UNIDADE 1 – CONCEITOS INICIAIS SOBRE UML
MÓDULO 1 – UML E O DESENVOLVIMENTO DE SISTEMAS

01

1 - DIAGRAMAS E REQUISITOS DA DISCIPLINA UML

Nesta disciplina iremos tratar sobre a UML - Unified Modelling Language.

A UML, em português, **Linguagem de Modelagem Unificada** é uma linguagem ou notação de diagramas para especificar, visualizar e documentar modelos de *software* orientados por objetos.

A UML não é um método de desenvolvimento, o que significa que não lhe diz como será a interface (telas) do sistema, ou mesmo como programá-lo, mas ajuda a visualizar as principais características, grupos de funcionalidades e como os componentes do sistema se comunicam.

A UML é oficialmente composta por quatorze diagramas que representam as diferentes visões, partes e características de um sistema de software. Assim como podemos ter um mapa geográfico, um mapa geopolítico, um mapa hidrográfico, um mapa agropecuário e um mapa de altitudes que representam cada um uma visão diferente sobre o mesmo assunto, cada diagrama da UML também permite compreender o sistema sobre aspectos diferentes.

Os diagramas definidos na UML 2.0 são:

1. Diagrama de Caso de Uso;
2. Diagrama de Classe;
3. Diagrama de Objetos;
4. Diagrama de Sequência;
5. Diagrama de Máquina de Estados;
6. Diagrama de Atividade;
7. Diagrama de Componente;
8. Diagrama de Implantação;
9. Diagrama de Pacotes;
10. Diagrama de Estruturas Compostas;
11. Diagrama de Visão Geral de Interação;
12. Diagrama de Comunicação;
13. Diagrama de Tempo;
14. Diagrama de Perfil.

Diagrama de Caso de Uso
Representa as pessoas e/ou usuários do sistema (chamados de atores), as situações de uso/funcionalidades do sistema (chamados de casos de uso), e seus relacionamentos. Basicamente, representa quais serviços (funcionalidades) o sistema deve oferecer aos usuários.

Diagrama de Classe
Representa as classes e os relacionamentos entre elas. Cada classe representa um conjunto de informação e/ou funcionalidades de um objeto (um item do sistema). Em outras palavras, representam o conteúdo de dados e funcionalidades que são utilizados e manipulados dentro do sistema.

Diagrama de Objetos
Representa uma condição específica ou um exemplo. São normalmente utilizados para representar uma configuração específica do sistema, o que é muito usado na realização de testes ou chamadas operacionais.

Diagrama de Sequência
Representa os objetos e uma sequência lógica de evolução das informações e funcionalidades em um determinado contexto. Foca na troca de mensagens entre um grupo de objetos e a sequência das mensagens.

Diagrama de Máquina de Estados
Representa os estágios (ciclo de vida) de um objeto à medida que ele evolui em um sistema.

Diagrama de Atividade
Representa os acontecimentos (atividades e mudanças) de acordo com os eventos ocorridos em alguma parte do sistema. Utilizado para representam o fluxo de dados e o fluxo de controle. Captura o fluxo entre objetos interligados.

Diagrama de Componente
Representa os componentes de programação de alto nível. Mostram a organização e o relacionamento entre os entregáveis (pacotes) do sistema.

Diagrama de Implantação
Representa a arquitetura de execução do sistema, composta por todos os componentes do sistema e seus relacionamentos. Contempla, por exemplo, a plataforma de hardware, os artefatos de *software*, o ambiente de *software* (como máquinas virtuais ou sistemas operacionais).

Diagrama de Pacotes
Representam os componentes do sistema que estão associados em partes maiores para futura distribuição.

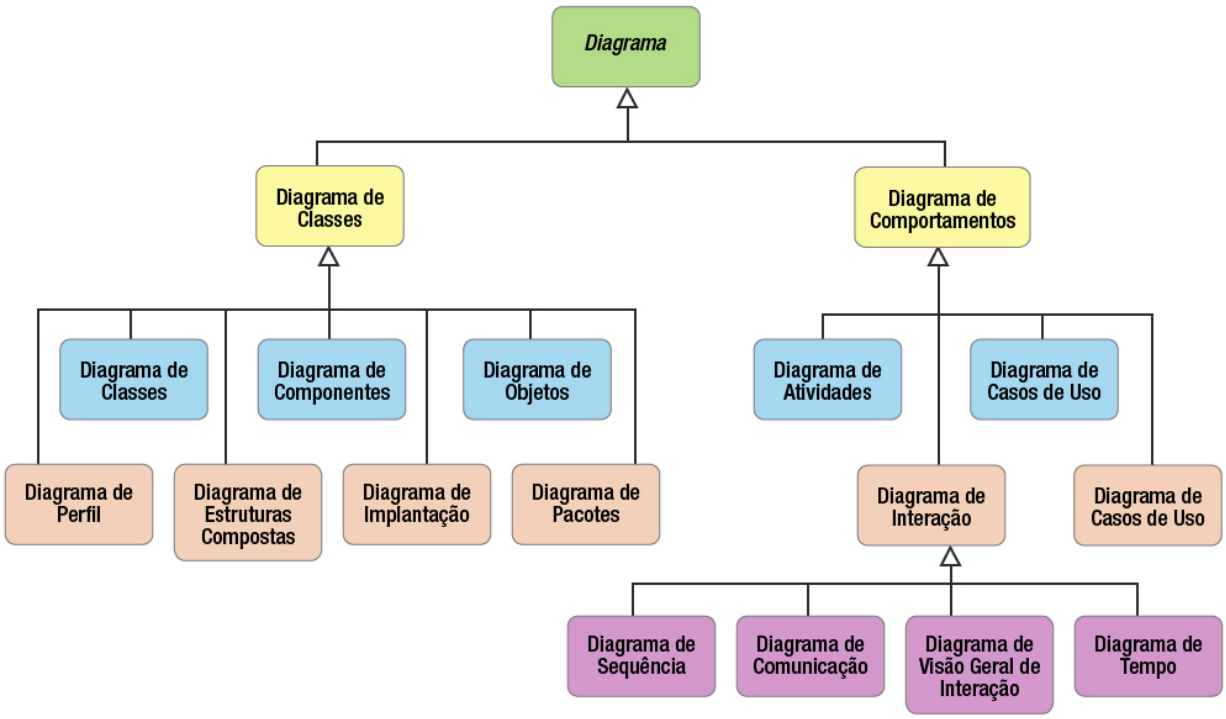
Diagrama de Estruturas Compostas
Representa como determinado pedaço do projeto foi feito. É especialmente útil em projetos muito complexos de estruturas compostas (baseadas em vários componentes integrados).

Diagrama de Visão Geral de Interação
Representa vários cenários possíveis e suas ações (um grupo de elementos trabalhando juntos para alcançar um objetivo).

Diagrama de Comunicação
Representa as mensagens entre um grupo de objetos e o relacionamento entre eles.

Diagrama de Tempo
Representa as mudanças e o relacionamento delas com o tempo sob a ótica do funcionamento real da aplicação. Este diagrama é raramente utilizado.

Diagrama de Perfil
Representa as diferentes versões das funcionalidades disponibilizadas pela aplicação de acordo com perfis distintos de usuários. Este diagrama é raramente utilizado.



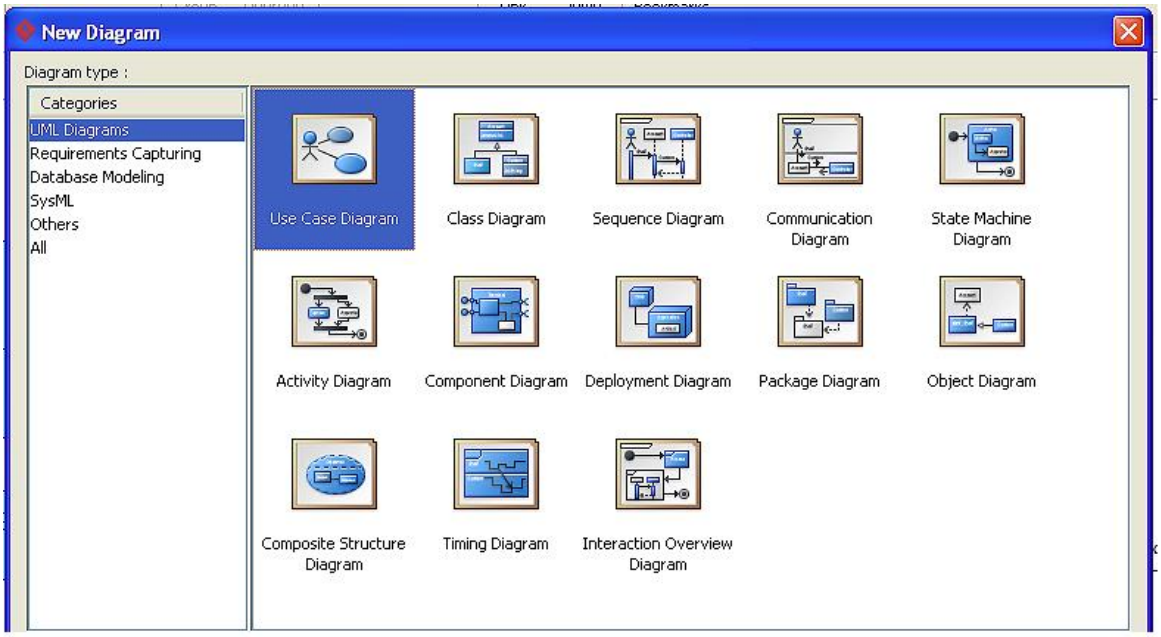
1.1. Pré-requisito para a disciplina

Para realizarmos os exercícios desta disciplina, iremos precisar de um *software* de modelagem. Há várias opções gratuitas, como o *software*livre Visual Paradigm.

Outro *software* similar é o Violet UML Editor, também gratuito.

Observações:

1: caso você já tenha feito a disciplina de Java, provavelmente você já deve ter o *software* "Eclipse". Nesse caso, não será necessário instalar o Visual Paradigm, visto que o Eclipse também permite criar os diagramas da UML.



Tela com os principais diagramas da UML no Visual Paradigm

2: não é escopo desta disciplina realizar um treinamento de Visual Paradigm, Violet UML, Eclipse ou qualquer outra ferramenta de UML. Para tanto, caso você tenha dificuldade no uso dessas ferramentas, pesquise no Youtube ou Google vídeos ou manuais de como utilizá-las.

Visual Paradigm
Para baixar o Visual Paradigm, acesse o link: http://www.visual-paradigm.com/download/community.jsp. Nesta página há um link para outras versões do programa (como para Windows 32 bits ou Linux). Faça o *download* e instale o programa. Informe seu nome e e-mail na tela de instalação e receba no seu e-mail o código de ativação. Informe esse código no sistema e pronto, você já pode utilizar o *software*.

Violet UML Editor
O Violet UML Editor também é gratuito e está disponível aqui:
http://sourceforge.net/projects/violet/files/latest/download?source=files

03

Para que você possa aproveitar ao máximo essa disciplina, é necessário o conhecimento prévio dos seguintes assuntos:

- **Orientação a objeto**: conceitos como classe, método, evento, objeto, chamada, herança e outros são fundamentais para a diagramação UML.
- **Metodologia de Desenvolvimento de Sistemas baseado em Método Unificado** (como o RUP ou o Open UP): conceitos como iniciação, elaboração, construção e transição, que representam o ciclo de criação de um projeto de *software*. Tratamos na UML principalmente das fases de Iniciação e Elaboração, a documentação elaborada nestas fases servirá de embasamento para a programação de *software* realizada na fase de construção.

Esses conteúdos fazem parte do seu curso, então aproveite para compreendê-los bem!

04

1.2. Tipos de diagramas

Na UML há basicamente três tipos de diagramas:

- Diagramas estruturais;
- Diagramas de comportamento;
- Diagramas de interação.

Outra forma de se organizar os diagramas é de acordo com suas características essenciais:

Diagramas estáticos	Diagramas dinâmicos	Diagramas funcionais
Representam as funcionalidades do sistema que não mudam com o tempo. Esta categoria é similar aos diagramas estruturais.	Representam como o sistema evolui (se comporta) com o tempo. Essa categoria envolve os diagramas de estado e de tempo.	Representam os detalhes dos comportamentos e os algoritmos, ou seja, como o sistema realiza o comportamento esperado. Essa categoria inclui os diagramas de caso de uso, de interação e de atividades.

Diagramas estruturais
São aqueles que representam a estrutura do *software*, independente das funcionalidades do sistema. Por essa razão, eles normalmente não mudam com a evolução do sistema (manutenção evolutiva). Estão intimamente ligados à tecnologia empregada no projeto.

Diagramas de comportamento
São aqueles que representam como o sistema reponde a uma chamada ou como ele se comporta ao longo do tempo.

Diagramas de interação
Esses diagramas são um tipo especial de diagramas de comportamento. Esses diagramas representam a colaboração (troca de mensagens) entre os componentes do projeto, ou mesmo com o projeto e outros sistemas adjacentes (integrados e/ou relacionados).

05

Você também pode utilizar os diagramas para responder informações distintas em momentos específicos do projeto de software. Um exemplo seria:

- **Quem usa o sistema?**

Os atores (usuários do sistema) nos respectivos casos de uso (representando o propósito do sistema).

- **Do que o sistema é feito?**

Representado nos diagramas de classe que apresenta a estrutura lógica e o diagrama de componentes que define a estrutura física.

- **Onde os componentes do sistema estão localizados?**

Os diagramas de implantação indicam o plano de onde os componentes do sistema deverão ser instalados.

- **Quando eventos importantes do sistema acontecem?**

Os diagramas de estado e de interação apresentam quais causas (eventos) fazem com que o sistema reaja e que tarefas são então executadas.

- **Como o sistema irá funcionar?**

Os diagramas de comunicação e de estrutura representam as interações em um nível suficientemente detalhado para o desenho (modelagem) e implementação (codificação).

06

2 - IDEIA GERAL SOBRE A UML

Você já ouviu falar que uma imagem vale mais do que mil palavras? Pois é, esse ditado popular tão antigo e amplamente utilizado traduz pura verdade quando falamos de projetos de sistemas de informação. **Diagramar é uma técnica de representar uma ideia em um desenho**. Para a grande maioria das pessoas é mais fácil compreender e relembrar uma ideia desenhada do que uma ideia escrita. Além disso, um desenho pode inserir detalhes que dificilmente seria possível caso só utilizássemos palavras.

