

DESCRIÇÃO

Visão geral da modelagem do domínio da aplicação com a UML, por meio de modelo de Casos de Uso, modelo de Classes, Diagrama de Objetos e Diagrama de Pacotes.

PROPÓSITO

Aprender as ferramentas básicas de modelagem conceitual de um sistema usando a UML é fundamental para a construção e a especificação sólida dos requisitos de um sistema, bem como o entendimento profundo sobre o domínio do negócio em que esse sistema esteja inserido.

PREPARAÇÃO

Antes de iniciar este conteúdo, instale em seu computador um programa que o permita elaborar modelos sob a forma de diagramas da UML (Linguagem Unificada de Modelagem). Nossa sugestão inicial é o Astah Free Student License, e será necessário usar seu e-mail institucional para ativar a licença. Preencha os dados do formulário, envie e aguarde a liberação de sua licença em seu e-mail institucional. Ao receber a licença, siga as instruções do e-mail e instale o produto em seu computador.

Sugestões de links adicionais de programas livres para modelagem de sistemas em UML podem ser encontradas em buscas na internet.

OBJETIVOS

MÓDULO 1

Identificar requisitos funcionais de um sistema com uso de Diagrama de Casos de Uso

MÓDULO 2

Reconhecer casos de uso por meio de especificações textuais

MÓDULO 3

Identificar classes e seus relacionamentos em um domínio de aplicação com uso de Diagrama de Classes

MÓDULO 4

Empregar Diagramas de Objetos e de Pacotes para apoiar a especificação de um sistema

INTRODUÇÃO

Aprenderemos técnicas e notação para modelagem de requisitos e conceitos de domínio de sistemas. A *Unified Modeling Language* (UML) ou Linguagem de Modelagem Unificada é uma linguagem ou notação visual para especificação de sistemas orientados a objetos. Essa notação vem sendo adotada como padrão para modelagem em organizações, pois tem como objetivo principal prover um conjunto de ferramentas poderosas para análise e projeto de sistemas.

O foco deste conteúdo são os Diagramas de Casos de Uso, Diagrama de Classes e Objetos e Diagrama de Pacotes, com seus respectivos conceitos e elementos, muito úteis para arquitetos, analistas e projetistas de software na fase inicial do processo de desenvolvimento: a especificação de requisitos e modelagem conceitual do domínio.

Em Engenharia de Software, a palavra domínio denota ou agrupa um conjunto de sistemas ou de áreas com funcionalidades similares dentro dos sistemas. Assim, é possível definir domínio aplicativo como uma coleção de aplicações de software com determinadas características em comum. Logo, domínio é um conjunto de características de uma família de problemas que determinada aplicação pretende solucionar.

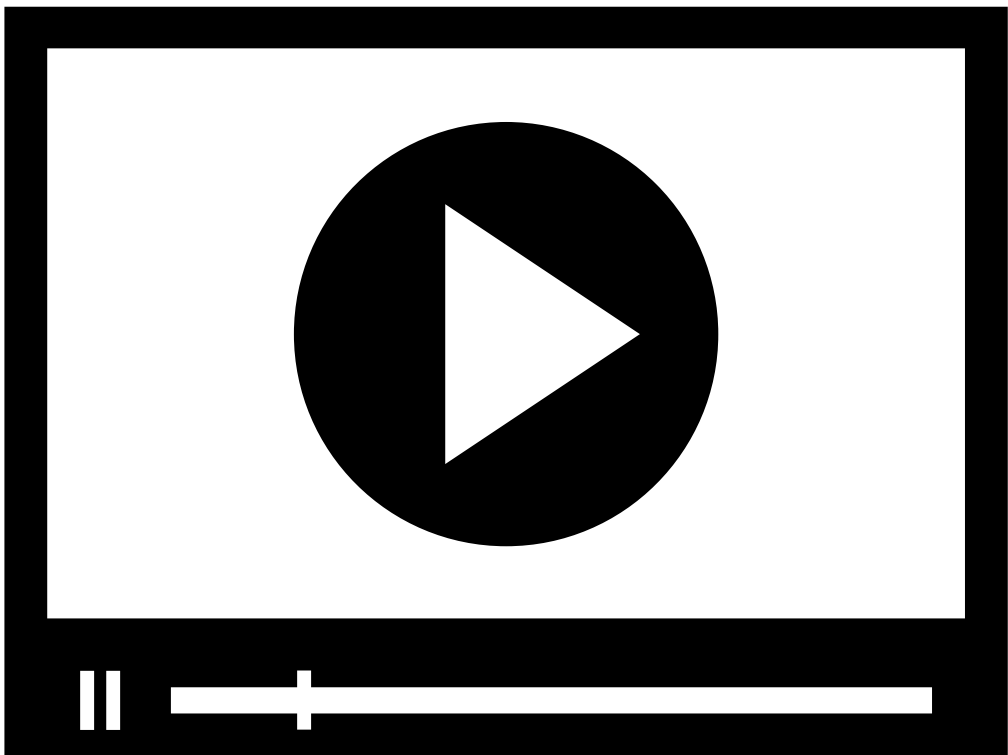
ATENÇÃO

Como se observará ao longo do conteúdo, essa definição de domínio não deve ser confundida com o domínio de um atributo de classe, que é o conjunto de valores possíveis que esse atributo pode assumir.

MÓDULO 1

- Identificar requisitos funcionais de um sistema com uso de Diagrama de Casos de Uso

REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS



DESVENDANDO REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS: TEORIA E PRÁTICA

Neste vídeo, vamos explorar os conceitos de Requisitos Funcionais e Não Funcionais. Vamos explicar o que são, por que são cruciais no desenvolvimento de software e como identificá-los na prática. Prepare-se para dominar esses elementos essenciais para o sucesso de qualquer projeto de software.



O desenvolvimento de um sistema começa pelo entendimento do problema de negócio e o levantamento de quais requisitos esse sistema de software deve satisfazer para resolvê-lo. Não existe uma definição padronizada para requisito na literatura de Engenharia de Software. A norma ISO/IEC/IEEE (IEEE, 1990) apresenta os seguintes enunciados para explicar o que é requisito:

Uma condição ou capacidade do sistema solicitada por um usuário, para resolver um problema ou atingir um objetivo.

Uma condição ou capacidade que deve ser atendida por uma solução para satisfazer um contrato, uma especificação, um padrão ou quaisquer outros documentos formais impostos.

Documentação da representação das condições ou capacidades apresentadas nos dois anteriores.

Necessidades quantificadas e documentadas, desejos e expectativas do patrocinador, clientes e outras partes interessadas.

Os requisitos são o ponto de partida para todo o processo de desenvolvimento, sendo a base para as atividades de análise e projeto, bem como implementação e testes. De acordo com Sommerville (2011), a área de Engenharia de Software faz uma distinção, em termos de nível de detalhamento de requisitos, entre requisitos de usuário e requisitos de sistema. O primeiro expressa os requisitos abstratos de alto nível, e o segundo revela uma descrição detalhada do que o sistema deve fazer.

REQUISITOS DE USUÁRIO SÃO DECLARAÇÕES, EM UMA LINGUAGEM NATURAL COM DIAGRAMAS, DE QUAIS SERVIÇOS O SISTEMA DEVERÁ FORNECER A SEUS USUÁRIOS E AS RESTRIÇÕES COM AS QUAIS ESTE DEVE OPERAR.

SOMMERVILLE, 2011

REQUISITOS DE SISTEMA SÃO DESCRIÇÕES MAIS DETALHADAS DAS FUNÇÕES, SERVIÇOS E RESTRIÇÕES OPERACIONAIS DO SISTEMA DE SOFTWARE.

SOMMERVILLE, 2011

Em uma perspectiva mais técnica, os requisitos de software são normalmente classificados como requisitos funcionais e requisitos não funcionais.

REQUISITOS FUNCIONAIS SÃO DECLARAÇÕES DE SERVIÇOS QUE O SISTEMA DEVE FORNECER, DE COMO O SISTEMA DEVE REAGIR A ENTRADAS ESPECÍFICAS E DE COMO O SISTEMA DEVE SE COMPORTAR EM DETERMINADAS SITUAÇÕES.

SOMMERVILLE, 2011

ATENÇÃO

Requisito funcional é uma funcionalidade do sistema, é algo que o sistema deve realizar para prover um resultado para o usuário. Requisitos funcionais pressupõem interação do sistema com usuários.

Veja uma lista com alguns exemplos de enunciados dos chamados requisitos funcionais:

O usuário deve ser capaz de pesquisar por nome na base de dados disponível.

O sistema deve designar um identificador único (NUMERO_PEDIDO) para cada pedido do usuário.

O sistema deve apresentar o documento selecionado para leitura do usuário.

Perceba que, em todos os exemplos, estamos definindo atividades que o sistema deve realizar, e, em todas essas atividades, o usuário está envolvido, seja fornecendo entradas ou recebendo saídas (resultados).

A especificação de um sistema precisa considerar também como esses requisitos serão entregues. Ou seja, quais características de qualidade e quais restrições desejamos impor ao sistema. Assim, definimos os chamados requisitos não funcionais.

REQUISITOS NÃO FUNCIONAIS SÃO RESTRIÇÕES AOS SERVIÇOS OU FUNÇÕES OFERECIDOS PELO SISTEMA. INCLUEM RESTRIÇÕES DE TIMING, RESTRIÇÕES NO PROCESSO DE DESENVOLVIMENTO E RESTRIÇÕES IMPOSTAS PELAS NORMAS. AO CONTRÁRIO DAS CARACTERÍSTICAS INDIVIDUAIS OU SERVIÇOS DO SISTEMA, OS REQUISITOS NÃO FUNCIONAIS, MUITAS VEZES, APLICAM-SE AO SISTEMA COMO UM TODO.

SOMMERVILLE, 2011

Requisitos não funcionais (RNF) são frequentemente mais críticos do que os requisitos funcionais (RF). Deixar de atender a um RNF pode significar a inutilização de todo o sistema. Os RNF podem afetar a arquitetura geral de um sistema em vez de apenas componentes individuais. No caso de um website, um único RNF pode originar uma série de RF relacionados, que, por sua vez, definem os serviços necessários.

Existem diversas classificações que visam a organizar e prover melhor entendimento sobre os RNF. Um exemplo desse tipo de classificação é apresentado na lista a seguir.

2

Requisitos organizacionais: ambientais, operacionais, processo de desenvolvimento.

3

Requisitos externos: reguladores, éticos, legais, contábeis.

★ EXEMPLO

Aqui estão alguns exemplos de enunciados de requisitos não funcionais:

O sistema deve estar disponível durante todas as horas normais de trabalho (2ª a 6ª, 8h30 às 17h). Períodos de não operação dentro desses intervalos não podem ultrapassar 2 segundos em um dia.

O sistema deve requerer autenticação dos usuários para permitir acesso.

O sistema deve implementar disposições de privacidade dos usuários de acordo com a Lei Geral de Proteção de Dados.

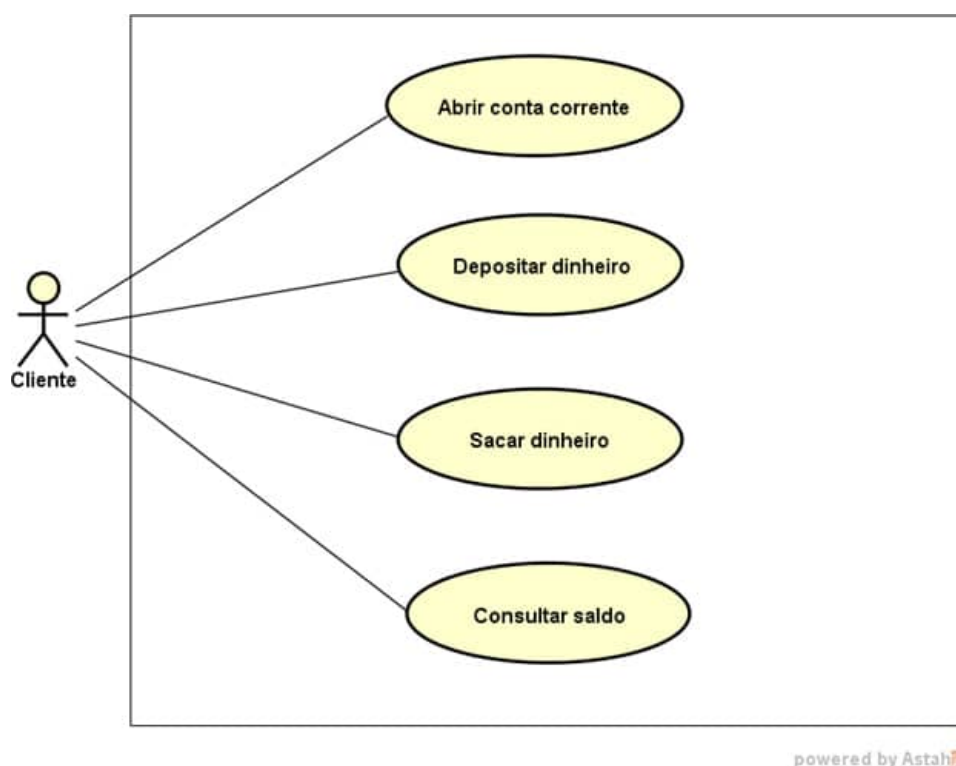
Você pode perceber, nesses exemplos, que esses requisitos não indicam o que o sistema realiza, mas como ele entrega uma ou mais de suas funcionalidades.

