Estratégia
CONCURSOS

Aul

Engenharia de Software p/ TRF 3ª Região (Técnico Jud-Técnico em Informática) - FCC

Professor: Diego Carvalho, Equipe Informática e TI

SUMÁRIO	PÁGINA

UML: VISÃO GERAL, MODELOS E DIAGRAMAS

Vamos primeiro resumir o contexto histórico! Havia uma empresa chamada Rational Software Corporation. *Sim, pessoal... é aquela mesma criadora do RUP!* Em 1995, ela conseguiu reunir três dos pesquisadores de Engenharia de Software mais proeminentes do mundo: James Rumbaugh, Grady Booch e Ivar Jacobson – conhecidos como *The Three Amigos* (imagem abaixo).

Rumbaugh era o criador da Técnica de Modelagem de Objetos (TMO). Já Booch era o criador do método de Projeto Orientado a Objetos (POO). Por fim, Jacobson era o criador do método de Engenharia de Software Orientada a Objetos (ESOO). Esses três caras trabalhavam para a Rational, mas cada um seguia seus métodos e técnicas de modelagem próprios.

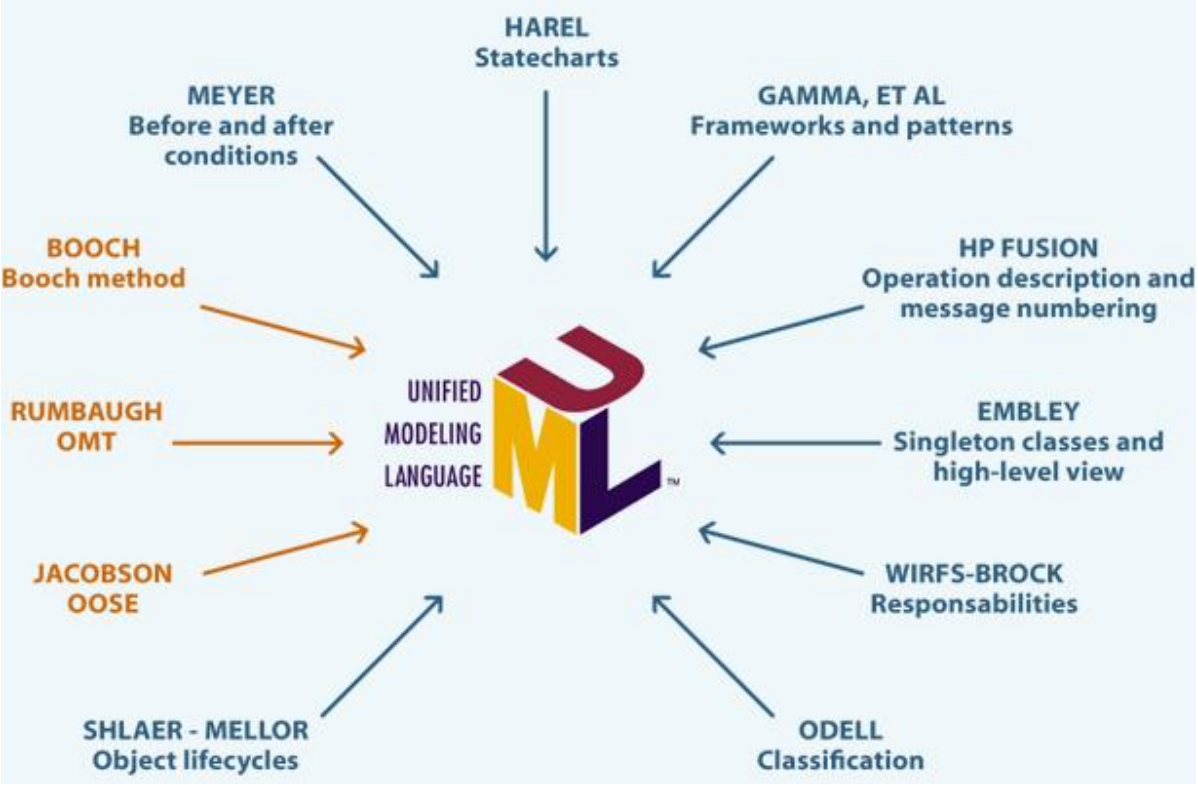
Aliás, naquela época não havia uma linguagem de modelagem dominante. Havia diversas linguagens, cada uma com vantagens e desvantagens. Foi aí que a Rational se perguntou: *Por que a famosa e moderna tecnologia de Orientação a Objetos estava demorando tanto para ser adotada de fato?* A resposta foi, entre outras, que havia um excesso de linguagens de modelagem.

Ora, ela não gostou da resposta e requisitou aos seus notáveis pesquisadores que encontrassem uma solução! *E o que eles fizeram?* Reuniram-se, consultaram outros pesquisadores, consolidaram seus métodos com as informações obtidas e padronizaram tudo em uma linguagem de modelagem não-proprietária: *Unified Modeling Language (UML)*.

Naquela época, os fornecedores estavam com medo de que um padrão controlado pela Rational desse uma vantagem competitiva desleal à empresa. Então, eles incitaram a Object Management Group (OMG) – consórcio de empresas

responsável por estabelecer padrões que suportem interoperabilidade – a tomar partido dessa padronização.

Essa consolidação surgiu por meio de intensas discussões com representantes de tecnologias concorrentes. A UML é o resultado de vários métodos e visões distintas de modelagem, como mostra a imagem abaixo. Em agosto de 1997, foi publicada a UML 1.1 e, posteriormente, foi editada como um padrão internacional chamado ISO/IEC 19501:2005.



Observem que a UML contou com a participação de outras empresas (IBM, HP, Oracle, etc). Sendo assim, a OMG adotou a versão 1.1 como um padrão oficial! A revisão 1.2 foi só para melhorar as aparências, já a versão 1.3 trouxe mudanças mais significativas. A revisão 1.4 acrescentou conceitos de componentes e perfis, e a revisão 1.5 adicionou a semântica de ação.

Com o passar do tempo, a linguagem passou a integrar conceitos de diversos outros métodos de orientação a objetos. Ademais, corrigiu diversos *bugs* e inconsistências até alcançar maturidade suficiente para avançar até a versão UML 2.0, em meados de 2005. A partir de então, passou por diversas e pequenas atualizações até chegar à versão atual: UML 2.5.

A UML pode ser definida como uma linguagem gráfica para especificar, visualizar e documentar artefatos primariamente de um sistema de software. *Por que primariamente, professor?* Porque ela tem sido usada efetivamente em diversas outras áreas, a saber: telecomunicações, defesa, aeroespacial, bancária, eletrônica, financeira, entre outras.

Portanto, a UML não está limitada a modelagem de software. Aliás, ela é uma linguagem tão expressiva que pode modelar outros sistemas, tais como um fluxograma do sistema judiciário, o comportamento de um sistema de saúde pública ou um projeto de hardware. Se a prova disser que é uma linguagem exclusiva de software, vocês já sabem o que marcar!

Alguns a definem como uma linguagem padrão de modelagem visual usada para modelagem de negócio e processos similares; além da análise, projeto e implementação de sistemas baseados em software. Trata-se de uma linguagem comum para analistas de negócio, arquitetos de software e desenvolvedores usada em sistemas de software já existentes ou novos.

A UML é intencionalmente independente de processos, i.e., pode ser aplicada no contexto de processos diferentes (Ex: RUP). Ademais, ela não é linguagem completa, ou seja, dado apenas um diagrama, não é possível entender completamente o comportamento do sistema; e não é completamente visual, ou seja, alguns conceitos não têm notação gráfica alguma.

Mas por que utilizar a UML? Bem, Martin Fowler diz que é por conta da comunicação e do entendimento. Um bom diagrama frequentemente pode ajudar uma equipe a entender um problema e transmitir uma ideia. A notação gráfica é um meio termo entre a imprecisão da linguagem natural e o detalhamento excessivo de uma linguagem de programação.

Pensem em como seria especificar um sistema só escrevendo ou falando! *Já imaginaram a quantidade de ambiguidades?* A linguagem natural é simples demais! Agora imaginem como seria especificar um sistema usando C++! *Já imaginaram a reação do usuário ao ver 400 linhas de código para entender uma classe*? Códigos detalham demais!

Bem, esse já seria um excelente motivo para se utilizar a UML. *Ora, mas se resume a isso?* Não, ela é uma linguagem completamente não-dependente de tecnologia. *Professor, isso quer dizer que é possível usá-la com Linguagem Estruturada?* Sim, com

C, Cobol, Pascal, etc. É independente de linguagem de programação, ferramentas, entre outros.

Hoje em dia, ela se tornou não somente a notação gráfica mais dominante dentro do mundo orientado a objetos, como também uma técnica popular nos círculos não orientados a objetos. Por fim, cabe salientar que a UML é uma verdadeira ferramenta de planejamento. Ela ajuda a apresentar uma visão geral do sistema atual e futuro.

Na UML, existe a definição de uma linguagem formal utilizada para especificar as restrições sobre os elementos de um modelo. *Quem é capaz de me responder como se chama essa linguagem?* Chama-se Object Constraint Language (OCL). Trata-se de uma linguagem declarativa para descrever regras que se aplicam aos modelos UML. Pode-se dividir a UML em quatro especificações:

- Infraestrutura da UML: especificação que contém o core da arquitetura, perfis e estereótipos.
- Superestrutura da UML: especificação que contém os elementos de modelagem estáticos e dinâmicos.
- Object Constraint Language (OCL): especificação que contém a linguagem formal usada para descrever expressões em Modelos UML.
- Intercâmbio: especificação que contém formatos de intercâmbio de diagramas para a UML.

A UML contém alguns mecanismos de uso geral muito importantes, tais como: estereótipos, notas explicativas, tagged values, restrições e pacotes:

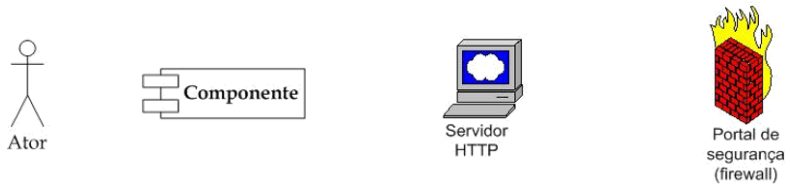
- Estereótipo: mecanismo utilizado para estender o significado de determinado elemento em um diagrama.
- Notas Explicativas: mecanismo utilizado para definir informação que comenta ou esclarece alguma parte de um diagrama.
- Tagged Values: mecanismo utilizado para predefinir propriedades para determinados elementos.

- Restrições: mecanismo utilizado para especificar restrições sobre um ou mais valores de um ou mais elementos de um modelo.

- Pacotes: mecanismo utilizado para agrupar elementos semanticamente relacionados (mais detalhes quando virmos Diagrama de Pacotes).

Estereótipos permitem adaptar ou personalizar modelos com construções específicas para um domínio, plataforma ou método de desenvolvimento particular. Trocando em miúdos, é um mecanismo de extensão que dá mais poder e flexibilidade à UML. Podemos ter estereótipos de dois tipos: predefinidos pela linguagem ou definidos pela equipe de desenvolvimento. *Como assim, professor?*

Estereótipos predefinidos já vêm nativamente na linguagem (Ex: <<interface>>, <<document>>, <<control>>, <<entity>>). No entanto, a equipe de desenvolvimento pode criar seus próprios estereótipos! *Como?* Basta colocar o nome do elemento delimitado pelos símbolos << e >>. Além disso, os estereótipos podem ser definidos textualmente ou graficamente.

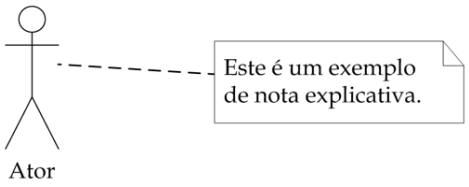


Os estereótipos gráficos são representados por um ícone que lembre o significado do conceito ao qual ele está associado. Na imagem acima, os dois estereótipos à esquerda são predefinidos pela própria linguagem; já os dois à direita são exemplos de possíveis estereótipos que podem ser construídos pela equipe de desenvolvimento. *Bacana?*

Rapaziada, vamos resumir o que vimos! Estereótipo é um mecanismo de uso geral utilizado para aumentar as capacidades da Linguagem UML! Ora, antes era tudo muito limitado, agora eu tenho infinitas possibilidades para representar um sistema. Há duas classificações para estereótipos: podem ser predefinidos ou definidos pela equipe de desenvolvimento.

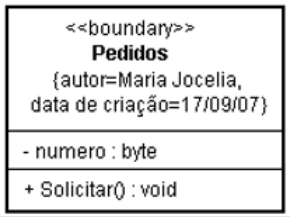
Pode ser classificado também em estereótipos textuais e gráficos: os primeiros devem vir delimitados pelos símbolos << e >>; os segundos devem vir com um ícone que lembre o conceito sendo representado. Essas duas classificações são independentes, logo é possível ter estereótipos gráficos ou textuais sendo predefinidos ou definidos pela equipe de desenvolvimento. *Ficou claro agora?*

Notas explicativas são utilizadas para definir informação que comenta ou esclarece alguma parte de um diagrama. Podem ser descritas em texto livre; também podem corresponder a uma expressão formal utilizando a linguagem de restrição de objetos da UML (OCL). Graficamente, as notas são representadas por um retângulo com uma *"orelhinha"* – como é mostrado na imagem abaixo.



O conteúdo da nota é inserido no interior do retângulo e este é ligado ao elemento que se quer esclarecer ou comentar através de uma linha tracejada. Ao contrário dos estereótipos, as notas textuais não modificam nem estendem o significado do elemento ao qual estão associadas, i.e., não criam algo novo – apenas explicam um elemento do modelo sem modificar sua estrutura ou semântica.

Elementos UML possuem propriedades predefinidas! As classes, por exemplo, possuem: Nome, Lista de Atributos e Lista de Operações. As Tagged Values (ou Etiquetas Valoradas) são utilizadas para definir outras propriedades para determinados elementos de um diagrama. A partir da UML 2.0, só é possível utilizá-las em conjunto com estereótipos!



A todo elemento da UML está associada alguma semântica. Isso quer dizer que cada elemento gráfico dessa linguagem possui um significado bem definido que, uma vez entendido, fica implícito na utilização do elemento em algum diagrama. As restrições permitem estender ou alterar a semântica natural de um elemento gráfico. *Bacana? Entendido?*

Esse mecanismo geral especifica restrições sobre um ou mais valores de um ou mais elementos de um modelo. Restrições podem ser especificadas tanto formal quanto informalmente. A especificação formal se dá pela OCL e a especificação informal se

dá por texto livre ou linguagem natural; devem vir delimitadas por chaves e aparecer dentro das notas explicativas.

Na década de 90, a arquitetura de software padecia de alguns problemas graves. Muitas vezes, enfatizava-se exageradamente só um aspecto do desenvolvimento de software e, outras vezes, a arquitetura não se direcionava aos interesses de todos os interessados. Então, em 1995, Philippe Kruchten publicou um artigo buscando solucionar essa questão.

Este artigo propunha organizar a descrição da arquitetura de software usando diversas visões concorrentes, cada uma direcionada a um conjunto de interesses específicos. Sabe-se que a arquitetura de software lida com abstrações e, para descrevê-la, o autor se utilizou de cinco visões ou perspectivas principais, como mostra a imagem abaixo.

- Visão Lógica (de Projeto): é a visão da arquitetura do sistema sob o ponto de vista dos usuários finais, apresentando os requisitos funcionais do software que suportam a arquitetura e fornecem serviços. Principais diagramas utilizados: Classe, Objetos e Pacotes.

