**Engenharia de Requisitos de Software**

**UNIDADE 1 - TIPOS DE REQUISITOS E ELICITAÇÃO DE REQUISITOS DE SOFTWARE**

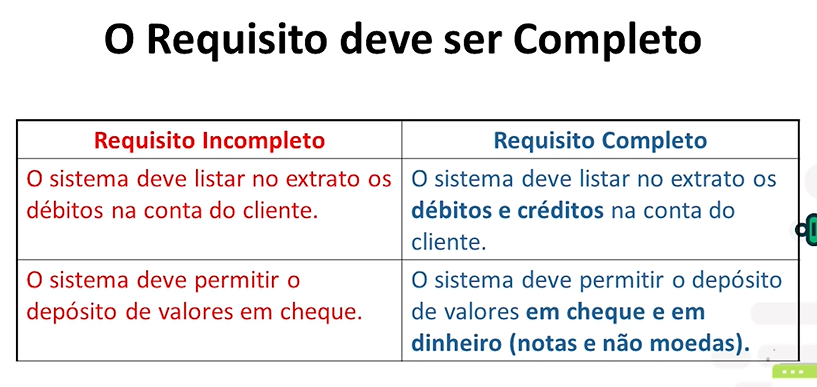
**Tema 1: Características e Tipos de Requisitos**

**CONCEITOS E CARACTERÍSTICAS DE REQUISITOS**

O **Requisito**é uma característica que um produto deve possuir para que seja aceito.

A **Engenharia de Requisitos de Software**é um conjunto de técnicas empregadas para levantar, detalhar, documentar e validar os requisitos de um produto.

O Requisito deve ser  **Completo,**descrevendo integralmente a necessidade a ser atendida. O Requisito deve ser **Correto**, descrevendo uma necessidade exata e associada ao software em desenvolvimento. O Requisito deve ser **Exequível,**isto é, ele deve ser possível de ser executado.



O Requisito deve ser **Priorizado** entre os níveis obrigatório, desejável ou opcional. Deve ser possível alterar o Requisito, portanto, ele deve ser **Modificável.**O Requisito deve ser **Preciso,**isto é, sua redação deve ser clara e livre de ambiguidade. Deve-se conseguir testar o Requisito, logo o requisito deve ser **Verificável.**Deve-se conseguir associar o Requisito ao código e vice-versa, garantindo que o Requisito seja **Rastreável.**

**REQUISITOS FUNCIONAIS**

O **requisito funcional** está diretamente ligado à funcionalidade do software.

Ele descreve a função que o produto deverá realizar em benefício do usuário.

Ele tem o Foco no Negócio e não em tecnologia.

Nos requisitos funcionais, deve-se imaginar uma tecnologia perfeita, esquecendo um pouco dos detalhes técnicos e restrições tecnológicas para a implementação, pois isto é assunto para requisitos não-funcionais.

**REQUISITOS NÃO-FUNCIONAIS**

O **requisito não-funcional** tem foco na Tecnologia, nas restrições da plataforma do software a ser desenvolvido.

Os Requisitos não-funcionais do produto especificam ou restringem o comportamento do software. Exemplos incluem os requisitos de desempenho que especificam a rapidez com que o sistema deve executar e quanta memória ele requer. Temos também os requisitos de confiabilidade que estabelecem a taxa aceitável de falhas. Temos ainda os requisitos de proteção e os requisitos de usabilidade.

Esta classificação de requisitos não-funcionais pode variar entre diferentes autores de Engenharia de Software. Os Requisitos Não-Funcionais são muito importantes para que o software esteja adequado à plataforma tecnológica com bom desempenho e confiabilidade.

**Tema 2: Elicitação de Requisitos de Software**

**PROCESSOS DA ENGENHARIA DE REQUISITOS E TÉCNICAS DE ELICITAÇÃO**

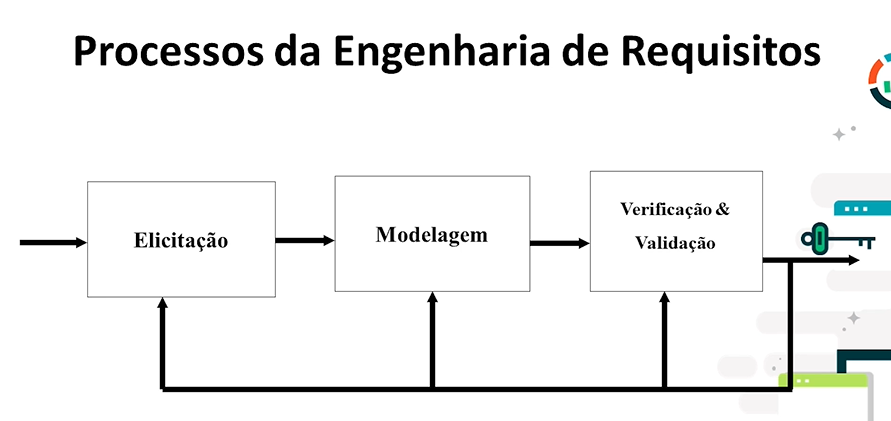
Elicitar quer dizer explicitar, extrair, obter o máximo de informação para o conhecimento do objeto em questão. As nossas fontes de informação podem ser clientes, usuários, documentos, normas, e até mesmo outros sistemas de informação que vão ter interface com o sistema que se pretende desenvolver.