É por isso que desde há muitos anos são utilizadas técnicas para expressar ideias em desenhos: uma planta de uma casa, um desenho arquitetônico de uma ponte, uma simulação em 3D de um automóvel são exemplos de técnicas de desenho e projeto.

Para o desenvolvimento de *software*, a melhor técnica de se desenhar um sistema, por meio da qual se apresenta uma ideia do que o sistema será no futuro, é a **prototipação**.

Por meio da **prototipação** criamos uma simulação do que o sistema será quando concluído, incluindo como acessar e navegar no sistema, como acessar as funcionalidades, como se parecerão os cadastros, pesquisas e relatórios.

Entretanto, já iniciar o projeto de informação por meio do protótipo não é uma boa ideia, pois quando criamos um protótipo, é necessário e prudente possuir uma visão completa do que o sistema faz e como ele funciona. Para isso, utilizaremos outras técnicas de diagramação que nos permitirão entender o que o nosso cliente quer com o sistema e como ele deve interagir internamente para funcionar conforme o esperado.

A UML surgiu como uma ideia de promover a comunicação e a produtividade entre os desenvolvedores de sistemas orientados a objeto, mas o verdadeiro poder da UML é a facilidade com que novos desenvolvedores compreendem o que o sistema deve fazer. Por isso, não só no desenvolvimento de novos sistemas, mas também durante a manutenção dos sistemas já existentes a diagramação UML tem muito a contribuir.

07

Os principais objetivos da UML é realizar as seguintes tarefas:

• Especificação

• Visualização

• Desenho da arquitetura do sistema

• Construção

• Simulação e teste

• Documentação

Especificação

Entender o que o sistema deve fazer sob a visão do usuário.

Visualização

Permitir uma visão do todo e como as partes do sistema se integram e interagem entre si.

Desenho da arquitetura do sistema

Definir entre os milhares de possibilidades qual será a estrutura interna (arquitetura) do sistema, quais tecnologias serão utilizadas e como elas serão utilizadas.

Construção
Definir um projeto de construção segundo uma ordem lógica que permita, futuramente, a integração das diversas partes de *software* que compõe o sistema por completo.

Simulação e teste

Definir regras para as quais o sistema será testado antes de ser entregue aos usuários finais (colocado em produção), assim, poderão ser identificados problemas de performance, segurança ou mesmo erros de código.

Documentação

De modo amplo, documentar o sistema de forma a que qualquer desenvolvedor tenha a capacidade de desenvolvê-lo de forma padronizada e, futuramente, alterá-lo, consertá-lo ou criar novas funcionalidades para o sistema.

08

2.1. Processo de modelagem de *software*

A forma pela qual o sistema será modelado e documentado (que é o que a UML pretende) depende do modelo de desenvolvimento de *software* proposto no projeto.

Atualmente há, basicamente, dois modelos largamente difundidos, cada um tem suas vantagens e desvantagens:

Modelo de Desenvolvimento	Vantagens	Desvantagens
Modelo Unificado (Unified Process)	• Indicado principalmente para projetos grandes; • Permite a documentação completa de todo o projeto do sistema; • Baseia-se nos modelos de fábrica de software mais comuns do mercado.	• É mais complexo; • Demora mais; • O projeto tende a custar mais caro.
Modelo Ágil	• Indicado para projetos pequenos ou médios; • Produz resultados mais rápidos; • O projeto geralmente fica economicamente mais barato.	• Documenta apenas o essencial ao projeto; • Pode deixar falhas, principalmente nas integrações dos componentes.

Os conceitos que aprenderemos nesta disciplina permitem a aplicação de qualquer um dos modelos citados.

09

Ainda em se tratando do modelo unificado, é importante relembrar que a modelagem evolui em dois sentidos com o tempo. Na medida em que o tempo do projeto passa:

- Passamos de uma visão mais geral e macro do projeto, para uma visão mais detalhada do que o sistema deve fazer;

Após a definição da visão geral, o projeto do sistema é dividido em pedaços chamados de iterações e, em cada iteração, os detalhes pertencentes àquele conjunto são detalhados mais e mais até o nível de entendimento e documentação desejado.

Uma boa metodologia de desenvolvimento de sistema deve prever quais documentos serão feitos e o nível de detalhamento de cada um de acordo com as características do projeto, dessa forma você poderá otimizar o tempo e o custo do projeto que melhor seja adequado à situação. Lembre-se: nem sempre é possível (e mesmo desejável) criar todos os documentos da UML para todas as situações do projeto.

A visão dos usuários sobre um determinado sistema nem sempre é completa e logicamente organizada. Por isso, durante a modelagem de um sistema, é muito comum que o profissional que está realizando esse serviço sugira mudanças estruturais e organizacionais no projeto do sistema de forma a produzir um *software* melhor. Por isso, para uma boa modelagem de sistema, o projeto de *software* precisa de alguém que conheça muito bem sobre a matéria que trata o *software*, de forma a colaborar com os usuários e analistas na melhor organização e desenho do sistema em si.

Também é prudente que analistas de UML procurem por outros projetos similares ao que ele está trabalhando, pois é muito mais fácil conhecer e melhorar algo que já existe parecido do que "reinventar a roda a partir do zero".

10

Fazendo uma analogia com a construção civil, ao modelarmos um sistema é como realizar as seguintes atividades da construção civil para a criação de um prédio:

- Fazer um **desenho arquitetônico** para ter uma visão geral externa do prédio. Da mesma forma, criaremos uma visão geral do sistema sob o ponto de vista do usuário.
- Fazer a **planta** de divisão dos ambientes internos. Também identificaremos as funcionalidades do sistema.
- Produzir as **plantas elétricas e hidráulicas** de forma a estruturar e promover funcionalidades e recursos para cada área e divisão do prédio. Da mesma forma, iremos criar diversos documentos, com objetivos diferentes, que serão integrados entre si, juntando-os dentro de um projeto único (foco e objetivo único).

11

2.2. Abstração

Abstração refere-se à capacidade humana de traduzir o que está escrito e diagramado e imaginar como aquilo será no futuro.

Ao ler a planta de uma casa conseguimos entender a disposição e tamanho dos cômodos. Ao ver um mapa de uma estrada, conseguir interpretar quais serão as cidades pelas quais passaremos ao longo do

nosso trajeto. Sob essa mesma ideia, ao criarmos a documentação UML, precisaremos ter a capacidade de interpretar os diagramas e textos criados naquilo que futuramente chamaremos de sistema!

Quando modelamos um sistema, o material que produzimos não completa exatamente TODAS as características do sistema, mas sim aquelas**essenciais** para formar a estrutura do sistema.

As demais características do sistema são informadas à equipe de projeto por meio de documentos auxiliares como protótipos, desenhos de tela, exemplos de gráficos e relatórios, escolha de fontes, tamanhos e cores, e outros.

12

2.3. Da generalização para a especialização

Uma das formas mais tradicionais de se especificar e modelar um sistema de informação é partir de um conceito mais genérico, central, geral, amplo, grosso modo (generalização) para um conceito mais detalhado, específico, determinado, definido, claro, restrito (especialização).
Utilizaremos essa técnica no fluxo de trabalho de modelagem de um projeto de sistema de informação: partiremos de diagramas que representam conceitos genéricos, passando por diagramas que representam conceitos intermediários, até chegarmos a diagramas que representam conceitos específicos e detalhados.
À medida que o projeto de sistema de informação vai evoluindo, novas ideias são clarificadas, definidas e detalhadas. Muitas vezes é necessário voltar aos documentos já elaborados para refiná-los, corrigi-los ou adequá-los aos projetos.

13

3 - Automação por meio da arquitetura dirigida ao modelo

A arquitetura dirigida ao modelo (do inglês – Model-Driven Architecture – MDA) é uma metodologia de desenvolvimento de *software* que permite a estreita relação entre os diagramas UML e o código fonte. Ferramentas de desenvolvimento de *software* que suportam essa tecnologia fazem com que qualquer mudança feita nos diagramas UML automaticamente reflita no código fonte do programa. O contrário também é verdadeiro, qualquer mudança no código fonte atualiza a documentação UML. Desta forma, conseguem-se os seguintes benefícios:

• A documentação do projeto nunca fica desatualizada em relação ao código fonte.
• Facilita o desenvolvimento.
• Modelos de alto nível independente de plataformas (tecnologias).

A documentação do projeto nunca fica desatualizada em relação ao código fonte.
Sempre que o desenvolvedor altera, corrige, cria ou retira parte do sistema, a documentação automaticamente reflete essas mudanças.

Facilita o desenvolvimento.
Por criar toda a estrutura do projeto, fica mais fácil programas o sistema. Também diminui erros de

programação por falta ou excesso de código em divergência com o que foi estipulado.

Modelos de alto nível independente de plataformas (tecnologias).
Com isso, os modelos de alto nível ficam abstratos, o que permite que seja possível realizar mudanças de tecnologia sem alterar a especificação do projeto (portabilidade). Exemplo: poderia, hipoteticamente, trocar o banco de dados de uma aplicação de Oracle para Ms SQL e o *software* de desenvolvimento automaticamente adequaria o código fonte de acordo com os parâmetros e características da outra linguagem. Ao programados cabe apenas autorizar a conversão do sistema, sem necessidade de programação complementar.

14

4 - Perfis profissionais de TI que utilizam UML

Basicamente, são três os perfis profissionais que utilizam a UML:

Analistas de Negócio, Modeladores de sistema

- São os profissionais que descrevem as necessidades dos usuários em um conceito sistematizado (que se tornará um sistema de informação). Fazendo uma analogia com a medicina, são eles que descrevem os sintomas e os problemas dos pacientes.

Analistas de Sistema, Designers, Engenheiros de *Software*, Administradores de dados

- São os profissionais que exploram as possibilidades de soluções e tecnologias a fim de resolver as necessidades dos usuários. Na ideia de uma analogia com a medicina, eles entendem a doença e prescrevem o remédio e/ou a terapia adequada.

Implementadores, Desenvolvedores, Programadores

- São os profissionais que criam o *software* a partir da solução definida e escolhida. Atualmente, as modernas linguagens de programação atuam em conjunto com os modelos da UML criando não só o sistema de informação do projeto de *software* proposto, mas também programas auxiliares de testes e até mesmo o modelo de dados da aplicação. Na analogia com a medicina, eles são os enfermeiros e terapeutas que aplicam o remédio ao paciente e controlam realização da terapia proposta.

5 - Conceitos Gerais

Para trabalharmos com UML é essencial que alguns conceitos de orientação a objetos estejam bem claros para você:

Item	Descrição	Exemplo
Objeto	Refere-se a algo útil que tem uma identidade, uma estrutura e um comportamento.	O aparelho de ar-condicionado que está na minha sala é único em relação a todos os demais aparelhos do prédio. Ele está situado no teto, próximo à janela e mede aproximadamente 1m de largura por 20 cm de altura. Ele se comporta bem, resfriando o ar quente da sala, deixando-a confortável para o trabalho.
Classe	Um conjunto de objetos similares quanto à estrutura e comportamento.	Todos os aparelhos e ar-condicionado resfriam o ar. Eles têm um controle que permite ligar, desligar e regular a temperatura para um determinado nível.
Abstração	Descreve a essência de um objeto para um propósito.	Um diagrama elétrico de um aparelho de ar-condicionado descreve como a energia elétrica percorre seus circuitos, dessa forma, você sabe manipular internamente o ar-condicionado sem tomar um choque elétrico.
Encapsulamento	Descreve apenas o que eu preciso saber para usar um objeto	Para operar o ar-condicionado, eu não preciso saber como funciona o compressor, as tubulações e o conversor térmico. Basta saber ligá-lo à energia elétrica, pressionar o botão de ligar e regular a temperatura ao nível que eu desejar.
Informação oculta	Mantém as coisas simples suprimindo detalhes.	A maioria das pessoas não precisa saber que o aparelho de ar condicionado precisa de um disjuntor elétrico de 15 amperes de energia. Também não é relevante nesse contexto falar como o aparelho é fixado à janela ou onde está conectado o duto de saída de água.

Agregação	Informa tudo o desejável sobre o objeto.	O ar-condicionado resfria o ar da sala puxando o ar quente por um duto, resfriando-o e expelindo-o de volta por outro duto. Há ainda a possibilidade de puxar o ar externo ao ambiente ao invés do ar da sala. Ele é composto por uma unidade interna com ventilador e um filtro, um condensador externo, tubulação ligando essas duas unidades, conexões elétricas e um controle remoto.
Generalização	Informa o que é comum entre os objetos.	Todo aparelho de ar-condicionado possui um filtro de ar e um ventilador interno para movimentar o ar.
Especialização	Informa o que é diferente em um objeto em particular. É o oposto da generalização.	O meu aparelho de ar-condicionado possui um mostrador digital na cor verde que representa a temperatura atual do ambiente. No controle remoto há outro mostrador, desta vez azul, que representa a temperatura que o aparelho está configurado para funcionar.
Herança	Objetos específicos herdam características comuns de objetos genéricos.	Assim como o meu aparelho de ar-condicionado, todos os aparelhos de ar-condicionado são capazes de resfriar um ambiente.
Polimorfismo	Nomes idênticos para operações semelhantes em objetos diferentes.	Todo aparelho, de qualquer natureza, tem a tecla “on”, essa tecla é a que liga o aparelho. Assim como toda tecla “off” desliga qualquer tipo de aparelho.

16

No dia a dia, nós reproduzimos e utilizamos vários dos conceitos da orientação ao objeto sem perceber. Entender esses conceitos é importante, pois facilita o estudo da matéria.

Quando olhamos um mapa de uma estrada que liga duas cidades quaisquer, por exemplo, nós não vemos a estrada real, mas sim uma representação daquilo que encontraremos ao dirigir pelo caminho. Ao “ler” o mapa, interpretamos as linhas e curvas que vemos como a estrada que iremos percorrer. Essa interpretação do desenho na estrada de fato é o que chamamos de “abstração”. Quando modelamos um *software* não vemos nos diagramas que criamos o *software* em si, mas sim uma representação daquilo que queremos construir. O mapa também não contém as árvores, as placas e sinalizações, pontes e viadutos que encontramos pelo caminho. Apenas as informações relevantes estão lá presentes, as demais são omitidas. Assim é na modelagem, apenas o que é realmente importante deve estar

presente, os demais detalhes surgirão em outro momento com documentos de apoio, protótipos, exemplos etc. A modelagem deve focar naquilo que é essencial para o sistema funcionar de maneira completa.

Quando traçamos um plano de viagem, incluímos informações de previsão do tempo, condição das estradas, tempo previsto para a viagem, lugares para abastecer o veículo e se alimentar, postos de pedágio previstos, horário de saída e chegada.