DIAGRAMA DE CASOS DE USO

A visão de casos de uso da UML apresenta o conjunto de funções (ou usos) que o sistema deve executar para atender às necessidades dos usuários. No modelo de casos de uso, o sistema é visto sob a perspectiva do usuário. A documentação dos requisitos funcionais em um Diagrama de Casos de Uso dá início ao processo de análise do sistema e permite entender sua abrangência e complexidade.

O objetivo do Diagrama de Casos de Uso é apresentar uma visão geral dos requisitos funcionais do sistema e, portanto, ele é o digrama mais abstrato, flexível e simples da UML.

Esse diagrama é composto pelos seguintes elementos: **casos de uso**, **atores** e **relacionamentos**. A figura 1 mostra um exemplo de Diagrama de Casos de Uso para um sistema bancário.



📷 Figura 1 – Exemplo de Diagrama de Casos de Uso, elaborado com ferramenta ASTAH (2021).

Os possíveis usos de um sistema são representados pelos casos de uso no diagrama. Um **Caso de Uso** pode estar diretamente associado a um ou mais requisitos funcionais, identificados na etapa de levantamento de requisitos do processo de desenvolvimento.

Conforme observado na figura 1, o caso de uso assume a forma gráfica de uma elipse contendo o seu nome no interior; nela, podemos observar quatro casos de usos representados:

Abrir conta corrente

Depositar dinheiro

Sacar dinheiro

Consultar saldo

O caso de uso é o elemento fundamental do Modelo de Casos de Uso.

📢 ATENÇÃO

Embora não exista um padrão para nomear casos de uso, a boa prática recomenda utilizar uma frase verbal iniciando com verbo no infinitivo, denotando a ação prevista na funcionalidade representada pelo Caso de Uso.

O segundo elemento do Diagrama de Casos de Uso é o **Ator**, que é representado no Diagrama de Casos de Uso por um boneco palito, com o nome na parte inferior.

O ator é quem utiliza ou interage em um caso de uso, ou seja, é uma entidade externa que interage com o sistema, mas que não faz parte dele.

Interagir significa trocar informações com o sistema, seja fornecendo ou recebendo essas informações. Na figura 1, temos a representação do ator **Cliente**, que interage com os quatro casos de uso. Por exemplo, no caso de uso **Consultar Saldo**, o ator Cliente informa os dados de sua conta e recebe como resposta do sistema o valor de seu saldo.

O terceiro elemento do Diagrama de Casos de Uso são os **Relacionamentos**, que permitem associar:

Casos de uso com seus atores

Casos de uso com outros casos de uso

Atores com outros atores

⊕ SAIBA MAIS

A forma mais comum e obrigatória dos relacionamentos é entre atores e casos de uso, que assume a representação gráfica de segmento de reta (uma linha cheia) ligando o ator ao Caso de Uso no qual ele interage.

Na figura 1, observamos que o ator Cliente interage com os quatro casos de uso presentes nesse modelo, portanto, esses elementos gráficos estão ligados por uma reta.

A seguir, vamos detalhar cada um desses elementos.

ATORES

Medeiros (2004) explica que atores atuam em casos de uso realizando atividades. O ator não é uma entidade específica, mas representa um papel ou aspecto particular de uma entidade. O ator pode representar papéis executados por usuários humanos, um componente de

hardware, outros sistemas com os quais o sistema em desenvolvimento se comunica. Desse modo, o nome do ator deve expressar esse papel de forma clara. No exemplo da figura 1, o ator é o papel de cliente do banco. Esse papel pode ser assumido pelos diversos clientes do banco.

RESUMINDO

Durante a execução de uma funcionalidade do sistema (um ou mais requisitos funcionais), o ator precisa prover entradas (informações) para o sistema e deve receber respostas dele.

O ator não é uma entidade específica, mas representa um papel ou aspecto particular de uma entidade. O ator pode representar papéis executados por usuários humanos, um componente de hardware, outros sistemas com os quais o sistema em desenvolvimento se comunica. Desse modo, o nome do ator deve expressar esse papel de forma clara. No exemplo da figura 1, o ator é o papel de cliente do banco. Esse papel pode ser assumido pelos diversos clientes do banco.

Tipos de atores que geralmente aparecem nos sistemas são:

CARGOS

ORGANIZAÇÕES E SUAS DIVISÕES

OUTROS SISTEMAS

HARDWARE

CARGOS

Por exemplo: empregado, cliente, gerente, vendedor, comprador etc.

ORGANIZAÇÕES E SUAS DIVISÕES

Por exemplo: fornecedor, agência reguladora, administradora de cartões, setor de vendas etc.

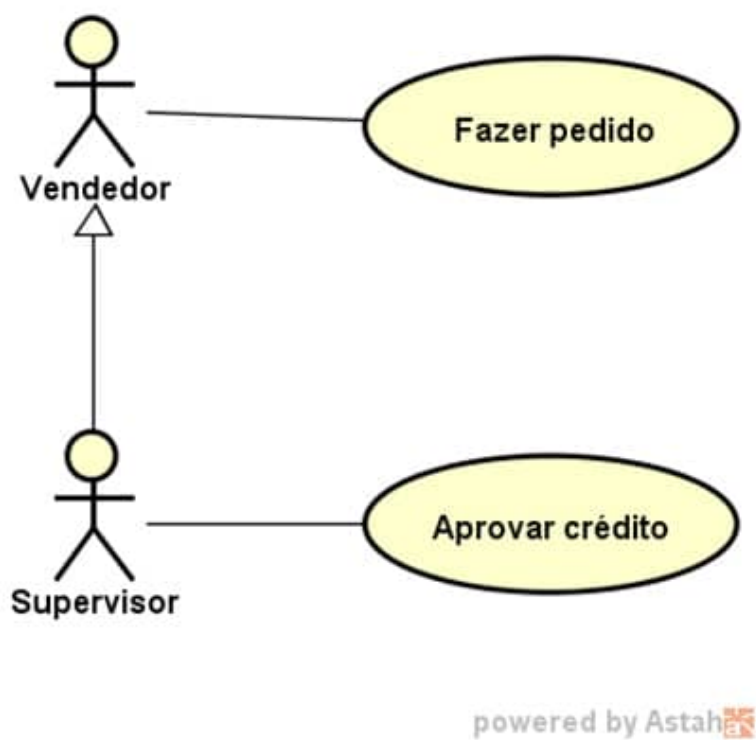
OUTROS SISTEMAS

Por exemplo: sistema de vendas, sistema de cobrança, sistema de pagamento etc.

HARDWARE

Por exemplo: sensor de temperatura, leitora de código de barras etc.

O relacionamento de generalização entre atores indica que uma instância do primeiro ator interage com os mesmos tipos de instância de casos de uso do segundo. Porém, o contrário não é verdade. Os casos de uso associados ao ator mais específico nesse relacionamento são exclusivos desse ator, que herda os casos de uso associados ao ator mais genérico. A figura 2 mostra um exemplo desse tipo de relacionamento, no qual percebemos que tanto o ator Vendedor quanto o ator Supervisor interagem com o caso de uso Fazer pedido. No entanto, apenas o Supervisor participa do caso de uso Aprovar crédito.



📷 Figura 2 – Exemplo do relacionamento de generalização entre atores, elaborado com ferramenta ASTAH (2021). Imagem: Flavia Maria Santoro.

Normalmente, o ator inicia a sequência de interações em um caso de uso. Portanto, a forma mais fácil de começar a construir um modelo de casos de uso é identificar todos os atores.

O analista de sistemas pode observar as fontes que devem fornecer informações a serem processadas pelo sistema e para quem os resultados do processamento devam ser fornecidos.

Bezerra (2015) sugere algumas perguntas úteis para guiar os analistas de sistemas:

1. QUE ÓRGÃOS, EMPRESAS OU PESSOAS UTILIZARÃO O SISTEMA?

2. QUE SISTEMAS OU EQUIPAMENTOS VÃO SE COMUNICAR COM O SISTEMA A SER CONSTRUÍDO?

3. ALGUÉM DEVE SER INFORMADO DE ALGUMA OCORRÊNCIA NO SISTEMA?

4. QUEM ESTÁ INTERESSADO EM CERTO REQUISITO FUNCIONAL DO SISTEMA?

A partir daí, durante a identificação dos casos de uso, o analista poderá perceber a existência de outros atores não identificados nessa etapa.

CASO DE USO

Caso de Uso é a descrição de uma sequência de interações entre um sistema e um ou mais atores externos a esse sistema. De acordo com Bezerra (2015), os casos de usos permitem a um observador externo ao sistema entender quais são as funcionalidades fornecidas pelo sistema e quais são os resultados externos produzidos por elas. Porém, não é possível saber como o sistema trabalha internamente para produzir esses resultados. Um modelo de casos de uso pode conter vários casos de uso, dependendo do tamanho e da complexidade do sistema.

ATENÇÃO

Casos de uso não se limitam à sua representação gráfica; tão importante quanto isso é a sua descrição textual. As descrições devem explicar de forma clara, simples e de fácil entendimento, o que o sistema faz, sem entrar nos detalhes de como ele deve fazê-lo. Casos de uso facilitam o processo de comunicação com usuários e são utilizados para documentar a interação do sistema com eles. Desse modo, casos de uso são úteis, tanto para os profissionais envolvidos no desenvolvimento do sistema (analistas, projetistas, programadores, testadores) quanto para seus usuários e clientes. A descrição textual de um caso de uso é uma espécie de história por meio da qual contamos como um usuário realiza uma ação utilizando o sistema.

Casos de uso podem relatar as diversas formas de utilização de uma funcionalidade do sistema. Assim, podemos definir cada uma dessas formas por meio de cenários.

Cenário pode ser entendido como a instância de execução de um caso de uso. Portanto, diversos cenários para o mesmo caso de uso podem ser escritos. Os cenários podem ser usados para a especificação dos casos de testes do sistema.

Para identificar os casos de uso, podemos pensar em dois tipos que sempre estarão presentes nos sistemas: primário e secundário.

CASOS DE USO PRIMÁRIOS

Estão relacionados diretamente aos objetivos dos atores, ou seja, são as funcionalidades que agregam valor aos usuários. Bezerra (2015) sugere uma lista de perguntas para auxiliar os analistas de sistemas na descoberta dos casos de uso primários de um sistema:

Quais são as necessidades e os objetivos de cada ator em relação ao sistema?

Que informações o sistema deve produzir?

O sistema deve realizar alguma ação que ocorre regularmente no tempo?

Para cada requisito funcional, existe um caso de uso (ou mais) para atendê-lo?

Alguns casos de uso são iniciados não por um ator, mas por um evento temporal. Desse modo, normalmente o ator é o tipo de entidade que recebe a informação resultante (por exemplo, um cliente que receba uma notificação do sistema).

CASOS DE USO SECUNDÁRIOS

Esses casos não apresentam uma relação direta de valor para os atores, mas são imprescindíveis para o funcionamento do sistema. Por exemplo, a manutenção de cadastros (inclusão, alteração, consulta e exclusão das diversas informações que alimentam o sistema). Um sistema bancário precisa cadastrar as informações sobre os clientes.

RELACIONAMENTOS

Os relacionamentos são responsáveis por estabelecer as associações entre atores e casos de uso.

Todos os atores devem estar associados explicitamente aos casos de uso com os quais interagem. Além disso, podemos estabelecer relacionamentos entre os casos de uso ou entre os atores do sistema. O Modelo de Casos de uso permite os seguintes tipos de relacionamentos: comunicação, inclusão, extensão e generalização.

RELACIONAMENTO DE COMUNICAÇÃO

É uma linha simples que mostra a que caso de uso determinado ator está associado, ou seja, esse ator interage com o sistema por meio do caso de uso. Um ator pode se relacionar com mais de um caso de uso, e todos os atores devem estar associados a pelo menos um caso de uso. Esse relacionamento foi mostrado na figura 1.

📢 ATENÇÃO

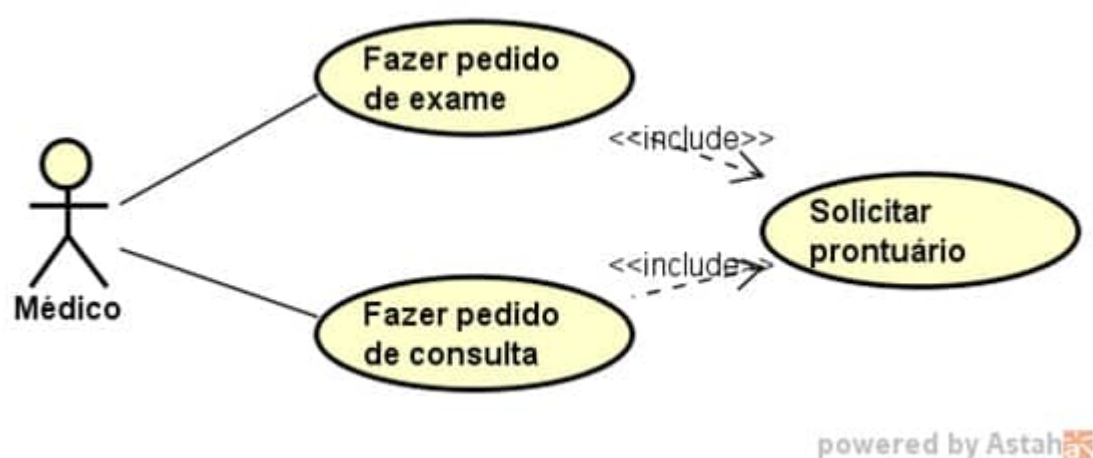
Um erro comum na elaboração de diagramas de casos de uso é estabelecer relacionamentos de comunicação entre casos de uso. Não existe troca de informações entre casos de uso, uma vez que são funcionalidades independentes entre si.