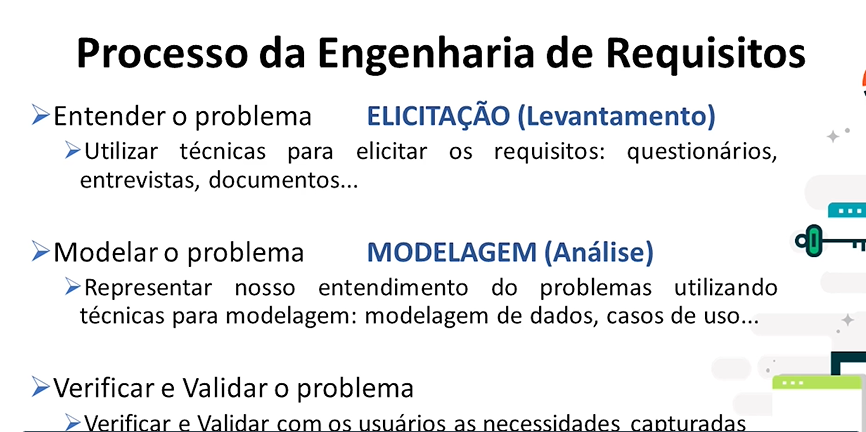
Em seguida, estão detalhadas várias técnicas de **Elicitação de Requisitos**.

A **entrevista** deve ser marcada com antecedência e você deve preparar um roteiro com as principais questões. A entrevista pode ser individual ou em grupo. Uma das vantagens da entrevista é que você pode conseguir motivar os entrevistados ao longo do processo.  Você pode também incluir, retirar ou alterar a ordem das perguntas de acordo com o ritmo das entrevistas. Por outro lado, você tem de tomar cuidado para a entrevista não virar uma reunião longa e chata, deixando os entrevistados desatentos.

Já no **Brainstorming,**não há bem um roteiro prévio e as partes interessadas apresentam livremente sugestões e requisitos para  o sistema a ser desenvolvido.

Você deve atuar como um facilitador do processo, buscando diversidade de ideias.





O **Questionário** pode ter só perguntas fechadas ou pode ter também algumas perguntas abertas. Geralmente, a maioria é fechada para permitir um maior alcance e facilitar o registro dos resultados. A vantagem do questionário é a praticidade, pois você encaminha para os seus usuários e combina um prazo comum de resposta.

O **protótipo** é muito indicado para aquele usuário que tem dificuldade de expressar o que deseja, então ele precisa ver um esboço de como vão ficar as telas para expressar melhor suas necessidades. Você pode construir este protótipo com um formulário HTML, slides ou até mesmo com a linguagem em que vai desenvolver o sistema.

**Étnografia** é uma observação da realidade em que você passa um período observando como os usuários trabalham ou como os clientes fazem as compras.

**Estudo de Documentos**é quando se tem que interpretar quais são os requisitos derivados destes documentos.

**UNIDADE 2  - MODELAGEM, VALIDAÇÃO E VERIFICAÇÃO DE REQUISITOS DE SOFTWARE**

**Tema 1: Modelagem de Requisitos**

**MODELAGEM DE REQUISITOS A INTRODUÇÃO À UML**

A Modelagem de Requisitos também é chamada de Análise de Requisitos por alguns autores. Um dos produtos da elicitação de requisitos é uma lista de requisitos funcionais e não-funcionais. Na etapa da Modelagem de Requisitos, usa-se a notação UML da Análise Orientada a Objeto para detalhar os requisitos, desenhando figuras que são modelos representativos do software.

 A sigla da UML quer dizer Unified Modeling Language, ou seja, linguagem de modelagem unificada. A letra L quer dizer que a UML é uma linguagem, uma notação, um jeito padrão para construir modelos, para especificar software baseado nos conceitos da Orientação a Objetos. Já a letra M quer dizer Modelagem. A letra U quer dizer que a UML é uma notação unificada, um jeito universal de desenhar os modelos.

A UML surgiu em 1997 como foi resultante de uma iniciativa de várias multinacionais na área de TI lideradas por 3 pesquisadores e consultores na área de engenharia de software que são considerados os pais da UML. A especificação completa da UML é encontrada em [www.uml.org (Links para um site externo.)](http://www.uml.org/) .

É importante lembrar que a UML não é uma metodologia nem um processo de software. A UML não diz  como e nem em qual  ordem fazer.

A UML possui 3 tipos de diagramas:

* Diagramas de Estrutura,
* Diagramas de Comportamento, (O Diagrama de Casos de Uso UML se enquadra em qual categoria de Diagramas UML?)
* Diagramas de Interação.

Explicação de cada uma:

Os diagramas de Estrutura - enfatizam a visão estática. (Dentro dele existem dois diagramas de estrutura: o diagrama de classes e o diagrama de pacotes.)

Os diagramas de Comportamento - enfatizam a visão dinâmica. Neste você vai conhecer diagrama de caso de uso, que é o mais usado para a modelagem de requisitos.)

Por fim, temos os diagramas de interação que representam a comunicação entre os objetos do sistema.