- Visão de Desenvolvimento (ou Implementação): é a visão da arquitetura do sistema sob o ponto de vista do programador, apresentando a organização

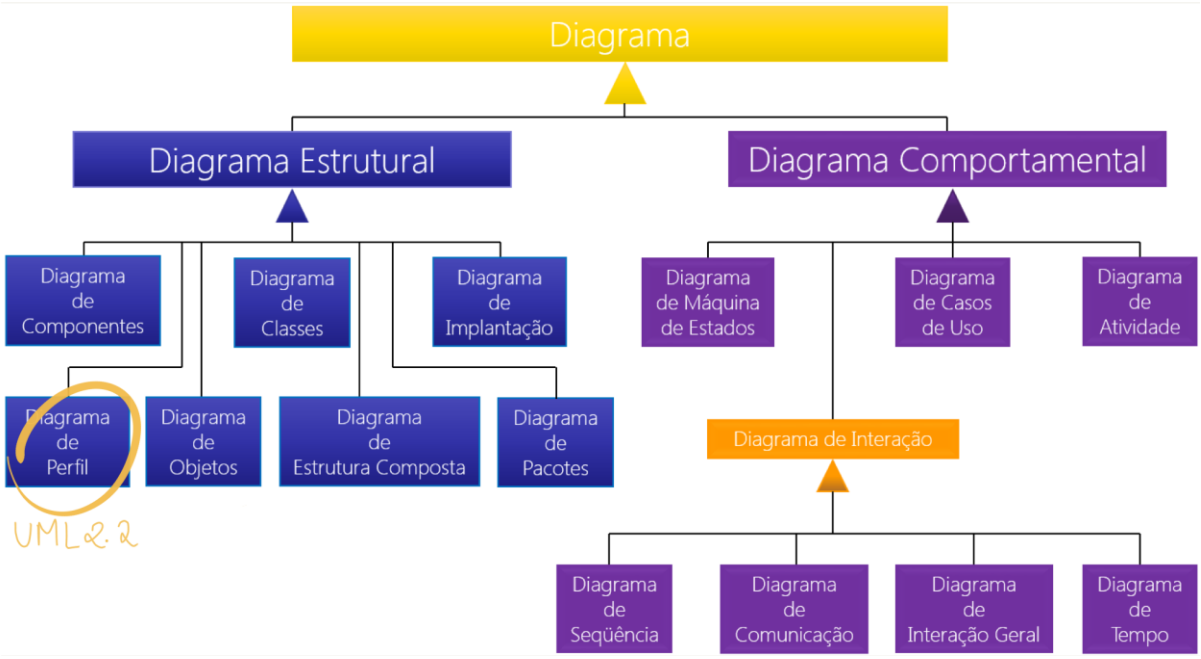
estática dos módulos que formam o software. Principais diagramas utilizados: Componentes.

- Visão de Processo: é a visão da arquitetura do sistema sob o ponto de vista do integrador, apresentando requisitos não-funcionais (Desempenho, Escalabilidade, etc). Principais diagramas utilizados: Sequência, Estrutura Composta, Máquina de Estados e Atividade.

- Visão Física (ou Implantação): é a visão da arquitetura do sistema sob o ponto de vista do engenheiro de sistemas, apresentando a topologia ou distribuição física dos componentes. Principais diagramas utilizados: Implantação e Componentes.

- Visão de Casos de Uso (Cenários): é a visão da arquitetura do sistema sob o ponto de vista de todos os usuários das outras visões e avaliadores, apresentando a consistência e validade do sistema. Principais diagramas utilizados: Casos de Uso.

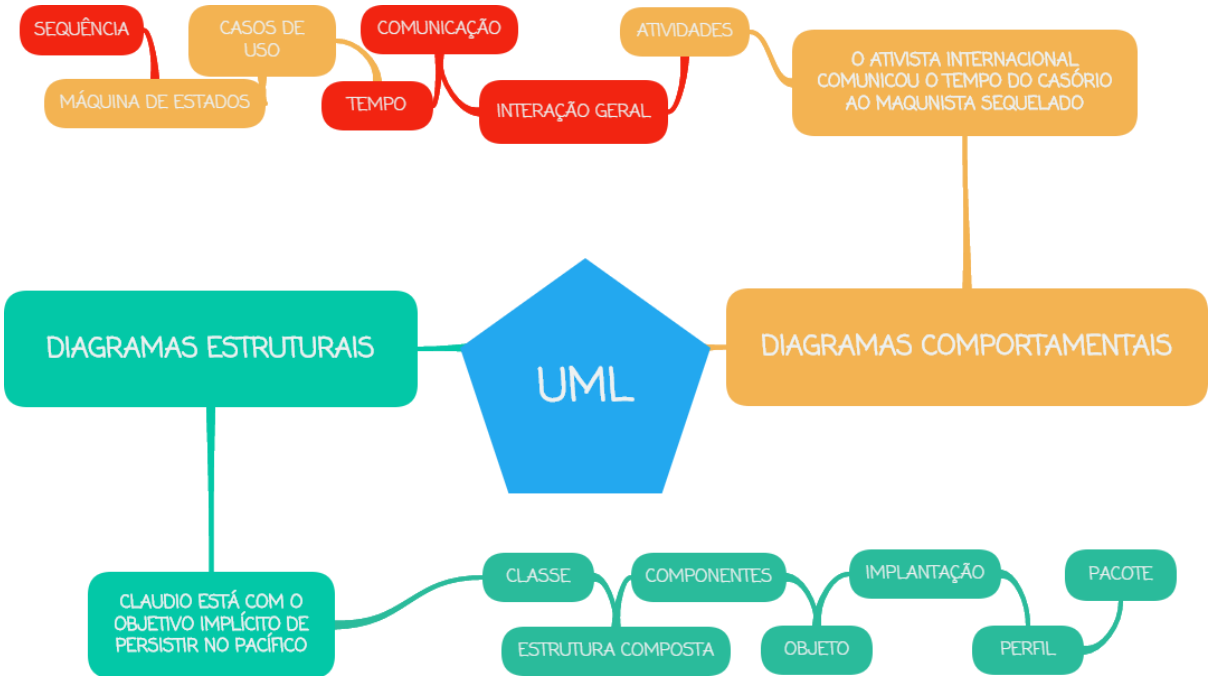
Galera, a Modelagem Orientada a Objetos ocorre quase que sempre por meio da Unified Modeling Language (UML). Portanto, nosso foco aqui será nos Diagramas UML! Eles são capazes de modelar sistemas orientados a objetos e nós veremos um por um cada um dos catorze! Atenção nesse assunto! A UML 2.4.1 descreve 14 tipos de diagramas oficiais como mostra a imagem abaixo:



Diagramas Estruturais: representam aspectos estáticos do sistema sob diversas visões diferentes. Em outras palavras, esses diagramas apresentam a estrutura do sistema inalterada há qualquer momento por não levarem em consideração o tempo em sua representação. São eles: Componente, Classes, Implantação, Perfil, Objetos, Estrutura Composta e Pacotes.

Diagramas Comportamentais: representam aspectos dinâmicos do sistema como um conjunto de mudanças. Podemos dizer, em outras palavras, que esses diagramas apresentam como os processos e funcionalidades do programa se relacionam. São eles: Máquina de Estados, Casos de Uso, Atividade, Sequência, Comunicação, Interação Geral e Tempo.

- Diagramas de Interação: são diagramas comportamentais que consideram o relacionamento dinâmico e colaborativo entre os objetos do sistema e suas trocas de informações. Eles enfatizam o controle de fluxo e dados entre as coisas do sistema que estão sendo modeladas (Ex: Objetos). São eles: Sequência, Comunicação, Interação Geral e Tempo.



IMPORTANTE

Infelizmente, esses conceitos devem ser memorizados. Decorem o nome de todos os diagramas, respondendo quais são estruturais, comportamentais e de interação! Façam isso nem que vocês tenham que tatuar o nome de cada um em partes do corpo. Decorem! Decorem! Decorem! Decorem! Decorem!

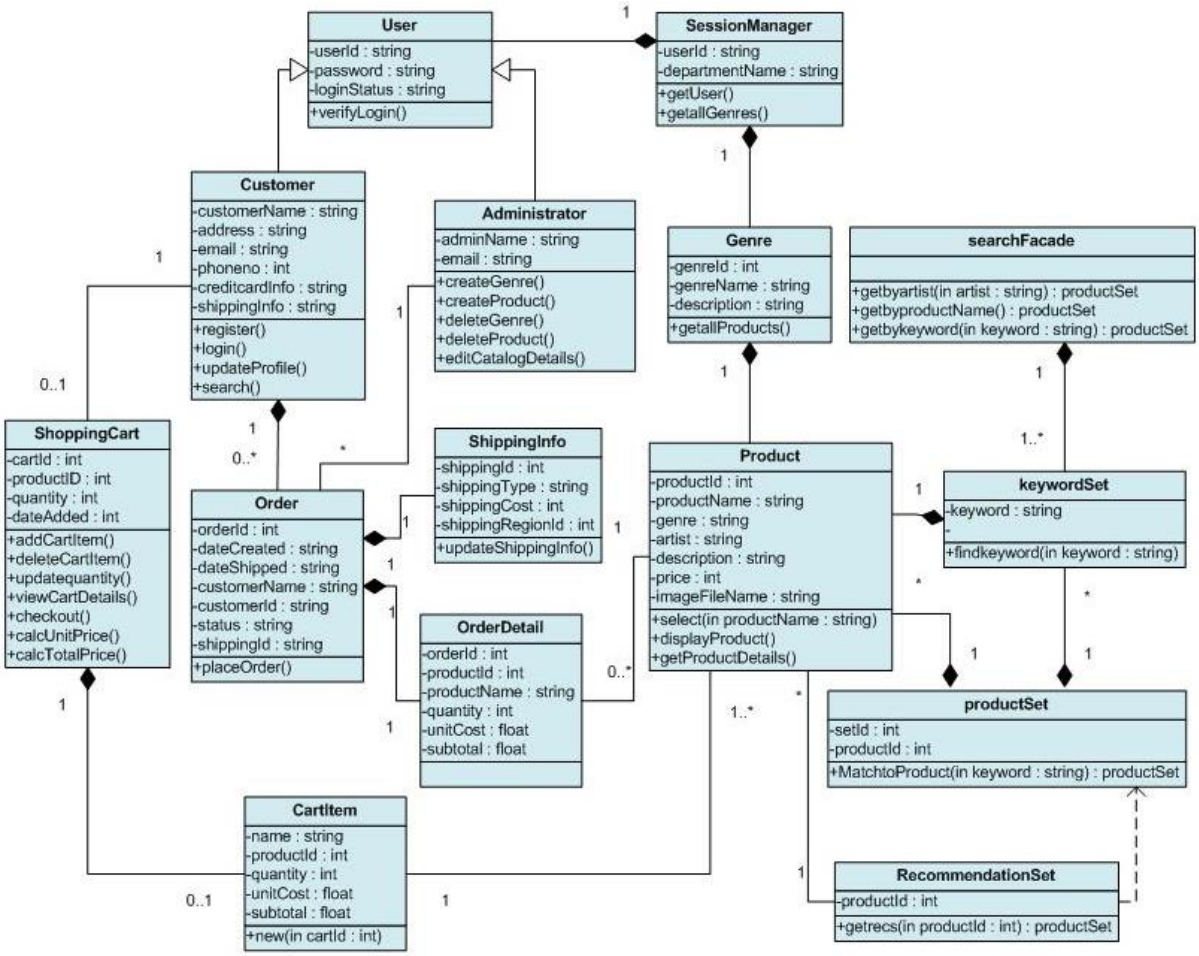
Professor, eu estou exausto de tanto decorar coisas! Pessoal, eu vou dizer o que me ajudou um pouco no momento de memorizar esses diagramas! Eu decorei as duas frases acima (uma para os estruturais e uma para os comportamentais). Elas contêm as letras iniciais de cada diagrama. A partir daí, eu fiquei tentando lembrar o nome de todos os diagramas incansavelmente até decorar.

DIAGRAMAS ESTRUTURAIS

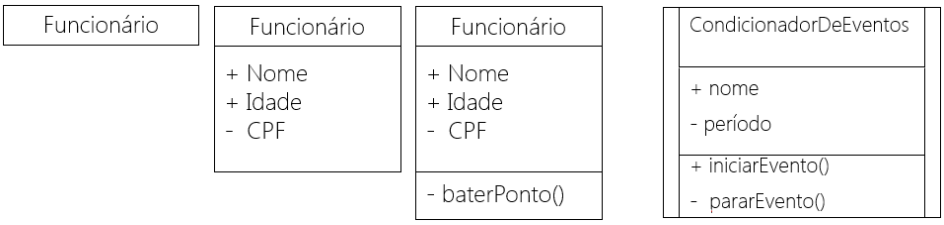
DIAGRAMA DE CLASSES

Esse é facilmente o diagrama mais conhecido de todos! Ele descreve as classes e interfaces presentes no sistema, suas características, restrições e os vários tipos de relacionamentos estáticos entre seus objetos. Representam-se também as propriedades e as operações de uma classe, assim como as restrições que se aplicam à maneira como os objetos estão conectados.

Professor, o que é uma classe? Classe é uma estrutura classificadora que abstrai um conjunto de objetos que compartilham características, restrições e semânticas similares. Ela define, também, o comportamento de seus objetos através de métodos e o estado por meio de atributos. A imagem abaixo apresenta as partes de um Diagrama de Classe (Classe, Interface, Relacionamentos, etc):



Professor, só existe uma forma de representar classes? Não! Pode-se representar de diversas formas, dependendo do nível de abstração. A imagem abaixo apresenta maneiras distintas de se representar uma classe: primeiro, apenas com nome da classe (mais abstrata); segundo, com nome da classe e suas propriedades; e terceiro, com nome da classe, propriedades e operações (mais concreta).



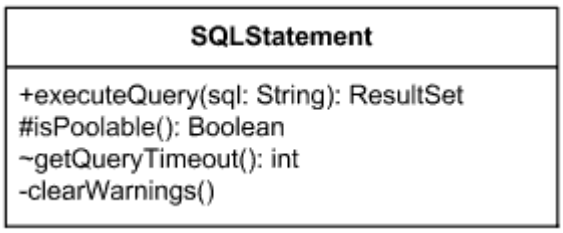
Professor, e essa última classe estranha? Bem, essa é a representação de uma Classe Ativa, que tem por objetivo representar classes cujos objetos têm um ou mais processos (threads). *Eu posso inserir detalhes de implementação da linguagem de programação que eu escolhi?* Sim, você pode inserir peculiaridades de uma linguagem de programação particular.

Galera, agora vamos responder a algumas perguntas relevantes. *Professor, como se representa um atributo estático na UML?* Bem, basta sublinhar o nome do atributo! *Professor, como se representa uma operação abstrata?* Bem, basta escrever seu nome em itálico! *E como se representa uma operação estática?* Bem, basta escrever seu nome sublinhado!