Os relacionamentos de inclusão (include), extensão (extend) e generalização são estabelecidos entre casos de uso.

RELACIONAMENTO DE INCLUSÃO

Tem como objetivo principal a reutilização. Suponha que uma parte dos passos de execução de um caso de uso se repita em diversos outros casos de uso. Podemos criar um caso de uso separado e “incluir-lo” nos demais. A inclusão funciona de forma semelhante à chamada de uma função em um programa. A representação gráfica do relacionamento de inclusão é uma linha pontilhada com o estereótipo <<include>> apontando do caso de uso inclusor para o incluído.

A figura 3 mostra um exemplo desse tipo de relacionamento.



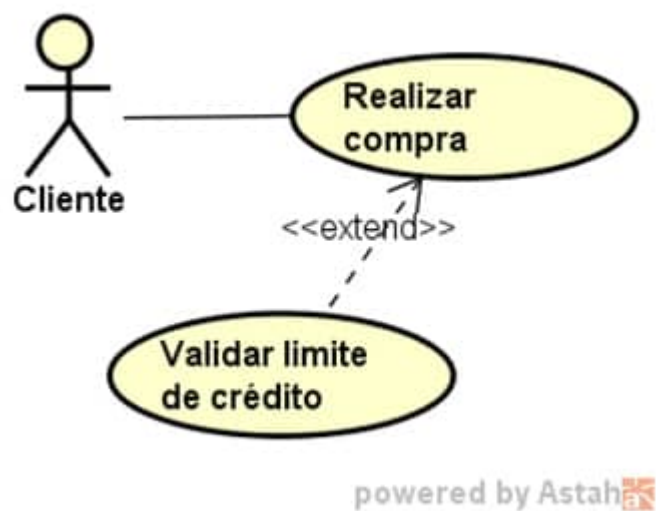
📷 Figura 3 – Exemplo do relacionamento de inclusão entre casos de uso, elaborado com ferramenta ASTAH (2021)

No exemplo da figura 3, o ator Médico participa de dois casos de uso: Fazer pedido de exame e Fazer pedido de consulta. Em ambos os casos de uso, é necessário realizar os passos do caso de uso Solicitar prontuário. Portanto, este caso de uso é incluído nos dois primeiros.

RELACIONAMENTO DE EXTENSÃO

Tem como objetivo representar situações em que diferentes sequências de passos possam ser inseridas no mesmo caso de uso. Ou seja, dependendo de uma condição, o caso de uso segue um caminho diferente. Desse modo, um caso de uso estende outro, permitindo um comportamento eventual. Note que a existência do caso de uso estendido deve ser independente da existência de quaisquer casos de uso que o estendam.

A figura 4 mostra um exemplo do relacionamento de extensão.



📷 Figura 4 – Exemplo do relacionamento de extensão entre casos de uso, elaborado com ferramenta ASTAH (2021)

Observe no exemplo da figura 4, em que o ator Cliente interage com o caso de uso Realizar compra. O caso de uso Validar limite de crédito pode ser necessário nesse contexto, portanto, é estendido para Realizar compra. Ambos os casos de uso representam funcionalidades independentes no sistema.

RELACIONAMENTO DE GENERALIZAÇÃO

Pode existir entre dois casos de uso ou entre dois atores (conforme vimos em Atores). Esse relacionamento permite que um caso de uso (mais específico) herde características de outro (mais genérico).

Na generalização entre casos de uso, quando um caso de uso herda de outro, significa que as sequências de passos de execução do caso de uso mais genérico (pai) valem também para o mais específico (filho). Além disso, o caso de uso herdeiro participa de qualquer associação do qual o caso de uso mais genérico participe. Isso significa que um ator que interage com o caso de uso pai pode também interagir em qualquer caso de uso filho.

A figura 5 mostra um exemplo do relacionamento de generalização:



📷 Figura 5 – Exemplo do relacionamento de generalização entre casos de uso, elaborado com ferramenta ASTAH (2021)

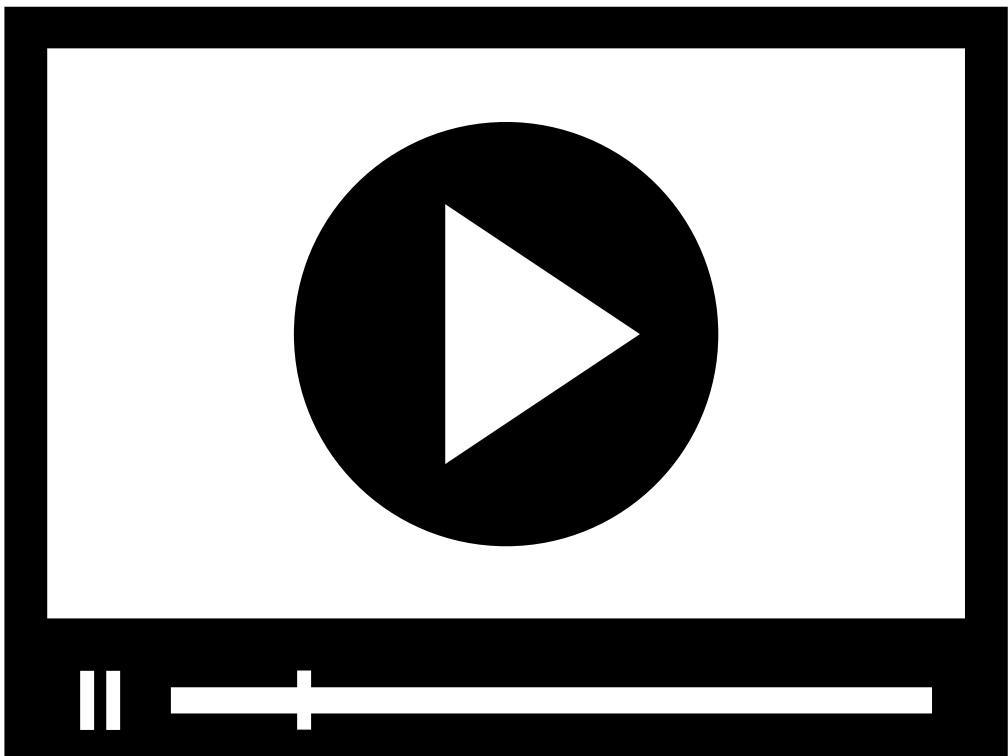
Como podemos observar na figura 5, o caso de uso Cadastrar empregado executa uma sequência de passos de cadastro para qualquer funcionário. Já os casos de uso mais específicos (Cadastrar vendedor e Cadastrar supervisor) herdam esses passos e ainda possuem passos específicos para os tipos de funcionários Vendedor e Supervisor, respectivamente.

Os tipos de relacionamento entre casos de uso abordados permitem manter a descrição dos casos de uso mais simples e sem repetições.

A inclusão deve ser usada quando um comportamento se repetir em mais de um caso de uso.

A extensão é normalmente usada quando um comportamento que não é executado em todas as instâncias de um caso de uso deva ser descrito.

A generalização deve ser aplicada em uma situação em que dois ou mais casos de uso possuam comportamentos comuns.



A IMPORTÂNCIA DO MODELO DE CASOS DE USO NA ANÁLISE DE REQUISITOS

A especialista Flavia Maria Santoro fala sobre o modelo de casos de uso, sua finalidade, seus elementos e sua importância na especificação de requisitos funcionais.



VERIFICANDO O APRENDIZADO

MÓDULO 2

-
- ⦿ Reconhecer casos de uso por meio de especificações textuais

ESPECIFICAÇÃO E FORMATOS DE CASOS DE USO

Os casos de uso incluídos pelos analistas no Diagrama de Casos de Uso devem ser especificados em um formato de narrativa textual, para complementar a representação gráfica sob forma de diagramas. Desse modo, o Modelo de Casos de Uso é formado pelo diagrama e pelo conjunto de todas as descrições dos casos de uso.

A descrição textual permite o entendimento mais preciso de qual funcionalidade do sistema trata aquele caso de uso. Ela serve como ponto de partida para construção de outros modelos, tais como, Modelo de Classes, ainda na fase de análise, e o Diagrama de Sequência, na fase de projeto do desenvolvimento do sistema.

COMENTÁRIO

A UML não propõe uma descrição formal de casos de uso e não existe uma forma padronizada para a escrita dessa especificação. Os analistas e, muitas vezes, as empresas definem seus padrões com base em exemplos, boas práticas e necessidades particulares de cada projeto de desenvolvimento. As especificações podem ser bem simples, abstraindo os detalhes da execução do caso de uso, ou mais complexas, incluindo todo o passo a passo a ser seguido para obter os resultados esperados.

Uma vez que não existe um padrão definido pela UML, a descrição textual de um caso de uso pode ser feita em diferentes formatos. O que há em comum em todos esses formatos é a **forma de expressão narrativa**.

O texto de um caso de uso é uma história que conta as ações ou a sequência de passos de um ou mais atores e do sistema para atingir determinado objetivo. Essa narrativa pode ser escrita em um parágrafo contínuo, em passos numerados ou em uma tabela contendo uma coluna para cada ator e uma coluna para o sistema.

ATENÇÃO

A descrição de um caso de uso deve ser clara, livre de jargões e termos técnicos, portanto, de fácil entendimento não só para a equipe de desenvolvimento, mas para os usuários finais, pois o modelo de casos de uso é lido e validado por todos os envolvidos.

FORMATO CONTÍNUO

A narrativa é um texto livre. Como um exemplo, considere o caso de uso Consultar saldo na figura 1. Uma especificação em formato contínuo para esse caso de uso seria:

Este caso de uso inicia quando o Cliente chega ao caixa eletrônico e insere seu cartão. O Sistema requisita a senha do Cliente. Após o Cliente fornecer sua senha e esta ser validada, o Sistema exibe as opções de operações possíveis. O Cliente opta por consultar saldo. O Sistema exibe o saldo da conta e o caso de uso termina.

FORMATO NUMERADO

Aqui, a descrição desse caso de uso seria:

1	O Cliente insere cartão no caixa eletrônico.
2	O Sistema solicita a senha.
3	O Cliente digita senha.
4	O Sistema valida a senha e exibe menu de operações disponíveis.
5	O Cliente seleciona a opção consultar o saldo.
6	O Sistema exibe o saldo da conta.

FORMATO TABULAR

Cliente	Sistema
---------	---------

Insere cartão no caixa eletrônico.	
	Solicita a senha.
Digita senha.	
	Valida a senha e exibe menu de operações disponíveis.
Seleciona a opção consultar o saldo.	
	Exibe o saldo da conta.

 **Atenção!** Para visualizaçãocompleta da tabela utilize a rolagem horizontal

CENÁRIO PRINCIPAL E CENÁRIOS ALTERNATIVOS

A especificação de um caso de uso descreve a sequência de passos de interação entre o sistema e seus atores. A descrição do exemplo do caso de uso Abrir conta corrente, na figura 1, mostra uma sequência, um fluxo ou passo a passo mais usual, ou seja, aquela em que acontece exatamente o que era esperado nessa interação.

+ SAIBA MAIS

Essa sequência é chamada de cenário (ou fluxo) principal do caso de uso. Nesse cenário, o ator consegue atingir seu objetivo seguindo o caminho “feliz”; em outras palavras, nada de excepcional acontece. De acordo com Lima (2005), é o fluxo mais óbvio, no qual todas as ações são bem-sucedidas.

Cada caso de uso possui apenas um cenário principal e ele deve ser descrito de forma clara e concisa, sem uso de jargão computacional. Não se esqueça: casos de uso devem ser escritos do ponto de vista do usuário e, sempre que possível, com os termos que os usuários possam compreender.

Agora, imagine que, em nosso exemplo, o Cliente digitou a senha errada. O que aconteceria no fluxo descrito? Nada, pois esse problema não é previsto nesse cenário!

Para essa situação e algumas outras, o modelo de casos de uso prevê a especificação dos **cenários alternativos e cenários de exceção**.

CENÁRIOS ALTERNATIVOS

Nesta situação, o ator pode escolher formas diferentes para chegar ao seu objetivo. Portanto, deve ser possível, apesar de não obrigatório, especificar cenários alternativos. Em nosso exemplo da consulta de saldo, o Cliente poderia optar por imprimir o saldo em vez de consultá-lo diretamente.

Um cenário alternativo não é uma exceção, é outra forma de fazer a mesma coisa!

CENÁRIOS DE EXCEÇÃO

Devem ser explicitados exceções, problemas ou possíveis erros. Um cenário de exceção representa uma anomalia não prevista no cenário principal, como o Cliente digitar sua senha errada. O analista deve escrever cenários para tratar esse e outros desvios do curso normal de um caso de uso, ou mesmo cancelar a realização do caso de uso em questão.

De acordo com Bezerra (2015), esses cenários possuem as seguintes características:

Representam erros de operação no cenário principal.



Não têm sentido fora do contexto do caso de uso no qual ocorre (ou seja, não são casos de uso em si).



Precisam indicar em que passo o caso de uso continua ou, conforme for, indicar que o caso de uso está finalizado.

No exemplo do caso de uso Abrir conta corrente, um cenário de exceção para a situação em que o Cliente digitasse sua senha errada poderia ser especificado da seguinte forma:

CENÁRIO PRINCIPAL

O Cliente insere cartão no caixa eletrônico.

O Sistema solicita a senha.

O Cliente digita senha.

O Sistema valida a senha e exibe menu de operações disponíveis.

O Cliente seleciona a opção consultar o saldo.

O Sistema exibe o saldo da conta.

CENÁRIO DE EXCEÇÃO “SENHA INCORRETA”

No passo 4, caso a senha seja incorreta:

4a. O Sistema informa que a senha está incorreta e solicita que o Cliente digite novamente.