Todas essas informações reunidas em um plano de viagem único e completo é o que chamamos de **agregação**. Agregar é juntar em um pacote tudo aquilo que diz respeito a um terminado assunto. Ao modelar sistemas, precisamos compreender as informações afins e agregá-las para obter a melhor coesão possível.

17

Já o termo **encapsular** refere-se a esconder os detalhes internos daquela funcionalidade de quem a acessa, o que é importante não só para dar mais segurança à aplicação, mas também para não poluir o próprio ambiente de desenvolvimento. Por exemplo, você sabe que ao ligar o aparelho de ar-condicionado ele funcionará, você não precisa saber como os fios, motor, tubulações e demais peças estão conectadas internamente. Pois bem, essas peças todas estão encapsuladas em um objeto chamado ar-condicionado.

Dica: quando você precisar esconder as partes internas de um objeto, use a notação da UML de agregação. Dessa forma, você estará escondendo a complexidade de todo um objeto do mundo exterior que o acessa (os demais objetos que interagem com ele).

A composição é um tipo especial de agregação, para entender a composição é mais fácil quando comparamos as duas:

Composição	**Agregação**
Representa um único objeto, indivisível, cujo ciclo de vida é integrado, se evento encerra a vida de parte do objeto, todo o objeto é extinto.	Representa um objeto constituído pelo relacionamento de vários outros objetos. Cada subparte tem seu ciclo de vida próprio, o encerramento de uma parte não interfere no ciclo de vida das demais subpartes.

Ex. Se um aparelho de ar-condicionado queima o condensador, é correto pensarmos que todo o aparelho estragou, pois a parte “condensador” é indispensável para o funcionamento do ar-condicionado.	Ex. Se um caminhão carrega uma carreta, e essa carreta é removida do caminhão, ele continua funcionando normalmente, sem nenhuma interferência. A remoção da carreta não faz com que o caminhão torne-se imprestável, o ciclo de vida dele continua independentemente da carreta.

Observe que a diferença básica é a força com a qual os objetos estão ligados entre si. Na composição, a força é extrema e vital, já na agregação é uma espécie de “parceria suave”.

18

5.1. Algumas outras palavras técnicas no mundo orientado ao objeto

Diferentemente do mundo estruturado antigo, onde tínhamos apenas os conceitos de variáveis, constantes, fórmulas, funções e procedimentos, o mundo orientado ao objeto trouxe consigo uma série de novos elementos. Vejamos os principais:

Elemento	Significado	Exemplo
Componente	Um objeto do mundo real feito em código de *software* que é completo e pode ser melhorado ou substituído por outro objeto, sem que o usuário saiba a diferença.	Você pode substituir seu aparelho de DVD por um outro aparelho de DVD mais moderno com funcionalidades iguais ou superiores. Da mesma forma, você pode substituir um módulo do código do seu sistema por outro módulo que funcione melhor.
Desenvolvimento baseado em componentes	Uma forma de construir sistemas modularmente, onde você pode facilmente substituir unidades de código sem afetar o restante do sistema.	Desenvolver um sistema utilizando componentes Enterprise Java Beans, Microsoft .Net ou CORBA.
Interface	Uma especificação que diz o que o objeto deve fazer (mas não como fazer).	Um aparelho de DVD deve possuir saídas de áudio padrão (exemplo, tipo RCA).
Padrão (Pattern)	Descrição de como os programadores resolvem um problema/necessidade que	Usar um adaptador padrão para fazer com que uma classe de interface existente realize uma

	ocorre frequentemente.	tarefa determinada.
Framework (sem uma boa tradução para o português)	Framework é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação. Ditam a arquitetura da aplicação.	Você pode implementar um programa de reserva de hotel usando o framework orientado a eventos utilizando telas GUI.
Ferramenta de modelagem UML	Um *software* utilizado para desenhar diagramas de UML de maneira integrada e inter-relacionada e documentar sistemas de informação. Alguns ainda criam parte do código fonte da aplicação.	Há vários *software*s de modelagem UML, aqui você encontra uma lista de vários *software*s de modelagem UML: http://goo.gl/Yo0CQ4.
Ciclo de vida	Uma sequência genérica de passos desde o início até o fim que qualquer um do time de desenvolvimento precisa seguir para considerar o projeto de *software* corretamente desenvolvido (bem feito).	Para a maioria dos projetos de *software* é aplicável um ciclo de vida que começa com uma etapa de análise, seguida por uma de desenvolvimento.
Metodologia (de desenvolvimento de *software*)	Uma definição detalhada de todas as tarefas necessárias para desenvolver um sistema de informação para uma determinada equipe de desenvolvimento.	Há vários modelos proprietários e públicos que tratam deste assunto. RUP, OpenUp, OMT, Scrum, eXtreme Programming, são alguns exemplos.

19

6 - Criando aplicações com programação orientada a componentes

Antigamente, quando falávamos de programação estruturada, os programadores normalmente criavam os programas baseando-se numa sequência lógica: primeiro se criava a tela principal do sistema, com o login de acesso e o menu de funcionalidades, depois criava-se os cadastros básicos, depois os cadastros mais complexos, depois as operações do sistema e por fim, os relatórios. Quase todas as equipes utilizavam essa ideia de processo de criação de *software*.

Entretanto, para o mundo orientado a objetos essa técnica mostrou-se ineficiente.
Construir *software* orientado a objetos tem outra lógica, outra ideia: agora, os programadores criam os diversos pedaços (componentes) de *software*, cada um especializado em um tipo de tarefa. Exemplo: cria-se um componente de interface de usuário, outro de gerenciamento de login e segurança, outro

para cadastros, outro para relatórios, tudo de forma bem genérica. Depois, é hora de montar o sistema, juntando os diversos componentes em outros componentes mais específicos, que realizam as tarefas pontuais. Para isso, reutilizam-se os componentes genéricos integrando-os por meio de componentes mais especializados.

A maioria das linguagens de programação moderna, como Java, Ms .Net, PHP e outras, já possuem prontos todos os componentes genéricos utilizáveis por qualquer tipo de *software*. Desta forma, o trabalho do programador fica "simplificado" na arte de juntar esses componentes naquilo que é o propósito do projeto de *software*.

20

Ainda, por meio desses componentes (que já foram testados exaustivamente), o risco de incorrer em erros de *software* é cada vez menor. Assim como a segurança, a qualidade e a performance devem ser cada vez maiores. Dessa forma, a criação de programas orientados ao objeto deve seguir as regras:

1. **Criar componentes**;
2. **Separar o que o componente pode fazer do como o componente faz**;
3. **Promover um método padrão para comunicação entre os componentes**;
4. **Permitir que seus componentes existam em um ambiente padrão**.

Use diagramas da UML para descrever como as partes da sua aplicação foram montadas:

- Use os diagramas de classes, de composição de estruturas, de sequência e de comunicação para descrever como a parte interna dos seus componentes atuam.
- Os diagramas de classe mostram os atributos e as operações de cada objeto pertencente aos seus componentes.
- Os diagramas de composição de estrutura apresentam as partes internas que formam cada componente.
- Os diagramas de sequência mostram a interação entre os componentes ao longo do tempo.
- Os diagramas de comunicação representam a complexidade das interações internas das partes dos seus componentes.

Criar componentes
Escrever (ou utilizar) unidades de *software* como grupo de objetos que cooperam entre si, os quais você pode reutilizá-los em várias partes da sua aplicação ou mesmo em várias aplicações. Nesta etapa é muito comum utilizar "bibliotecas" de *software* feitas por outras pessoas. Exemplos: bibliotecas para editar texto, bibliotecas para tratar de assinatura digital, bibliotecas para tratar de arquivos multimídia, bibliotecas para tratar criptografia, etc.

Separar o que o componente pode fazer do como o componente faz
Você precisa declarar interfaces nos seus componentes. Cada interface especifica o nome da operação e os parâmetros necessários para ela funcionar. Quando um componente invoca a interface de outro componente, ele não precisa saber absolutamente nada de como aquela tarefa será realizada. Exemplo: quando seu programa faz uma chamada em um componente que reproduz um arquivo MP3, você não precisa saber como o componente fará para tocar a música, ela simplesmente soará por conta da codificação interna que o componente de áudio possui para tocar músicas MP3. Para você, basta utilizar as operações preestabelecidas para tocar, parar, adiantar e/ou voltar a música para o início.

Promover um método padrão para comunicação entre os componentes
Para permitir que os componentes possam ser substituídos (quando necessário), você precisa padronizar a modo exato com o qual os componentes conversam entre si. Os padrões CORBA e Microsoft COM são dois exemplos de padrões de comunicação.

Permitir que seus componentes existam em um ambiente padrão
Os seus componentes devem ser capazes de a) criar instâncias de outros componentes; b) descobrir quais interfaces outros componentes disponibilizam; c) registrar a si próprios, de modo que outros componentes possam encontra-los e invocá-los. O Enterprise Java Beans (EJB) é um bom exemplo de ambiente de componentes. O EJB provê meios padrão para criar, registrar, encontrar, executar e deletar componentes.

21

6.1. Usando ferramentas de UML

Diagramas UML são fáceis de desenhar, entretanto, requerem certa dose de “dom artístico” para ficarem legíveis e bem organizados. Você poderia desenhar diagramas UML usando um papel e uma caneta, mas certamente seria impossível manter as atualizações no diagrama bem como ler e interpretar dezenas e dezenas de diagramas.

O que precisamos é de uma ferramenta de modelagem UML, formalmente conhecida como **ferramentas CASE** (Computer-Aided *Software*Engineering – Engenharia de *Software* Assistida por Computação). Uma ferramenta de modelagem faz muito mais do que desenhar diagramas.

Com essa ferramenta é possível:

- Criar diagramas inter-relacionados, onde é possível ver o detalhamento de um diagrama em outro, ou a sequência de um em outro.
- Desenhar notações UML corretamente. É importante que você mantenha o padrão exato de cada item, pois não faria sentido precisarmos de uma legenda toda vez que analisássemos um diagrama novo.
- Organizar as anotações e diagramas em pacotes. Em projetos grandes, a quantidade de itens que compõe o seu projeto pode ser de centenas ou milhares. Ter uma ferramenta que permita não só organizar, mas também realizar pesquisas (localizar) é fundamental.
- Engenharia reversa. Essa é uma funcionalidade extremamente útil: a partir de um código fonte a ferramenta de UML é capaz de gerar a documentação dela. Essa mesma funcionalidade é utilizada para manter a sincronia entre os diagramas e o código fonte.
- Relatórios. Toda ferramenta de UML deve ser capaz de imprimir não só os diagramas, mas também listagem de classes, objetos e dados estatísticos sobre a modelagem.
- Geração de código. Essa funcionalidade trabalha em sincronia com a engenharia reserva, ela é que gera parte da estrutura do software a partir dos diagramas criados.

22

Existem mais de 200 ferramentas de UML atualmente, a escolha daquela mais adequada ao seu projeto deve levar em conta:

- o custo da ferramenta (e disponibilidade de recurso financeiro),
- a tecnologia utilizada,
- o *framework* de desenvolvimento,
- o tamanho do projeto, e
- a expertise da equipe.

Considere ainda as seguintes características de projetos de *software*:

• Sistema de informação.

• Sistemas com comportamento baseados em tempo real.

• Sistemas de banco de dados.

• Sistemas WEB.

Sistema de informação
Se o seu projeto pretende processar informação, você precisa de uma ferramenta que trabalhe bem todos os diagramas da UML (deve ser bem completa).

Sistemas com comportamento baseados em tempo real
Neste caso, você precisa de uma ferramenta que trabalhe bem com questões temporais e eventos. Procure uma ferramenta que implemente bem diagramas de estado.

Sistemas de banco de dados
Neste caso, você provavelmente precisará mapear transações ou data warehouse (armazém de dados). Dessa forma, considere ferramentas que incluam representação de diagramas lógico e físico de dados, e que gerem código SQL para criação de bancos compatível com a tecnologia de banco de dados da sua aplicação.

Sistemas WEB
Aqui você provavelmente precisará de ferramentas que sejam capazes de criar scripts de código e serviços WEB. Estruturas de dados em formato XML, código em cliente (cliente-side code), e operações nos serviços (server-side code) são alguns dos exemplos de tecnologia que sua ferramenta deverá suprir.

23

Resumo

Neste módulo, aprendemos que:

a. UML é uma linguagem ou notação de diagramas para especificar, visualizar e documentar modelos de software orientados por objetos.

b. Atualmente são 13 os tipos de diagramas, cada qual com seu uso particular.

c. Nem todos os diagramas são aplicáveis a todos os tipos de sistemas. Cada projeto de sistema requer um conjunto específico de diagramas, cabe à equipe de projeto decidir quais diagramas fazem sentido em cada contexto do projeto.

d. A diagramação UML é aplicável a qualquer metodologia de desenvolvimento, desde que orientada a objetos.

e. Os diagramas UML podem ser classificados quanto aos tipos: estruturais, de comportamento e de interação. Também podem ser classificados quanto às suas características essenciais: estáticos, dinâmicos e funcionais.

f. Ferramentas modernas de modelagem incluem a tecnologia MDA que permite a sincronização entre o código fonte e a modelagem do sistema.

g. Há vários profissionais de TI que utilizam os diagramas UML para suas tarefas do dia a dia. Cada profissional tem um uso particular para os diagramas.

h. Os conceitos de UML estão estritamente ligados aos conceitos de programação orientada a componentes.

i. Há centenas de opções de escolha de ferramentas de diagramação UML, cada uma pode ser mais bem adequada a uma ou outra equipe. Há de se fazer uma boa análise antes de adquirir ou utilizar uma ferramenta de modelagem UML em um projeto.