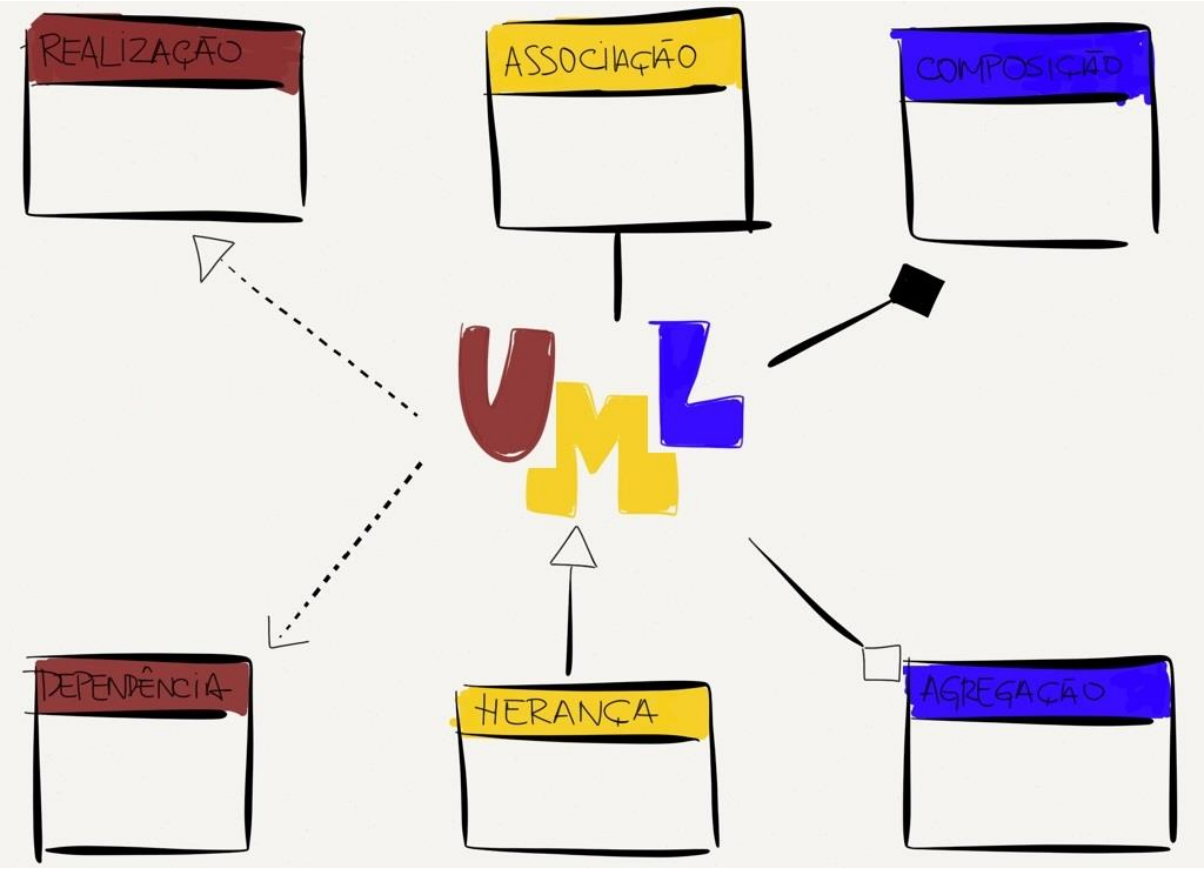
MODIFICADOR/ESPECIFICADOR			CLASSE	SUBCLASSE	PACOTE	TODOS
UML	PÚBLICO	+	X	X	X	X
	PROTEGIDO	#	X	X		
	PACOTE	~	X		X	
	PRIVADO	-	X			
JAVA	PÚBLICO	+	X	X	X	X
	PROTEGIDO	#	X	X	X	
	DEFAULT	~	X		X	
	PRIVADO	-	X			

E como se representa a visibilidade de atributos e operações? A tabela acima apresenta a diferença de visibilidade em JAVA e UML. Observem duas diferenças sutis: na UML, um elemento protegido não é visível para elementos dentro do

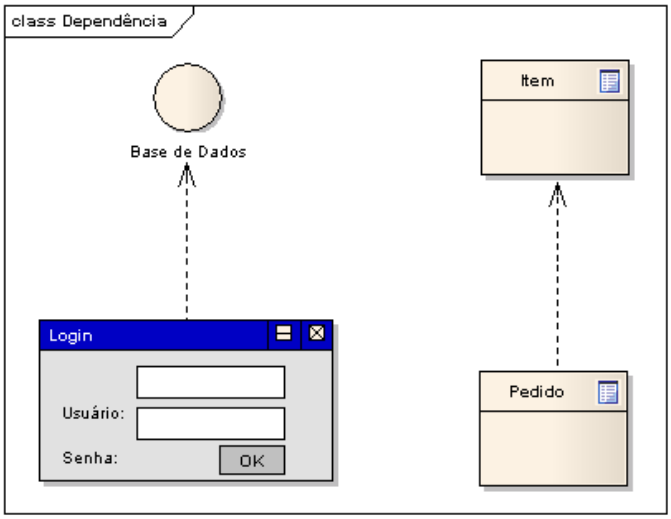
mesmo pacote; e o nível Pacote da UML tem o mesmo nome do nível Default em JAVA! Pessoal, isso cai bastante em prova! Podemos ver um exemplo abaixo:



Pessoal, agora vamos falar sobre os tipos de relacionamentos em um diagrama de classes. Eles representam as conexões entre classes, objetos, pacotes, tabelas, entre outros. Há três tipos de relacionamentos entre classes: Dependência, Generalização e Associação, como mostra a imagem abaixo. No entanto, eles se desdobram em vários outros! Vejamos...



Relacionamento de Dependência: é um relacionamento direcionado e semântico entre dois ou mais elementos que ocorre se mudanças na definição de um elemento (independente) causarem mudanças ao outro elemento (dependente). Em outras palavras, é quando a classe cliente é dependente de algum serviço da classe fornecedora, como mostra a imagem abaixo.

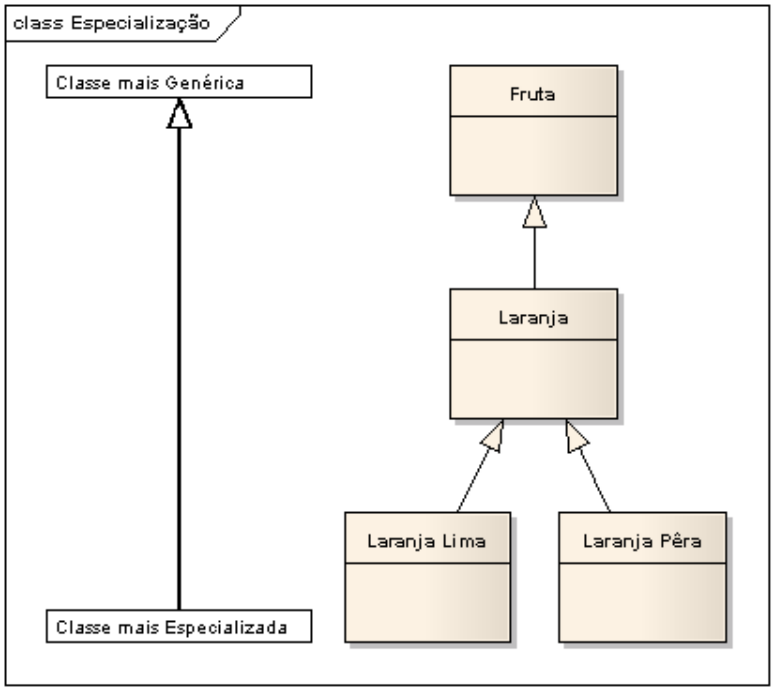


À esquerda, uma Classe *Login* depende de uma Interface Base de Dados. *Como assim, professor?* O diagrama indica que a Classe *Login* utiliza algum método da Interface Base de Dados, logo caso esse método seja modificado, pode haver danos à *Login*. Lembrando que esse relacionamento pode ocorrer entre duas classes ou entre uma classe e uma interface.

À direita, uma Classe Pedido depende de uma Classe Item. *Como assim, professor?* O diagrama indica que a Classe Pedido utiliza algum método da Classe Item. Portanto, caso se modifique um método da Classe Item, pode haver danos na classe dependente, que é a Classe Pedido. Logo, Pedido é a classe dependente e Item é a classe independente.

IMPORTANTE

Observem que o Relacionamento de Dependência é representado por uma seta tracejada que aponta para classe independente. Em outras palavras, a Classe Pedido depende da Classe Item. Os relacionamentos <include> e <extend> também são relacionamentos de dependência.



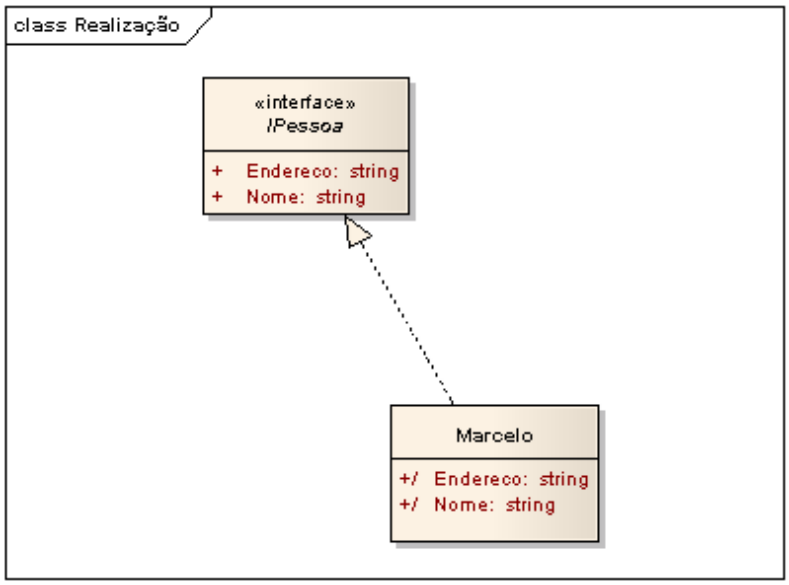
Relacionamento de Generalização/Especialização (Herança): indica que a subclasse é uma especialização da superclasse ou que a superclasse é uma generalização da subclasse. Qualquer instância da subclasse é também uma instância da superclasse. É conhecido como relacionamento de herança, relacionamento de extensão ou relacionamento "é-um".

Observem que a imagem acima apresenta um relacionamento em que Fruta é uma generalização de Laranja, assim como Laranja é uma generalização de Laranja Lima e Laranja Pêra. Tudo isso pode ser dito em termos de especialização: Laranja Lima e Laranja Pêra são especializações de Laranja, assim como Laranja é uma especialização de Fruta.

IMPORTANTE

Observem que o Relacionamento de Generalização/Especialização é representado por uma linha com um triângulo que aponta para a classe genérica.

Relacionamento de Realização: relacionamento entre dois elementos em que um elemento realiza (implementa/executa) o comportamento que o outro elemento especifica. Costuma-se dizer que um dos elementos especifica um contrato e o outro elemento realiza esse contrato. A imagem abaixo mostra a Classe Marcelo, que realiza uma Interface Pessoa.

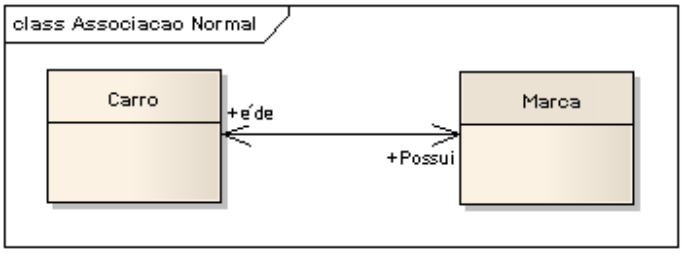


IMPORTANTE

Observem que o Relacionamento de Realização é representado por uma linha tracejada com um triângulo que aponta para a interface.

Relacionamento de Associação: relacionamento estrutural entre objetos e especifica os objetos de uma classe que estão ligados a objetos de outra classe. São eles:

- Simples: é um tipo de relacionamento mais forte que o relacionamento de dependência e indica que uma instância de um elemento está ligada à instância de outro elemento. São representados por uma linha sólida com ou sem setas de navegabilidade. Ademais, pode haver nomes para a associação e indicação de multiplicidade[1].



Na imagem acima, há uma associação simples entre Carro e Marca. Trocando em miúdos, significa que o carro é de uma marca e a marca possui diversos carros.

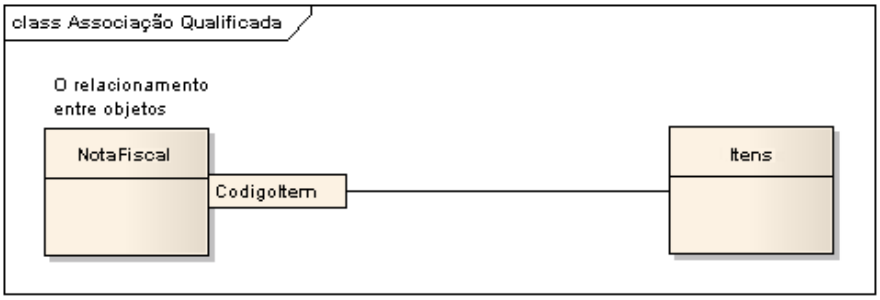
[1] Multiplicidade permitem representar a informação dos limites inferior e superior da quantidade de objetos aos quais outro objeto pode estar associado. Um objeto pode ser um caso de uso, uma classe, um ator, etc.

Observem que se excluir-se um carro, a marca continua a existir. Assim como se excluir-se uma marca, o carro continua a existir. Portanto, esse é um relacionamento de associação simples.

IMPORTANTE

Observem que o Relacionamento de Associação Simples é representado por uma linha sólida que pode ou não ter setas de navegabilidade unidirecionais ou bidirecionais.

- Qualificada: é um tipo de relacionamento similar à associação simples, contudo possui um qualificador, que é um atributo do elemento-alvo capaz de identificar uma instância dentre as demais. Ela ocorre em associações um-para-muitos ou muitos-para-muitos em que se deseja encontrar um elemento específico dada uma chave.



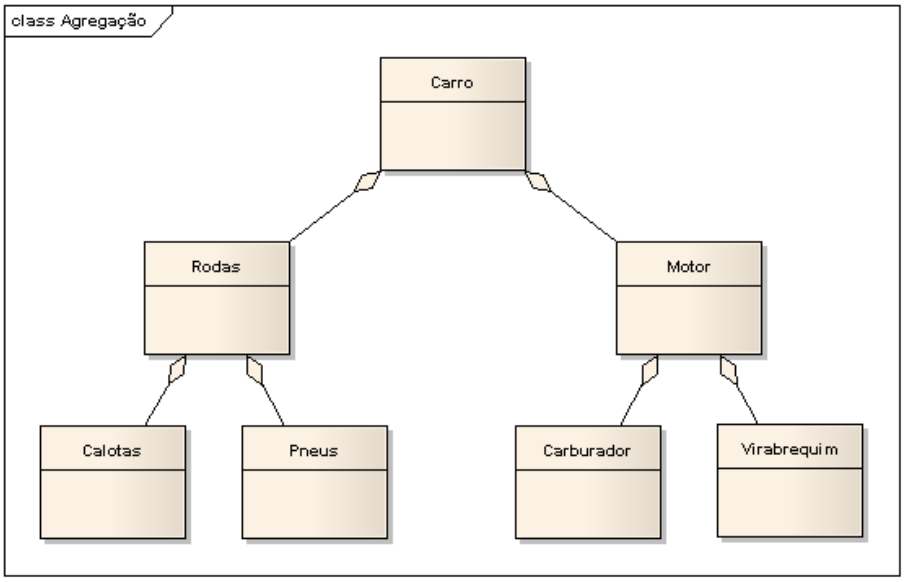
Na imagem acima, há uma associação qualificada entre NotaFiscal e Itens. Pensem comigo: *uma nota fiscal pode ter milhares de itens, mas e se por acaso eu quiser encontrar um item específico?* Ora, eu posso utilizar um identificador de itens como qualificador de NotaFiscal. Portanto, cada item está associado a um código de item e a uma nota fiscal.

IMPORTANTE

Observem que o Relacionamento de Associação Qualificada é representado por uma linha sólida com um retângulo ao lado da classe de cardinalidade 1 contendo o qualificador.

- Agregação: é um tipo de associação, porém mais forte, em que o todo está relacionado às suas partes de forma independente. Nesse tipo de relacionamento, as partes têm existência própria. Portanto, elas existem por si

só, isto é, a parte existe sem o todo. É representado por uma linha com um diamante vazio na extremidade referente ao todo.



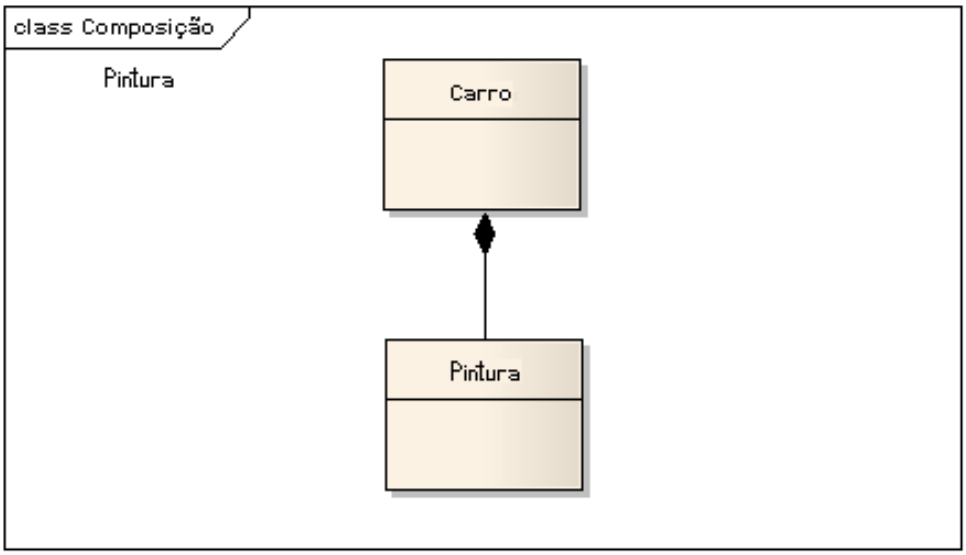
Na imagem acima, há uma agregação entre Carro e Rodas, Carro e Motor, Rodas e Calotas, Rodas e Pneus, Motor e Carburador, e Motor e Virabrequim. Vamos interpretar apenas a agregação entre Carro (Todo) e Rodas (Partes). *Respondam-me: a roda pode existir sem um carro?* Sim, claro! Logo, esse é um relacionamento de agregação.