4b. O caso de uso retorna ao passo 3.

Observe que o cenário de exceção indica o ponto do cenário principal no qual o problema pode ocorrer e o ponto exato para onde o fluxo deve retornar. O analista deve especificar todas as possíveis exceções ou problemas usando esses tipos de cenários.

ESTRUTURA DE ESPECIFICAÇÃO DE CASOS DE USO


Como a UML não define um padrão para especificação de casos de uso, você já observou que existem algumas formas práticas de apresentar a narrativa ou o passo a passo de execução do caso de uso. No entanto, é comum as empresas ou as equipes de desenvolvimento definirem seus próprios padrões de estrutura e os tipos de informação a serem apresentados, de modo a manter a qualidade dos documentos de requisitos de software. Nesse sentido, você encontrará diversas sugestões de modelos em livros ou sites na internet.

Um exemplo bastante útil é a estrutura descrita por Bezerra (2015). Veja, no quadro 1, uma adaptação dessa proposta, que destaca os campos que poderiam compor a descrição de um caso de uso. Você pode tomar como base essa proposta e criar um padrão para seus projetos.

1. Identificador: código único para cada caso de uso que permite fazer a rastreabilidade com outros elementos e modelos. Por exemplo, com casos de teste e regras de negócio, UC01.

<p>2. Nome: cada caso de uso tem um nome único que o identifica; é o nome que aparece no Diagrama de Casos de Usos. O nome do caso de uso é uma frase verbal iniciada por verbo no infinitivo. É importante ter cuidado com coerência e clareza ao nomear um caso de uso. Por exemplo, o nome do caso de uso UC01 é “Criar conta corrente”.</p>
<p>3. Objetivo: declaração sucinta do objetivo do ator no caso de uso. Para o caso de uso Criar conta corrente, o Objetivo seria: “O objetivo deste caso de uso é permitir que o Cliente do banco crie uma nova conta corrente”.</p>
<p>4. Ator primário: nome do ator que inicia o caso de uso ou ator que recebe o resultado do caso de uso; um caso de uso possui apenas um ator primário. Para o caso de uso Criar conta corrente, o Ator primário seria: “Cliente”.</p>
<p>5. Atores secundários: nomes dos demais atores que atuam no caso de uso. Para o caso de uso Criar conta corrente, um ator secundário seria: “Gerente da agência”.</p>
<p>6. Pré-condições: condições assumidas como verdadeiras para que o caso de uso tenha início; um caso de uso pode conter zero ou mais pré-condições. Por exemplo, em um caso de uso Criar conta corrente, uma pré-condição poderia ser “Cliente cadastrado com CPF válido”.</p>
<p>7. Cenário (ou fluxo) principal: sequência de passos que normalmente acontece quando o caso de uso é utilizado com sucesso; toda descrição de caso de uso deve ter um fluxo principal.</p>
<p>8. Cenários (ou fluxos) alternativos e de exceção: sequência de passos que apresentam formas de execução diferentes da descrita no fluxo principal ou situações de exceções e erros que devem ser tratadas pelo sistema. Não há obrigatoriedade de existirem fluxos alternativos ou de exceção em um caso de uso.</p>
<p>9. Pós-condições: estado do sistema após o caso de uso ter sido executado; tipicamente, uma pós-condição descreve que uma informação foi alterada, excluída ou inserida no sistema. Uma boa prática é usar verbo no particípio passado para descrever pós-condições. Por exemplo, em um caso de uso Criar conta corrente, uma pós-condição seria “conta corrente criada”.</p>
<p>10. Regras de negócio: a descrição de um caso de uso pode fazer referência a uma ou mais regras de negócio. Aprenda mais sobre Regras de Negócio em Explore +.</p>

 **Atenção!** Para visualizaçãocompleta da tabela utilize a rolagem horizontal

 Quadro 1 – Proposta de estrutura para especificação de casos de uso. Quadro: Bezerra (2015), adaptado por Flavia Maria Santoro.

ESPECIFICAÇÃO DE CASOS DE USO COM INCLUSÃO E EXTENSÃO

A UML também não define uma forma padrão para a escrita de casos de uso de inclusão ou extensão. No entanto, geralmente fazemos referência ao caso de uso incluído/estendido na descrição do caso de uso que o inclui/estende.

CASOS DE USO COM INCLUSÃO

Para a situação dos casos de uso de inclusão, a seguinte sintaxe pode ser usada: Include (nome do caso de uso incluso) no ponto da descrição da sequência de interações em que o caso de uso deve ser inserido.

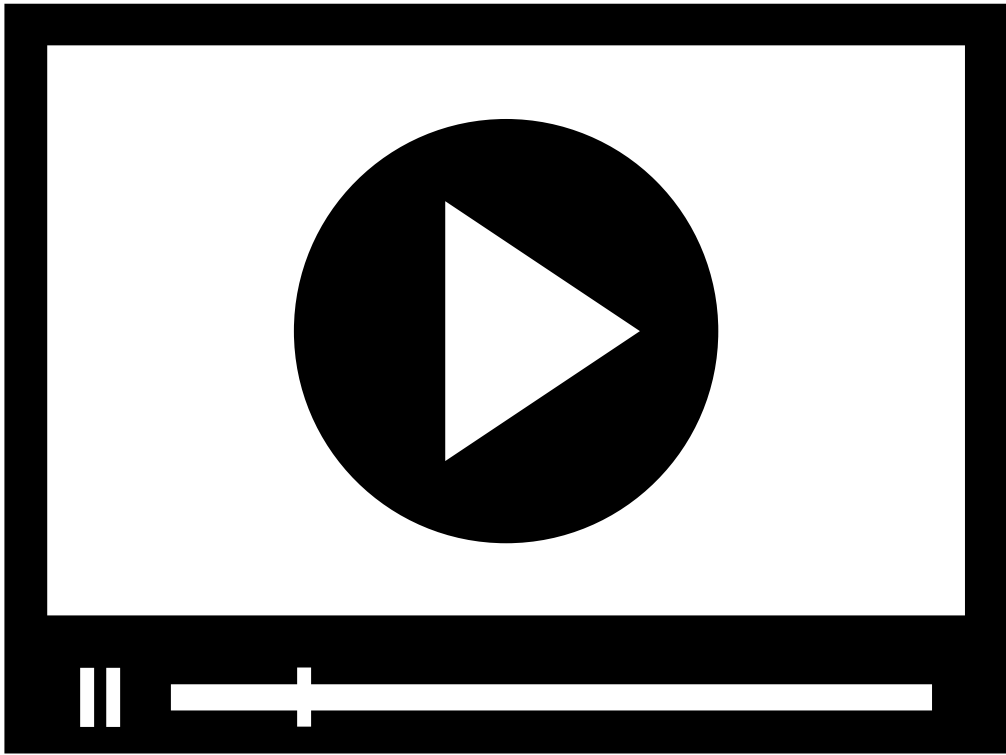
CASOS DE USO COM EXTENSÃO

Para a **situação dos casos de uso de extensão**, devemos lembrar que o caso de uso estendido existe independentemente de outros casos de uso que o estendam. Ele possui a descrição completa de sua sequência de passos. Se o ator escolhe acessar o caso de uso estendido, ele executa o fluxo de ações desse caso de uso e retorna para o ponto do cenário em que estava antes.

A especificação dos pontos de extensão em um caso de uso é feita normalmente dentro da descrição textual do caso de uso extensor. A descrição de um ponto de extensão deve indicar em que ponto (ou pontos) do caso de uso estendido pode-se inserir o comportamento do extensor, e podem existir vários pontos no estendido em que o extensor pode ser ativado.

📢 ATENÇÃO

Além da inclusão e extensão, um relacionamento de generalização requer uma forma de especificação, na descrição do caso de uso herdeiro, de quais passos do caso de uso pai estão sendo redefinidos. O analista pode criar marcadores no caso de uso pai. Esses marcadores podem, então, ser referenciados na descrição do caso de uso filho para especificar que passos estão sendo redefinidos ou onde está sendo definida uma extensão do comportamento do pai.



DESCRIÇÕES TEXTUAIS DE CASOS DE USO

A especialista Flavia Maria Santoro fala sobre as formas de especificação dos casos de uso por meio de descrições textuais.



VERIFICANDO O APRENDIZADO

MÓDULO 3

🕒 Identificar classes e seus relacionamentos em um domínio de aplicação com uso de Diagrama de Classes

CONCEITO E ELEMENTOS DE UMA CLASSE

A abordagem de desenvolvimento de sistemas orientada a objetos assume que um sistema é uma coleção de entidades – os objetos – que se comunicam e realizam ações. A interação entre objetos é a forma de execução de uma funcionalidade do software.

Essa abordagem se baseia nos seguintes princípios listados por Bezerra (2015):

Qualquer coisa é um objeto.

Objetos realizam tarefas por meio da requisição de serviços a outros objetos.

Cada objeto pertence a uma classe, ou seja, classe é uma entidade que agrupa e abstrai objetos similares.

Uma classe funciona com repositório para comportamento associado aos objetos instanciados a partir dela.

Classes são organizadas em hierarquias.

📢 ATENÇÃO

A UML é a linguagem padrão que permite a elaboração dos diversos modelos e das perspectivas de um sistema na abordagem da orientação a objetos. Classes (e seus objetos) são os componentes básicos dessa abordagem.

Objetos são entidades do mundo real que podemos observar. Por exemplo, um objeto pode ser um carro, uma fruta, uma empresa, um empregado da empresa, um concerto de música, um restaurante, um pedido de comida nesse restaurante, entre muitos outros.

Quando pensamos nesses objetos em termos de grupos de coisas, por exemplo, cada pessoa que assiste a aulas em uma turma é um objeto diferente, mas, se abstrairmos esses objetos, observando suas características comuns, criamos o conceito de Aluno, o qual é denominado, então, de Classe. Uma classe de objetos com características comuns, tais como: estão matriculados em uma turma e assistem a aulas.

Uma classe pode ser definida como uma descrição de atributos (características) e serviços (ações) comuns a um grupo de objetos.

Bezerra (2015) ressalta que uma classe pode ser entendida como uma espécie de molde, a partir do qual objetos podem ser construídos. Portanto, é comum afirmarmos que um objeto é uma instância de uma classe. Cada objeto do mundo real é único, uma classe abstrai o tipo de objeto e permite que novos objetos sejam criados a partir desse modelo. Se uma nova pessoa se matricula na turma, ela é um novo objeto instanciado a partir da classe Aluno.

Quando estamos tratando da modelagem de um sistema, somente um subconjunto de características do grupo de objetos pode ser relevante. Portanto, uma classe representa uma abstração das características relevantes do mundo real para aquele conjunto de objetos.

★ EXEMPLO

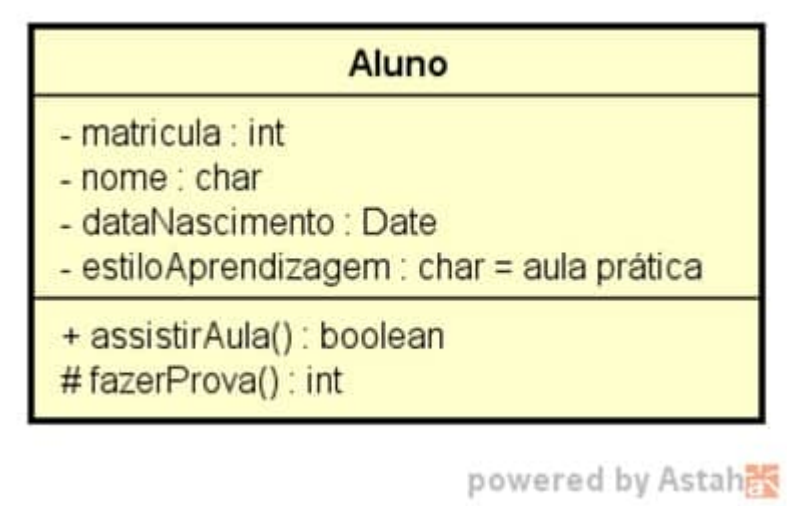
Voltando aos nossos alunos, as características relevantes desses objetos, para um sistema acadêmico, poderiam ser número de matrícula, idade e estilo de aprendizagem; mas outras, tais como o esporte favorito ou número do calçado, talvez não façam parte do interesse desse sistema.

A estrutura de uma classe é composta de atributos e de operações.

Os atributos descrevem os dados que deverão ser armazenados pelos objetos dessa classe (suas características). Cada atributo de uma classe é de um tipo de dados e possui um conjunto de valores que esse atributo pode assumir, chamado de domínio. Veja um exemplo na

figura 6.

As operações apresentam as ações que os objetos de uma classe podem realizar. Como são ações, costumam ser nomeadas com uso de um verbo e um complemento, e terminam com um par de parênteses (funções em linguagens de programação). Ao contrário dos atributos, que para cada objeto têm o seu próprio valor, os objetos de mesma classe compartilham as suas operações. Em outras palavras, todos os objetos de uma classe realizam as mesmas ações. Veja o exemplo de uma operação da classe Aluno na figura 6.



📷 Figura 6: Exemplo de Classe e seus atributos e operações, elaborado com ferramenta ASTAH (2021)

Na figura 6, observamos que a representação de uma classe é feita por meio de um retângulo com três compartimentos: nome da classe, lista de atributos e lista de operações. Você também pode perceber a sintaxe de especificação dos atributos e operações.