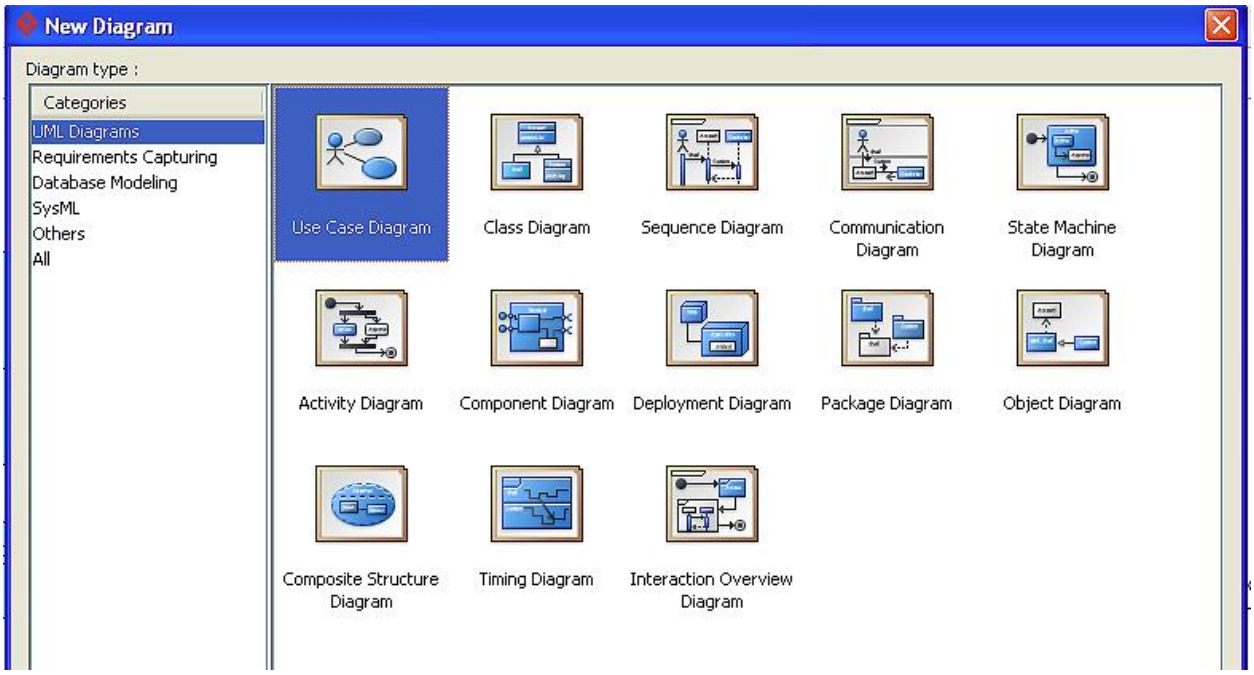
UNIDADE 1 – CONCEITOS INICIAIS SOBRE UML
MÓDULO 2 – DIAGRAMAS DE CASO DE USO

01

1 - CASOS DE USO

Neste módulo você aprenderá os conceitos mais comuns e utilizados no dia a dia no desenvolvimento de aplicações orientadas a objetos. Tanto atuando como um modelador de sistemas quanto como um programador, nós iremos familiarizá-los com os conceitos de casos de uso. Você verá detalhes importantes sobre a notação de modelagem de objetos por meio da UML e verá dicas importantes e boas práticas. Também você aprenderá a identificar os problemas mais comuns que você deve evitar.

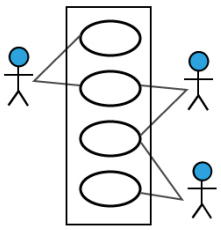
O início de um projeto de *software* é também o momento mais importante do projeto. Saber utilizar bem o tempo gasto nessa fase é primordial para obter uma boa concepção do projeto. Nesse momento você deverá ser capaz de identificar os principais objetivos do projeto e quais são as pessoas e mecanismos que interagem com o sistema.



Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de caso de uso.

02

O diagrama de casos de uso é o diagrama que apresenta o teor menos técnico de todos os diagramas. Isso se deve ao fato de este diagrama demonstrar como as pessoas esperam que o sistema funcione. Portanto, por ter a visão das pessoas comuns (usuários) é de se esperar que os termos que apareçam nele sejam relativamente fáceis das pessoas comuns entenderem.



Layout básico de um diagrama de caso de uso.

O diagrama de casos de uso incorpora os seguintes objetivos:

- Identificação dos usuários relevantes para o sistema (lembrando que um usuário pode ser também uma máquina ou outro sistema).
- Os serviços que os usuários executarão no sistema (algo como as funcionalidades que eles esperam que o sistema produza).
- Os serviços que os usuários precisam prover para o sistema (informações e procedimentos que devem ser inseridos e executados no sistema).

03

O diagrama de caso de uso pode representar uma atividade manual do dia a dia de uma organização a qual se quer ser transformada em uma atividade gerenciada por um sistema de informação. Muitas vezes, os insumos para se criar casos de uso são as teorias da administração, de gestão de pessoas, do direito civil, do direito comercial, pois são essas teorias (e também leis, regulamentos, regras etc.) que definem a forma de se trabalhar. Dessa maneira, evita-se um erro muito comum na criação de sistemas: ao invés de se criar sistemas que devem fazer a atividade do jeito certo, criam-se sistemas que executam as atividades informadas de maneira errada ou incompleta pelas pessoas que as realizam no dia a dia nas empresas.

É por isso que o trabalho do profissional de informática que faz a modelagem de sistemas não deve ser apenas o de documentar a rotina de trabalho atual, mas também procurar certificar-se que as instruções que lhe são passadas pelos usuários é a forma correta (e completa) de se trabalhar.

Dessa maneira, podemos concluir que as **atividades de elaboração de casos de uso** podem incluir:

- Entrevistas com os futuros usuários do sistema ou com pessoas especialistas no assunto (exemplo: entrevistar uma pessoa formada em Contabilidade para tratar de um sistema de contabilidade);
- Pesquisas em documentos, legislação, regras, manuais, livros e padrões de mercado;
- Comparativos com outros sistemas semelhantes que já existam.

Geralmente, ao modelar um sistema de informação temos dois cenários: o **atual**, onde há um problema para ser resolvido, e um **futuro**, que representa a situação que se quer alcançar.

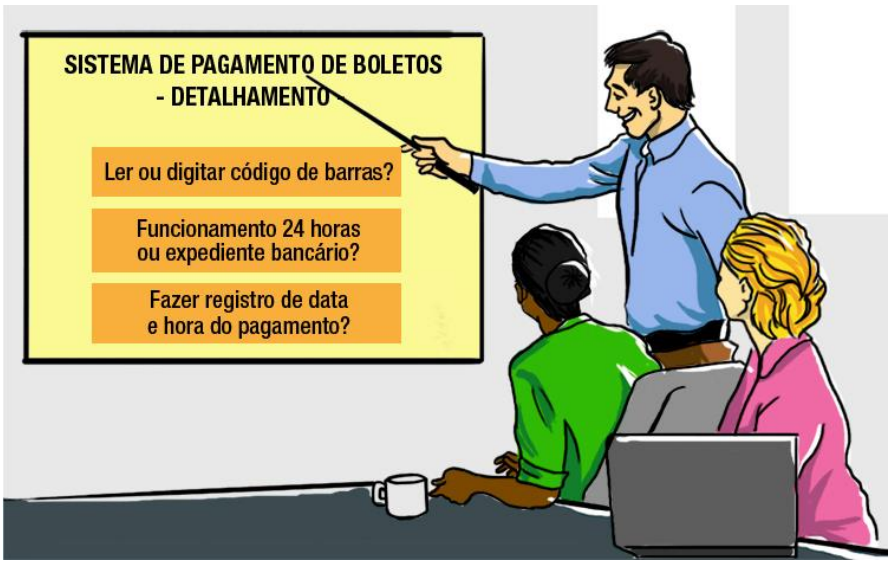
Modelar sistemas é saber traçar o melhor caminho entre a situação atual e o cenário futuro. Essa reta que liga esses dois pontos é o escopo do projeto, e o profissional que sabe construir essa reta da maneira mais inteligente, simples e eficaz, certamente criará o melhor projeto de *software* possível.

04

Para se definir os requisitos de um projeto de sistema de informação é necessário considerar duas dimensões:

- O escopo, ou seja, todas as funcionalidades que o sistema deve realizar; e
- O nível de detalhes (ou requisitos) que o sistema deve cumprir.

Por exemplo, não basta apenas dizer que um sistema qualquer deve permitir que o usuário faça o pagamento de um boleto bancário por meio do sistema, mas deve incluir todos os detalhes técnicos, tais como: que o boleto terá seu código de barras digitado no sistema (e não escaneado por um scanner de código de barras), que o sistema deverá funcionar 24h por dia (e não somente no horário de expediente bancário), que o sistema deverá guardar em registro interno quem foi a pessoa que realizou o pagamento, entre outras informações.



O escopo do projeto deve definir claramente quais são as fronteiras da aplicação. O que ela deve fazer e o que ela não faz, mas que é necessário para o projeto funcionar. Muitas vezes, como podemos estar tratando de sistemas que conversam com outros sistemas, então é importante deixar bem claro qual é a fronteira que limita essas duas aplicações e como elas trocarão informações entre si.

05

Um diagrama de caso de usos pode representar tanto a visão geral e total de todo um sistema, como pode ser detalhado em diagramas mais específicos até o ponto de representar apenas uma pequena funcionalidade do sistema. É comum termos relacionamentos do tipo um diagrama “pai”, mais genérico, que aponta para vários diagramas “filhos” (ou até mesmo netos, bisnetos etc.), mais detalhados e específicos.

Para permitir a documentação de todo o sistema, em seus diversos níveis, a UML provê o conceito de pacotes. A ideia dos pacotes é similar à de diretórios que podem conter subdiretórios e/ou arquivos. Cada subdiretório representa subconjuntos do sistema e cada arquivo um subdiagrama. Para o analista que irá modelar o sistema, o ideal é que se parta de uma visão mais macro e com o tempo ele vai detalhando até o ponto em que julgar aquele nível de detalhe ideal para a compreensão do projeto de *software*.

06

Como um sistema de informação é criado para desempenhar um conjunto de funcionalidades, é de se esperar que o levantamento de informações para a modelagem desse sistema comece pela identificação das funcionalidades que fazem parte do escopo do projeto.

A lista de funcionalidades deve:

- Definir todas as funcionalidades e seus desdobramentos em funcionalidades menores, mais específicas.
- Definir como as funcionalidades interagem entre si, por onde começam e onde terminam, quais são aquelas que podem funcionar de maneira isolada e quais dependem de outras.
- Definir os comportamentos específicos que o sistema deve realizar, o que é sucesso (e está ok), e o que pode ser considerado errado (e deve ser desfeito ou cancelado).
- Definir todas as interfaces com os usuários, sistemas, componentes etc.
- Prover um cenário de plano de testes desde o início do projeto (e não no final).

Os diagramas de caso de uso não devem focar no **processo de trabalho** ao longo do tempo, mas sim nos **objetivos** daquela funcionalidade. Dessa forma, focamos em **como satisfazer o usuário**, e não em como conduzi-lo à resposta esperada. Trabalhando dessa forma, a solução aparecerá como o menor caminho para se alcançar o resultado, evitando-se assim atividades que podem não agregar valor para o sistema.

07

Sob essa ideia, nossos futuros sistemas podem não ser uma representação “sistematizada” do que ocorre no presente, mas sim uma reinvenção da forma de se trabalhar, buscando alternativas mais vantajosas e eficientes. Em outras palavras, o sistema não deve focar em como as pessoas trabalham atualmente, mas procurar meios de alcançar os resultados do trabalho de modo mais simples, prático, barato, seguro, eficiente e rápido. E tudo isso é responsabilidade do modelador de sistemas! Exemplo prático.

Exemplo prático
Pense em um cinema que sempre vendeu ingressos na sua bilheteria de forma manual e agora é solicitado a você um sistema para modernizar a venda de bilhetes. Neste cenário você analisa a situação e percebe que empresa (cinema) possui os seguintes objetivos com o futuro sistema:

a) facilitar a venda de ingressos,
b) gastar menos tempo com a venda de ingressos,
c) reduzir custos de mão-de-obra na venda de ingressos,
d) aumentar as vendas de ingresso.

Dessa forma, ao invés de propor que se crie apenas um sistema para gerenciar a venda de bilhetes, você possa sugerir a criação de uma solução completa de tecnologia que permita a venda de bilhetes pela internet, por smartphones e em máquinas automáticas nas proximidades do sistema. Isso tudo facilitaria a vida do cliente e certamente potencializaria as vendas de ingressos. Percebe como a solução pode ser muito melhor do que o problema original?

08

2 - IDENTIFICANDO INFORMAÇÕES PARA OS DIAGRAMAS DE CASO DE USO

Conforme já dissemos, uma das primeiras atividades em um projeto de sistema de informação é definir o **contexto** e o **escopo** proposto para a aplicação. Para isso, você precisará responder às seguintes questões:

- Quais funcionalidades devemos incluir e excluir?
- Como o sistema se relaciona a outros sistemas no ambiente de trabalho?
- Quem usará o sistema?
- De quem ou do que o sistema depende para realizar suas atividades?
- Quais produtos ou resultados o sistema deve prover?
- Por que os usuários precisam dessas funcionalidades no sistema?

Dessa forma, pensando na teoria do encapsulamento, os diagramas de caso de uso representação o que o sistema faz e suas interfaces, e não como o sistema age internamente para realizar suas tarefas.

Geralmente os propósitos e interfaces dos sistemas se mantêm com o tempo, o que muda é a forma como internamente as coisas acontecem. Assim, os componentes do sistema podem evoluir e serem substituídos sem afetar a interação deles para com os usuários.

09

A prioridade na modelagem de sistemas é definir o seu propósito e suas interfaces.

O **propósito** representa a justificativa pela qual o sistema deve existir.

As **interfaces** representam os canais de comunicação entre os usuários do sistema (que chamaremos a partir de agora de **atores**) e as funcionalidades (que chamaremos de **casos de uso**).

O diagrama de casos de uso é composto por seis tipos de elementos gráficos que representam os atores, os casos de uso e os diferentes tipos de relacionamento entre eles, além da nomenclatura utilizada. O objetivo dos diagramas de caso de uso é prover uma visão externa entre o relacionamento do sistema e do mundo exterior.

Exemplo prático, um sistema hipotético de compra de ingressos de cinema (seja ela pela internet, *smartphone* ou equipamentos localizados na sala de cinemas) deveria oferecer aos usuários as seguintes possibilidades:

- Ver quais filmes estão passando e horários das próximas seções;
- Mostrar quais seções estão disponíveis para compra (e talvez quais já estejam esgotadas);
- Reservar assentos e comprar ingressos, pagando com cartão de crédito.

