Como realizar um diagrama de classes

Classes e objetos: As classes são representadas em um diagrama como apenas um conceito, em forma de desenho no diagrama ou em forma de texto se for usado em código fonte. Quando essas classes dentro de um software são materializadas, ela tornar-se objeto.

O diagrama de classes ilustra graficamente de como vai ser uma estrutura de software, utilizando isso, será interligado com os componentes de sua estrutura.

UML e Agilidade: Como o nome diz parecer ser, Agilidade não significar uma produção de software mais apressada e sim com mais velocidade, considerando também a entrega do produto em menor tempo possível. Quando se fala em Agilidade também é considerado a eficiência e uma boa comunicação, sendo assim, se um desenvolvedor entende de uma maneira diferente os requisitos, o custo para consertar os erros serão grandes, fora o tempo perdido.

O uso dos diagramas UML é uma ferramenta bastante favorável para a cultura ágil, tendo visto o objetivo de transmissão de ideia entre os membros da mesma equipe.

Objetivos: O diagrama de classes especifica componentes do software e interligam sua estrutura. É importante entender como funciona as caixas ligadas com as setas, para poder compreender um modelo de classes com mais eficiência.

Na produção do software, pode ser considerado as seguintes aplicações do diagrama para diferentes finalidades:

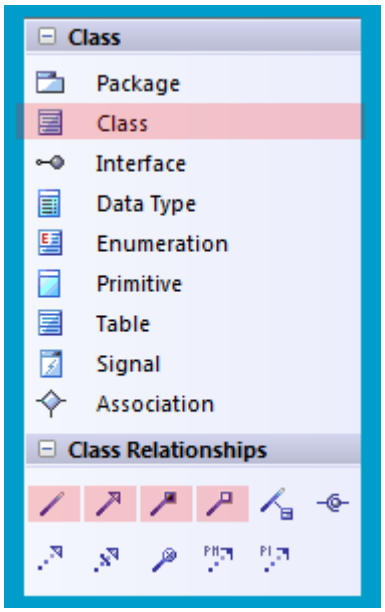
- **Design**: Irá definir um modelo para ser seguido, sendo assim, esse modelo será em um software que ainda não existe, porém será específico para um modelo antes da construção de um software executável.

- **Engenharia reversa**: Para a produção do diagrama, deverá analisar uma estrutura já existente de um software, ou seja, a partir dessa estrutura será feita uma leitura das classes e suas relações.

- **Esboço de ideias**: Sem precisar necessariamente de uma ferramenta case, um desenho de modelos estruturais para o compartilhamento de ideias e alinhamento entre analistas de sistema.

Seu Uso: Existe três tipos de conceitos que são necessários na UML para poder entender: Diagramas, elementos e relacionamentos.

Na parte de elementos, são onde fica as formas gráficas que compõe cada diagrama, sendo assim são denominadas como Elementos, é onde são contidas as sintaxes nos diagramas. no uso dos elementos, cada um deles tem um objetivo específico, combinando cada um deles, acaba formando o diagrama.



- **Class**: É a classe, ela é utilizada quando é para demonstrar visualmente a classe no diagrama.

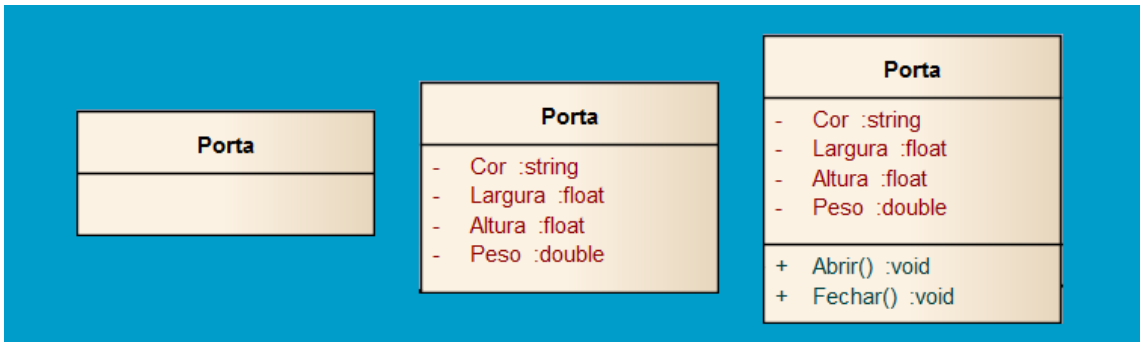
- **Association**: É um modelo de relacionamento usado entre classes. Aplicando em classes que são independentes, mas terá momentos em que vão poder ter uma relação conceitual.