Um atributo é especificado no seguinte formato:

[VISIBILIDADE] NOME: TIPO = VALOR_INICIAL

Visibilidade de um atributo diz respeito ao seu nível de acesso, ou seja, permite definir quais atributos de um objeto são acessíveis por outros objetos. Essa propriedade serve para implementar o encapsulamento da estrutura interna da classe. A visibilidade de um atributo pode ser:

PÚBLICA

(+)

PROTEGIDA

(#)

PRIVATIVA

(-)

DE PACOTE

(~)

Somente as propriedades que são realmente necessárias ao exterior da classe devem ser definidas com visibilidade pública. Todo o resto é escondido dentro da classe por meio das visibilidades protegida (visível para subclasses da classe em que o atributo foi definido) e privativa (invisível externamente à classe em que o atributo está definido). Com visibilidade de pacote, o atributo é visível a qualquer classe que pertença ao mesmo pacote no qual está definida a classe. Como veremos no módulo seguinte, um pacote é um agrupamento de diagramas UML.

O elemento nome, única parte obrigatória na sintaxe do atributo, corresponde ao nome do atributo. O elemento tipo, que especifica o tipo do atributo, é dependente da linguagem de programação na qual a classe será implementada. Por exemplo: int, float, boolean. No exemplo da figura 6, o atributo matrícula é do tipo int.

É possível declarar o valor inicial que o atributo assume quando um objeto daquela classe for instanciado. Desse modo, sempre que um objeto de uma classe é instanciado, o valor inicial declarado é automaticamente definido para o atributo. Em nosso exemplo, quando um novo aluno é inserido no sistema, assumimos que seu estiloAprendizagem é aula prática.

Uma operação é especificada no seguinte formato:

VISIBILIDADE NOME(PARÂMETROS): TIPO-RETORNO {PROPRIEDADES}

O elemento nome corresponde ao nome dado à operação, em que normalmente usamos verbos que denotem a ação realizada por essa operação (no exemplo, assistirAula). Uma operação é ativada (executada) quando um objeto envia uma mensagem para outro, sendo essa mensagem o nome da operação.

Do mesmo modo que os atributos, a visibilidade pode ser pública (+), privada (-), protegida (#) ou de pacote (~).

Os parâmetros de uma operação são informações que ela recebe do objeto que envia a mensagem para requisitar a execução da operação. Uma operação pode ter zero ou mais parâmetros, que devem ser separados por vírgulas. Parâmetros têm seu tipo definido: tipo retorno representa o tipo de dados do valor retornado por uma operação.

VISÃO GERAL DO DIAGRAMA DE CLASSES

O Diagrama de Classes da UML é o modelo em que representamos as classes de um sistema, incluindo os detalhes sobre seus atributos e suas operações, e como essas classes se relacionam.

Portanto, a elaboração do Diagrama de Classes é fundamental em todo o processo de desenvolvimento que segue a abordagem da orientação a objetos.

Os diversos componentes de um Diagrama de Classes especificam as classes que serão realmente implementadas (ou seja, codificadas em uma linguagem de programação), sendo a base para a criação dos principais objetos e as interações entre eles.

ATENÇÃO

Góes (2014) explica que o Diagrama de Classes é o modelo mais importante da UML, pois permite a representação dos elementos da base de dados do sistema. Portanto, facilita a comunicação entre a equipe de desenvolvimento e com os usuários finais.

O Diagrama de Classes também explicita as demandas de dados e informações dos atores, que são declaradas nos casos de uso.

Diagramas de classes são usados para expressar graficamente os conceitos a serem manipulados por um sistema, bem como as ações esperadas, e para fornecer uma descrição independente de implementação dos tipos utilizados em um sistema, ou seja, os modelos de dados.

COMENTÁRIO

Costumamos dizer que esse diagrama destaca um aspecto estrutural estático do sistema, pois representa a estrutura interna do sistema, mas não como os objetos do sistema interagem no decorrer do tempo. As operações mostram quais são as possíveis interações entre os objetos, mas não explicitam em que momento serão invocadas.

O Diagrama de Classes é utilizado na construção do modelo de classes desde o nível de análise até o nível de especificação. O diagrama evolui à medida que o processo de desenvolvimento do sistema avança.

O QUE QUEREMOS DIZER COM “EVOLUI”?

RESPOSTA

Significa que vamos acrescentando detalhes e refinando o diagrama, tanto em termos de atributos e operações quanto na forma como associamos as classes.

É comum chamarmos esse diagrama que evolui sucessivamente de modelo de classes de análise, modelo de classes de projeto e modelo de classes de implementação, fazendo referência às etapas do processo de desenvolvimento.

MODELO DE CLASSES DE ANÁLISE

É composto por elementos identificados na etapa de análise; é basicamente formado por classes relacionadas a conceitos do domínio da aplicação. Seu objetivo é descrever o problema representado pelo sistema a ser desenvolvido. É um modelo conceitual; com foco em o que o sistema deve tratar, não considera características técnicas da solução a ser utilizada. Em termos práticos, nesse modelo, incluímos as classes, a lista de atributos e a lista básica de operações, sem os detalhes de tipos de dados, ou parâmetros das funções, por exemplo. O modelo de casos de uso e o modelo de classes de análise são os dois principais modelos criados na fase de análise.

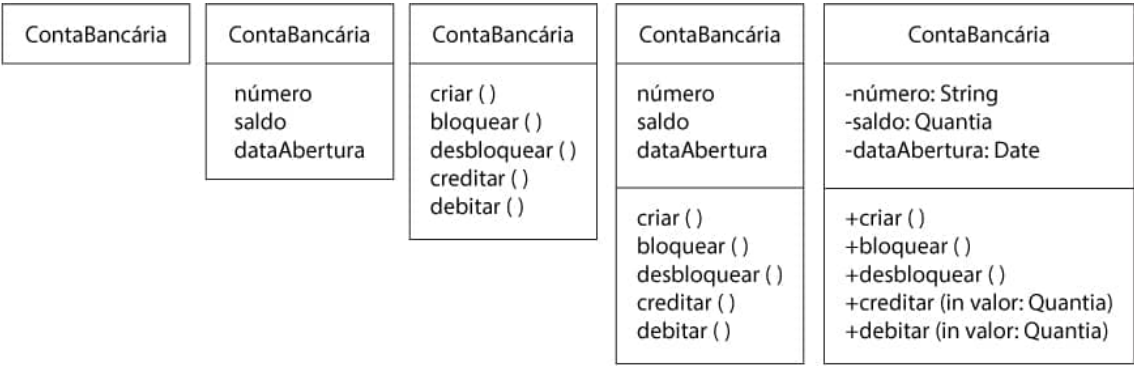
MODELO DE CLASSES DE PROJETO

É construído na atividade de projeto do desenvolvimento e considera alguns detalhes da solução de software a ser utilizada, mas ainda em um nível alto de abstração. Nessa etapa do processo de desenvolvimento, é comum percebermos a necessidade de criar novas classes, uma vez que o foco agora está em como o sistema deve funcionar. Ou seja, como o sistema vai entregar suas funcionalidades. Os detalhes incluídos são relacionados, por exemplo, com tipos e domínio dos atributos; parâmetros, entradas e saídas das operações etc. Novas classes criadas podem ser de interfaces do sistema, bancos de dados, entre outras.

MODELO DE CLASSES DE IMPLEMENTAÇÃO

É a codificação das classes definidas nas fases de análise e projeto em alguma linguagem de programação, normalmente uma linguagem orientada a objetos (por exemplo, C++, Java, C# etc.). O modelo de implementação é construído na atividade de codificação de um processo de desenvolvimento

O exemplo da figura 7, extraído da obra de Bezerra (2015), ilustra a representação de uma classe em diferentes níveis de abstração. Observe que, em cada iteração do processo de desenvolvimento, são acrescentados mais detalhes sobre os atributos e operações.



📷 Figura 7 – Exemplo de Classe ContaBancária em diferentes níveis de abstração.

Na especificação mais detalhada da classe ContaBancária do exemplo da figura 7, você pode observar a presença dos símbolos – e + antes dos nomes dos atributos e das operações, que representam sua visibilidade. Nessa mesma representação, temos as operações com seus parâmetros de entrada especificados (a Quantia em creditar e debitar).

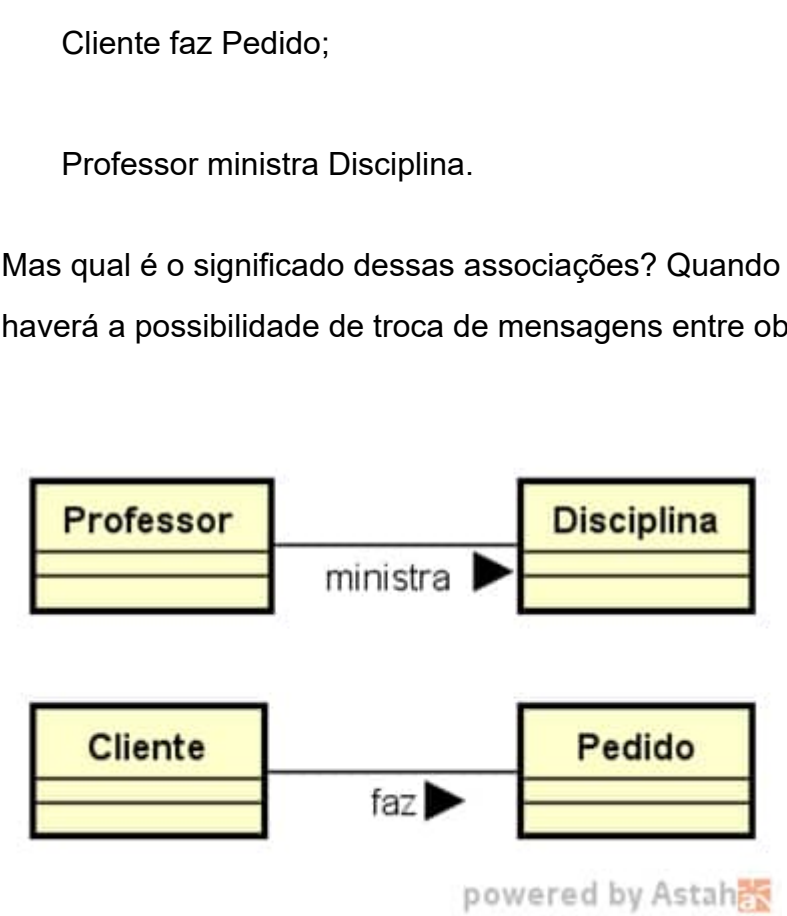
RELACIONAMENTOS NO DIAGRAMA DE CLASSES

Você já viu que especificar uma classe diz respeito a definir seus atributos e operações. Mas classes não existem isoladas, elas só fazem sentido quando se relacionam a outras, pois os objetos originados a partir delas trocam informações uns com os outros; e nisso reside a implementação de um sistema orientado a objetos.

Dizemos que os objetos colaboram uns com os outros. O desenho do Diagrama de Classes mostra como os objetos instanciados a partir dessas classes podem se relacionar. Existem diferentes tipos de relacionamentos possíveis, bem como diversos detalhes que podem ser acrescentados no diagrama para melhorar sua semântica. Trataremos deles a seguir.

A forma de relacionamento mais comum é chamada de associação e é representada no Diagrama de Classes por uma linha (normalmente um segmento de reta) ligando as classes às quais pertencem os objetos relacionados.

A figura 8 ilustra associações entre classes:



📷 Figura 8 – Associações simples entre classes, elaborado com ferramenta ASTAH (2021)

As associações possuem diversas características que podem ser representadas, por escolha do analista, para prover um melhor entendimento do modelo: nome, multiplicidades, tipo de participação, direção de leitura e papéis.

NOME

Conforme a figura 8 mostra, os nomes das associações (ministra, faz) descrevem a relação que é estabelecida entre os objetos das classes

associadas. Normalmente, são usados verbos ou expressões verbais para nomear associações.

MULTIPLICIDADES

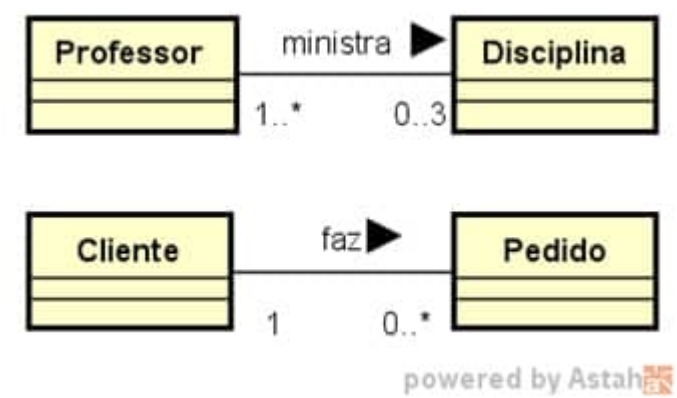
As associações permitem representar a informação dos limites inferior e superior da quantidade de objetos aos quais outro objeto pode estar associado: são as chamadas multiplicidades. Cada associação em um Diagrama de Classes possui duas multiplicidades, uma em cada extremo da linha que a representa. Os símbolos que representam uma multiplicidade são apresentados na tabela 1.


Apenas 1	1..1 (ou 1)
Zero ou Muitos	0..* (ou *)
Um ou Muitos	1..*
Zero ou Um	0..1
Intervalo Específico	li..ls

 **Atenção!** Para visualizaçãocompleta da tabela utilize a rolagem horizontal

 Tabela 1 – Multiplicidades em um Diagrama de Classes. Tabela: Flavia Maria Santoro

Na figura 9, mostramos o exemplo com as multiplicidades correspondentes. Esses novos diagramas expressam mais do que o anterior. Eles informam que um professor pode ministrar nenhuma (zero) e no máximo três disciplinas (0..3). Uma disciplina precisa ser ministrada por pelo menos um professor, mas pode ser ministrada por muitos (1..*). Já um cliente pode fazer nenhum (zero) pedido, sem um limite máximo (0..*), e um pedido precisa ser feito por um cliente (1).



