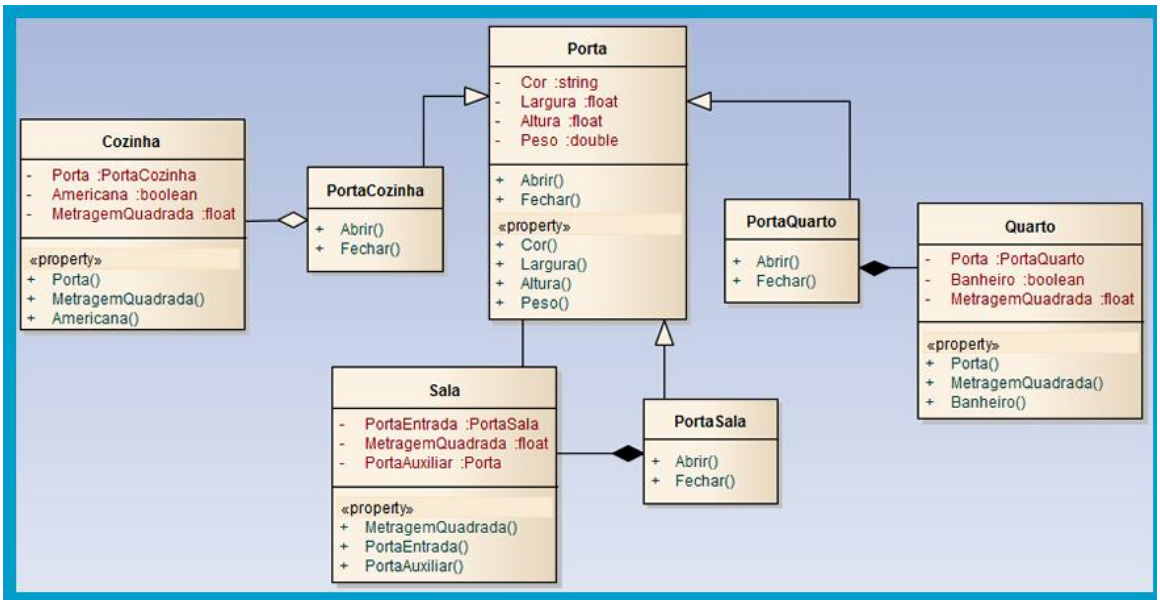
- Uma equipe está discutindo um problema e algum profissional quer esboçar (esboço = rascunho, “rabisco”) como pensa na solução, em termos de arquitetura.
- Um profissional quer mostrar apenas as dependências entre as classes do sistema, para uma análise de impacto ou contextualização da arquitetura.
- Não há necessidade de entrar em maiores detalhes sobre as classes, apenas identificá-las e ilustrar suas relações.



O diagrama acima já é um pouco diferente do primeiro, pois além dos compartimentos com os nomes das classes, possui também um outro compartimento contendo os **atributos da classe** (as classes sem atributos são classes “filhas” de outras [generalização], que portanto, implicitamente possuem todos os atributos que a classe “mãe” possui).

Este tipo de representação acima é muito usado/recomendado quando:

- O objetivo é demonstrar as classes, seus relacionamentos, seus atributos e não há necessidade de detalhar as operações da classe.
- O profissional precisa (ou entende ser necessário) dar mais contexto às classes, detalhando seus atributos, para que se compreenda melhor o escopo de cada classe do modelo e como isso compõe o entendimento sobre as relações entre as classes.



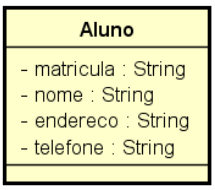
O diagrama acima já é bem diferente do primeiro e do segundo, pois além dos compartimentos com os nomes das classes e atributos, possui também um outro compartimento contendo as **operações da classe**.
Este tipo de representação acima é muito usado/recomendado quando:

- O objetivo é demonstrar as classes, seus relacionamentos, e cada classe com seu escopo completo.
- Quando a empresa realiza projeto formal do software, utilizando ferramentas case, modelos de classes que serão utilizados para outra empresa (ou a mesma até) para entendimento sobre o software a ser construído.
- Muito cobrado em empresas que prestam serviço para órgãos públicos através de licitação.
- O profissional precisa (ou entende ser necessário) dar 100% de contexto às classes, detalhando seus atributos e suas operações, para que se compreenda melhor o escopo de cada classe do modelo e como isso compõe o entendimento sobre as relações entre as classes.

https://www.ateomomento.com.br/uml-diagrama-de-classes/

ATIVIDADES

01. Baseado nas definições dos Requisitos Funcionais, desenvolva o Diagrama de Classes para o Sistema Acadêmico (Utilize o nome da Classe e os atributos para o desenvolvimento do diagramas)Não esqueça de realizar a ligação (associação) entre as classes.
Ex.



02.Leia o minimundo abaixo e desenvolva o diagrama de classes:

Empresa de Taxi Aéreo

Em uma empresa de taxi aéreo deseja-se armazenar informações de possíveis clientes: uma pessoa que possui Nome, RG, CPF, Endereço (Logradouro, CEP, Número), Telefone(s) (Celular(es), Residencial ou Comercial) pode ser um Cliente ou um Funcionário. O primeiro possui data de cadastro, e pode indicar a empresa para outro Cliente. Um Funcionário tem um Salário e um a Data de contratação. Este pode ser tanto um Piloto, com seu respectivo Brevê, quando um Vendedor, que pode dar descontos. Um Veículo possui Modelo, Autonomia, Capacidade, Velocidade, Número de Chassi e Tipo. Quando um Cliente negocia com um Vendedor, pode ser gerado um Contrato que contém Protocolo e Valor. Quando tal Contrato é efetivado, ocorre o agendamento do Voo, que possui ID do Protocolo, Data agendada, Duração, Número de voo e os Aeroportos de origem e destino. O Voo é realizado quando um Piloto pilota um Veículo, e então é registrado o Horário em que o Veículo partiu. A Negociação possui ID e Data.