- Generalization: É um tipo de relacionamento onde a classe generalizada fornece recursos para a classe especializada.

- **Compose**: É um tipo de relacionamento onde a classe composta depende de outras classes para existir (por assim dizer). Por exemplo, para a classe A existir é necessário que a classe B exista para suprir as necessidades da classe A.

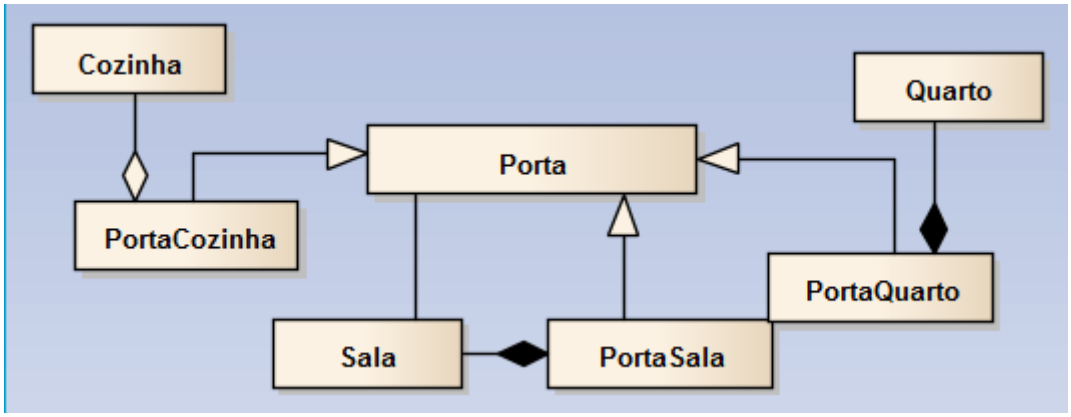
- **Aggregate**: É um relacionamento onde a classe agregada usa outra classe para existir, entretanto, pode viver sem ela. Por Exemplo uma Classe A necessita da Classe B para existir e utiliza a Classe C, sem a Classe C a Classe A continua existindo.

Utilizando:



Uma classe, na UML possui três compartimentos, sendo: Nome (que é o primeiro), Atributos(segundo) e Operações(terceiro).

Uma observação importante é que em um diagrama de classes, nem sempre será necessário apresentar os menores detalhes.



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)**

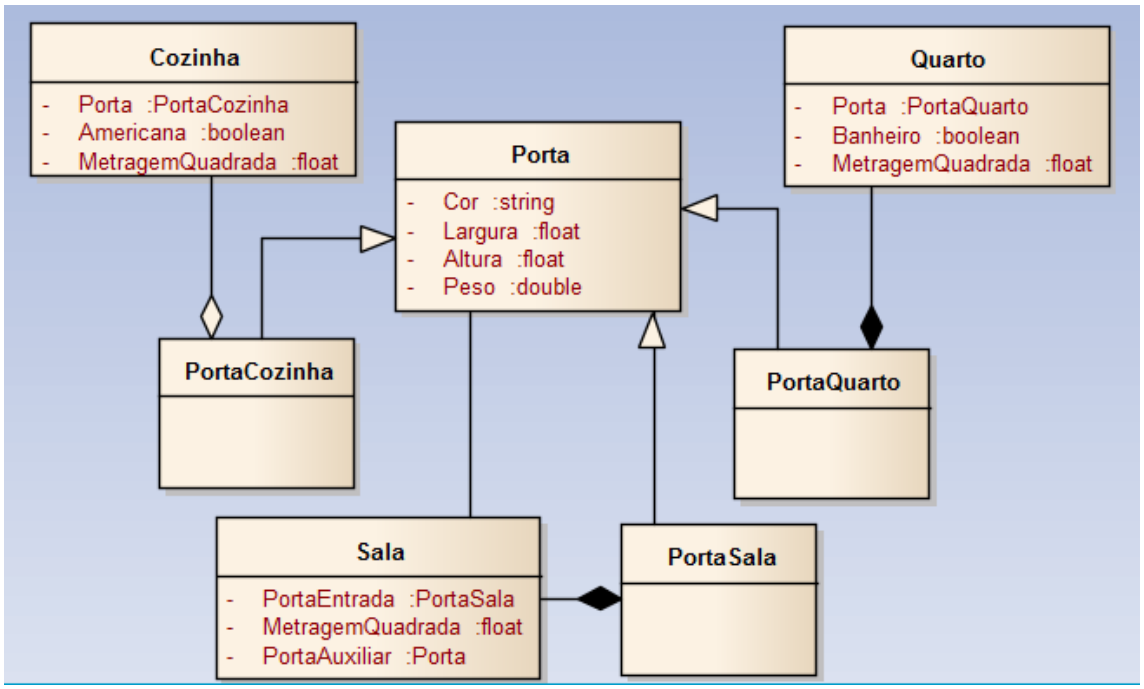
- **PortaQuarto** generaliza **Porta**, que tem todas as características que **Porta** têm, além das suas específicas. **(Generalização)**