 Figura 9 – Associações simples entre classes com multiplicidades, elaborado com ferramenta ASTAH (2021)

TIPO DE PARTICIPAÇÃO

Indica a necessidade ou não da existência dessa associação entre objetos. A participação pode ser **obrigatória** ou **opcional**. Se o valor mínimo da multiplicidade de uma associação é igual a 1 (um), significa que a participação é obrigatória. Caso contrário, a participação é opcional. Por exemplo, a participação de Cliente e Professor nos exemplos da figura 9 é obrigatória, e a de Disciplina e Pedido é opcional. Isso significa que pode existir professores e clientes sem necessariamente estarem ministrando disciplinas ou fazendo pedidos.

DIREÇÃO DA LEITURA

A direção de leitura indica como a associação deve ser lida. Essa direção é representada por um pequeno triângulo posicionado próximo a um dos lados do nome da associação (como nos exemplos da figura 9).

PAPÉIS

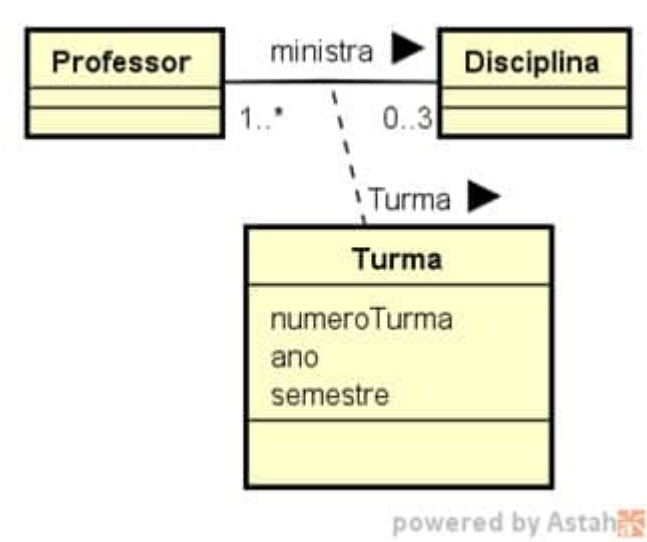
Quando um objeto participa de uma associação, ele tem um papel específico nela. Uma característica complementar à utilização de nomes e de direções de leitura é a indicação de papéis (roles) para cada uma das classes participantes em uma associação.

CLASSES ASSOCIATIVAS

São classes que estão ligadas a associações, em vez de estarem ligadas a outras classes. São também chamadas de classes de associação. Esse tipo de classe normalmente aparece quando duas ou mais classes estão associadas e seja necessário manter informações sobre a associação que existe entre elas.

É muito comum a necessidade de classes associativas em relacionamentos muitos para muitos, ou seja 0..* 0..*. Uma classe associativa é representada pela mesma notação utilizada para uma classe comum. A diferença é que aquela é ligada por uma linha tracejada a uma associação.

A figura 10 mostra um exemplo de classe associativa. Um professor ministra uma disciplina em determinada turma (que possui número, ano e semestre específicos).

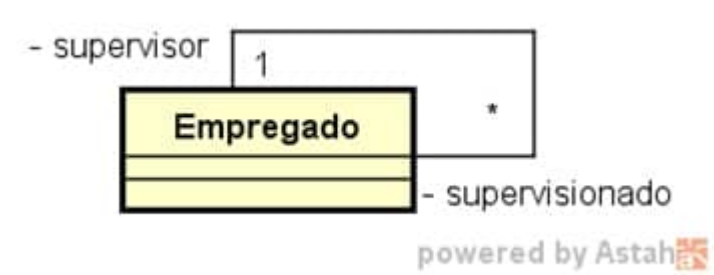


📷 Figura 10 – Exemplo de classe associativa, elaborado com ferramenta ASTAH (2021)

AUTOASSOCIAÇÃO

Tem como objetivo relacionar objetos da mesma classe, mas cada objeto tem um papel distinto nessa associação. Por exemplo, na figura 11, objetos da classe Empregado estão associados entre si, um empregado como supervisor, que supervisiona outros empregados.

A figura 11 ilustra, também, o uso de papéis (roles) para distinguir o lado supervisor (multiplicidade 1) do lado supervisionado (lado muitos) da autoassociação.



📷 Figura 11 – Exemplo de autoassociação, elaborado com ferramenta ASTAH (2021)

TUDO-PARTE

Existe uma categoria especial de relacionamento que possui uma semântica (significado) particular: todo-parte.

Uma relação todo-parte entre dois objetos indica que um dos objetos está contido (é uma parte do) no outro, ou seja, o objeto do lado todo contém o do lado parte.

A UML define dois tipos de relacionamentos todo-parte: a agregação e a composição.

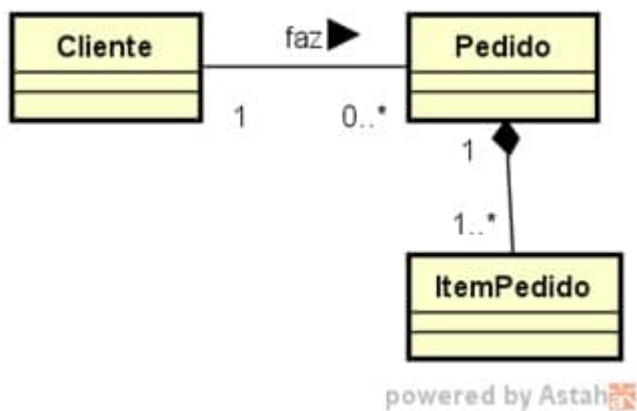
📢 ATENÇÃO

A **agregação** e a **composição** são casos especiais da associação, logo, todas as características válidas para uma associação valem para agregações e composições. Agregações/composições são assimétricas no sentido que, se um objeto A é parte de um objeto B, o objeto B não pode ser parte do objeto A.

Agregações e composições propagam comportamento, pois um comportamento que se aplica a um todo, automaticamente se aplica às suas partes. Nas agregações/composições, as partes são normalmente criadas e destruídas pelo todo.

Uma agregação é representada graficamente como uma linha que conecta as classes relacionadas, com um diamante (losango) branco perto da classe que representa o todo. Uma composição é representada por meio de um diamante negro.

A figura 12 apresenta um exemplo de relacionamento de composição. Um pedido é composto de vários itens de pedidos.



📷 Figura 12 – Exemplo de relacionamento de composição, elaborado com ferramenta ASTAH (2021)

A diferença entre agregação e composição é sutil, pois não é muito bem definida na UML. Ambas são relacionamentos todo-parte, mas pode-se dizer que, na agregação, a parte pode existir independentemente do todo; na composição, a existência da parte depende da existência do todo.

No exemplo da figura 12, não faria sentido a existência de ItemPedido se não existisse Pedido.

★ EXEMPLO

Um exemplo de agregação seria a associação entre Equipe e Jogador, em que este último é parte da Equipe; entretanto, um Jogador existe independentemente da existência da Equipe. Isso significa que a destruição de uma Equipe (todo) não implicaria na destruição do Jogador (parte), como ocorre na composição.

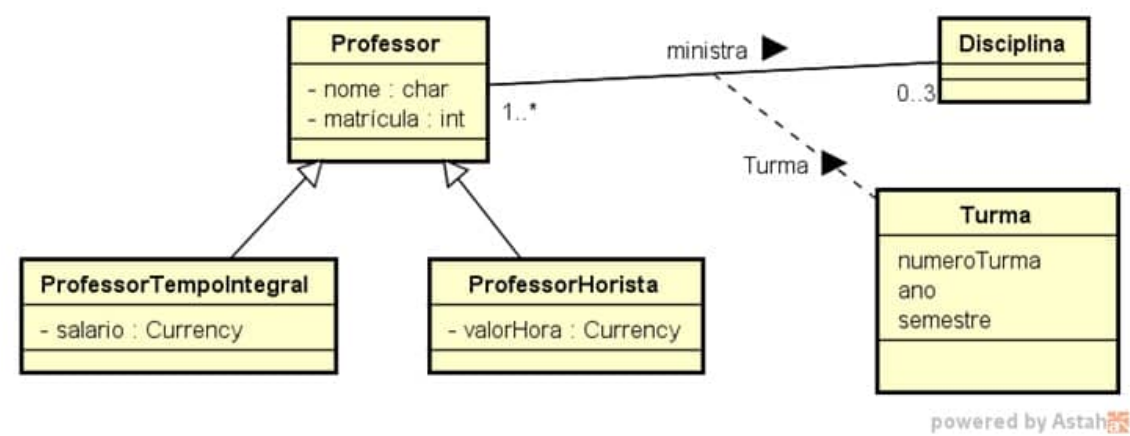
HERANÇA

Outro relacionamento com semântica específica é o de herança, ou relacionamento de generalização/especialização. Esse tipo de relacionamento, diferente dos que você estudou, significa um relacionamento de generalidade ou especialidade entre as classes envolvidas. Por exemplo, o conceito animais é mais genérico que o conceito mamífero, que, por sua vez, é mais genérico que o conceito macaco.

Nesse tipo de relacionamento, as classes mais específicas herdam características das classes mais genéricas.

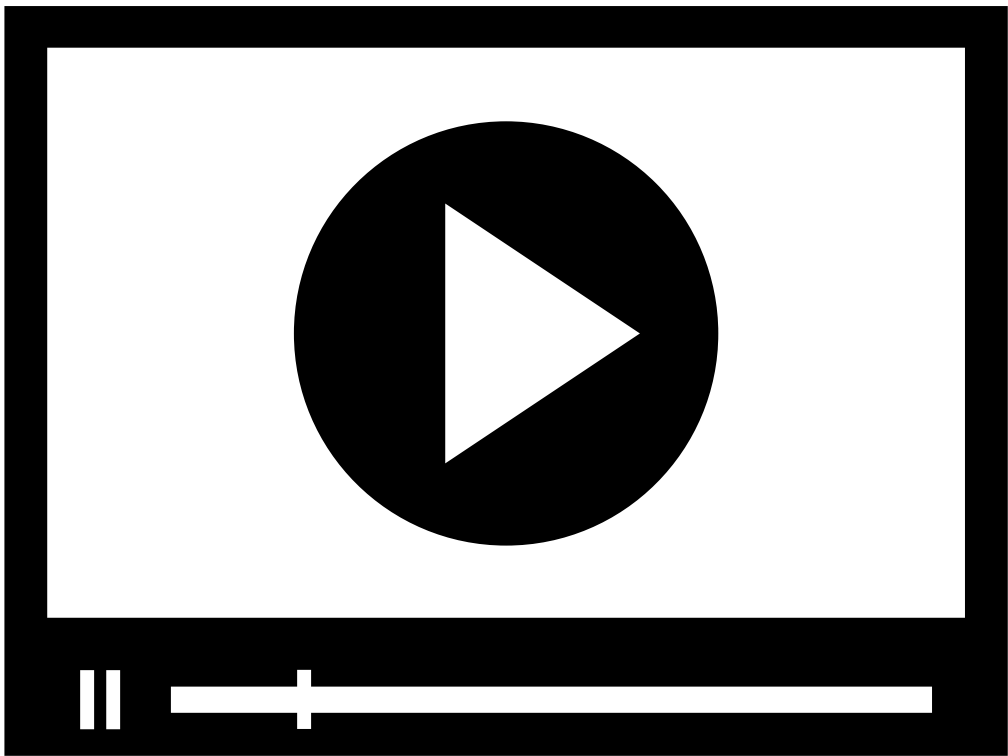
Usamos o termo subclasse para denotar a classe que herda as propriedades de outra classe mediante uma generalização. A classe que possui propriedades herdadas por outras classes é chamada de superclasse. No Diagrama de Classes, a herança é representada por uma flecha partindo da subclasse em direção à superclasse. Um relacionamento de herança representa um conjunto de objetos que compartilham um conjunto comum de propriedades (atributos, operações, associações), características inerentes à superclasse, além de possuírem características próprias.

A figura 13 mostra um exemplo do relacionamento de herança.



📷 Figura 13 – Exemplo de relacionamento de herança, elaborado com ferramenta ASTAH (2021)

Nesse exemplo, um professor pode ser do tipo Tempo Integral ou Horista. Ambos possuem nome e matrícula, porém, o Professor Tempo Integral possui salário, e o Professor Horista possui valor de hora paga. Note que, além dos atributos da classe genérica (no exemplo, Professor), as classes especializadas herdam também as operações (não especificadas no exemplo), assim como os relacionamentos (no exemplo, a associação ministra) da classe genérica.



O DIAGRAMA DE CLASSES NO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS

A especialista Flavia Maria Santoro fala sobre o modelo de classes, sua finalidade, seus elementos e sua evolução ao longo do processo de desenvolvimento de sistemas.



VERIFICANDO O APRENDIZADO



MÓDULO 4

🕒 Empregar Diagramas de Objetos e de Pacotes para apoiar a especificação de um sistema

VISÃO GERAL DO DIAGRAMA DE OBJETOS

Além do Diagrama de Classes, a UML define um segundo tipo de diagrama na perspectiva estática e estrutural do sistema: o Diagrama de Objetos.

Assim como objetos são instâncias de classes, os Diagramas de objetos podem ser vistos como instâncias de Diagramas de Classes.