**MODELAGEM DE CASOS DE USO UML**

O diagrama de caso de uso é um dos mais importantes diagramas da UML. Ele é muito usado na modelagem de requisitos, pois possui uma notação fácil que auxilia o diálogo com os usuários.

O diagrama de caso de uso é usualmente o primeiro diagrama UML utilizado na fase de requisitos, sendo um diagrama de alto nível de abstração utilizado nas reuniões de levantamento com os usuários. O diagrama de caso de uso permite detalhar os Requisitos Funcionais identificados na etapa de elicitação. Não se utiliza diagramas de caso de uso para requisitos não-funcionais.

O diagrama de caso de uso também pode servir como um contrato entre as partes, pois ele delimita o escopo do sistema, isto é, o que está dentro do sistema e o que está fora do sistema, que são as interfaces do sistema que você está desenvolvendo com outros sistemas.

O diagrama de caso de uso tem 3 elementos principais: **Atores, Casos de Uso e Relacionamentos**. Os relacionamentos podem ser entre atores e caso de uso, ou entre caso de uso e caso de uso ou ainda entre atores e atores.

O **ator**representa aquelas entidades que interagem com o sistema. São exemplos de atores os usuários, clientes, funcionários, alunos, empresas, órgãos do Governo ou outros tipos de pessoas que se relacionam com o sistema. Atores também podem ser sensores ou outros sistemas, tais como sistema de contabilidade, sistema de RH.

 É importante lembrar que o próprio sistema não deve ser ator se você estiver modelando este sistema.

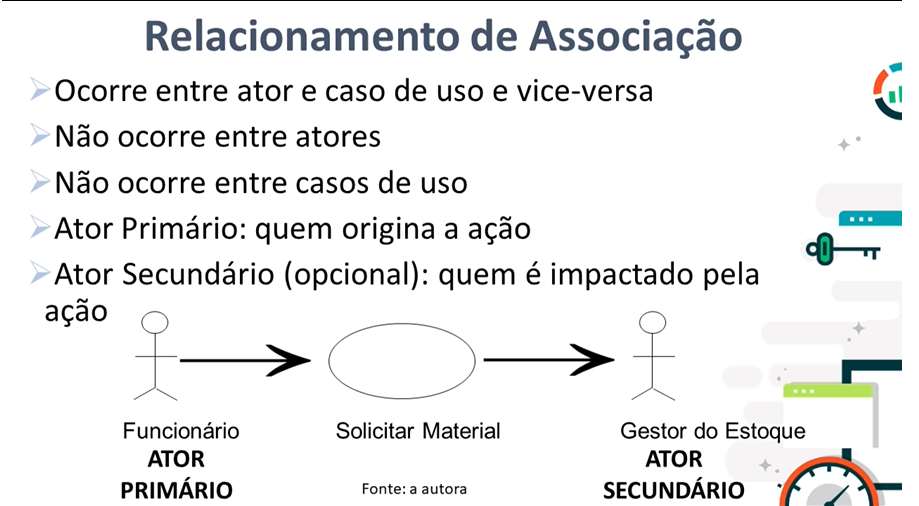
Já o **caso de uso** é uma transação realizada pelo sistema para atender a uma solicitação de um ator. Um Diagrama de Caso de uso vai conter vários casos de uso.

É importante lembrar que o diagrama de caso de uso não tem a ideia de sequência. Portanto, a ordem em que os eventos ocorrem não é indicada.

Os **relacionamentos** podem ser de **Associação, Generalização, Inclusão e Extensão**.

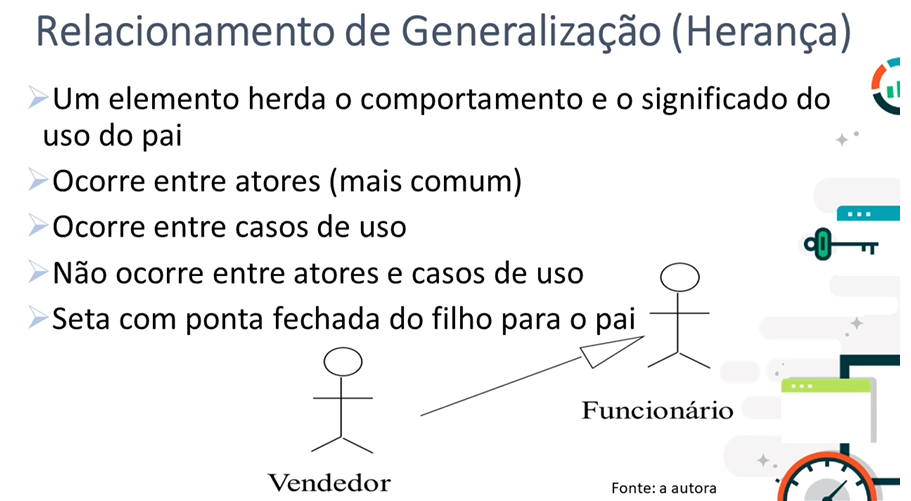
O relacionamento de **Associação** é o relacionamento mais comum representado por uma seta que liga atores a casos de uso. Esta seta com a ponta aberta não pode ser usada para ligar ator com ator nem caso de uso com caso de uso. Você pode também ligar atores com casos de uso simplesmente com uma reta, pois a UML também permite isso. No entanto, preferimos sempre usar a seta para ser mais legível na indicação do fluxo de informação.