IMPORTANTE

Observem que o Relacionamento de Agregação é representado por uma linha sólida com um diamante vazio na classe agregadora.

- Composição: é um tipo de agregação[2], porém mais forte, em que o todo está relacionado às partes de forma dependente. Nesse relacionamento, as partes não têm existência própria. Logo, não existem por si só, i.e., a parte não existe sem o todo. É representado por uma linha com um diamante cheio na extremidade referente ao todo.

[2] Inclusive, é chamado algumas vezes de Agregação por Composição.



Na imagem acima, há uma composição entre Carro e Pintura. *Respondam-me: a pintura pode existir sem o carro?* Não, não faz o menor sentido! A existência do carro é requisito fundamental para a existência da pintura. Ademais, essa pintura é exclusiva daquele carro e não pode ser compartilhada. Logo, na composição, o todo tem sempre cardinalidade 1!c

IMPORTANTE

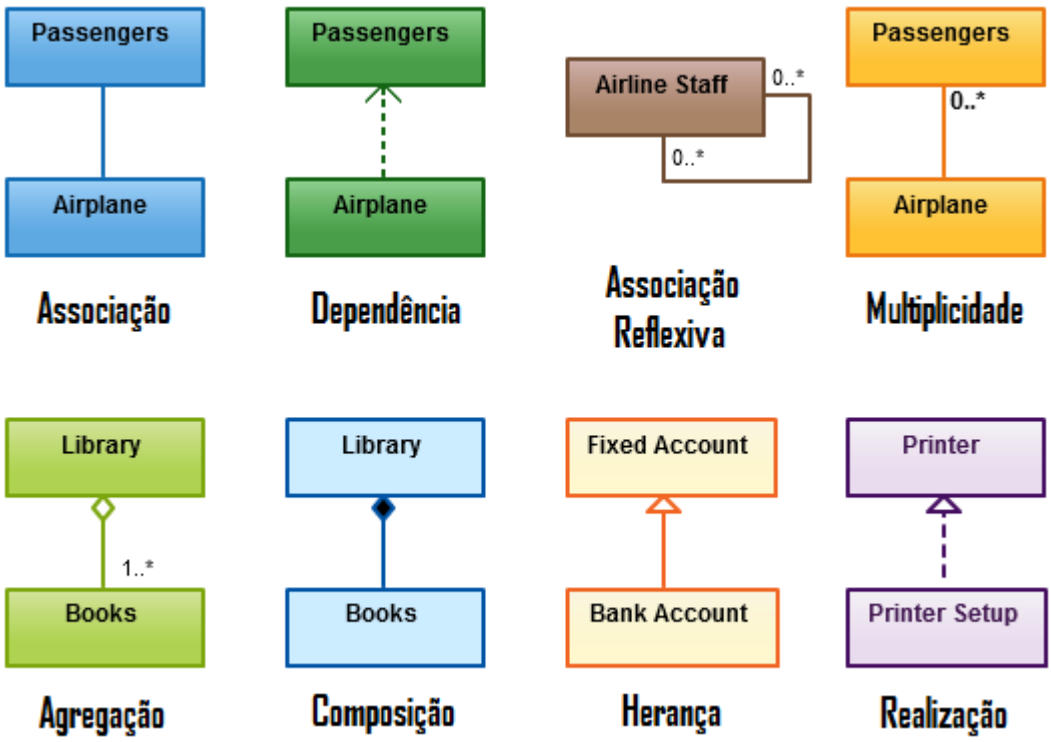
Observem que o Relacionamento de Composição é representado por uma linha sólida com um diamante cheio na classe compositora. Pessoal, quando eu aprendi isso, decorei assim: Diamante Cheio = Composição. Portanto, Diamante Vazio = Agregação.

Quando utilizar Diagramas de Classe? Os Diagramas de Classe são a espinha dorsal da UML; portanto, você irá utilizá-los o tempo todo. O maior problema é que eles são tão ricos que podem ser complexos demais para usar. Dessa forma, não tente utilizar todas as notações de que você dispõe; uso de diagramas de classes conceituais são muito uteis na exploração da linguagem do negócio.

Busque manter o software fora da discussão e manter a notação mais simples; não desenhe modelos para tudo; em vez disso, concentre-se nas áreas principais. É melhor ter poucos diagramas que você utiliza e os mantém atualizados do que ter muitos modelos esquecidos e obsoletos. O maior perigo é que você pode focalizar exclusivamente a estrutura e ignorar o comportamento.

Portanto, ao desenhar diagramas de classe para entender o software, sempre os faça em conjunto com alguma forma de técnica comportamental. Se você estiver indo bem, vai ficar trocando entre as técnicas frequentemente. Galera, é possível fazer uma aula inteira somente sobre os diagramas de classe, mas aqui temos que ser mais objetivo, abordando os assuntos mais frequentes.

A imagem abaixo apresenta um resumão de diversos relacionamentos:



- **Associação:** mostra que os passageiros e o avião possuem alguma ligação;
- **Dependência:** mostra que o avião depende de passageiros;
- **Associação Reflexiva:** mostra que a equipe do avião se relaciona entre si;
- **Multiplicidade:** mostra que um avião possui zero ou mais passageiros;
- **Agregação:** mostra que uma biblioteca tem um ou mais livros (independentes)[3];
- **Composição:** mostra que uma biblioteca tem livros (necessariamente)[4];
- **Herança:** mostra que uma conta bancária é filha de conta fixa;
- **Realização:** *setup* da impressora implementa funcionalidades da impressora;

[3] Nesse entendimento, livros continuarão a existir mesmo sem bibliotecas;
[4] Nesse entendimento, livros só podem existir como parte de uma biblioteca;

DIAGRAMA DE OBJETOS

O Diagrama de Objetos (ou Diagrama de Instâncias) é uma variação do Diagrama de Classes. Contudo, aqui não se trata da estrutura geral, mas de cada instância específica do sistema. Portanto, no diagrama de objetos não há Pessoa, há "João". Não há Carro, há "Pálio". Não há Cachorro, há "Totó". *Entenderam?* No diagrama de objetos, personaliza-se cada instância com seus valores.

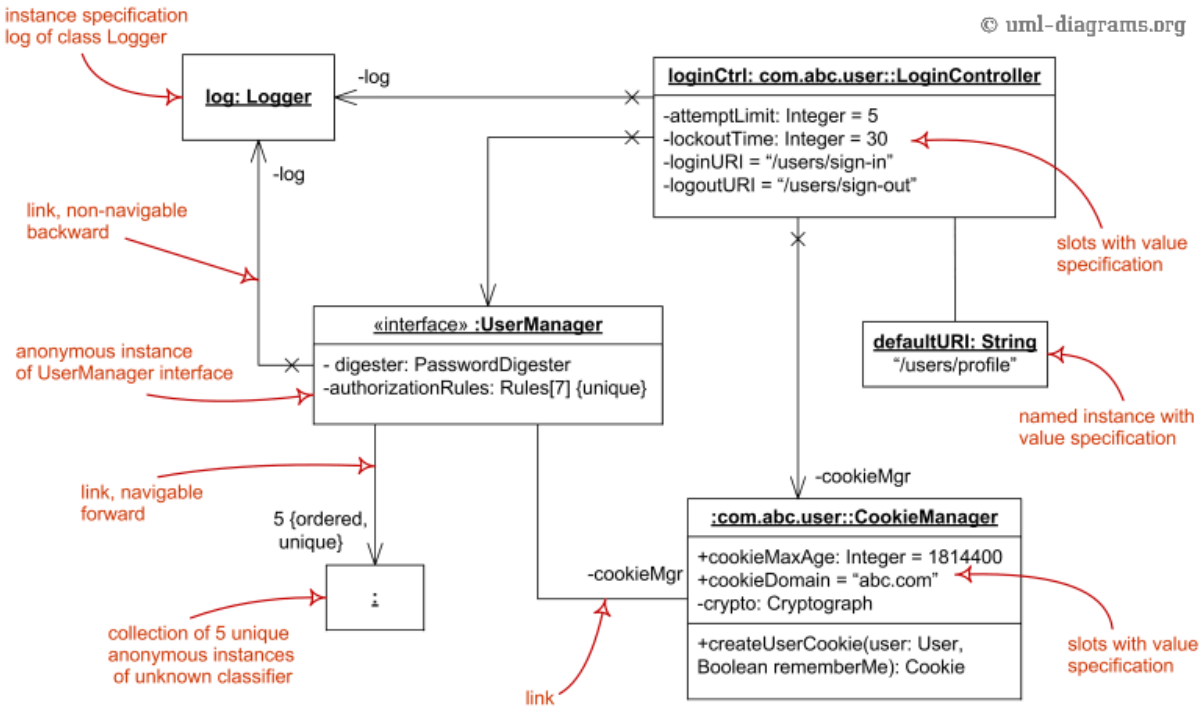
Costuma-se dizer que o diagrama de objetos representa uma fotografia estática do sistema em um dado momento de execução, portanto esses diagramas não refletem o sistema genericamente, mas de forma específica – em um determinado instante. Como ele mostra instâncias, em vez de classes, ele é frequentemente chamado Diagrama de Instâncias.

Esses diagramas são úteis para mostrar exemplos de objetos interligados. Em algumas situações, pode-se definir a estrutura de um sistema precisamente com um diagrama de classes, mas a estrutura ainda é difícil de entender. Neste momento, o diagrama de objetos pode ser bastante útil para facilitar a compreensão de um determinado domínio.

Ele é um momento instantâneo dos objetos em um sistema em um determinado ponto no tempo. Você pode usar um diagrama de objetos para mostrar um exemplo de configuração de objetos. Você pode dizer que os elementos de um diagrama de objetos são instâncias caso os nomes estejam sublinhados. Cada nome assume a forma **nome de instância : nome da classe**.

As duas partes do nome são opcionais. Se você usar apenas o nome da classe, deve incluir os dois-pontos. Você pode mostrar valores para atributos e vínculos também. Rigorosamente, elementos de um diagrama de objetos são especificações de instâncias, em vez de instâncias verdadeiras. O motivo é que é válido deixar atributos obrigatórios vazios ou mostrar especificações de instâncias de classes abstratas.

Esses diagramas são raramente utilizados. A única utilidade prática e direta dos diagramas de objetos é a de ilustrar a formação de relacionamentos complexos de um diagrama de classes, como associações reflexivas. Com o objetivo de facilitar a atividade de validação, podemos construir diagramas de objetos para ilustrar e esclarecer certos aspectos de um diagrama de classes.



Rapaziada, podemos representar o nome de um objeto de diversas formas:

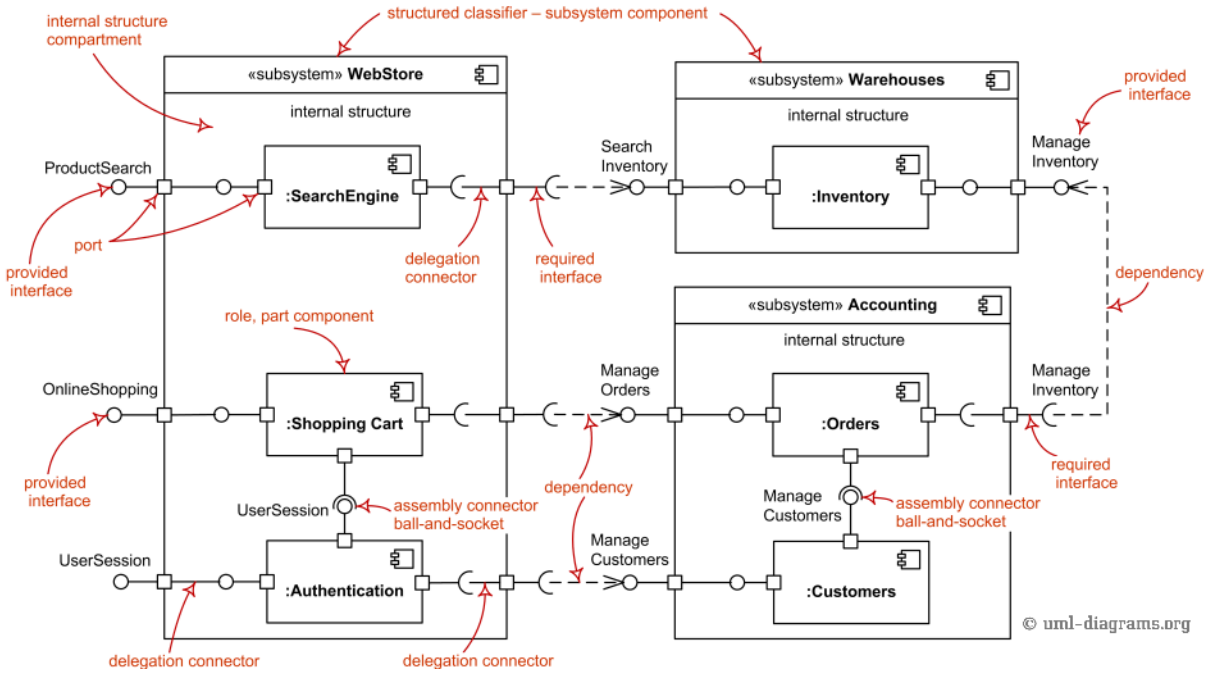
Representação do Nome	Descrição
:	Significa que se trata de uma instância anônima de uma classe anônima;
:Cliente	Significa que se trata de uma instância anônima da classe Cliente;
novoCliente:	Significa que se trata de uma instância novoCliente de uma classe anônima;
novoCliente:Cliente	Significa que se trata de uma instância novoCliente de uma classe Cliente;
novoCliente:Clientes::Cliente	Significa que se trata de uma instância novoCliente de uma classe Cliente de um pacote Clientes;

Quando utilizar o Diagrama de Objetos? Os Diagramas de Objeto são úteis para mostrar exemplos de objetos interligados. Eventualmente, define-se uma estrutura precisamente com um diagrama de classes, mas a estrutura pode ser ainda é difícil de entender. Nesses casos, exemplos de diagrama de objetos podem auxiliar. Eles podem ser vistos como um diagrama de comunicação sem as mensagens.

DIAGRAMA DE COMPONENTES

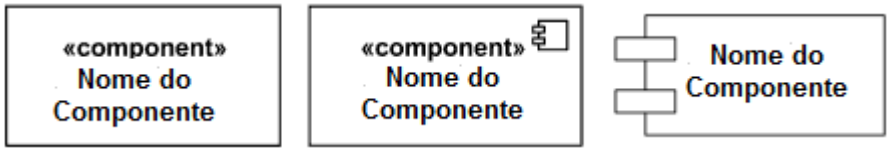
O Diagrama de Componentes representa o sistema sob uma perspectiva funcional, expondo a organização de seus módulos e as relações entre seus componentes por meio de interfaces. *Professor, o que é são os componentes?* É uma unidade independente, que pode ser utilizada ou substituída com/por outros componentes para formar um sistema complexo.

Os componentes representam peças que podem ser adquiridas e atualizadas independentemente. Ora, um sistema inteiro pode ser construído baseado em componentes e essa é uma decisão tanto de negócio quanto técnica. Existe, inclusive, um modelo de desenvolvimento de software baseado em componentes. Os componentes podem ser tabelas, documentos, etc.



Como mostra a imagem acima, os componentes são representados como retângulos com o símbolo de componente no canto superior direito e com nome escrito dentro ou abaixo do componente. Frequentemente, eles vêm com estereótipos para definir seu tipo e seus relacionamentos. A grande vantagem do uso de componentes é a modularidade.

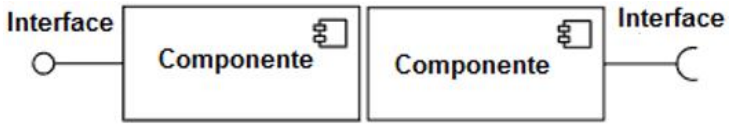
Galera, é possível representar componentes das seguintes três formas:



Um componente pode apresentar um estereótipo, i.e., uma definição do que é este componente. Os principais estereótipos são:

- Arquivo: determina que o componente é um arquivo de dados do sistema;
- Biblioteca: determina que o componente é uma biblioteca de código;
- Documento: determina que o componente é um documento de sistema;
- Executável: determina que o componente é um arquivo executável;
- Tabela: determina que o componente é uma tabela de um banco de dados.

Temos, também, a Interface Fornecida, que designa uma interface que o próprio componente possui e oferece para outros componentes – o componente só pode ser acessado pela interface fornecida; e Interface Requerida, que designa uma interface necessária para que componente se comunique com outros componentes. Esta interface será conectada em uma interface fornecida de outro sistema.