- **Sala** deve ter **PortaSala**, não podendo existir se não tiver. **(Composição)**

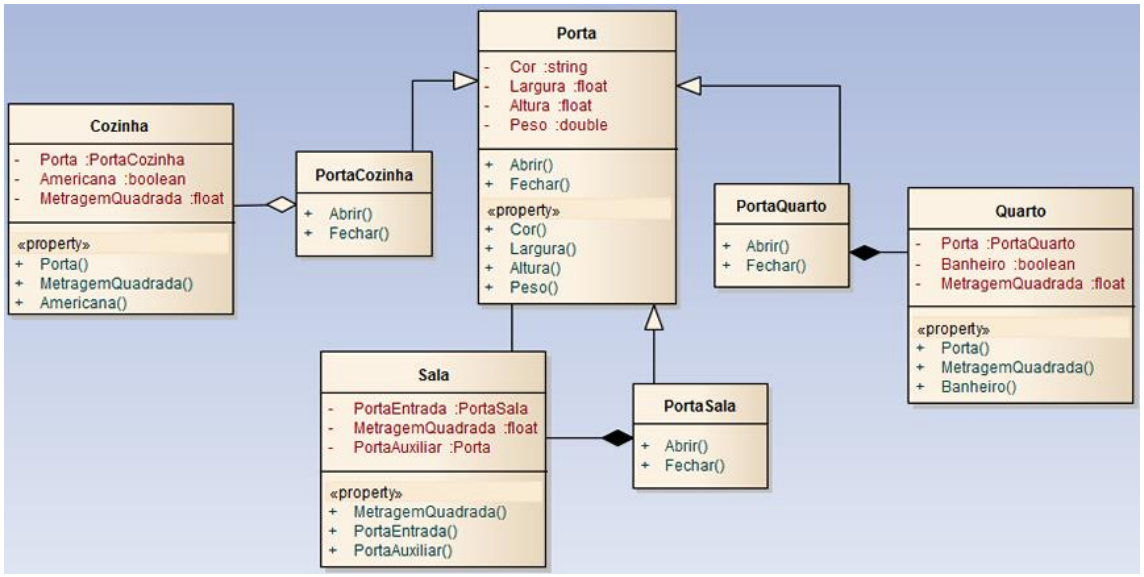
- **PortaSala** generaliza **Porta**, que tem todas as características que **Porta** têm, além das suas específicas. **(Generalização)**

- **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)**.

Obs: Esse tipo de representação é bastante comum entre uma equipe que está discutindo um problema e algum profissional quiser fazer um rascunho, também um desenvolvedor querer mostrar apenas as dependências entre as classes do sistema, para uma análise.



A demonstração deste diagrama consiste muito no compartimento de **atributos de classe**, onde é bastante usado quando: O objetivo é demonstrar classes, seus relacionamentos e seus atributos. Também o desenvolvedor necessita de dar mais contextos as classes, sendo assim detalhando seus atributos.



Neste último exemplo, esse diagrama possui o compartimento de **operações da classe**, é utilizado quando:

- Demonstração de classes e seus relacionamentos;

- Quando a empresa demonstra o projeto formal do software, utilizando as ferramentas case.

- Nesse compartimento, o profissional precisa dar 100% de contexto as classes, sendo assim, detalhando seus atributos e suas operações para a melhor compreensão do escopo de cada classe.