O ator que inicia a ação do caso de uso é chamado Ator Primário. O ator que é afetado pela ação é chamado Ator Secundário. Um caso de uso pode ter zero ou vários atores secundários.



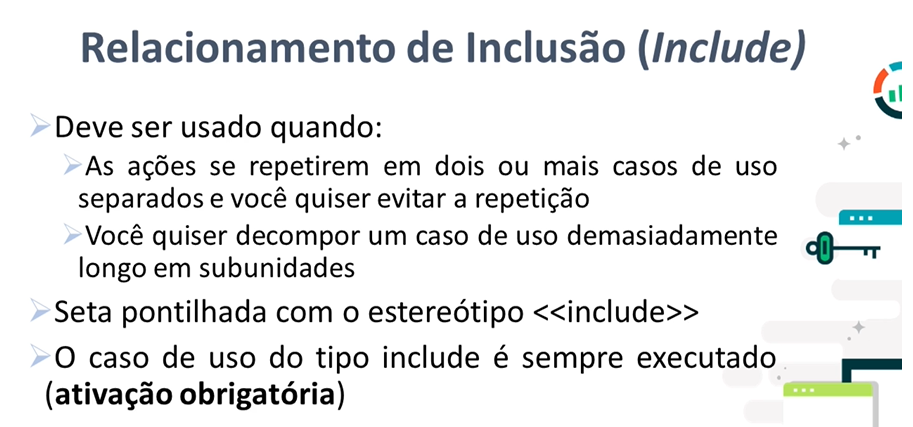
O relacionamento de **Herança**pode acontecer entre elementos iguais, ou seja, entre ator e ator e também entre caso de uso e caso de uso, mas é mais comum entre ator e ator.

A **Generalização** ocorre quando um elemento herda o comportamento do pai.

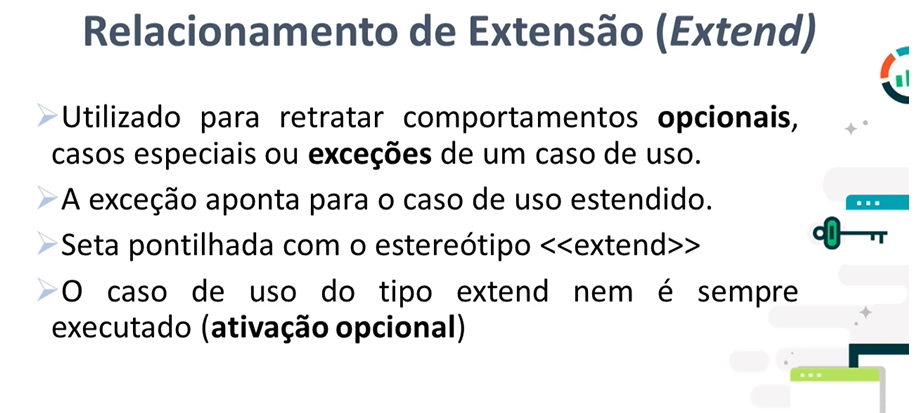


Para diferenciar da Associação, a seta é com ponta fechada na Generalização sempre no sentido do filho para o pai. Podemos ter Generalização entre casos de uso quando um caso de uso herda comportamentos de outro caso de uso.

O relacionamento de **Inclusão,** também chamado de Include que só ocorre entre casos de uso. Ele é muito usado quando temos rotinas comuns entre as várias transações do sistema. Ele é representado por uma seta pontilhada escrita include do caso de uso que o ativa para a rotina geral.



O relacionamento de **Extensão**é representado por uma seta pontilhada, mas com o estereótipo extend. Ele é usado para tratar exceções, casos especiais. Ao contrário do include que tem ativação obrigatória, no extend, a ativação é opcional.