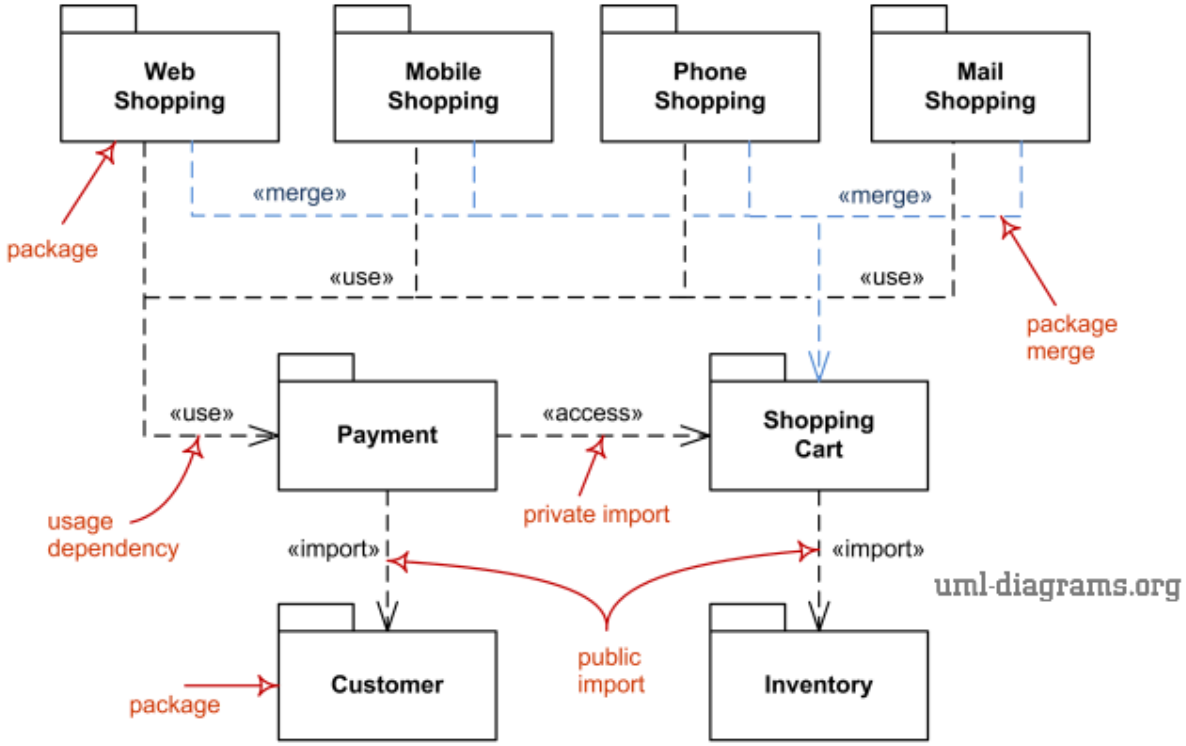


Quando utilizar Diagramas de Componentes? Use Diagramas de Componentes quando você estiver dividindo seu sistema em componentes e quiser representar seus relacionamentos por intermédio de interfaces ou também quando quiser representar a decomposição de componentes em uma estrutura de nível mais baixo. *Entendido, galera?* Exercícios!

DIAGRAMA DE PACOTES

Ele representa os pacotes e suas dependências. *Professor, o que é um pacote?* É um agrupamento lógico que permite reunir elementos da UML em unidades de mais alto nível. Comumente, pacotes agrupam classes, mas pode-se agrupar qualquer coisa. Esses elementos podem ser membros de um único pacote com uma estrutura hierárquica.

Os pacotes de nível superior são divididos em subpacotes que possuem seus próprios subpacotes e assim por diante. Esses pacotes ilustram a arquitetura de um sistema como um agrupamento de classes em um alto nível de abstração. O diagrama pode ser desenhado como uma pasta contendo várias classes ou pastas hierárquicas.



Quando utilizar Diagramas de Pacote? São extremamente úteis em sistemas de grande porte, para obter uma visão das dependências entre os principais elementos de um sistema. Esses diagramas correspondem bem às estruturas usuais de programação. A representação de diagramas de pacotes e dependências o ajuda a manter as dependências de uma aplicação sob controle.

Os Diagramas de Pacotes representam um mecanismo de agrupamento em tempo de compilação. Para mostrar como os objetos são compostos em tempo de execução, recomenda-se a utilização do Diagrama de Estrutura Composta.

Percebam que não é um diagrama que cai bastante em prova, mas é importante saber o básico sobre ele.

DIAGRAMA DE IMPLANTAÇÃO (OU INSTALAÇÃO)

O Diagrama de Implantação apresenta o *layout* físico de um sistema, revelando quais partes do software são executadas em quais partes do hardware. Também conhecido como Diagrama de Instalação ou também Diagrama de Distribuição, pode representar toda a estrutura de hardware e requisitos mínimos onde o sistema será de fato executado.

Os itens principais do diagrama são nós conectados por caminhos de comunicação. Um nó é algo que pode conter algum software. Um dispositivo é um hardware, que pode ser um computador ou uma peça de hardware mais simples conectada a um sistema; um ambiente de execução é um software que contém a si mesmo ou contém outro software (Ex: Sistema operacional ou processo contêiner).

Os nós contêm artefatos, que são manifestações físicas do software: normalmente, arquivos. Esses arquivos podem ser executáveis (Ex: .exe, binários, entre outros) ou arquivos de dados, arquivos de configuração, documentos HTML, etc. A listagem de um artefato dentro de um nó mostra que ele está instalado nesse nó do sistema que está em execução.

Você pode mostrar artefatos como caixas de classe ou listando o nome dentro de um nó. Se você os mostrar como caixas de classe, poderá adicionar um ícone de documento ou a palavra-chave <<artifact>>. Você pode rotular nós ou artefatos com valores para indicar diversas informações interessantes a respeito do nó, tais como fornecedor, sistema operacional, localização ou qualquer coisa que você desejar.

Frequentemente, você terá vários nós físicos executando a mesma tarefa lógica. Você pode mostrar isso com várias caixas de nó ou declarar o número como um valor afixado. Muitas vezes, os artefatos são a implementação de um componente. Para mostrar isso, você pode usar um valor indicado na caixa do artefato. Os caminhos de comunicação entre os nós indicam como as coisas se comunicam.

Você pode rotular esses caminhos com informações sobre os protocolos de comunicação utilizados. Os nós e artefatos não são de responsabilidade do programador, mas da equipe de implantação do sistema, que deve se preocupar com a configuração e topologia dos elementos de software sobre os elementos de *hardware*.

Quanto usar os Diagramas de Implantação (ou Instalação). Bem, eles são úteis para mostrar o que é instalado e onde; portanto, qualquer instalação mais complicada

pode fazer bom uso deles (Ex: uma configuração e arquitetura de sistema em que estarão ligados componentes, representados pela arquitetura física de hardware, processadores, entre outros).

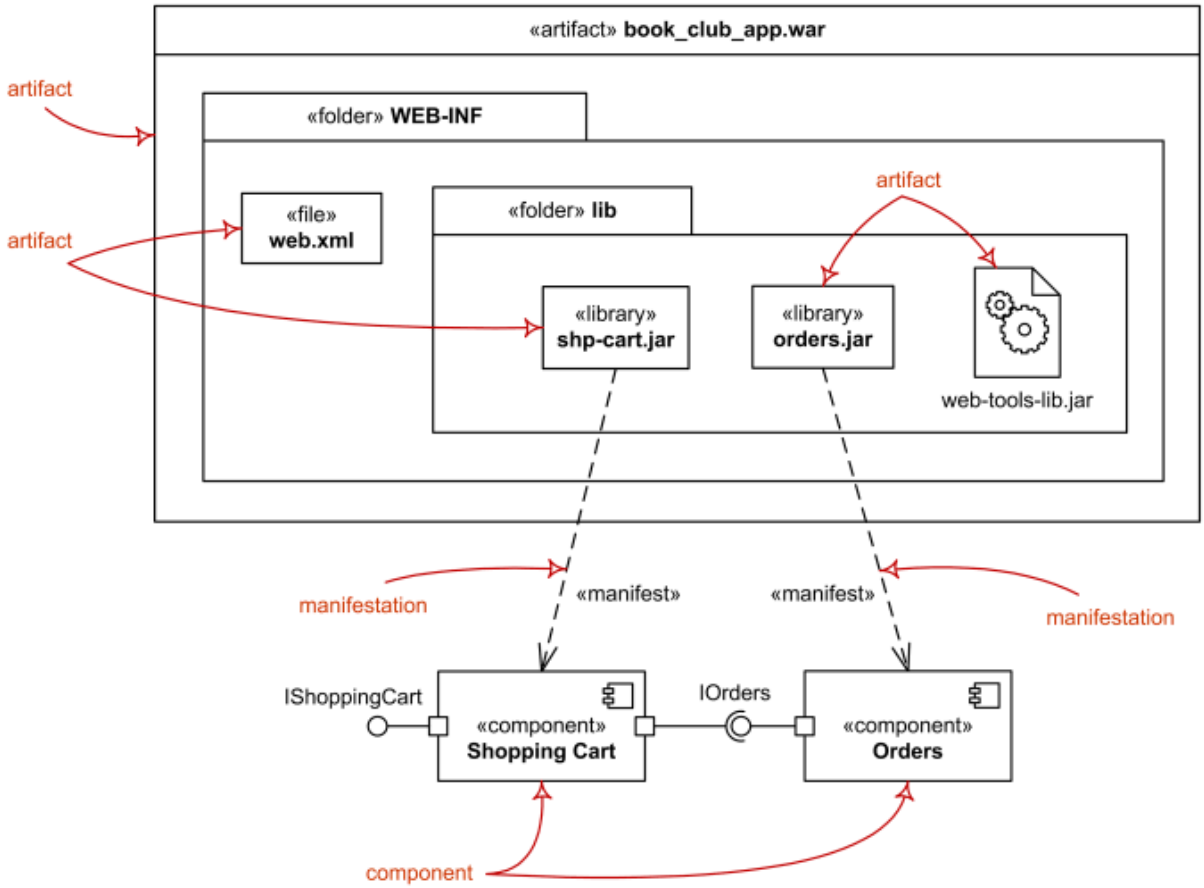


DIAGRAMA DE PERFIL

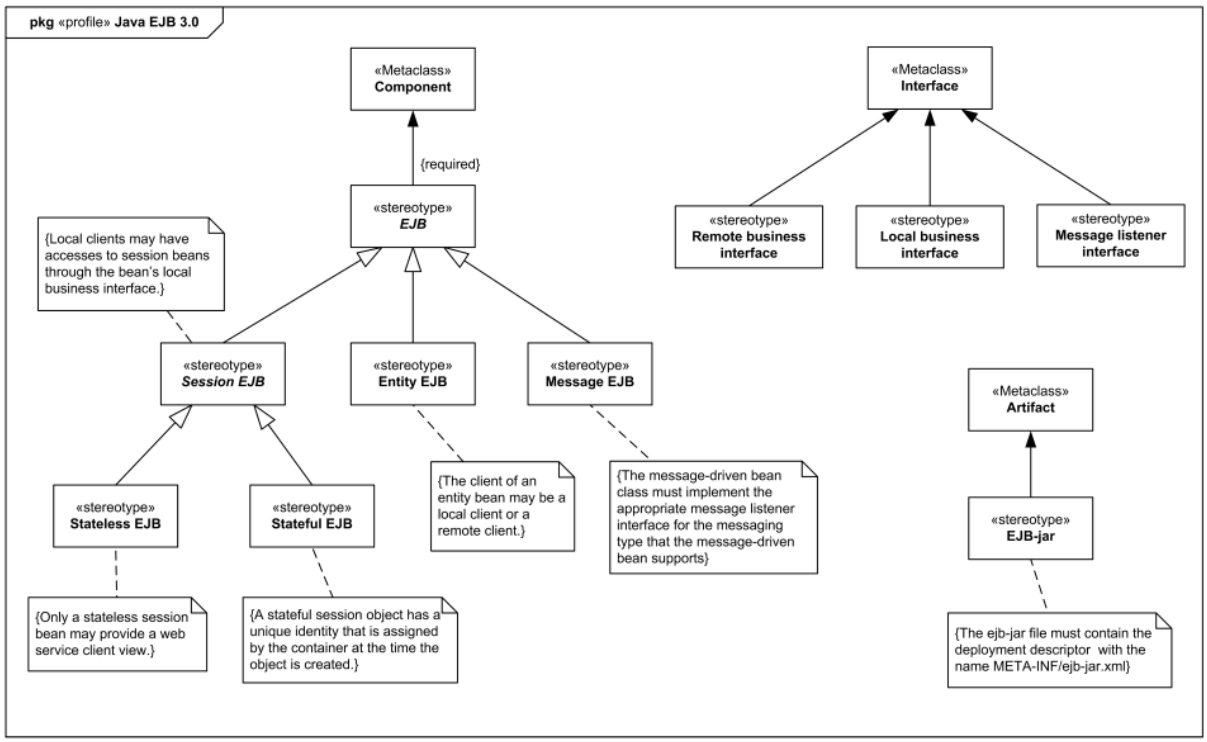
Frequentemente, vale a pena definir uma versão de UML adequada a uma determinada finalidade ou área de domínio. Por exemplo, se você deseja usar um modelo de UML personalizado para geração de código em uma determinada linguagem, é útil definir estereótipos que possam ser aplicados aos elementos para fornecer dicas ao gerador de código.

Entretanto, os estereótipos definidos seriam diferentes para, por exemplo, Java e C++. Da mesma forma, você poderia usar a UML para modelar bancos de dados – é possível ajustar a UML utilizando perfis. *Professor, o que é um perfil?* Trata-se de uma UML com um conjunto de estereótipos predefinidos, valores atribuídos, restrições e classes de base.

Ele também seleciona um subconjunto dos tipos de elementos da UML para uso e define uma versão especializada da UML para uma determinada área. Como o perfil é criado sobre elementos comuns, ele não representa uma nova linguagem e pode ser suportado por ferramentas comuns da UML. A maioria dos modeladores não construirá seus próprios perfis.

A maioria dos perfis será construída por criadores de ferramentas, criadores de frameworks e projetistas similares aos recursos gerais. Entretanto, muitos modeladores usarão os perfis. São como as bibliotecas de sub-rotina tradicionais – alguns especialistas as criam, mas muitos programadores as usam. *Sacaram?* Portanto há bastante reusabilidade.

O Diagrama de Perfil permite representar esses novos elementos, operando no nível de metamodelos. Imaginem que eu quero utilizar a UML para representar uma rede de computadores. *A UML tem símbolos para representar roteadores, switches, etc?* Não! Para tal, podem-se utilizar estereótipos. *Como?* Ora, eu desenho um retângulo e escrevo nele a expressão <<roteador>> ou <<switch>>.



A UML oferece diversos estereótipos padronizados, podemos citar <<interface>>, <<extends>> e <<include>>. No entanto, vamos supor que haja um relacionamento *Pessoa saca Dinheiro*. Não há um estereótipo <<sacar>>, todavia ele pode ser criado e representado usando Diagramas de Perfil por meio da expressão <<estereótipo>>. Continuando: se eu não quiser representar um ator por meio de um *stickman*, eu posso utilizar um retângulo com o nome <<ator>> ou <<stickman>>.

Detalhe: um modelo tipicamente contém elementos que são instanciados a partir de um metamodelo. O papel típico de um metamodelo é definir a semântica para a forma de modelar elementos dentro de um modelo sendo instanciado. Um modelo captura uma visão de um sistema físico. Um modelo é uma abstração do sistema com um certo propósito.

Por exemplo: descrever aspectos estruturais ou comportamentais do software. Este propósito determina o que deve ser incluído no modelo e o que é irrelevante. Assim o modelo descreve completamente aqueles aspectos do sistema físico que são relevantes ao propósito do modelo, no nível apropriado de detalhe. Já um metamodelo define uma linguagem para expressar modelos.