10

3 - DESCRIÇÃO DO CASO DE USO

O diagrama de caso de uso por si só geralmente não traduz o entendimento e os requisitos suficientes para a modelagem da situação, necessitando, portanto, de um texto narrativo complementar, chamado de descrição do caso de uso (ou narrativa do caso de uso). Não há uma forma padrão estabelecida para essa descrição, muito embora a maioria das organizações utilizem documentos de texto que contenha, ao menos, as seguintes informações:

- **Nome:** identificação do caso de uso.
- **Suposições:** condições que se julgam serem verdadeiras para a execução do caso de uso. Exemplo
- **Pré-condições:** situação prévia do sistema antes do processamento. Exemplo
- **Diálogo:** sequência de passos que o usuário deve fazer para utilizar o sistema corretamente. Exemplo
- **Pós-condições:** o que o sistema realiza para confirmar a operação. Exemplo

- **Exceções:** define como o sistema reage quando a operação não é executada a contento.Exemplos
- **Melhorias futuras:** ideias que são pensadas no momento da análise, mas não previstas no escopo atual. Essas ideias poderão ser parte futura de melhorias no sistema.
- **Questões abertas:** questões que ainda não estão claras de como o sistema deve reagir, essas questões serão discutidas com usuários e equipe de desenvolvedores a fim de se definir o comportamento ideal. Exemplos

Suposições - Exemplo
O usuário deve estar logado no sistema para que ele possa executar o caso de uso em questão.

Pré-condições - Exemplo
O sistema deve apresentar a tela de cadastro para o usuário.

Diálogo – exemplo
O usuário deve preencher os campos de nome e endereço, selecionar na lista a UF referente à cidade onde ele mora e executar o comando de “salvar”.

Pós-condições – exemplo

O sistema confere se o CPF do usuário é válido e, em caso positivo, armazena os dados no banco de dados, apresentando uma janela de diálogo que confirma a operação.

Exceções – exemplos
Algumas exceções podem ser:

a) caso o CPF do usuário não seja válido, o sistema deve cancelar a operação, apresentando janela de diálogo na tela informando que o CPF não é válido;
b) caso o sistema não consiga salvar os dados no banco de dados, o sistema deve informar em janela de diálogo que não foi possível salvar as informações, solicitando do usuário uma das opções a seguir:
    1) se o usuário deseja tentar salvar novamente ou
    2) se o usuário deseja cancelar a operação).

Questões abertas – exemplos
Podem ser questões do tipo:
O que o sistema deve fazer ao cancelar a operação? Deve apagar todos os dados da tela? Deve deixar os dados preenchidos para que o usuário possa tentar salvar posteriormente?

11

Vejamos a seguir um exemplo hipotético de um caso de uso de abertura de conta bancária.

• **Nome**: Abertura de conta bancária.

• **Suposições**

O cliente está de posse de toda a documentação necessária, o gerente do banco possui acesso às funcionalidades de cadastro de novo cliente e abertura de conta bancária no sistema do banco.

• **Pré-condições**

O gerente deve logar no sistema com sua conta pessoal e acessar a tela de funcionalidades do sistema.

• **Diálogo**

1. O gerente clica em “cadastrar novo cliente”
2. O gerente preenche os dados pessoais consultando os documentos apresentados.
3. O gerente clica em “incluir endereço residencial”.
4. O gerente preenche os dados de endereço e clica em “salvar”.
5. O gerente clica em “criar conta bancária”.
6. O gerente seleciona uma ou mais opções: “criar conta corrente”, “criar conta de poupança”, “criar conta especial com limite”, “criar conta associada a cartão de crédito”.
7. O gerente confirma a operação, imprime os documentos e solicita assinatura do cliente em todos.
8. O gerente digitaliza os documentos enviando-os para o sistema e arquivando os

documentos físicos na agência.
9. Fim.

• **Pós-condições**

O sistema informa que a conta foi aberta e apresenta instruções para geração de senha pessoal de acesso.

• **Exceções**

1. O sistema impede o cadastro de dados de CPF inválido.
2. O sistema impede a conclusão da abertura enquanto os documentos digitalizados não forem enviados para o sistema.

• **Melhorias futuras**

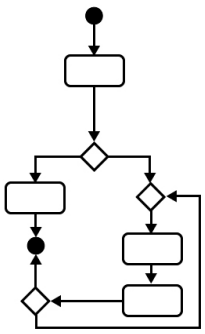
1. O sistema poderia permitir associar mais de um endereço para o cliente.
2. A geração da senha poderia ser feita no mesmo ato de criação da conta corrente.

• **Questões abertas**

1. Qual será o padrão para digitalização? PDF?
2. O sistema poderia permitir cadastro prévio de informações até que o cliente traga os documentos pessoais?
3. O sistema deve sugerir como padrão a abertura de todas as opções de conta e crédito?

12

4 -CENÁRIO DE CASO DE USO



Layout básico de um diagrama de cenário de caso de uso.

Uma das maiores forças do diagrama de casos de uso é que ele define o que o sistema (ou o componente) deve fazer. Para assegurar o sucesso do desenvolvimento, é necessário um plano de teste. Examinando as narrativas de caso de uso podemos identificar os cenários de caso de uso. Os “cenários”

são as formas que o sistema pode operar. Normalmente, em cada caso de uso é possível definir os seguintes cenários:

- O cenário que representa a sequência de atividades mais comum e curto do sistema (muitos autores chamam esse cenário de “caminho feliz”). Exemplo 1
- Outros cenários que representam alternativas, incorporando outras funcionalidades. Exemplo 2
- Cenários de exceção, onde o sistema deverá bloquear a operação, desfazer ações já feitas e/ou apresentar erros na tela. Exemplos

Os cenários de caso de uso podem tanto ser representados por um diagrama, utilizando um diagrama de atividades (veremos mais à frente), técnicas de fluxograma ou BPMN, ou utilizando narração descritiva.

Juntos, o diagrama de casos de uso, a descrição/narrativa do caso de uso, e os cenários compõem o conjunto de artefatos de caso de uso necessários para modelar um sistema.

Exemplo 1
O usuário seleciona o filme, o horário, a quantidade de bilhetes, define as poltronas onde quer sentar e efetiva a compra, realizando o pagamento.

Exemplo 2
O usuário pode pagar por meio de cartão de crédito ou por meio de transferência bancária.

Exemplos
A sessão escolhida está esgotada; o usuário não está satisfeito com as poltronas disponíveis e quer cancelar a operação (ou mudar de horário da sessão); o pagamento não pode ser validado e a operação deve ser cancelada.

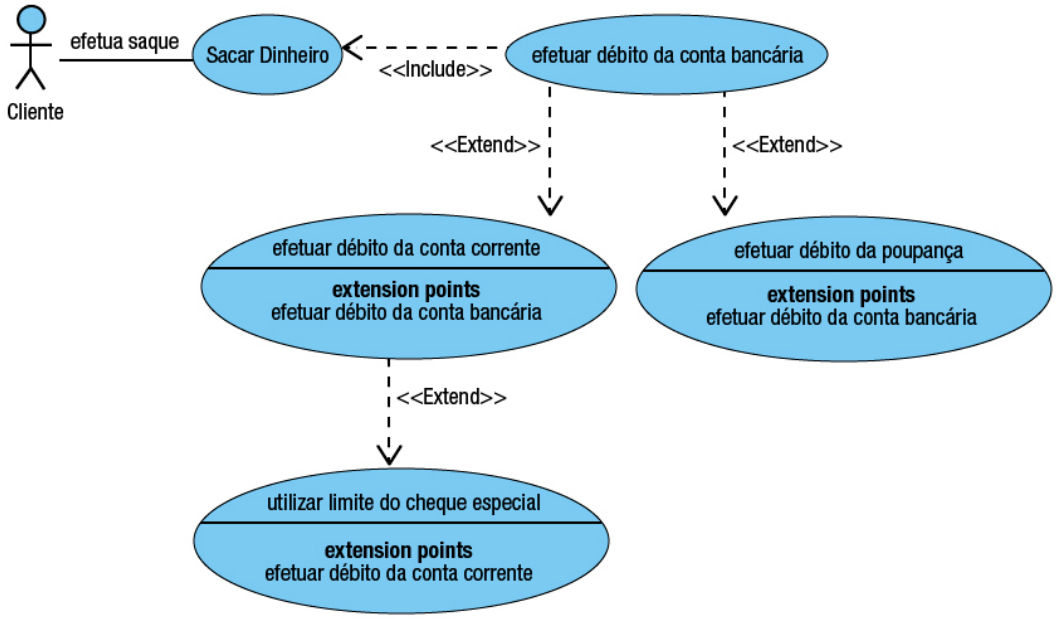
13

5 - DIAGRAMANDO UM CASO DE USO

Elementos que formam um diagrama de caso de uso:

- atores,
- casos de uso,
- associações,
- relacionamentos (<<include>> e <<extend>>) e

- generalizações.



Exemplo básico de um diagrama de caso de uso (utilizando a ferramenta Visual Paradigm).

14

• Ator

Representa um papel desempenhado por uma pessoa, um sistema, um equipamento, uma organização etc., que tem um interesse no sucesso da operação do sistema.

Exemplos: usuário, cliente, vendedor, operador do caixa, sistema bancário, empresa de correios, transportadora, departamento financeiro, setor de usinagem etc.

Um ator nunca será uma pessoa nomeada, mas sim um papel. Uma mesma pessoa pode executar, na prática, diversos papeis diferentes. Exemplo hipotético: o João, que é o gerente do banco XPTO, também é um cliente do banco, portanto, ora ele pode ser considerado no papel "gerente", ora no papel de "cliente". O ator é normalmente representado pelo desenho de um boneco, mas também pode ser substituído por uma imagem, logotipo ou outro desenho.

Exemplo de simbologia de um ator.

15

• Caso de uso

Identifica um comportamento chave do sistema (e não uma simples atividade). Sem esse comportamento, o sistema não resolveria a necessidade do ator. Lembre-se: um caso de uso deve ser sempre focado em um **objeto**.

Bons exemplos: “sacar dinheiro”, “efetivar compra do bilhete”, “reservar diária em hotel”, “cadastrar pessoa”.

Maus exemplos: “logar no sistema”, “selecionar o item no menu do sistema”, “preencher a tela com informações cadastrais”, “salvar dados”.

Muitas equipes de desenvolvimento preferem de usar verbos no infinitivo para os casos de uso, mas há também a possibilidade de utilizar verbos no presente (exemplo: “saca dinheiro”, “reserva hotel”). Mantenha sempre o padrão da sua equipe, evite misturar padrões. O caso de uso é representado por uma elipse com o nome do caso de uso escrito internamente.

Exemplo de simbologia de um caso de uso.

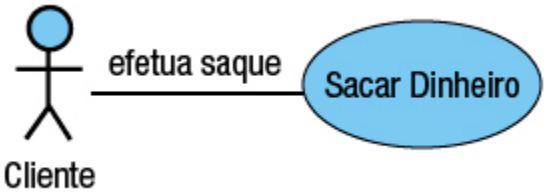
16

• **Associação**

Identifica uma interação entre os atores e os casos de uso. Cada associação inicia com um diálogo que deve ser explicado em um documento de narrativa do caso de uso. Cada narrativa em questão provê um conjunto de cenários que podem ajudar no desenvolvimento dos planos de testes para futura validação da análise, desenho e implementação do sistema.

Exemplo: se um ator do tipo “cliente” executa uma operação de sacar dinheiro em um caixa eletrônico de banco, então, há uma associação entre o ator “cliente” e o caso de uso “sacar dinheiro”.

A associação é diagramada por meio de uma linha comum (sem ser pontilhada).



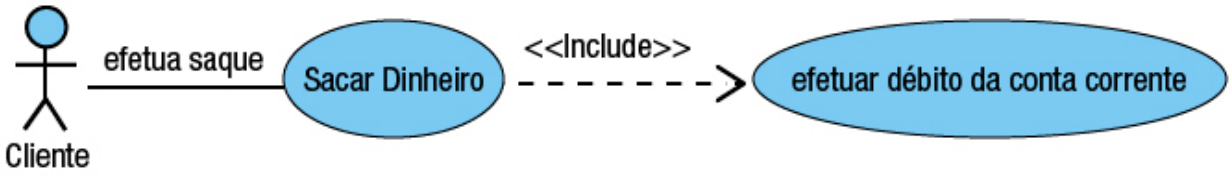
Exemplo de simbologia de uma associação de um ator a um caso de uso

17

• **Relacionamento do tipo “<<include>>”**

Identifica um caso de uso reutilizável (e compartilhado com outros casos de uso) que é obrigatoriamente incorporado na execução de outro caso de uso.

Exemplo: para que o caso de uso “sacar dinheiro” funcione, obrigatoriamente ele deve executar o caso de uso “efetuar débito da conta corrente”. O relacionamento do tipo “<<include>>” é representado por meio de uma seta pontilhada (com o texto <<include>>) que aponta para o item que incorporado ao caso de uso principal.



Exemplo de simbologia de um relacionamento do tipo include, ligando dois casos de uso

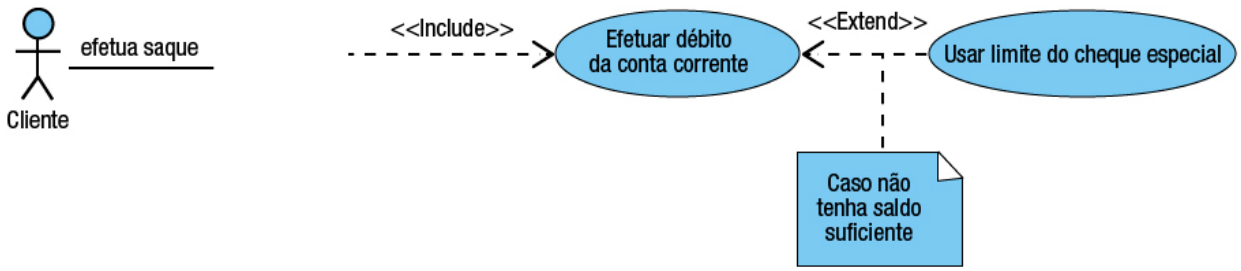
18

• Relacionamento do tipo “<<extend>>”

Identifica um caso de uso reutilizável que pode ser opcionalmente incorporado na execução de outro caso de uso. E, para ser incorporado, deve obedecer à uma certa condição.

Exemplo: para que o caso de uso “efetuar débito da conta corrente” funcione, pode ser necessário executar ou caso de uso “usar o limite do cheque especial”, que somente será executado caso o valor do saque seja maior que o valor disponível na conta corrente do cliente.

O relacionamento do tipo “<<include>>” é representado por meio de uma seta pontilhada (com o texto <<extend>>) que aponta para o item que invoca o caso de uso opcional (direção oposta do include). No exemplo abaixo vemos também a representação de uma nota explicativa (por meio de um ícone que representa uma nota).