**PRÁTICAS DE DIAGRAMA DE CASOS DE USO**

**Construa o diagrama de caso de uso UML para o seguinte sistema de seguradora.**

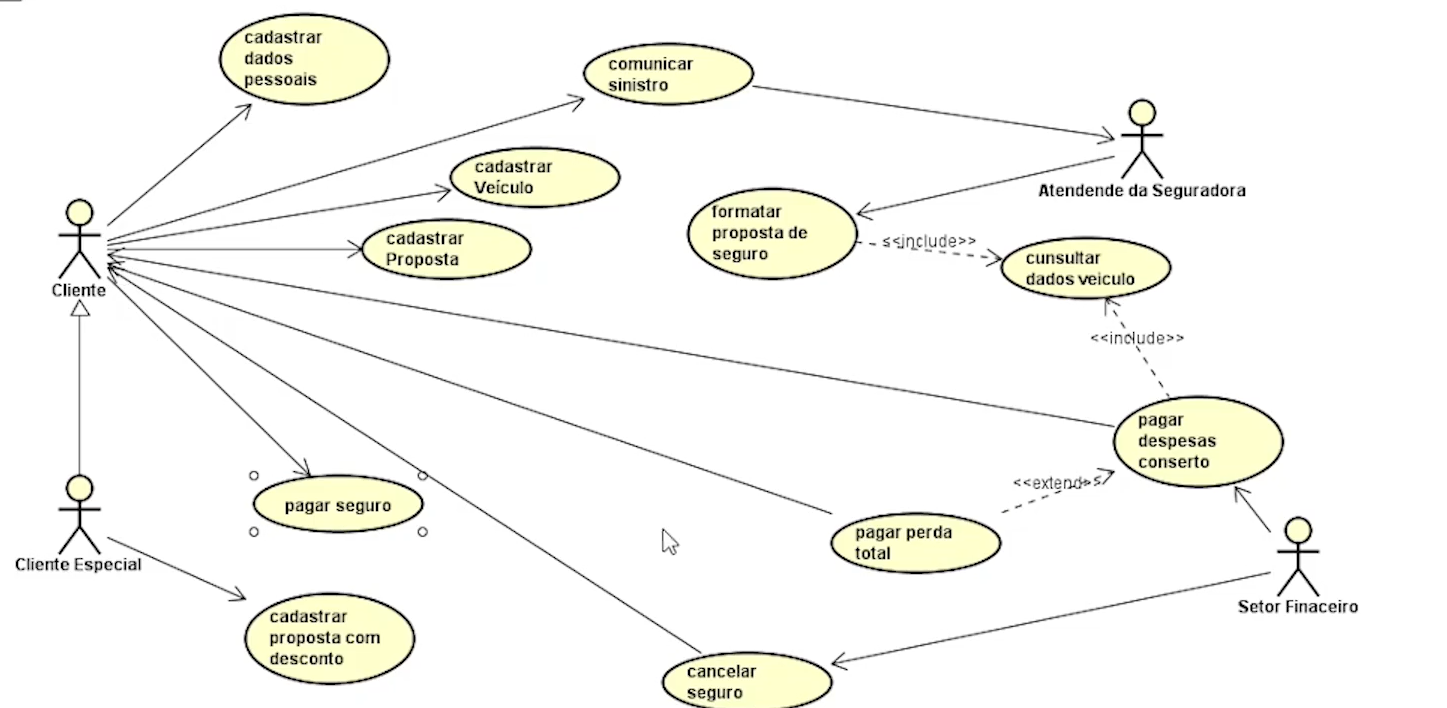
O cliente cadastra inicialmente seus dados pessoais, depois cadastra os dados do veículo e finalmente cadastra uma proposta de seguros para a Seguradora.

Alguns clientes são clientes especiais do programa de fidelidade da Seguradora e então cadastram uma proposta de seguros com descontos.

A seguradora formata a proposta de seguros, consultando para tanto o cadastro de veículos. Posteriormente, o cliente paga as prestações do seguro.

Caso algum cliente não pague, a seguradora cancela o seguro, comunicando o cliente.

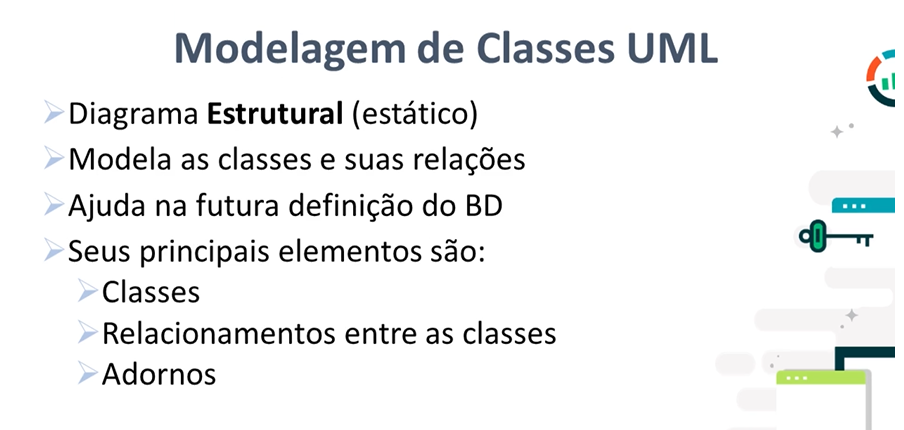
Em caso de acidente, o cliente comunica o sinistro à seguradora. A seguradora paga as despesas de conserto, consultando para tanto o cadastro de veículos. Em casos raros de sinistros mais graves, a seguradora paga ao cliente o valor da perda total do veículo.