Um Diagrama de Objetos UML representa uma instância específica de um Diagrama de Classes em determinado momento. Portanto, do ponto de vista gráfico, ele é bastante parecido com o Diagrama de Classes.

⊕ SAIBA MAIS

Bezerra (2015) afirma que um Diagrama de Objetos exibe uma “fotografia” do sistema em certo momento, mostrando as associações estabelecidas entre objetos, de acordo com suas interações e com os valores que seus atributos assumem.

Um Diagrama de Objetos mostra os valores de atributos do conjunto de objetos que ele representa e como esses objetos se relacionam entre si. Um objeto, ou uma instância de uma classe, possui uma identificação única (não existem dois objetos iguais) e os seus atributos assumem determinados valores em um momento específico no tempo.

Os valores dos atributos podem sofrer alterações e, nesse caso, dizemos que o objeto sofreu uma **mudança de estado**.

★ EXEMPLO

Por exemplo, o objeto Maria, criado a partir de uma classe Pessoa, possui como valor do atributo Endereço: Rua das Palmeiras. Quando Maria muda de endereço, o valor do atributo passa a ser o novo endereço: Avenida Vieira Souto. Assim, em um diagrama de estados, o objeto Maria seria representado com o valor do atributo Endereço quando o diagrama foi criado.

No Diagrama de Objetos, não faz sentido falar em operações, que são as mesmas para todos os objetos de uma classe. Portanto, nesse diagrama, o elemento principal, o Objeto, possui apenas o nome e os atributos com seus valores.

Em um Diagrama de Objetos, cada objeto é representado por um retângulo com dois compartimentos. O compartimento superior mostra a identificação do objeto na forma de texto sublinhado:

:NomeClasse

nomeObjeto: NomeClasse

Lembre-se de que cada objeto é único e, portanto, esse nome identifica exclusivamente um objeto. A identificação do objeto pode ser omitida.

Por exemplo:

Marta: Professor

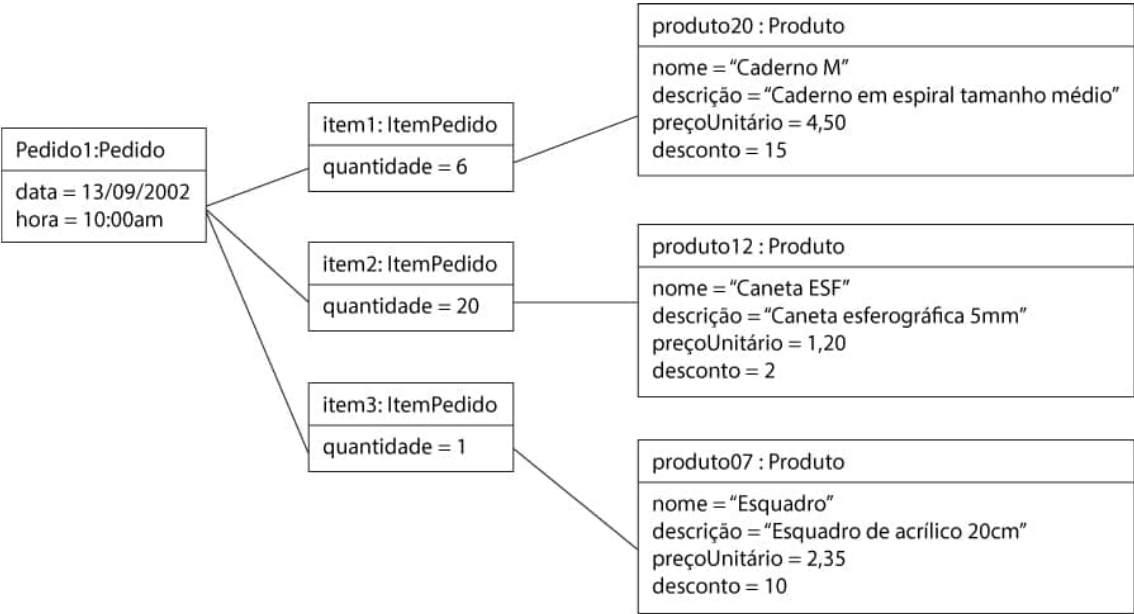
:Professor

No primeiro caso, o objeto da classe professor foi identificado com o nome Marta; no segundo, o analista optou por não caracterizar o objeto em si, significando que pode estar se referindo a qualquer objeto daquela classe.

A segunda representação é frequentemente usada na etapa de projeto do processo de desenvolvimento de sistemas, quando se representa a interação entre objetos (instâncias de classes) nos diagramas de sequência ou de comunicação.

No compartimento inferior (cuja utilização é opcional), aparecem os valores que os atributos definidos na classe assumem para este objeto. Dessa forma, deve mostrar uma lista de pares da forma: nome do atributo = valor do atributo.

A figura 14 reproduz o exemplo de Diagrama de Objetos mostrado por Bezerra (2015). Nesse exemplo, uma instância de pedido é associada a três instâncias de ItemPedido. Cada item, por sua vez, está ligado a um produto. Os valores para cada atributo de cada objeto são apresentados nesse diagrama.

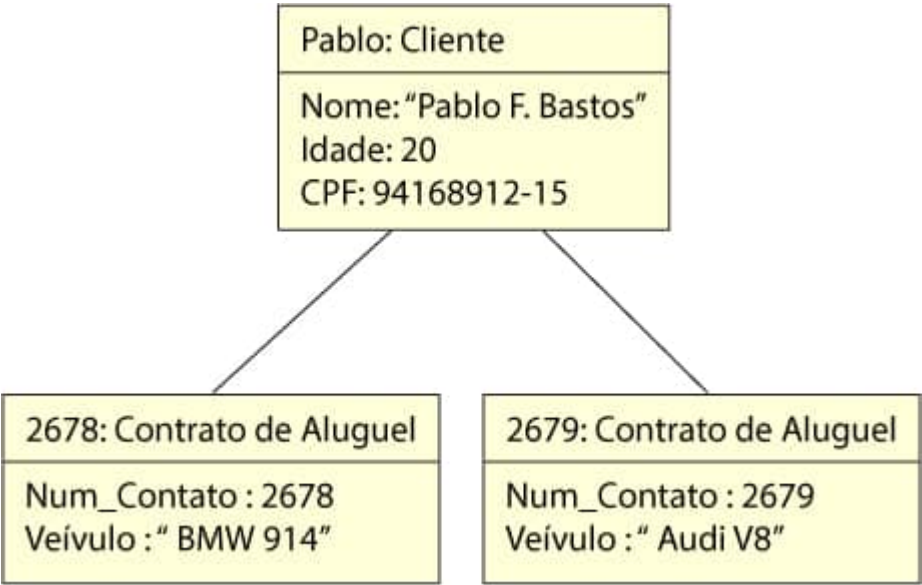


📷 Figura 14 – Exemplo de Diagrama de Objetos (BEZERRA, 2015)

Você pode observar também que não existem detalhes nos relacionamentos, tal como no Diagrama de Classes. Os relacionamentos no Diagrama de Objetos mostram uma situação específica em que aqueles objetos se encontram, não as possíveis relações entre eles.

No diagrama da figura 14, temos um Pedido que possui 3 Itens associados a ele, e 1 Produto associado respectivamente a cada Item.

A figura 15 mostra outro exemplo de Diagrama de Objetos, em que observamos que o Cliente Pablo está associado a 2 contratos de aluguel de carros diferentes. Cada contrato tem seu identificador e atributos próprios. Esse diagrama ajuda no entendimento da situação desse cliente.



📷 Figura 15 – Exemplo de Diagrama de Objetos

➕ SAIBA MAIS

De acordo com Bezerra (2015), Diagramas de Objetos não são muito utilizados na prática. Mas eles podem ser úteis na atividade de validação no processo de desenvolvimento, pois permitem ilustrar e esclarecer certos aspectos de um Diagrama de Classes.

Mesmo antes de criar um Diagrama de Classes, você pode criar um Diagrama de Objetos para descobrir fatos sobre elementos específicos e suas associações ou para esclarecer dúvidas por meio de exemplos específicos. Além disso, ajuda na construção dos diagramas de interação (de sequência ou comunicação), pois estes utilizam a mesma notação.

Em resumo, os Diagramas de Objetos podem ter as seguintes aplicações:

Ajudar na análise de uma interação específica.

Mostrar uma visão geral de alto nível do sistema.

Validar um Diagrama de Classes.

Os Diagramas de Objetos podem ser usados também para verificar se o sistema foi desenvolvido conforme os requisitos, e até mesmo para analisar se uma regra de negócio está sendo cumprida.

💡 DICA

Para criar esses diagramas, procure exibir associações entre objetos somente com uma ligação (por exemplo, uma linha única juntando dois objetos, sem setas ou outros detalhes), pois um Diagrama de Objetos tem por objetivo mostrar somente a existência de associações e não os tipos de associações, como em um Diagrama de Classes. Além disso, evite representar todos os objetos em um único Diagrama de Objetos, pois ele apresenta apenas um estado do objeto.

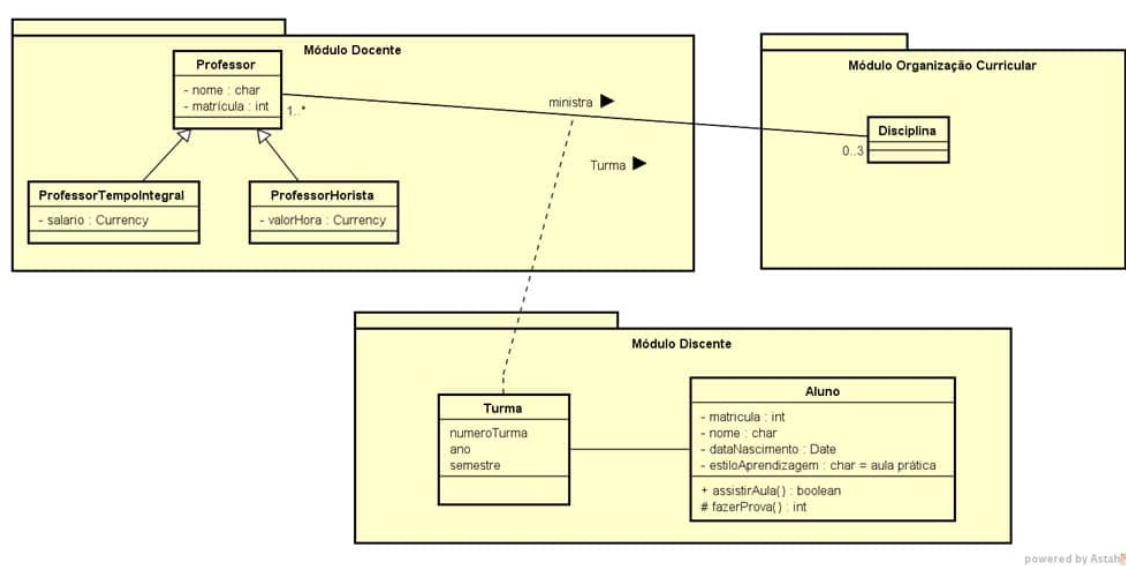
VISÃO GERAL DO DIAGRAMA DE PACOTES

Muitos sistemas possuem uma quantidade bastante grande de casos de uso e classes, portanto, representá-los em um único diagrama é inviável devido à sua complexidade para o entendimento. Para resolver esse problema, usamos o elemento da UML chamado de pacote (do inglês, *package*).

Os pacotes servem para agrupar diagramas, tais como classes, casos de uso, ou mesmo outros pacotes. Normalmente, os grupos de classes são associados com módulos de programação do sistema. Uma classe só pode aparecer uma vez no mesmo pacote, mas pode estar presente em vários pacotes diferentes.

Um pacote representa um grupo de elementos que pode se relacionar com outros pacotes por meio de uma relação de dependência. É possível definir a visibilidade (conforme estudamos no Módulo 3) de pacotes e dos elementos contidos nele. Pacotes são úteis para prover uma visão geral de um conjunto grande de objetos; para organizar um modelo grande e agrupar seus elementos.

A representação gráfica de um pacote é um retângulo com uma aba no canto superior esquerdo, com o nome do pacote na aba ou no centro do retângulo. A figura 16 mostra um exemplo de pacotes que agrupam conjuntos de classes. O pacote Módulo Docente agrupa as classes relacionadas com professores, o Módulo Discente agrupa as classes relacionadas com alunos, e o Módulo Organização Curricular agrupa classes pertinentes à matriz curricular, em um sistema acadêmico.