Exemplo de simbologia de um relacionamento do tipo extend, ligando dois casos de uso

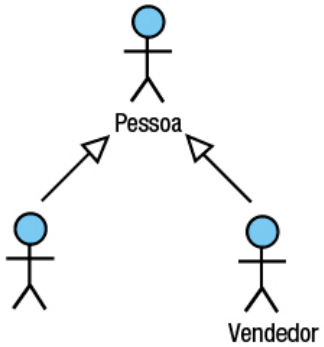
19

• Generalização/Herança

Identifica o relacionamento de herança entre atores ou entre casos de uso.

Exemplo: o ator “pessoa” pode ser a generalização dos atores “cliente” e “vendedor”, pois ambos compartilham (e herdam) propriedades similares ao ator “pessoa”. Já o caso de uso “abrir conta corrente” representa uma especialização do caso de uso “abrir conta bancária”, o mesmo vale para o caso de uso “abrir conta de poupança” que herda os comportamentos comuns de “abrir conta bancária”.

A generalização é representada por uma seta fechada lisa que sai do item mais específico e aponta para o item mais genérico.

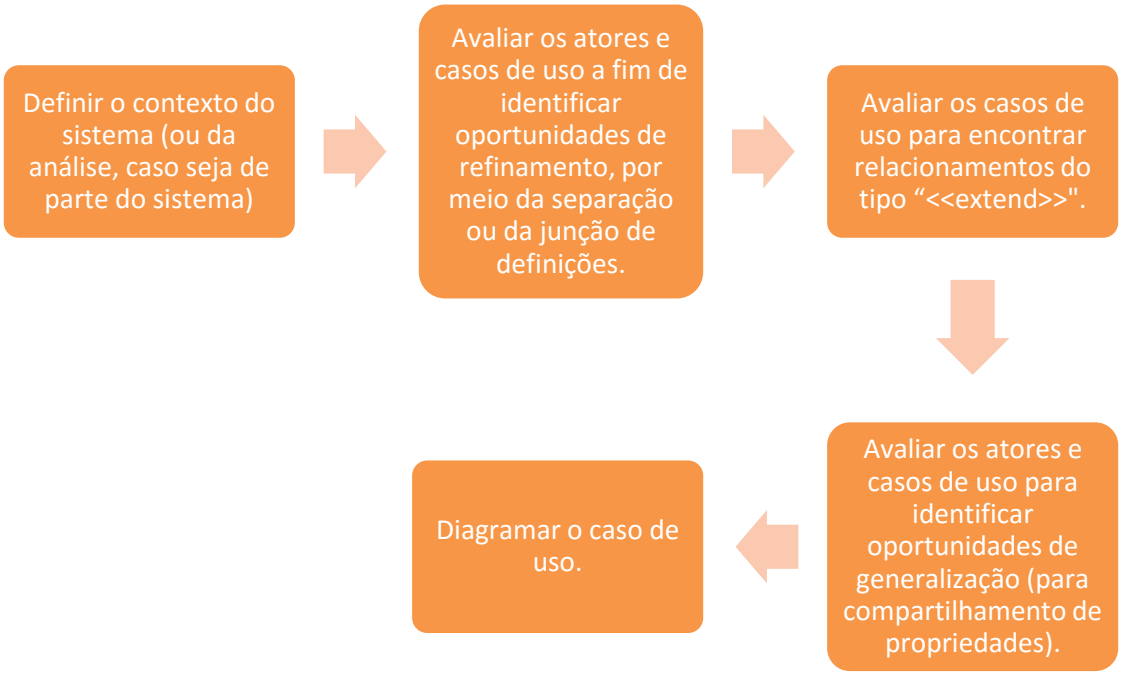


Exemplo de simbologia de generalização relacionando três atores (à esquerda) e três casos de uso (à direita)

20

5.1. Passo a passo para construção de diagramas de caso de uso

Para construir diagramas de caso de uso, você precisa:



Contexto do sistema
Na definição do contexto do sistema, deve-se:

1. Identificar os atores e suas responsabilidades;
2. Identificar os casos de uso e os comportamentos do sistema em termos de objetivos e/ou resultados que devem ser produzidos;

Documentar essas informações um documento a parte ou junto à narrativa de caso de uso.

6 - Exercício prático

A melhor forma de aprender a diagramar UML é executar exercícios práticos. Para esses exercícios que faremos agora, você não precisa nem de um software de UML, basta lápis e papel.

6.1. Máquina automática de venda de ingresso de cinema

Suponha que numa entrevista com um funcionário do cinema, ele tenha de explicado como funcionará a máquina de venda de ingressos: “As pessoas vão à maquininha e apertam na tela. Lá ele vai ver que filme está passando, aí ele escolhe o filme e o horário. Depois ele escolhe quantos ingressos quer comprar, depois ele escolhe onde vai sentar. Depois ele confirma, bota o cartão de crédito na máquina e compra o ingresso”.

Veja que a forma com a qual a pessoa narrou o fato precisa de uma série de ajustes de vocabulário para que a diagramação e documentação fiquem mais formal:

- A “pessoa” se tornará o “cliente”;
- A “maquininha” chamaremos de “máquina de venda de ingresso”;
- “Ver que filme está passando” escreveremos como “pesquisar filmes disponíveis”;
- “Ele escolhe o filme” transformaremos em “selecionar filme”;
- “E o horário” será “selecionar sessão do filme”;
- “Ele escolhe quantos ingressos quer comprar” será “selecionar quantidade de ingressos”;
- “Ele escolhe onde vai sentar” será “selecionar poltronas disponíveis”;
- “Ele confirma” será “confirmar operação de compra”;
- “Bota o cartão de crédito e compra” será ajustado para “realiza o pagamento com cartão de crédito”.

Sempre será necessário ajustar o diálogo entre o usuário/cliente e o texto que será documentado.

Feito isso, já conseguimos identificar os atores deste cenário; no caso, o único ator é o cliente.

22

Pensando em objetivos, percebemos que há dois objetivos no cenário:

1 - pesquisar e ver os filmes em cartaz e
2 - comprar bilhetes.

Para que esses dois objetivos sejam concretizados, outros menores devem existir para “auxiliá-los”, como por exemplo: selecionar assentos e efetivar pagamento. Uma informação nova é acrescentada ao contexto: a **multiplicidade**. Um cliente pode pesquisar um ou mais filmes, pode comprar um ou vários bilhetes, selecionar um ou vários assentos e, finalmente, realizar um único pagamento.

Para indicar a multiplicidade são usados os mesmos símbolos da orientação a objetos (1, 1..*, 0..*, etc., se você não se lembra desses símbolos, não se preocupe, eles serão revistos mais adiante). Com essas informações, já podemos desenhar nosso diagrama. Veja uma sugestão abaixo:

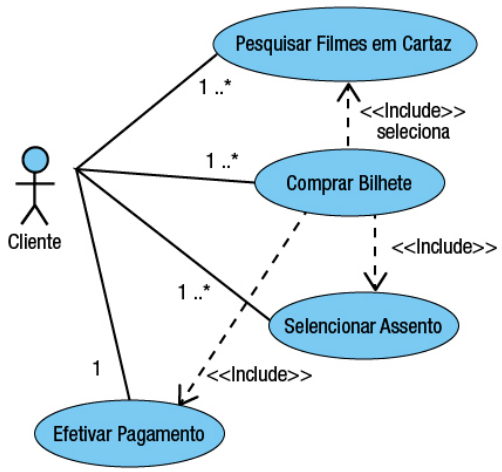


Diagrama de Caso de uso de Compra de Bilhete de Cinema

23

Com o diagrama em mãos, podemos realizar nova entrevista com o demandante do sistema a fim de se obter novas regras de negócio e assim poder descrever a narrativa de caso de uso. Uma sugestão hipotética do resultado dessa reunião de levantamento de informações poderia ser descrita nesta narrativa de caso de uso:

- **Nome**: Comprar bilhete de Cinema.
- Suposições
- Pré-condições

- Diálogo
- Pós-condições
- Exceções
- Melhorias futuras
- Questões abertas.

Suposições

1. Até 30 minutos após o início da sessão de cinema já iniciada, mas ainda com pelo menos um assento livre, a sessão deve ser considerada disponível para compra de bilhetes.
2. Após 30 minutos de início, a sessão de cinema deve ser considerada encerrada.
3. Após o esgotamento de assentos disponíveis, a sessão deve ser considerada esgotada.
4. Não deve ser possível a compra de bilhetes para sessões encerradas ou esgotadas.
5. Somente estará disponível a compra de bilhetes para sessões que acontecem no mesmo dia da compra.
6. O equipamento possui tela sensível ao toque para operação do sistema, todas as operações deverão ser realizadas por meio desta tela. Não há teclado, mouse ou outros botões físicos.
7. O equipamento possui dispositivo para leitura de cartão de crédito acoplado.
8. Caso o sistema permaneça por mais de 3 minutos sem que o cliente interaja com o sistema, toda a operação deve ser cancelada e voltada à tela de início.

O sistema deve possibilitar o cancelamento da operação em qualquer estágio da compra.

- **Pré-condições**
  1. Deve haver ao menos um filme em cartaz;
  2. Deve haver conectividade com sistema bancário (para pagamento com cartão de crédito);

Deve haver conectividade com o sistema central de gerenciamento do cinema (para pesquisa de informação sobre filmes disponíveis, sessões e assentos disponíveis).

- **Diálogo**
  1. Ao chegar à máquina de vendas, o usuário pode consultar algumas das informações já presentes da tela de inicial do sistema. Essa tela deve mostrar um resumo com todos os filmes em cartaz, apresentando uma imagem do filme, o nome do filme, o horário da próxima sessão de cada um e a quantidade de assentos disponíveis para compra da sessão. Caso esta sessão já esteja esgotada, o sistema deverá apresentar as informações para a sessão seguinte. Caso não exista sessão seguinte, o sistema deve apresentar a informação que a venda de ingressos está esgotada para aquele filme.
  2. O usuário clica em qualquer parte da tela para iniciar o uso do sistema.
  3. Ao clicar na tela, o sistema deve apresentar uma listagem de todos os filmes em

cartaz. Ao lado do nome de cada filme, o sistema deve apresentar todos os horários de início das sessões disponíveis para o dia. Para ser considerada disponível, é necessário ter ao menos um assento livre.

4. O usuário clica em uma sessão de cinema.
5. A tela apresenta uma imagem grande do filme, informando o horário de início e término da sessão e a quantidade de assentos disponíveis. A tela também apresenta opções para selecionar cada uma das outras sessões disponíveis e outro comando para cancelar a operação.
6. Se o cliente selecionar outra sessão disponível, a tela é atualizada com as informações respectivas da sessão selecionada.
7. Se o cliente clicar em cancelar, o sistema volta para a tela de início.
8. O cliente clica em “selecionar poltronas”.
9. O sistema apresenta uma tela com imagem representando todas as poltronas do cinema, marcando de vermelho escuro as poltronas indisponíveis e de azul claro as poltronas disponíveis. O sistema deverá incluir comandos para “cancelar a operação” (voltando para a tela de início), “selecionar outra sessão” (voltando para a tela anterior) e “finalizar compra”. A opção de “finalizar compra” só deve aparecer se ao menos uma poltrona (correspondente a um bilhete) estiver sido selecionada.
10. O cliente clica nas poltronas que deseja comprar. Ao clicar em uma poltrona, esta deve ser destacada em amarelo na tela do sistema. Para desmarcar uma poltrona selecionada, basta que o cliente clique novamente na mesma poltrona, assim, ela será desmarcada, retornando à cor azul. As poltronas que estiverem selecionadas, nesse momento deverão ficar em estado de reserva por 3 minutos. Dessa maneira, outros clientes utilizando outras máquinas de compra ou mesmo a bilheteria ficam impossibilitados de comprar a mesma poltrona. O sistema deve ser capaz ainda de atualizar as poltronas disponíveis em tempo real, a cada segundo, sincronizando informações de compra dos diversos pontos de venda, evitando assim a compra duplicada da mesma poltrona.
11. Após a seleção das poltronas desejadas, o cliente clica em “finalizar a compra”.
12. O sistema apresenta uma tela resumo com a quantidade de bilhetes, o valor total da compra, o filme e sessão selecionados, e as poltronas selecionadas. A categoria do bilhete a ser computada neste momento é a de bilhete comum.
13. O sistema de permitir a troca individual da categoria dos bilhetes, possibilitando ao cliente substituir pelas categorias estudante e terceira idade (que fornecem 50% de desconto no valor da compra).
14. Ao mudar de categoria de bilhete, o sistema deve recalcular o valor total da compra.
15. O sistema deve solicitar ao cliente inserir o cartão de crédito na máquina para efetivar a compra.
16. O sistema deve solicitar a senha do cartão de crédito e a confirmação do pagamento.
17. O sistema deve processar o pagamento e concluindo-o, imprimir os bilhetes do cinema, a nota fiscal e o comprovante de pagamento do cartão de crédito.

Pós-condições

Após a confirmação do pagamento a venda é concluída, emitindo os bilhetes e transformando os assentos daquela sessão em assentos permanentemente indisponíveis.

- **Exceções**
    1. Caso o cliente clique em cancelar, a qualquer momento, o sistema deve voltar à tela de início.
    2. Caso ocorra problema no pagamento, o sistema deve permitir a substituição ou outro cartão ou a possibilidade de se cancelar a compra de bilhetes.

Caso o sistema fique mais de 3 minutos sem uma ação do cliente, a operação deve ser cancelada e o sistema deve voltar à tela inicial.

- **Melhorias futuras**
    1. Na tela de listagem de todos os filmes e sessões, ao lado de cada sessão poderia ser informada a quantidade de assentos disponíveis.
    2. O sistema de pagamento poderia possibilitar também a operação de débito em conta (cartão de débito) ou até mesmo de transferência bancária.
    3. Criar uma aplicação para smartphones que também permita a venda de bilhetes.

- **Questões abertas**
    1. Ao cancelar a operação, a operação deve ser imediatamente cancelada ou o sistema deve apresentar uma janela de diálogo solicitando confirmação ao cliente se ele deseja realmente cancelar a operação?
    2. Ao abandonar a operação, as poltronas pré-selecionadas devem realmente ficar até 3 minutos em estado de reserva?

24

Quanta informação não é mesmo?! Parece complicado a princípio, mas com a prática do dia a dia, esse trabalho não será tão difícil quanto parece, apenas um pouco “grande” ou um pouco “demorado”, mas não será complexo nem difícil demais para você.