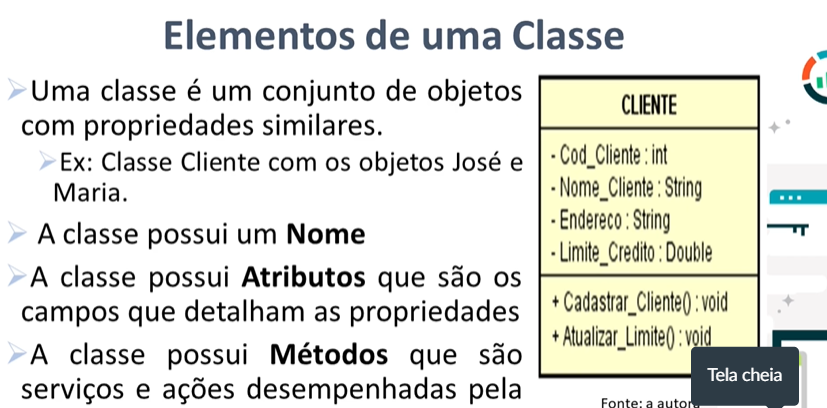
****

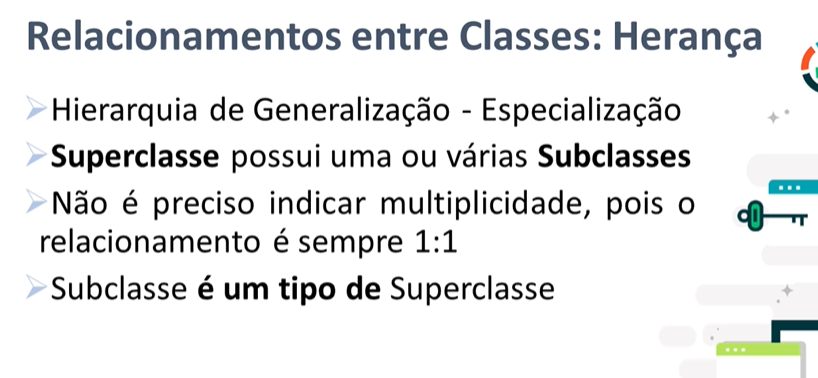
**MODELAGEM DE CLASSES UML**

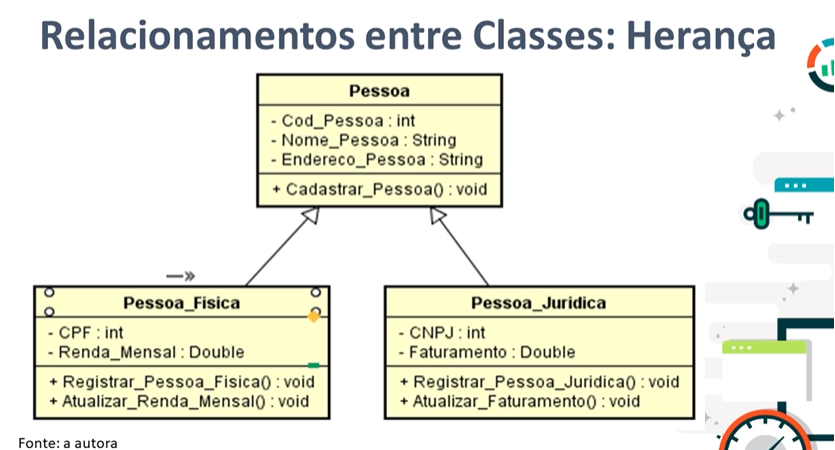
O Diagrama de Classes é um diagrama Estrutural da UML, pois ele define uma estrutura estática de relacionamentos entre as classes. Uma classe é um conjunto de objetos com propriedades similares. A classe possui um Nome. A classe possui Atributos que são os campos que detalham as propriedades. A classe possui Métodos que são serviços e ações desempenhadas pela classe.

O 1º. Tipo de ligação entre classes é a  **Herança** também chamada de Generalização-Especialização. Nesta situação, existe uma Superclasse que representa um conceito genérico e uma ou mais Subclasses (filhos) que são as especializações que herdam atributos e métodos da Superclasse pai. Neste relacionamento, existe a herança e não existe a multiplicidade porque a subclasse sempre é um tipo específico da superclasse, ou seja, o relacionamento é de 1 para 1. Neste relacionamento, usamos a mesma seta de ponta fechada indicada para representar a especialização de um caso de uso. A Subclasse (filho) sempre aponta para a Superclasse (pai).



****

****

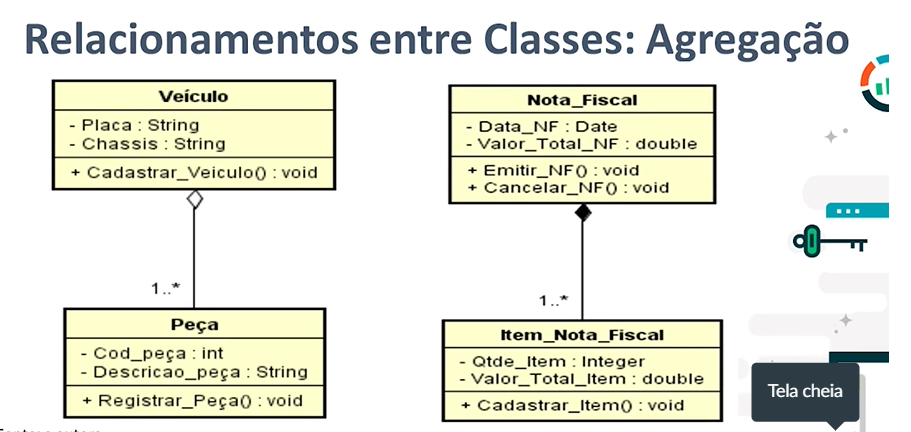
****

O 2º. Tipo de relacionamento é a **Agregação**, também conhecida como Estrutura Todo-Parte. Nesta estrutura em que um Todo possui várias Partes. Neste tipo de relacionamento, não há herança, ou seja, a parte não herda nada do todo. Mas vai existir uma relação de pertinência, pois as partes pertencem ao todo. Deve-se assim representar a Multiplicidade (1…\*). Na UML, o asterisco tem o mesmo significado do N de banco de dados, ou seja, quer dizer vários.