O papel de um metamodelo é o de definir a semântica para modelar elementos dentro de um modelo sendo instanciado, dessa forma um modelo é uma instância

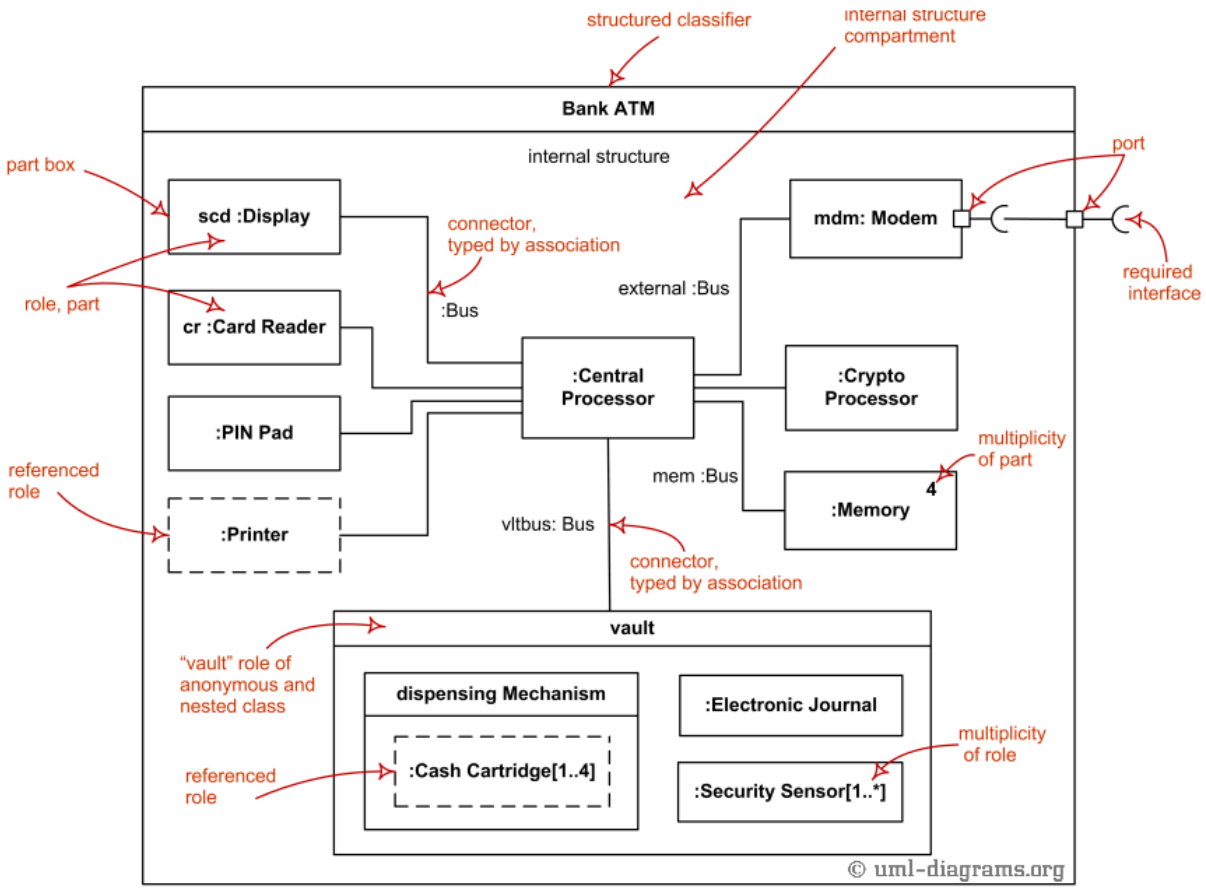
de um metamodelo. Já uma metaclasse é uma classe que pode ser estendida por meio de um ou mais estereótipos. Trata-se de uma classe do metamodelo (classe, interface, componente, associação, etc).

Em outras palavras, estereótipos podem ser aplicados a elementos em um Diagrama UML. Resumindo: um metamodelo é um modelo que modela outro modelo. A UML é um metamodelo no sentido de que ela possui diversos diagramas capazes de modelar a própria linguagem. Já uma metaclasse é uma classe capaz de especificar características comuns de várias outras classes.

DIAGRAMA DE ESTRUTURA COMPOSTA

O Diagrama de Estrutura Composta é utilizado para modelar colaborações internas de classes, interfaces e componentes para especificar uma funcionalidade. *O que é colaboração, professor?* É uma visão de um conjunto de entidades cooperativas interpretadas por instâncias que cooperam entre si para executar uma operação específica. Inclusive, uma colaboração pode conter outras colaborações.

Esse diagrama fornece meios de definir a estrutura de um elemento e de focalizá-la no detalhe, na construção e em relacionamentos internos. É um dos novos diagramas propostos na segunda versão de UML, voltado a detalhar elementos de modelagem estrutural, como classes, pacotes e componentes, descrevendo sua estrutura interna.



Quando utilizar o Diagrama de Estrutura Composta? As estruturas compostas entraram na UML 2.0, embora alguns métodos antigos tivessem algumas ideias semelhantes. Uma boa maneira de pensar a respeito da diferença entre pacotes e estruturas compostas é que os pacotes são um agrupamento em tempo de compilação e estruturas compostas mostram em tempo de execução.

Dessa forma, elas servem naturalmente para mostrar componentes e como eles são divididos em partes; assim, grande parte dessa notação é usada em diagramas de componentes. Sendo bastante sincero com vocês, até hoje esse diagrama não emplacou e, na verdade, sempre houve essa dúvida entre os membros do comitê organizador da UML.

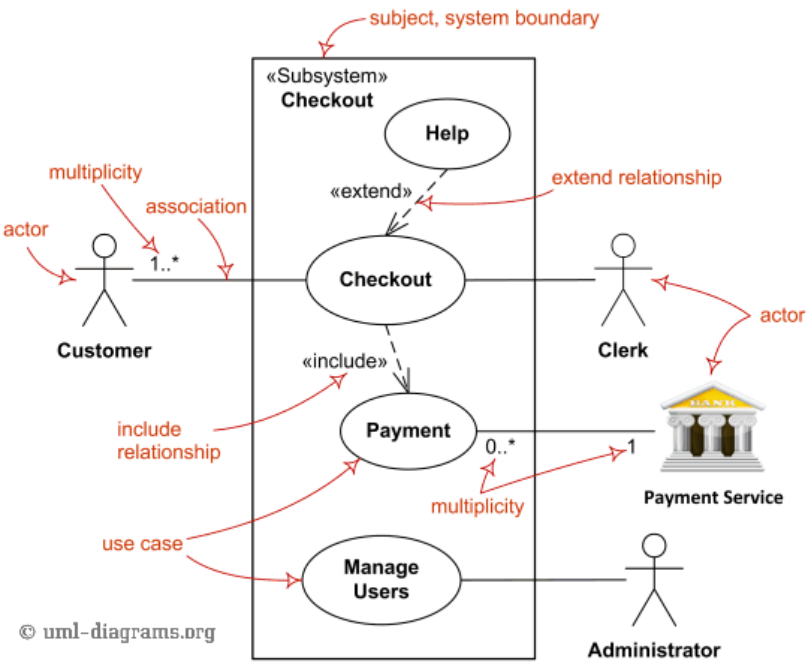
DIAGRAMAS COMPORTAMENTAIS

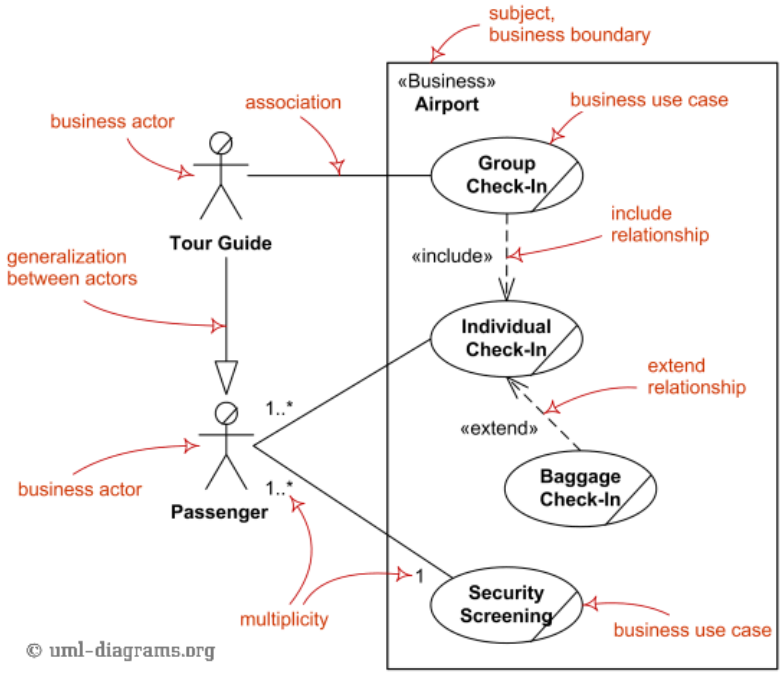
DIAGRAMA DE CASOS DE USO

Casos de Uso são uma técnica para captar os requisitos funcionais de um sistema. Eles servem para descrever as interações *de usuários com o sistema*, fornecendo uma narrativa sobre como o sistema é utilizado. *E o que é um cenário?* Cenário é uma instância de caso de uso, i.e., uma sequência de passos que descreve uma interação entre um usuário e o sistema.

Pensemos em uma loja virtual. O cenário de compra de um produto poderia ser: o cliente navega no catálogo e adiciona itens desejados à sua cesta de compras. Quando desejar pagar, descreve o endereço de entrega, fornece as informações do cartão de crédito e confirma a compra. O sistema verifica a autorização do cartão e confirma a venda.

Caso haja alguma falha em um dos passos, cria-se outro cenário. De todo modo, o Diagrama de Casos de Uso descreve um conjunto de funcionalidades do sistema e interações com elementos externos e entre si. Os Atores são os elementos externos que interagem com o sistema e são representados por um boneco (*Stickman*). Nas imagens abaixo, temos dois Diagramas de Casos de Uso (Sistema e Negócio):





Professor, eu pensava que um ator só podia ser um humano. Pois é, não é assim! Ele pode ser um humano, uma máquina ou outro sistema cuja interação executa uma ação significativa. Atores especificam um papel de uma entidade externa que se associam só entre si ou com casos de uso. Há quatro tipos de relacionamento relevantes!

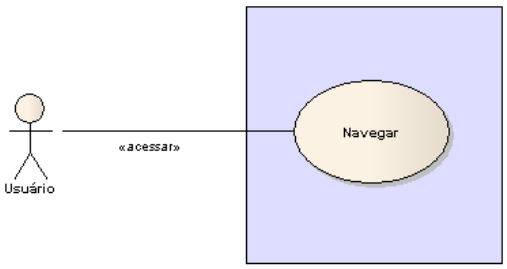
Um caso de uso conta uma história sobre como o usuário final interage com o sistema sob um conjunto de circunstâncias específicas. A história pode ser um texto narrativo, uma descrição geral de tarefas ou interações, uma descrição baseada em gabaritos ou uma representação esquemática. Independentemente de sua forma, um caso de uso representa o software ou sistema do ponto de vista do usuário final.

Há Casos de Uso Concretos e Abstratos. O primeiro é iniciado por um ator e constitui um fluxo completo de eventos, i.e., uma instância do caso de uso executa a operação inteira chamada pelo ator. O segundo jamais é instanciado diretamente, são incluídos, estendidos ou generalizados por outros casos de uso. Representa-se com nome em itálico!

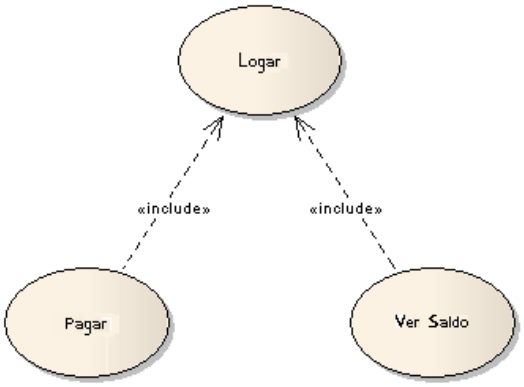
Há também Casos de Uso Primários e Secundários. O primeiro representa os objetivos dos atores. Esses casos de uso representam os processos da empresa que estão sendo automatizados pelo sistema de software. O segundo não traz benefício direto para os atores mas é necessário para que o sistema funciona adequadamente. Evidentemente, inicia-se pela identificação de casos de uso primários.

Quando um caso de uso concreto é iniciado, uma instância do caso de uso é criada. Ela também exibe o comportamento especificado por seus casos de uso abstratos associados. Logo, nenhuma instância separada é criada a partir de casos de uso abstratos. Essa distinção é importante, visto que os atores só podem enxergar casos de uso concretos.

- Relacionamento de Comunicação: também chamada de relacionamento de associação, o ator se comunica com o sistema por meio do envio e recebimento de mensagens. A imagem abaixo mostra a comunicação entre um ator e um caso de uso, representado por uma linha sólida no sentido do ator (Usuário) para o caso de uso (Navegar).



- Relacionamento de Inclusão: utilizado quando um mesmo comportamento se repete em mais de um caso de uso. A imagem abaixo apresenta o domínio de um *internet banking*. Observem que, para realizar um pagamento ou visualizar o saldo, é obrigatório que fazer *Login*. Logo, é um relacionamento obrigatório, representado por uma linha tracejada com uma seta na ponta[5].

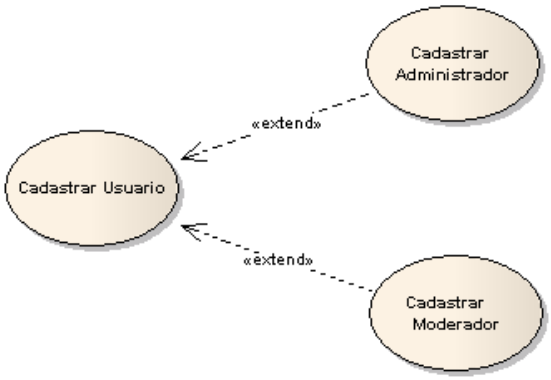


Explicando de uma forma mais simples de entender: quando o caso de uso A "inclui" o caso de uso B, significa que sempre sempre sempre sempre sempre que o caso de uso A for executado o caso de uso B também será executado. A

[5] A leitura é: o Caso de Uso *Pagar* e o Caso de Uso *Ver Saldo* incluem o Caso de Uso *Logar*, i.e., tanto para realizar pagamentos quanto para visualizar saldos, é obrigatório logar-se.

direção do relacionamento é do caso de uso que está incluindo para o caso de uso incluído.

- Relacionamento de Extensão: utilizado quando se deseja modelar um relacionamento alternativo. A imagem abaixo apresenta o contexto de um fórum de discussões. Observem que para cadastrar um usuário, há duas opções: moderador ou administrador. Logo, é um relacionamento opcional, representado por uma linha tracejada com uma seta na ponta[6].



Explicando de uma forma mais simples de entender: quando o caso de uso B estende o caso de uso A, significa que quando o caso de uso A for executado o caso de uso B poderá (poderá – talvez não seja) ser executado também. A direção do relacionamento é do caso de uso extensor (aqui o caso de uso B) para o caso de uso estendido (aqui o caso de uso A).

EM CASO DE DÚVIDA:

- **Inclusão:** A ------> B significa que A inclui B, ou seja, para realizar A, eu DEVO realizar B;
- **Extensão:** A ------> B significa que A estende B, ou seja, para realizar B, eu POSSO realizar A;

- Relacionamento de Herança: relacionamento entre atores, utilizado quando o ator filho é um uma especificação do ator genérico. É bastante útil para definir sobreposição de papéis entre atores e é representado com uma linha sólida com um triângulo no ator genérico. Na imagem abaixo, Vendedor é especialização de Pessoa. É representado por uma linha com um triângulo.

[6] A leitura é: o Caso de Uso *Cadastrar Administrador* e o Caso de Uso *Cadastrar Moderador* estendem a funcionalidade de Cadastrar Usuário, i.e., pode-se cadastrar usuário de duas maneiras distintas.

Abaixo segue uma tabela com as possibilidades de relacionamentos entre os elementos do modelo de casos de uso.

	Comunicação	Extensão	Inclusão	Herança
Entre Casos de Uso		X	X	X
Entre Atores				X
Entre Casos de Uso e Atores	X			

Quando utilizar Diagramas de Casos de Uso? Os casos de uso são uma ferramenta valiosa para ajudar no entendimento dos requisitos funcionais de um sistema. Uma primeira passagem nos casos de uso deve ser feita no início. Versões mais detalhadas dos casos de uso devem ser elaboradas apenas antes do desenvolvimento desse caso de uso.