Note ainda que para esse exemplo todos os fluxos opcionais já estão descritos no texto. Não seria realmente necessário criar um diagrama de cenário de caso de uso. Entretanto, se mesmo assim você desejar inserir um no seu projeto, abaixo você vê uma sugestão.

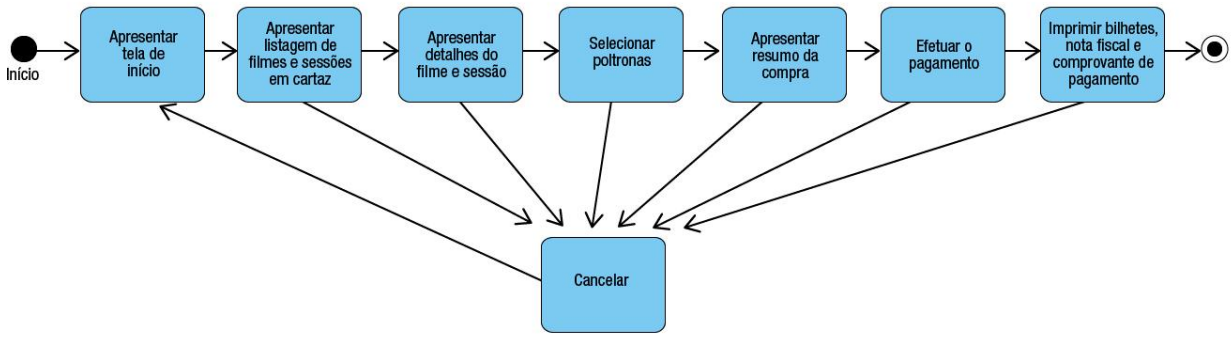


Diagrama de Sequência auxiliando o entendimento do caso de uso

Veremos o diagrama de sequência em detalhes em outro módulo.

25

7 - Casos de uso do tipo “manter alguma coisa”

A maioria dos sistemas de informação possui cadastros básicos, como os cadastros de cliente, de funcionários, de produtos, de cidade, de UF, de fornecedores, de usuários etc. Todos os cadastros básicos possuem quatro funcionalidades em comum em relação aos seus cadastros em bancos de dados.

Essas funcionalidades são chamadas pelos engenheiros de *software* de **CRUD**:

C – create (criar)	É a funcionalidade que cadastrar um novo item na base de dados.
R – report (reportar)	É a funcionalidade de poder pesquisar, encontrar e mostrar as informações cadastradas.
U – update (atualizar)	Refere-se à capacidade de poder atualizar uma informação previamente cadastrada com um dado mais atualizado.
D – Delete (excluir)	Funcionalidade de apagar informações do banco de dados.

Para realizar todas essas quatro operações (e uma quinta que é a de pesquisar/localizar), os engenheiros de *software*, por prática de mercado, definiram que todo sistema de informação precisa possuir ao menos um caso de uso para cada cadastro (tabela) no banco de dados denominado de caso de uso “manter”. Dessa forma, os primeiros casos de uso que você pode começar a identificar no seu sistema são os casos de uso do tipo “manter”.

Exemplos comuns desse tipo de caso de uso: manter cliente, manter funcionário, manter vendedor, manter produto, manter fornecedor, manter contas a pagar, manter contas a receber, manter cidades, manter UF etc.

26

Resumo

Neste módulo, aprendemos que:

a. Os projetos de *software* devem começar por meio da definição dos objetivos do *software*. Esta análise também nos ajudará a identificar os atores do sistema, que são pessoas e mecanismos que interagem com o sistema.

b. Há diversas fontes de informação para a elaboração de casos de uso. Documentos da empresa, regras de mercado, leis e entrevistas com os usuários são as opções mais comuns.

c. O desdobramento dos casos de uso trará as funcionalidades (e futuramente as classes) do sistema, e como elas interagem entre si e com outros sistemas.

d. Os casos de uso devem representar uma forma mais moderna e racional do trabalho atual, e não apenas a fiel repetição das atividades atuais.

e. Ao modelar sistemas, você pode trazer qualquer sugestão que ofereça algum tipo de melhoria ou benefício ao sistema e aos seus usuários.

f. A descrição de um caso de uso é uma narrativa complementar que fornece uma série de requisitos e informações sobre o funcionamento do caso de uso.

g. O diagrama de cenários de caso de uso representa o fluxo principal do caso de uso e outros fluxos opcionais e de exceção.

h. Relacionamentos do tipo “include” são obrigatórios e apontam para o caso de uso que “chama” o outro caso de uso.

i. Relacionamentos do tipo “extend” são opcionais e apontam para o caso de uso que é “chamado”. Ainda, esse tipo de relacionamento precisa de uma condição do uso do sistema para o caso de uso ser executado.

j. Atores são papéis representados no sistema por pessoas, subsistemas, equipamentos etc.

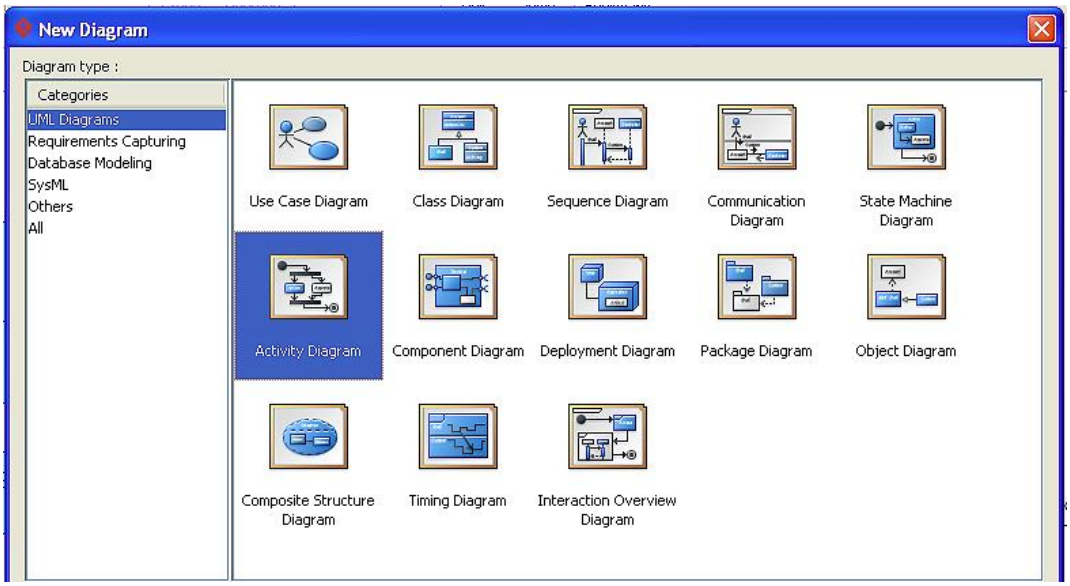
k. Os casos de uso representam objetivos do *software* a serem cumpridos.

l. A generalização é expressa por uma seta contínua que parte do item mais específico a aponta para o item mais genérico.

UNIDADE 1 – CONCEITOS INICIAIS SOBRE UML
MÓDULO 3 – DIAGRAMAS DE ATIVIDADES
01

1 - CARACTERÍSTICAS DO DIAGRAMA DE ATIVIDADES

Olá, seja bem-vindo a mais uma etapa do nosso estudo. Aqui você aprenderá os conceitos que estão por trás do comportamento do sistema, modelado por meio dos diagramas de atividades.



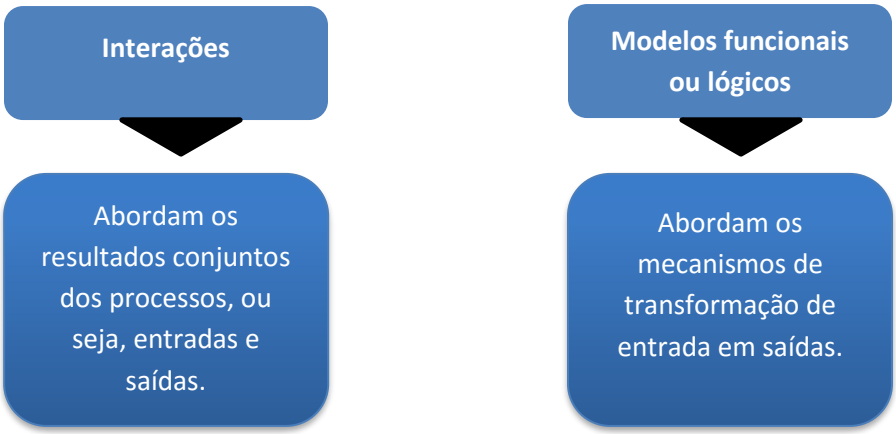
Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de atividades.

O diagrama de atividades é frequentemente visto como parte da visão funcional de um sistema, pois descreve os processos lógicos ou funções. Cada processo descreve uma sequência de tarefas e as decisões que regem quando e como eles são realizados.

É preciso compreender um processo a fim de escrever ou gerar o código correto para um comportamento.

02

Alguns autores juntam os aspectos funcionais e dinâmicos da modelagem. Isso se deve ao fato de que ambos expressam o comportamento do sistema. No entanto, para fins de aprendizado, é importante você saber distinguir a lógica da interação:



Além disso, a modelagem funcional adquiriu uma má reputação com o início da modelagem orientada a objetos (OO). Afinal, a orientação a objetos aborda as deficiências do modelo anterior, como a

modelagem funcional e de dados. Mas tanto a modelagem funcional quanto a modelagem de dados ainda fornecem informações valiosas sobre o desenvolvimento de *software*. Métodos de desenvolvimento OO não eliminam a necessidade dessas perspectivas valiosas; eles simplesmente trazem os dois conceitos juntos para fornecer um modelo mais abrangente e preciso de como as coisas funcionam. Modelagem funcional ainda é uma parte muito básica de qualquer *design* do aplicativo.

Assim, a UML preservou modelagem funcional sob a forma de diagramas de atividade, que é concebida para apoiar a descrição de comportamentos que dependem dos resultados dos processos internos, ao contrário de eventos externos, como nos diagramas de interação. O fluxo em um diagrama de atividades é impulsionado pela realização de uma ação. Em uma máquina de estado, o fluxo é conduzido por eventos ou condições externas associadas. Consequentemente, diagramas de atividades são úteis para operações que definem casos de uso e fluxos de trabalho que unem uma série de casos de uso.

03

2 - MODELANDO UM DIAGRAMA DE ATIVIDADES

Desde o passado até hoje em dia, muitos ambientes de negócios usam fluxogramas para representar processos de trabalho.

Processos são sequências lógicas de tarefas regidas por pontos bem definidos de controle, tais como as decisões, junções e separações.

A criação de um fluxograma é uma técnica simples e prática, com muitas aplicações. A UML oferece uma versão melhorada dos fluxogramas, sob a forma do diagrama de atividades, também chamado de **gráfico de atividades**.

Há pelo menos três lugares em um modelo onde um diagrama de atividades fornece informações valiosas:

- modelando fluxos de trabalho (workflow),
- descrevendo os casos de uso,
- especificando operações.

04

2.1 - Modelando fluxos de trabalho e casos de uso

Já aprendemos que um caso de uso modela um objetivo que o sistema deve atingir, a fim de ser bem sucedido. O diagrama de atividades mapeia o comportamento do usuário através de um procedimento, observando as decisões tomadas e as tarefas executadas. O procedimento pode explicar um único caso de uso, ou apenas parte de um caso de uso, ou mesmo muitos casos de uso unidos para criar um fluxo

de trabalho. Modelar um caso de uso é parte do processo de descoberta e validação para a modelagem de um sistema. Cada diagrama dá uma nova perspectiva sobre os recursos do sistema.

Um diagrama de atividades relacionado a um caso de uso explica como o ator interage com o sistema para cumprir a meta do caso de uso, incluindo as regras, informações trocadas, as decisões tomadas e produtos de trabalho criados.

Um diagrama de fluxo de trabalho (*workflow*) detalhado ao nível de atividades representa a ordem e as condições em que os casos de uso são executados em uma série (sequência).

Modelar o trabalho do usuário desta maneira não define essa versão particular do processo como a única forma correta do sistema trabalhar. Cada objetivo (caso de uso) pode ser satisfeito por qualquer número de processos válidos. Mas a criação de um modelo deste tipo provavelmente irá revelar os elementos essenciais do processo de uma forma que seja familiar para os utilizadores.

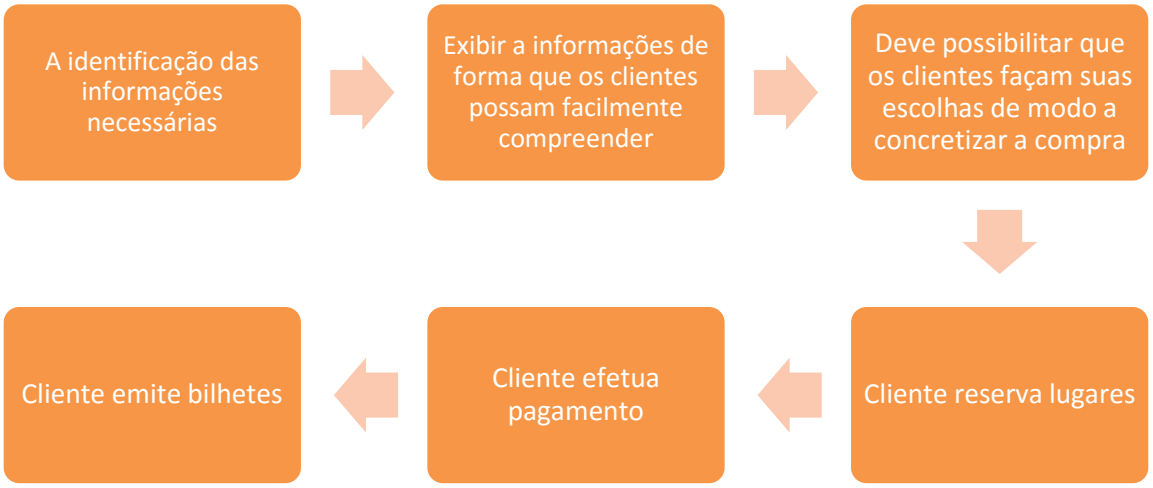
Elementos essenciais incluem:

05

O diagrama de atividades pode facilitar a entrevista para esclarecer as tarefas e as decisões a fim de encontrar as razões por trás delas. Ou seja, esta é a sua oportunidade de identificar como alcançar o objetivo do processo, e definir cada tarefa necessária para tornar o processo bem sucedido (passo a passo).