A agregação pode ser representada como:

Agregação Simples: losango não preenchido, pois a parte tem vida além do todo.

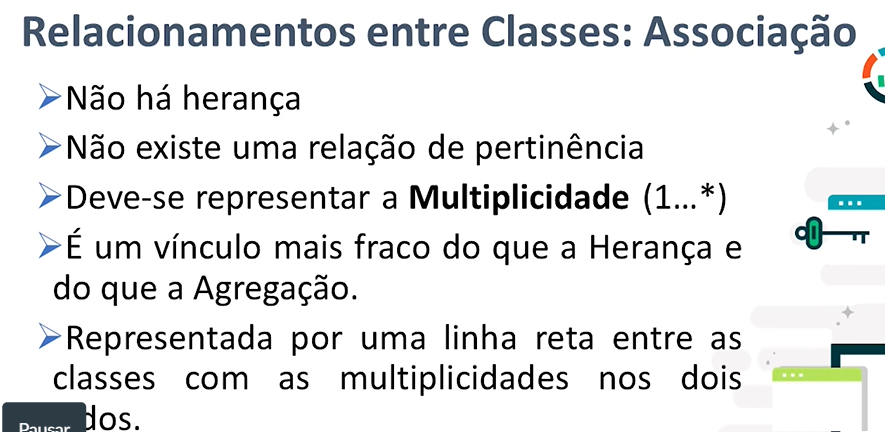
Agregação por Composição: losango preenchido, pois a parte não tem vida além do todo.

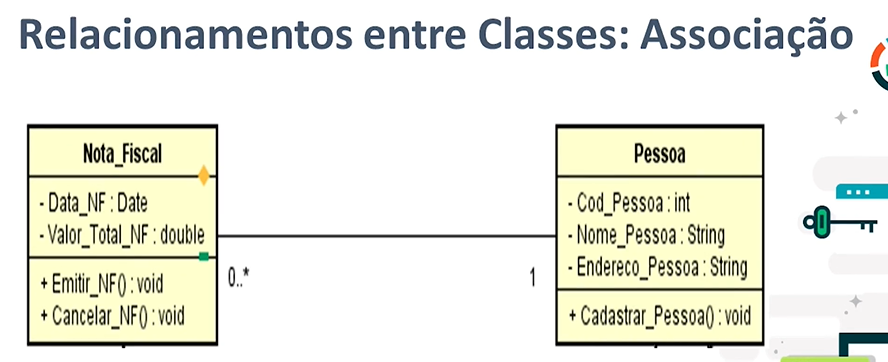


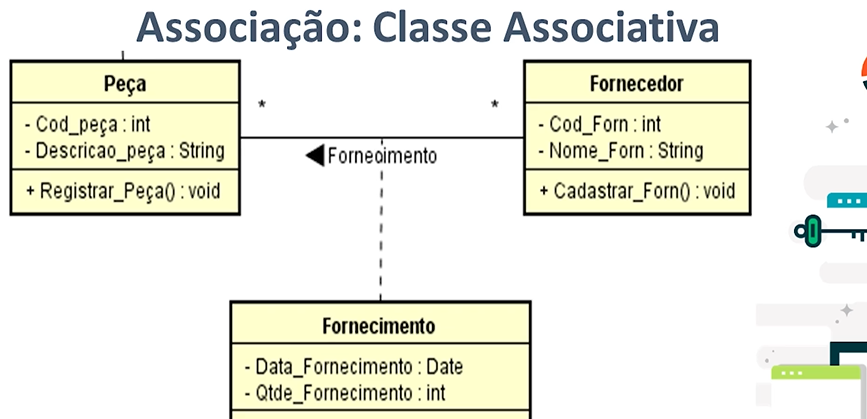
O 3º. Tipo de relacionamento entre classes que é a **Associação**. Este tipo é bem parecido com a Agregação, pois não temos a herança e temos a multiplicidade.

Na Associação, não existe esta relação de todo e parte. A ligação entre as classes é um vínculo mais fraco. A Associação é o tipo de relacionamento mais comum entre classes. Na Herança, usamos a expressão Classe A é um tipo de Classe B. Na Agregação, usamos a expressão Classe A faz parte da Classe B. Já na Associação, usamos a expressão Classe A está associada à Classe B.

Uma situação particular da Associação é quando temos uma multiplicidade de muitos para muitos entre classes. A classe associativa é representada por uma linha reta entre as classes com as multiplicidades com asterisco nos dois lados e por uma classe pendurada nesta associação.



****

****

**MODELAGEM DE PACOTES UML**

O Diagrama de Pacotes é um diagrama que deve ser feito após o término do Diagrama de Classes. Ele serve para empacotar as classes relacionadas. Tal qual o diagrama de classes, o Diagrama de Pacotes é um diagrama Estrutural (estático) da UML.

Um Pacote é um conjunto de classes relacionadas. Apesar de não ser algo exato e matemático, construir pacotes é um exercício de organização baseado no bom senso.

Para ajudar, devemos observar algumas regras em pacote.