📷 Figura 16 – Exemplo de representação de pacote, elaborado com ferramenta ASTAH (2021)

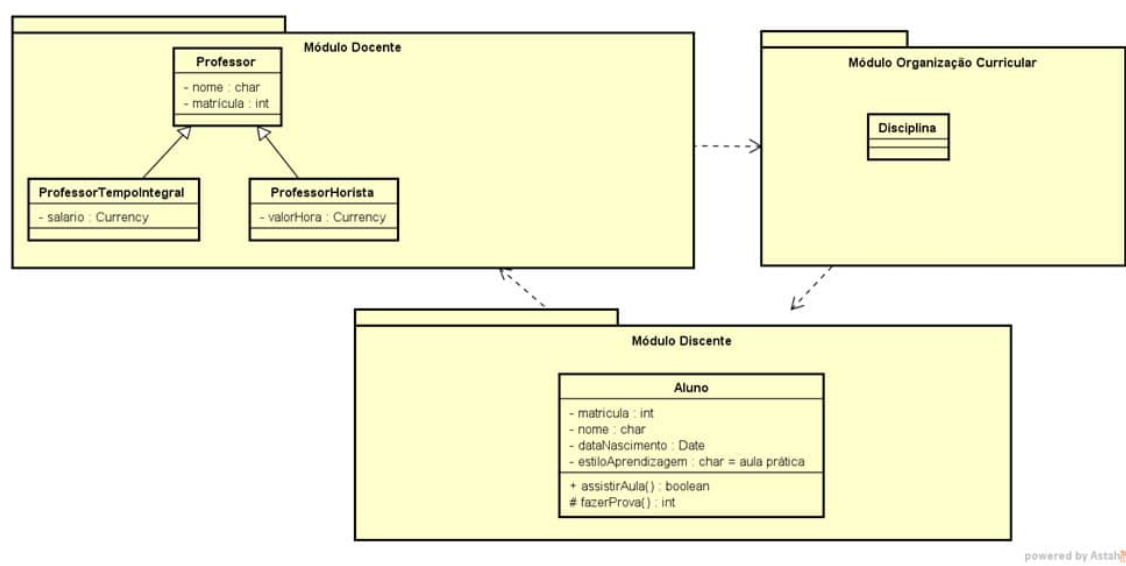
DESDOBRAMENTOS DOS DIAGRAMAS NO PACOTE

As ligações entre pacotes são relacionamentos de dependência. Um pacote P1 depende de outro pacote P2, se algum elemento contido em

P1 depende de algum elemento contido em P2.

A figura 17 mostra o Diagrama de Pacotes do exemplo do sistema acadêmico sem os relacionamentos explícitos entre as classes internas; apenas com a representação de dependências existentes entre os próprios pacotes. Ou seja:

Os pacotes se relacionam com outros pacotes por meio de um vínculo de dependência.



📷 Figura 17 – Exemplo de representação de dependências entre pacote, elaborado com ferramenta ASTAH (2021)

As dependências entre pacotes mostram que um pacote depende ou influencia outro. As dependências podem ser divididas em dois tipos: acesso e importação.

DEPENDÊNCIA DE ACESSO

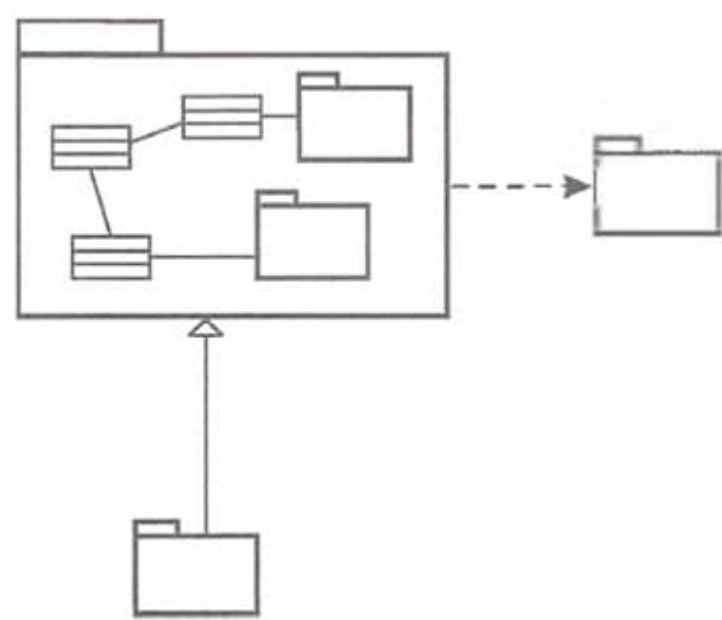
Indica que um pacote requer assistência das funções de outro pacote.

DEPENDÊNCIA DE IMPORTAÇÃO

Indica que uma funcionalidade foi importada de um pacote para outro.

Ambas são representadas por uma linha pontilhada com uma seta apontando para o pacote dependente.

Pacotes também podem formar grupos de pacotes e hierarquias. A figura 18 (Bezerra, 2015) mostra como pacotes podem ser agrupados dentro de outros pacotes, formando uma hierarquia de contenção.



📷 Figura 18 - Agrupamento de pacotes (BEZERRA, 2015)

Além disso, como já vimos, existem três tipos de visibilidade de elementos dentro dos pacotes:

+ (PÚBLICO)

Visível por todos que importam ou acessam o pacote.

(PROTEGIDO)

Visível só pelos pacotes filhos (por relação de generalização).

– (PRIVADO)

Visível só por outros elementos do pacote.

ATENÇÃO

Note que aqui não se aplica a visibilidade de pacotes (~), que é aplicável tão somente a atributo de classe que seja visível a qualquer classe pertencente ao mesmo pacote no qual está definida a classe.

No contexto da modelagem de casos de uso, os pacotes podem ser utilizados com diversos objetivos. Alguns desses objetivos são: estruturar o modelo de casos de uso com base nos grupos de usuários do sistema; definir a prioridade na qual os casos de uso serão desenvolvidos; e definir o grau de correlação entre os casos de uso.

O critério para criar os pacotes é subjetivo e depende da visão e das necessidades do projetista.

COMENTÁRIO

Normalmente, o projetista define uma semântica e aloca elementos similares em um pacote, pois esses elementos tendem a ser alterados em conjunto no mesmo pacote.

Um pacote bem estruturado tem as seguintes características:

É coeso em relação aos elementos incluídos.



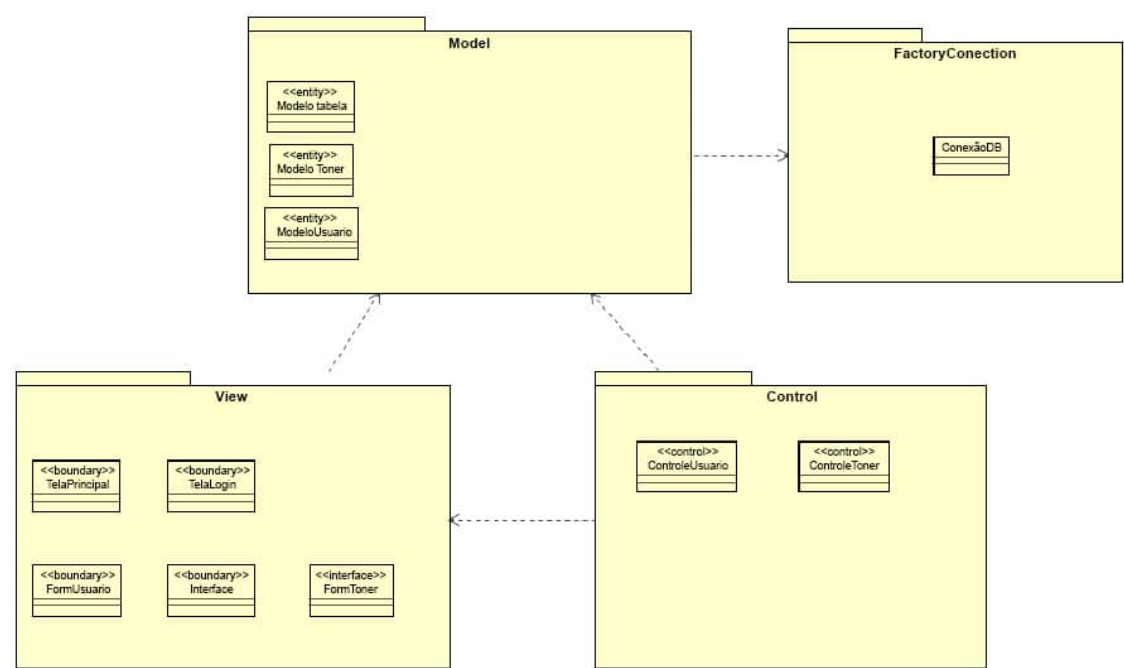
É fracamente acoplado em relação à exportação apenas dos elementos que os outros pacotes precisam enxergar.



Não contém muitos alinhamentos, ou seja, possui um conjunto equilibrado de elementos.

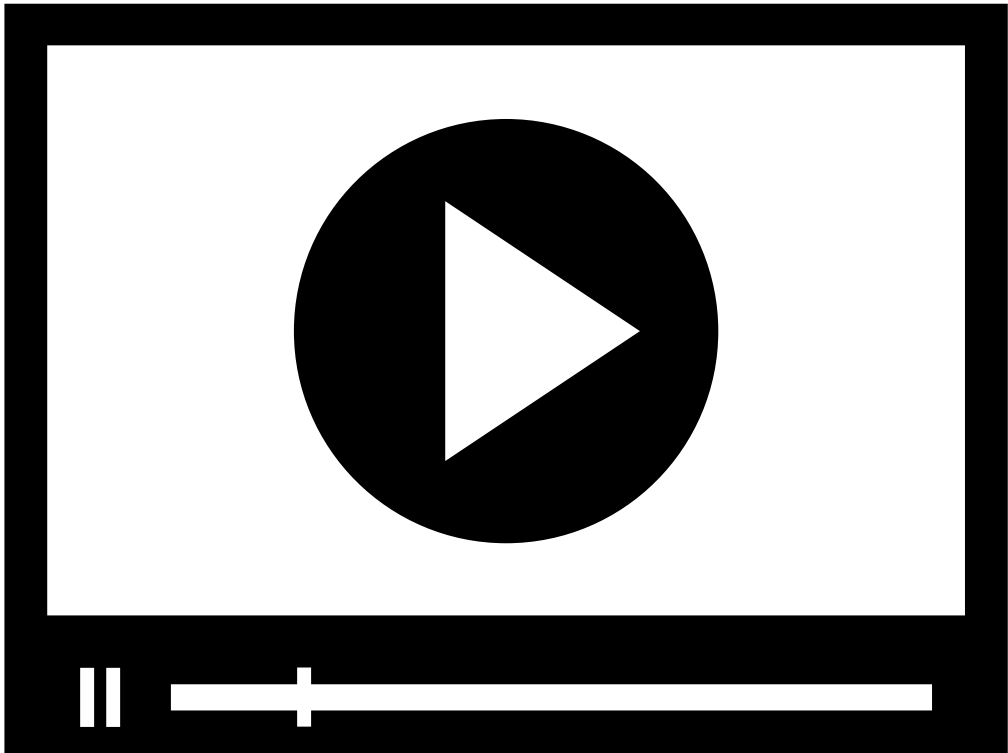
Ao especificar um pacote, é uma prática usar a forma simples de ícone retangular, obscurecendo o seu conteúdo. No caso da necessidade de apresentar o seu conteúdo interno (como nas figuras 16 e 17), deve-se mostrar apenas os elementos necessários para a compreensão do significado.

Esse diagrama também é muito utilizado para ilustrar a arquitetura de um sistema, mostrando o agrupamento de suas classes. A figura 19 mostra um exemplo de Diagrama de Pacotes usado com o objetivo de organizar as classes de acordo com as camadas da arquitetura do sistema. Tipicamente, um sistema pode ser organizado conforme o padrão de projeto Model (Modelo), View (Visão) e Control (Controle).



📷 Figura 19 – Exemplo de diagrama de pacotes

O pacote tem uma grande similaridade com a agregação. O fato de um pacote ser composto de modelos de elementos cria uma espécie de agregação de composição, pois, se esse pacote for destruído, todo o seu conteúdo também será.



USO DO DIAGRAMA DE PACOTES NA MODELAGEM DE SISTEMAS

A especialista Flavia Maria Santoro fala sobre o uso do diagrama de pacotes no agrupamento de diagramas de modelagem de sistemas.



VERIFICANDO O APRENDIZADO

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Estudamos os conceitos básicos para você iniciar a modelagem de um sistema orientado a objetos usando a UML. Observamos que a representação de requisitos funcionais deve ser feita por meio de um Diagrama de Casos de Uso, em conjunto com suas descrições textuais.

Você pode observar que a modelagem estrutural do sistema é feita com a identificação e representação de classes e seus objetos. Devemos definir seus atributos e operações e ficar atentos para os relacionamentos entre as classes, que podem ser associações simples, agregações e hierarquias. Vimos também que todos os elementos aprendidos podem ser agrupados em pacotes; além disso, podemos usar Diagramas de Objetos para apoiar no entendimento e análise de situações e interações específicas entre objetos.

Vimos que os diagramas da UML abordados neste tema podem ser usados na modelagem do domínio da aplicação, produzindo artefatos que serão refinados ao longo do processo de desenvolvimento do sistema. Na prática, nem todos esses diagramas são necessários em todos os tipos de sistema, destacando-se a relevância do diagrama de classes, por ser a base para a construção do banco de dados do sistema.



🎧 PODCAST

Agora, a especialista Flavia Maria Santoro encerra o tema destacando os pontos mais relevantes da modelagem do domínio, as boas práticas na especificação de requisitos, diagrama de casos de uso e diagrama de classes.



REFERÊNCIAS

ASTAH. **Astah UML Modeling Tool**. Consultado em meio eletrônico em: 31 jan. 2021.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Elsevier, 2015.

GÓES, W.M. **Aprenda UML por meio de estudos de caso**. São Paulo: Novatec, 2014.

IEEE STANDARD GLOSSARY of Software Engineering Terminology. *In*: IEEE Std. 610.12-1990, 31 dez. 1990.

LIMA, A. S. **UML 2.0: Do Requisito à Solução**. São Paulo: Érica, 2005.

MEDEIROS, E.S. **Desenvolvendo software com UML 2.0: definitivo**. São Paulo: Pearson Markron Books, 2004.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

Busque o artigo *Regras do Negócio, um fator chave de sucesso no processo de desenvolvimento de Sistemas de Informação*, dos professores Silvia Inês Dallavalle e Edson Walmir Cazarini, e aprenda sobre a importância das regras de negócio no desenvolvimento de sistemas.

CONTEUDISTA

Flavia Maria Santoro

CURRÍCULO LATTES