É importante lembrar que casos de uso representam uma visão externa do sistema. Como tal, não espere quaisquer correlações entre eles e as classes dentro do sistema. Com os casos de uso, concentra-se energia mais no texto do que no diagrama. A despeito de a UML não dizer nada sobre o texto do caso de uso, é esse texto que contém todo o valor da técnica.

Um grande perigo dos casos de uso é que as pessoas os tornam complicados demais e não conseguem prosseguir. Normalmente, você terá menos problemas fazendo pouco do que fazendo demais. Umas duas páginas por caso de uso está bom, para a maioria das situações. Se você tiver muito pouco, pelo menos terá um documento curto e legível; se tiver demais, dificilmente alguém o lerá e entenderá.

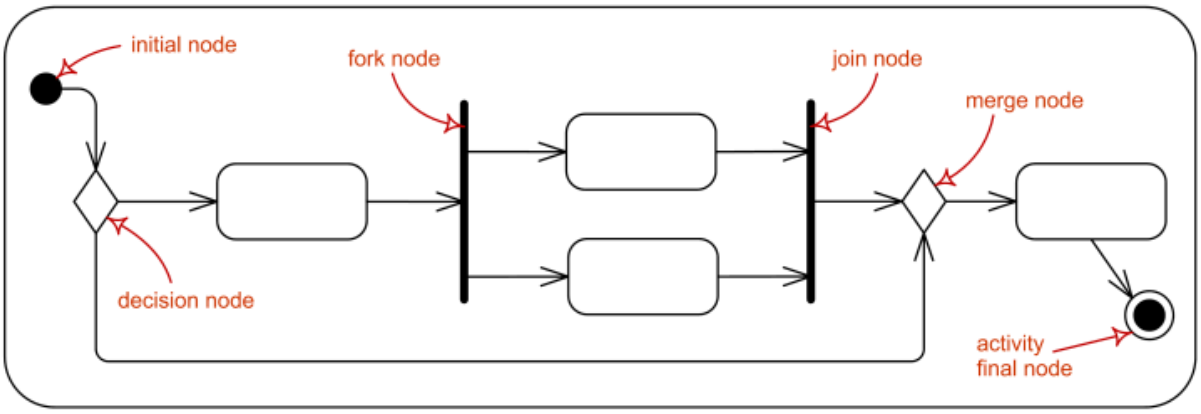
É importante salientar que Diagramas de Caso de Uso não são adequados para representar o desenho e não podem descrever mecanismos internos de um sistema.

DIAGRAMA DE ATIVIDADES

O Diagrama de Atividades descreve lógica de procedimento, processo de negócio e fluxos de trabalho. De várias formas, eles desempenham um papel semelhante aos fluxogramas, mas se diferenciam, pois suportam comportamentos paralelos. *Mas o que é uma atividade?* É um comportamento parametrizado representado como um fluxo coordenado de ações.

Notem que o diagrama de atividades tem um nível de abstração maior. Em outras palavras, pode-se dizer que ela não se preocupa com interações entre objetos, mas entre processos de negócios de mais alto nível. Os estados mudam quando uma ação é executada, ademais podem ser representados por meio de *swimlanes* (semelhante a raias de uma piscina olímpica).

Como apresenta a imagem abaixo, a *Swimlane* agrupa as atividades e as organizam de acordo com suas respectivas responsabilidades, com o auxílio de ações, bifurcações, fluxos e ramificações. São representadas como duas linhas paralelas, horizontais ou verticais, e seu nome em uma das extremidades. Qualquer nó de atividade entre essas linhas é considerado contido dentro de uma partição.



As ramificações especificam caminhos alternativos baseados em expressões booleanas – é representado com um diamante. A bifurcação é a divisão de um mesmo fluxo de controle em dois ou mais fluxos concorrentes: poderá ter uma única transição de entrada e duas ou mais transições de saída; abaixo da bifurcação, as atividades associadas com cada um dos caminhos prosseguem paralelamente.

Além da bifurgação, também temos a união. *O que seria isso, professor?* É a sincronização de dois ou mais fluxos de controle concorrentes: poderá ter duas ou mais transições de entrada e uma única transição de saída; uma barra de

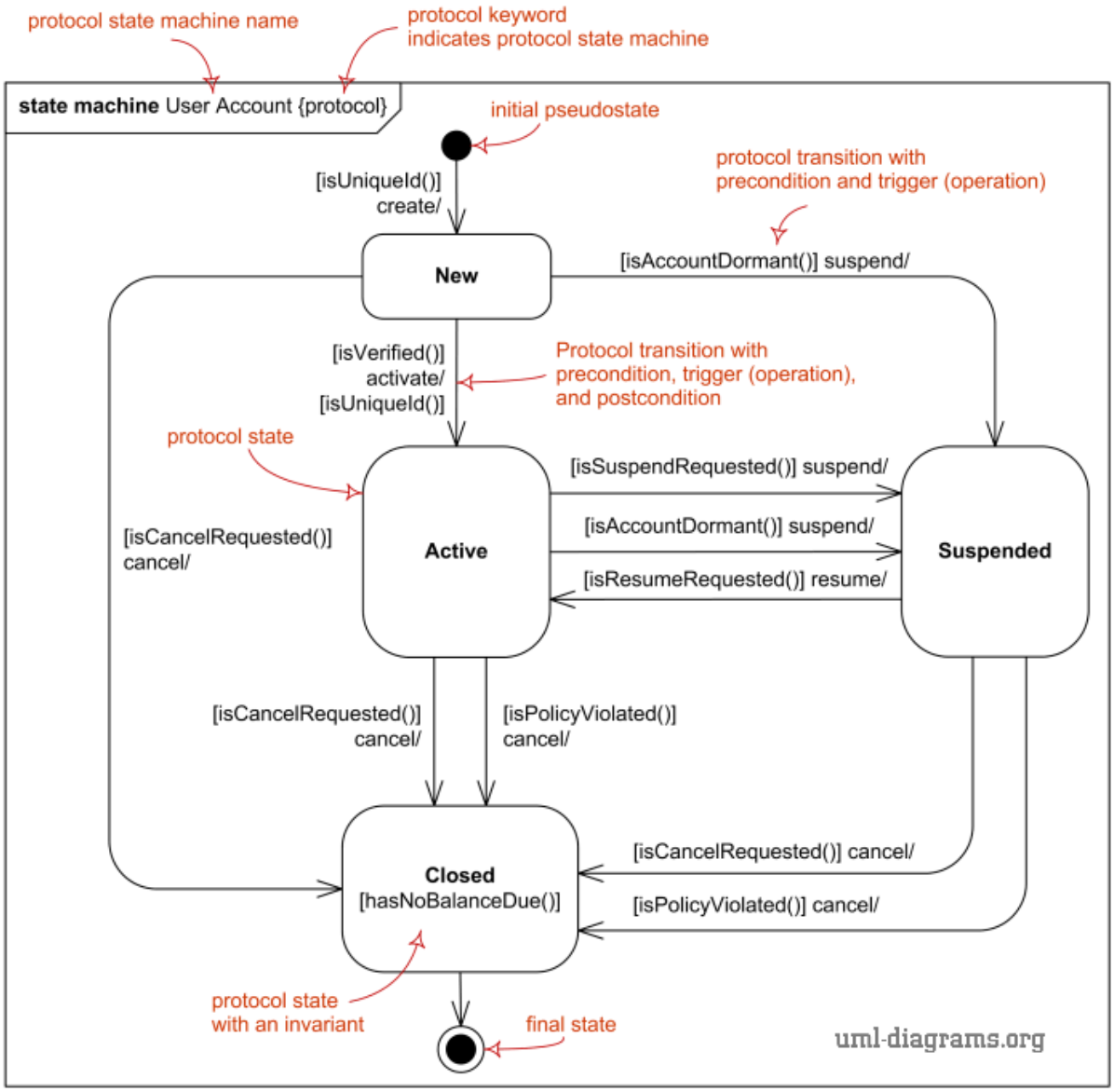
sincronização é usada para especificar bifurcação e união dos fluxos paralelos de controle. *Vocês entenderam?*

Quando utilizar o Diagrama de Atividades? A maior qualidade dos Diagramas de Atividades está no fato de que eles suportam e estimulam o comportamento paralelo. Isso os torna uma excelente ferramenta para modelagem de fluxos de trabalho e de processos. Na verdade, grande parte do avanço da UML 2.x é devido às pessoas envolvidas com fluxo de trabalho.

Você também pode usar um diagrama de atividades como um fluxograma compatível com a UML (mas suporta comportamento paralelo). Há iniciativas inovadoras que tentar utilizar a UML como uma linguagem de programação. *Como assim, professor?* A ideia é que ferramentas traduzam os diagramas diretamente em código executável – mas isso ainda está bastante incipiente.

DIAGRAMA DE MÁQUINA DE ESTADOS

Também conhecido como Diagrama de Transição de Estados, apresenta diversos estados possíveis de um objeto no decorrer da execução de processos de um sistema. Dessa forma, um objeto pode passar de um estado inicial para um estado final, por meio de uma transição, quando ocorre algum evento ou estímulo interno ou externo ao sistema.



O Diagrama de Máquina de Estados não mostra a interação entre objetos. Geralmente, ele mostra estados possíveis de um objeto específico. *O que é um Estado?* É a condição de um objeto em um determinado instante. *O que é uma Transição?* É a passagem de um estado para outro. *O que é uma ação?* É uma atividade que efetua a transição de estados.

Esse permite representar o ciclo de vida de objetos e como eles são afetados por eventos como erros, mensagens e condições. Eles se iniciam com um único estado inicial, mas podem ter vários estados finais. A imagem acima apresenta um objeto que faz pedidos de venda. Observem como é fácil de ler e auxilia a visualizar a complexidade do sistema.

É necessário distinguir Diagrama de Atividades e Diagrama de Máquina de Estados. No primeiro, o comportamento está expresso fundamentalmente nos nós do diagrama com cada nó representando um pedaço de comportamento. No segundo ocorre o contrário, ou seja, todo o comportamento se encontra nos arcos do diagrama.

Os nós do Diagrama de Estados representam o que está nos arcos do Diagrama de Atividades, e os nós dos Diagramas de Atividades representam o que está nos arcos dos Diagramas de Estado. Assim, apesar de visualmente bastante similares, do ponto de vista semântico, o que é representado em cada diagrama é exatamente o oposto um do outro.

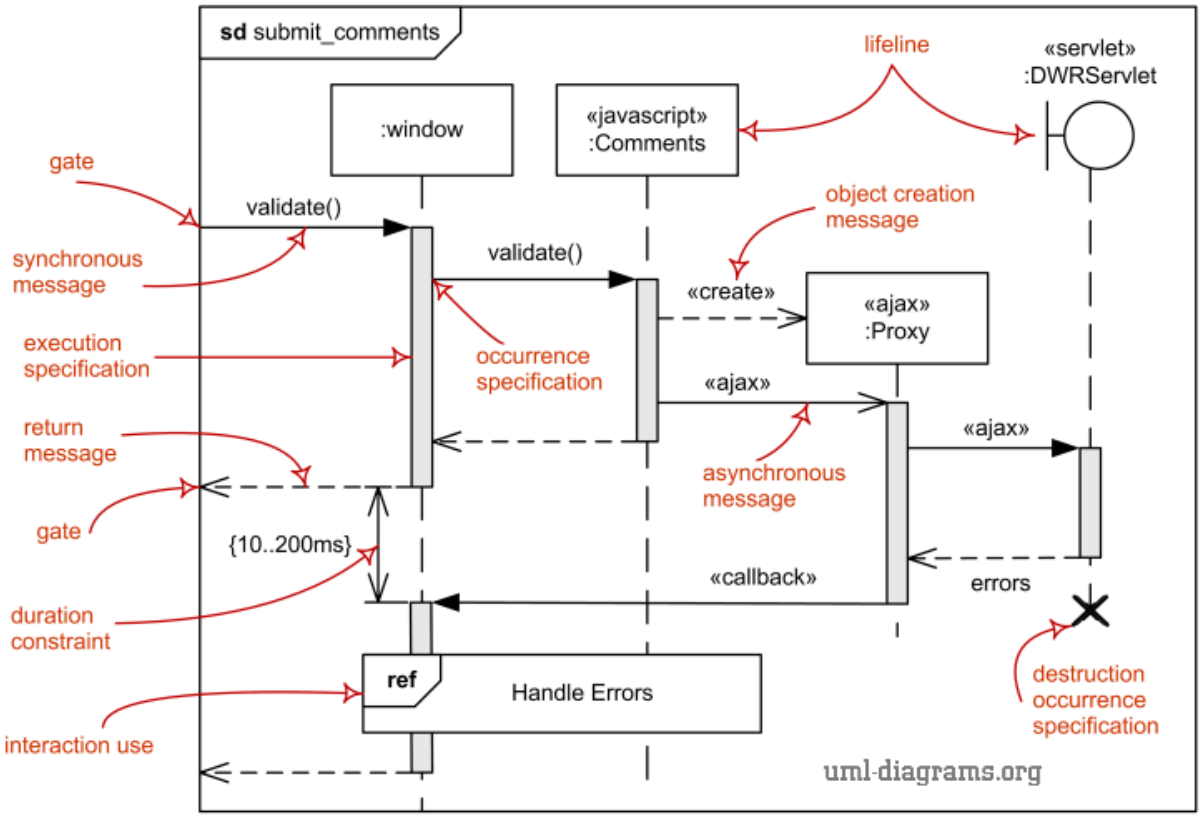
Quando utilizar Diagramas de Máquina de Estados? Eles são bons para descrever o comportamento de um objeto por intermédio de vários casos de uso. No entanto, esses diagramas são muito bons mesmo para descrever um comportamento que envolva vários objetos em colaboração. Para tal, é útil combinar diagramas de estados com outras técnicas.

Por exemplo, os Diagramas de Interação são bons para descrever o comportamento de vários objetos em um único caso de uso e os Diagramas de Atividades são bons para mostrar a sequência geral de atividades para vários objetos e casos de uso. Recomenda-se não o utilizar para todas as classes, mas apenaspara aquelas que exibem comportamento interessante e ajudem a entender o problema.

DIAGRAMA DE SEQUÊNCIA

O Diagrama de Sequência é um diagrama de interação que captura o comportamento de um único cenário, mostrando vários exemplos de objetos e mensagens que são trocadas dentro de caso de uso. Ele modela a interação entre os objetos, permitindo a visualização da execução de um ponto específico da aplicação, com ênfase na ordem temporal.

O diagrama de sequência possui dois eixos: horizontal e vertical. O primeiro representa os objetos envolvidos e o segundo representa o tempo em que a ação ocorre e é representado por uma linha tracejada (Linha de Vida). A imagem abaixo apresenta o diagrama de sequência de um caso de uso *Sacar* desde a inserção do cartão até o saldo ser gravado pelo banco.