☞ Um Pacote deve ter duas ou mais classes.

☞ Não faz sentido criar um pacote com uma única classe.

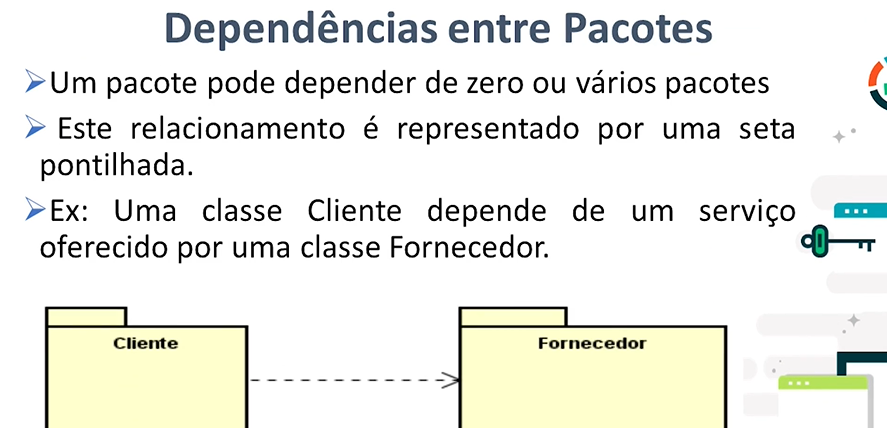
☞ O pacote tem um nome genérico ou o nome da sua classe mais importante.

☞ O pacote é representado por um ícone de Pasta.

☞ Uma classe pode pertencer a mais de um pacote se ela for muito importante.

☞ Nenhuma classe pode ficar de fora do diagrama de pacotes.

Um pacote pode depender de zero ou vários pacotes. Este relacionamento é representado por uma seta pontilhada.

****

**PRÁTICAS DE DIAGRAMA DE CLASSES**

**Construa o diagrama de classe para a seguinte aplicação de controle de uma farmácia, identificando os pacotes:**

Uma farmácia possui um nome, CNPJ e endereço. Uma farmácia possui vários funcionários. Todo funcionário tem um número de matrícula, nome, cargo e salário. A farmácia pode contratar, promover ou demitir funcionários. Os tipos de funcionários da farmácia são farmacêutico, atendente e gerente. Para o farmacêutico é preciso registrar o número do conselho profissional e para o atendente registra-se o percentual de comissão.

A farmácia vende vários tipos de produtos. Todo produto tem um código, descrição, preço unitário e quantidade em estoque.

Os produtos são vendidos para clientes através de uma nota fiscal. Uma nota fiscal é um documento oficial que formaliza um relacionamento comercial entre o cliente e a farmácia. Uma farmácia emite uma ou várias notas fiscais e um cliente possui várias notas fiscais. O cliente tem um nome e CPF.

A nota fiscal possui uma data de emissão e valor total. Um ou vários itens fazem parte de uma nota fiscal, onde cada item possui a quantidade comprada. Cada item da nota fiscal refere-se a um único produto e um produto pode estar presente em vários itens de nota fiscal.

**Tema 2: Validação e Verificação de Requisitos**

**VALIDAÇÃO E VERIFICAÇÃO DE REQUISITOS**

A **Validação** tem uma relação com a eficácia, ou seja, atingir o objetivo.

A Validação responde a seguinte pergunta: Estamos construindo o produto certo?

A Validação confere se estamos atendendo as necessidades dos clientes. A Validação está ligada com a relação entre o software e o mundo real.

Já a **Verificação** tem uma relação com a eficiência, ou seja, o processo de construção do produto. Estamos construindo o produto de maneira certa? A Verificação está associada à consistência entre modelos (Ex.: diagrama de casos de uso coerente com o diagrama de classes).

Por exemplo, se o software atende as necessidades do usuário, mas têm algumas telas com processamento lento, temos um problema de Verificação. Já se o software funciona bem, mas estão faltando funcionalidades, então temos um problema de Validação. Estes conceitos de Verificação e Validação são relacionados à Qualidade e são aplicados em outras áreas.

Vamos conhecer uma das técnicas usadas para validação de requisitos.

Uma destas técnicas é a Revisão Técnica. A revisão permite avaliar um software para determinar a sua adequação ao uso pretendido. A equipe de revisão deve ser formada por engenheiros de software, clientes, usuários e outros envolvidos. Nesta revisão, a especificação de requisitos é examinada em busca de:

* erros no conteúdo ou na interpretação,
* de informações faltantes,
* de inconsistências,
* de requisitos conflitantes ou irreais.

Uma das técnicas usadas para realizar a verificação é a Inspeção de Software. Ela tem por objetivo detectar e identificar anomalias no produto de software. São comuns as Inspeções de Código, quando parte da equipe faz um mutirão para revisar o código da aplicação, os comandos de banco de dados para identificar se tem alguma prática inadequada. Mas a inspeção pode ser aplicada em todas as fases do processo de software, até mesmo com os requisitos. Por exemplo, uma equipe pode conferir o diagrama de caso de uso produzido por outra equipe, verificando problemas de modelagem.

Por fim, convém lembrar que Verificação e Validação estão ligados com Testes de Software e com Qualidade.