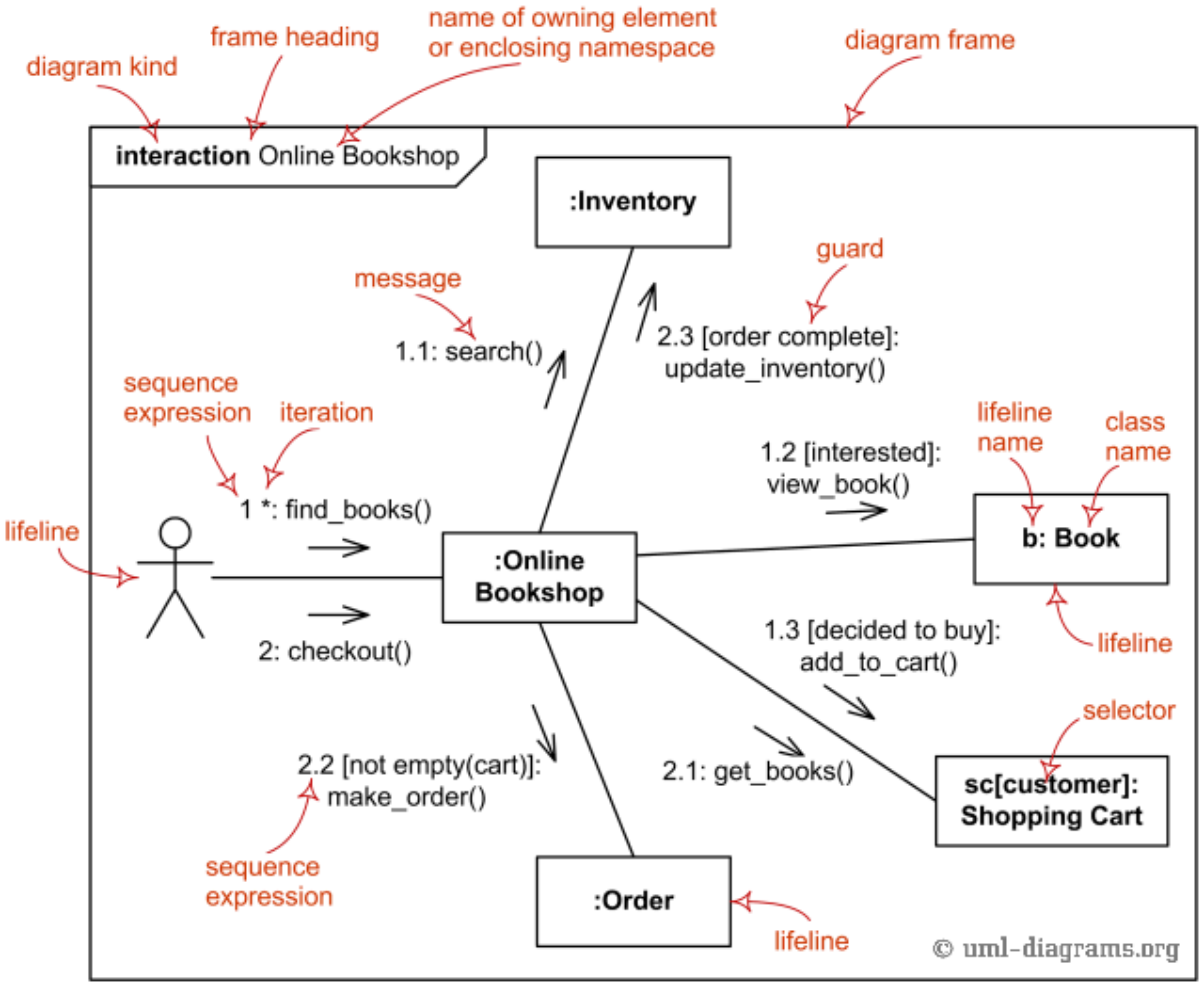
Eu disse que esse diagrama mostra a sequência temporal de mensagens trocadas entre objetos em um sistema. *Mas o que é uma mensagem?* É um serviço solicitado de um objeto para outro e suas respostas. Por exemplo, quando um objeto deseja utilizar um método definido por outro objeto, ele solicitando o serviço oferecido pelo outro enviando uma mensagem.

Quando utilizar os Diagramas de Sequência? Você deve utilizá-lo quando quiser observar o comportamento de vários objetos dentro de um único caso de uso. Eles são bons para mostrar as colaborações entre os objetos, mas não são tão bons para uma definição precisa do comportamento. Se quiser observar comportamento de um objeto em muitos casos de uso, utilize um Diagrama de Estados.

Se quiser observar o comportamento em muitos casos ou em muitas linhas de execução, considere o Diagrama de Atividades; se quiser explorar múltiplas interações alternativas rapidamente, talvez seja melhor usar cartões CRC, pois isso evita desenhar e apagar muitas vezes. Você explora alternativas de projeto com CRC e depois utiliza diagramas de sequência para capturar todas as interações.

DIAGRAMA DE COMUNICAÇÃO

O Diagrama de Comunicação também é uma espécie de diagrama de interação muito semelhante ao diagrama de sequência, mas com ênfase na ordem estrutural e, não, temporal. Em versões anteriores da UML, era conhecido como Diagrama de Colaboração[7]. *Mas como é possível verificar a ordem?* Ele possui números que identificam a sequência.



Geralmente, utiliza-se o diagrama de sequência para cenários mais complexos e o diagrama de comunicação para cenários mais simples. A imagem acima apresenta um cenário exatamente igual ao do Diagrama de Sequência, no entanto com foco na parte dinâmica do sistema. Observem que não há indicação cronológica, apenas sequencial (1, 2, 3, 4, 5...).

[7] Cuidado: O Diagrama de Comunicação era conhecido como Diagrama de Colaboração, mas ele não modela colaborações. *Quem modela colaborações, professor?* O Diagrama de Estrutura Composta!

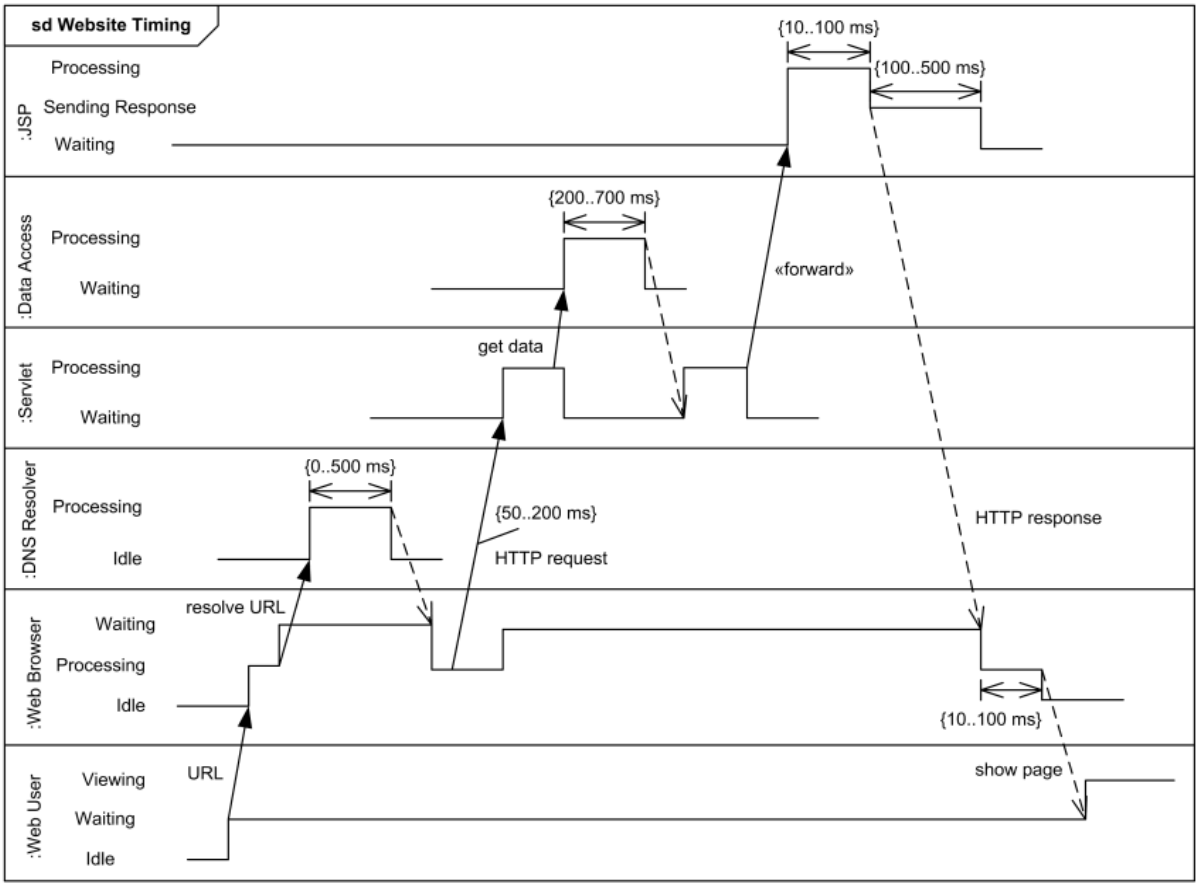
Sendo um tipo de diagrama de interação, o diagrama de comunicação mostra as mensagens trocadas entre objetos. Podemos dizer, inclusive, que o diagrama de comunicação é bastante semelhante estruturalmente ao diagrama de objetos, porém com setas e rótulos de mensagens nas ligações entre esses objetos. Um elemento particular do diagrama de comunicação é a ligação entre objetos.

Quando utilizar o Diagrama de Comunicação? A principal questão com os diagramas de comunicação é quando usá-los em lugar dos diagramas de sequência, mais comuns. Grande parte da decisão é questão de preferência pessoal: algumas pessoas gostam mais de um do que do outro. Frequentemente, isso direciona a escolha mais do que tudo.

Em geral, a maioria das pessoas parece preferir o Diagrama de Sequência. Uma abordagem mais racional diz que os diagramas de sequência são melhores quando você quer salientar a sequência de chamadas e que os Diagramas de Comunicação são melhores quando se quer salientar os vínculos. Os últimos são mais fáceis de alterar, de modo que eles são uma boa estratégia para explorar alternativas.

DIAGRAMA DE TEMPO

O Diagrama de Tempo (ou Temporização) também é uma espécie de diagrama de interação e descreve o comportamento dos objetos no decorrer do tempo e a duração na qual eles permanecem em determinados estados. Percebam que há um foco no tempo e na transição de estados, logo ele associa características do Diagrama de Sequência com o Diagrama de Máquina de Estados.



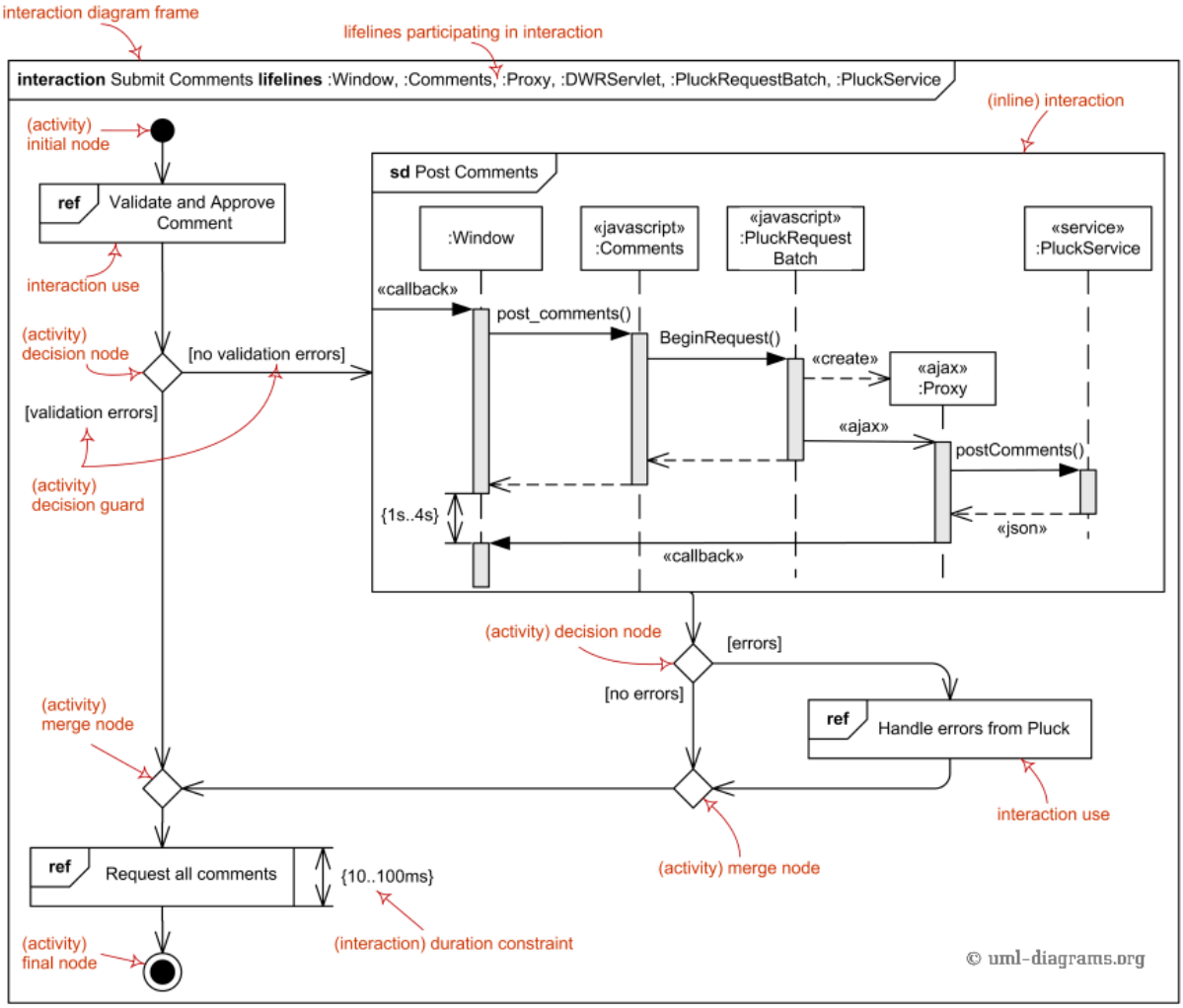
Os Diagramas de Tempo se focam nas restrições de temporização entre mudanças de estado em diferentes objetos: ou para um único objeto ou, de forma mais útil, para vários objetos. Esses diagramas são particularmente conhecidos dos engenheiros de hardware há um longo tempo. A imagem acima apresenta as restrições de tempo do diagrama.

O Diagrama de Tempo, ou de Temporização, descreve a mudança no estado ou na condição de uma instância de uma classe ou seu papel durante um tempo. É tipicamente utilizado para demonstrar a mudança no estado de um objeto no tempo em resposta a eventos externos. Esse diagrama somente passou a existir a partir da versão 2.0.

Quando utilizar os Diagramas de Temporização? Os Diagramas de Temporização são úteis para mostrar restrições de temporização entre mudanças de estado em diferentes objetos os diagramas são particularmente conhecidos dos Engenheiros de hardware. Há pouquíssimos exercícios sobre ele – acredito que talvez seja o diagrama menos cobrado em provas.

DIAGRAMA DE INTERACÃO GERAL

O Diagrama de Interação Geral (ou Visão Geral) é uma espécie de diagrama de interação que mistura o Diagrama de Atividades e o Diagrama de Sequência. Ele fornece uma visão geral do controle de fluxo entre objetos e engloba diversos tipos de diagramas de interação para demonstrar um processo geral. Ele mostra a troca de mensagens entre objetos e o estado de atividades.



Galera, os diagramas de interação preexistentes (Diagrama de Sequência e Diagrama de Comunicação) servem para representar as interações que ocorrem em um certo cenário de um caso de uso. Por outro lado, como seu próprio nome deixa transparecer, o Diagrama de Interação Geral tem o objetivo de dar uma visão geral dos diversos cenários de um caso de uso.

O renomado autor Martin Fowler afirma que pode-se considerar os diagramas de interação geral como diagramas de atividades nos quais as atividades são substituídas por pequenos diagramas de sequência ou como um diagrama de

sequência fragmentado, com a notação de diagrama de atividades usada para mostrar o fluxo de controle.

Quando utilizar os Diagramas de Interação Geral? Esses diagramas surgiram a partir da UML 2.0. Martin Fowler diz em seu livro que não gosta muito deles, pois acredita que eles misturam dois estilos que não se encaixam muito bem. De acordo com o autor, desenhe um Diagrama de Atividades ou use um Diagrama de Sequência, dependendo do que melhor atender seu propósito.

ÁREA DO ALUNO

<< Essa área é reservada para que os alunos enviem resumos, imagens, mnemônicos e dicas que podem auxiliar outros alunos >>

O queridíssimo aluno Jader Antônio Alves me mandou a seguinte mensagem:

Aluno: Jáder Antônio Alves

Aeee professor! Tudo certo?

Excelente teu curso, aborda o que cai de uma forma direta. Um assunto extenso como UML é complicado de tratar com objetividade, já comprei outros cursos inclusive bem mais completos, mas nenhum trata esse assunto dessa forma, ou são muito resumidos, ou são complexos e divagam bastante.

Pergunta: A ideia dos mnemônicos foi ótima. To usando uns para os padrões de projeto da GOF e usam imagens, percebi que as imagens dão um grande auxílio na memorização.

Fiz algumas para memorizar com tuas frases, ajudaram bastante, se quiser utilizar em uma apostila no futuro fique à vontade. Acho que vão ajudar outros alunos.

Imagem: https://i.imgur.com/9ucGSq7.png

Valeu! Abs!

Claudio Está Com Objetivo Implícito de Persistir no Pacífico
Classes Estrutura Composta Componentes Objetos Implantação Perfis Pacotes

O Ativista Internacional Comunicou o Tempo do Casório ao Maquinista Sequelado
Atividades Caso de Uso Máquina de Estados Sequência Comunicação Interação Geral Tempo

OS EXERCÍCIOS DE UML ESTARÃO NA PRÓXIMA AULA

(PARA A AULA NÃO FICAR MUITO GRANDE!)

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.