Uma vez que as regras essenciais para o processo foram identificadas, então é mais fácil de avaliar processos alternativos para satisfazer essas regras. Num sistema hipotético de **compra de bilhetes de cinema**, por exemplo, os clientes querem descobrir quais são os filmes, as sessões e os lugares disponíveis.

O objetivo básico é simples: mostrar a eles o que está disponível e capacitá-los para selecionar o que eles gostariam de comprar. As regras essenciais incluem:



06

No exemplo do sistema de bilhetagem de ingressos, tanto o objetivo quanto as regras associadas podem ser apoiados por muitos processos. Por exemplo, eles podem ser satisfeitos:

- por um ponto de venda (bilheteria) falando com um cliente ou por meio de uma máquina automática;
- por meio da impressão dos bilhetes em papel ou apenas a imagem deles em um *smartphone*,
- com o acesso ao sistema em uma página Web ou em aplicações para *smartphones* Apple e Android, por exemplo.

Observe que há muitas alternativas de como exibir o conteúdo e comunicar as escolhas.

O importante é que sem um requisito base é impossível criticar e avaliar com precisão o valor relativo de cada alternativa.

O diagrama de atividades é projetado para fornecer uma definição precisa dos requisitos básicos de uma perspectiva lógica, sequencial.

07

2.2 - Definindo métodos

O diagrama Atividade também pode ser usado para modelar a implementação de métodos complexos. Ao definir a implementação de uma operação, você precisa modelar a sequência de manipulação de

dados e transformações, lógica de programação e decisões. A modelagem dessas funções complexas pode evitar problemas futuros, quando você escrever o código, revelando todos os requisitos explicitamente no diagrama. A modelagem apresenta e permite o teste mental de todas as possibilidades possíveis, tornando-as mais fáceis de rever e corrigir.

É muito tentador resumir ou dispensar a modelagem as atividades, mas o verdadeiro valor da modelagem está em revelar o que sabe, para que ele possa ser contestado e verificado. Fazendo suposições visíveis (explícitas) a revisão e a correção tornam-se mais fácil (além de extremamente valioso).

As modernas ferramentas de UML implementam o diagrama de atividades com todas as construções lógicas que você encontra na maioria das linguagens de programação. Na verdade, ele se traduz muito bem em uma narrativa de código ou mesmo rascunho do código fonte.

As modernas ferramentas de UML implementam o diagrama de atividades com todas as construções lógicas que você encontra na maioria das linguagens de programação. Na verdade, ele se traduz muito bem em uma narrativa de código ou mesmo rascunho do código fonte.

08

3 - NOTAÇÃO DE UM DIAGRAMA DE ATIVIDADES

3.1. Atividades e transições

Uma **atividade** é uma etapa de um processo em que algum trabalho que está sendo feito. Poderia ser um cálculo matemático, encontrar alguns dados, manipulação de informações, ou a verificação dos dados.

A atividade é representada por um **retângulo** com os cantos arredondados e dentro dele um texto para descrever a ação. Por boa prática, as atividades sempre são escritas no idioma português utilizando verbos no infinitivo e um substantivo que dê a noção exata da ação que está sendo feita. Exemplos: calcular salário, pesquisar cliente, gravar dados do cliente, atualizar conta corrente, importar dados de venda, imprimir relatório fiscal etc.

Um diagrama de atividades contém sempre mais de uma atividade, ligadas por transições, representadas por setas que ligam cada atividade. Normalmente a transição ocorre porque a atividade é concluída. Por exemplo, você está atualmente na atividade de "ler a página." Quando você terminar a atividade, você alterna para a atividade "virar a página". Quando você terminar de virar a página, você volta para a atividade “ler a página”. Veja como representar graficamente essas duas atividades:

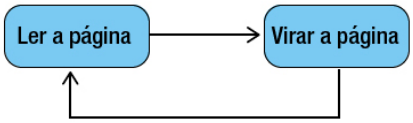


Diagrama de atividades representando o relacionamento entre duas atividades

09

3.1.1. Condição de guarda

Em um determinado cenário, uma transição de uma atividade para outra só pode ocorrer quando uma certa condição tornar-se verdadeira. Isso é especialmente útil quanto queremos representar um processamento contínuo até que todas as entradas tenham sido processadas.

Exemplos práticos para isso seriam:

- Incluir produto na nota fiscal, enquanto houver itens no pedido de compra a serem processados;
- Digitalizar documento, enquanto houver páginas a serem digitalizadas;
- Copiar arquivos, enquanto houver arquivos a serem copiados.

A representação de uma condição de guarda é feita por meio de um texto entre os sinais de "[" e "]". Exemplo: "[Enquanto houver arquivos a serem copiados]".

Para o exemplo do diagrama anterior, a página só pode ser virada quando todo o conteúdo da página atual tiver sido lido. Dessa forma, a representação do diagrama anterior poderia ser evoluída para:

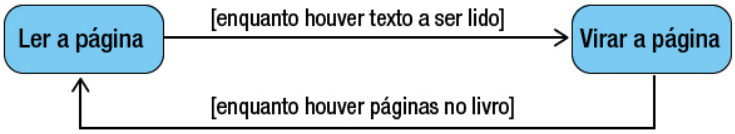


Diagrama de atividades representando o relacionamento entre duas atividades.

10

3.2. Início e fim de um diagrama de atividade

Para diagramar o início (que sempre é único) e o(s) encerramento(s) (que podem ser um ou vários) de um diagrama de atividades, utilizamos o símbolo de uma bola preta para simbolizar o início e uma bola preta com um aro externo para simbolizar o encerramento. O diagrama anterior poderia incluir a indicação de início e encerramento sendo assustado para:

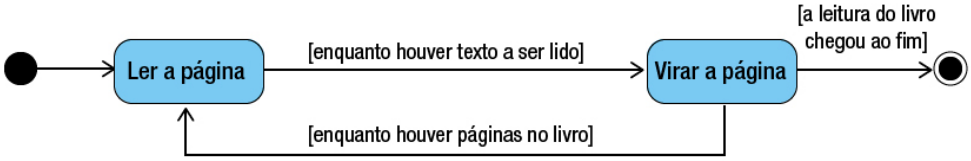


Diagrama de atividades incluindo a representação do início e do fim do processo.

11

3.3. Decisão

Uma decisão refere-se à possibilidade de o fluxo seguir um ou outro caminho dependendo de uma determinada condição.

Tanto na técnica de fluxograma quando na UML, a decisão é representada por um **losango**. A diferença é que no fluxograma geralmente há uma pergunta dentro do losango (exemplo: Qual foi a cor escolhida). Já na UML, não há a pergunta dentro do losango, mas, em cada linha de saída da decisão, há a representação da decisão escolhida entre os sinais de "[" e "]" (exemplos: [amarelo], [azul], [branco ou verde] – perceba que uma condição pode comportar mais de uma hipótese). Um exemplo de um diagrama com uma decisão está expresso a seguir:

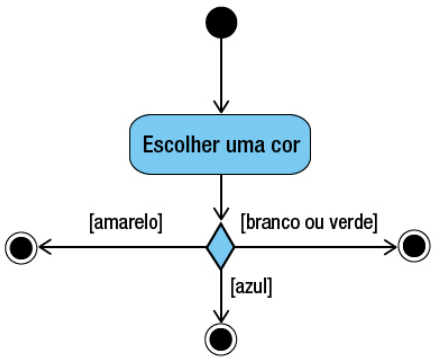


Diagrama de atividades representando uma condição.

As decisões podem ser descritas como expressões lógicas simples ou complexas, como por exemplo:

- [estoque >0]
- [existe vendedor disponível]
- [sexo = masculino]
- [sexo = masculino E estado civil = solteiro]
- [idade entre (18, 70)]
- [(sexo = masculino E estado civil = solteiro) OU (idade entre (18, 70)]

12

3.4. Junção

Junções representam a união de dois caminhos distintos.

A representação da junção também é feita por um losango que recebe todos os caminhos unidos. A condição para que o processo continue é a sensibilização de todos os caminhos juntados. No exemplo abaixo, perceba que a atividade “concluir a venda” só acontecerá quando as atividades “escolher itens” e “confirmar estoque disponível” forem concluídas. Enquanto qualquer uma delas não tenha sido concluída, a atividade “concluir a venda” não será executada.

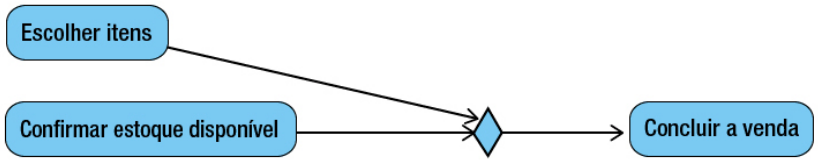


Diagrama de atividades representando uma junção.

A junção representa a necessidade de união de dois ou mais fluxos a fim de sensibilizar a entrada de outra atividade. É possível que uma atividade tenha mais de um fluxo de saída, portanto, a junção une fluxos e não atividades.

13

3.5. Atividades e ações

Dependendo da complexidade de uma atividade, ela pode ser detalhada em outro diagrama que represente atividades menores, mais facilmente compreensíveis. Entretanto, devemos ter cuidado ao detalhar demais, pois podemos ter um desenho altamente complexo e desnecessário. Você deve sempre utilizar o bom senso em saber até onde deve detalhar a atividade.

Todo item de alto nível é chamado de atividade pela UML, já o último nível de detalhamento, aquele que não pode ser mais detalhado em subpartes menores, é chamado de ação pela UML. Desta forma, dizemos que um processo é um conjunto de atividades, e uma atividade pode ser decomposta em ações.

Um exemplo prático o ajudará a entender melhor a ideia. No diagrama anterior, a atividade “confirmar estoque disponível” pode ser detalhada da seguinte maneira:

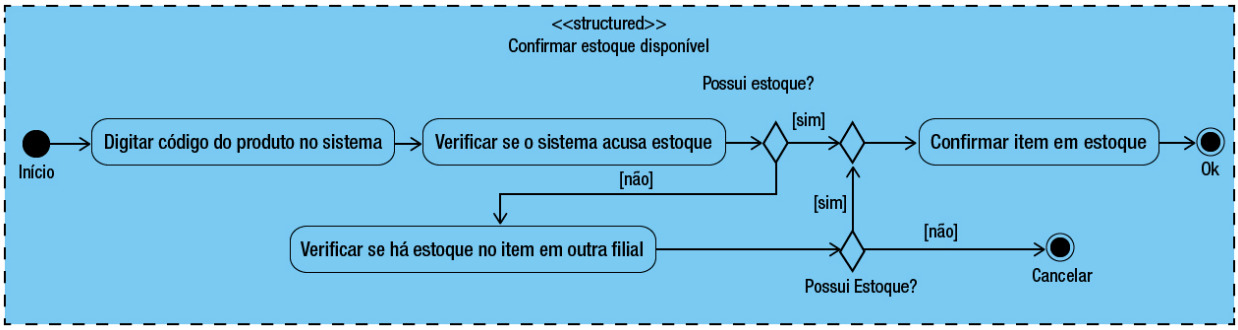


Diagrama de atividades representando uma divisão de uma atividade complexa em ações.

Quanto maior o detalhamento, maior é o controle sobre aquilo que será feito, entretanto, sua documentação fica mais complexa e mais inflexível quando as regras e alternativas de soluções adotadas.

14

3.6. Raias e Piscinas

a) Raias

Raias representam a separação de papéis.

Esse termo vem das raias das piscinas de natação, pois uma pessoa numa raia não invade outra raia. Aqui a ideia é a mesma: aquilo que acontece em uma raia restringe-se àquele escopo /sistema /dimensão /papel do usuário.

Dessa forma, as raias separam as atividades respectivas de cada papel de usuário em uma operação. Veja por exemplo um sistema de venda de um produto.

Poderíamos pensar, de forma bem simples, em dois papéis de pessoas que participam desse processo: o **cliente** e o **vendedor**. As atividades de cada papel ficariam em uma raia isolada. Resumidamente, teríamos um diagrama assim:

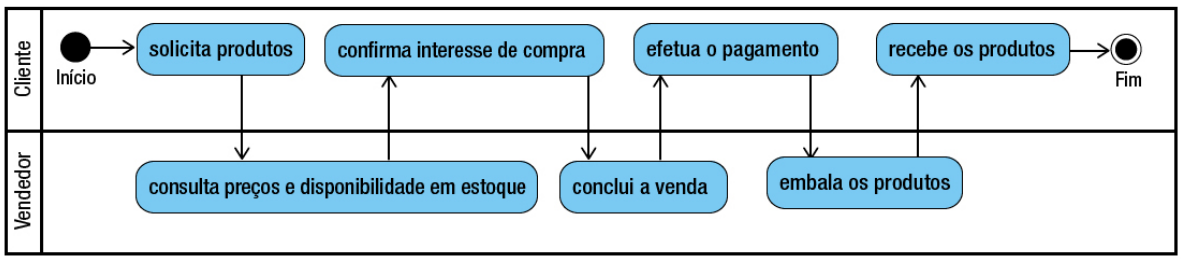


Diagrama de atividades resumido de um processo de venda.

Você seria capaz de evoluir esse diagrama incluindo os papeis do caixa e do estoquista? Refaça o diagrama, use lápis e papel se não tiver um software de UML disponível.

15

b) Piscinas

Para representar ambientes distintos (como sistemas distintos, ou operações no sistema e operações manuais), normalmente criamos “piscinas” separadas, ou seja, raias que não se tocam.

Enquanto raias separam papéis, piscinas separam ambientes. Graficamente, a diferença é que raias são desenhadas tocando-se lateralmente (retângulos lado-a-lado), enquanto piscinas são isoladas por um pequeno espaço em branco (retângulos que não se tocam).

Um diagrama de atividades pode conter quantas piscinas e raias forem necessárias. Inclusive um mesmo diagrama pode conter representações de raias e piscinas ao mesmo tempo.

Por exemplo, dentro de um sistema qualquer poderíamos usar raias para representar:

a. o que acontece dentro do universo do banco de dados;

b. outra raia para representar o que acontece no servidor da aplicação; e

c. uma última para o que acontece no computador do usuário (código em Java Script, por exemplo).

A seguir vamos ver um exemplo utilizando piscinas.

16

Exemplo do uso de piscinas

Suponha que você tenha uma atividade no seu sistema que se chama “efetuar pagamento do boleto bancário no site *homebanking* do banco” e que você queira detalhar todas as ações pertinentes a esta atividade. Em primeiro lugar, vamos determinar quantas piscinas existirão:

a. uma para a aplicação da empresa,

b. outra para o sistema bancário e

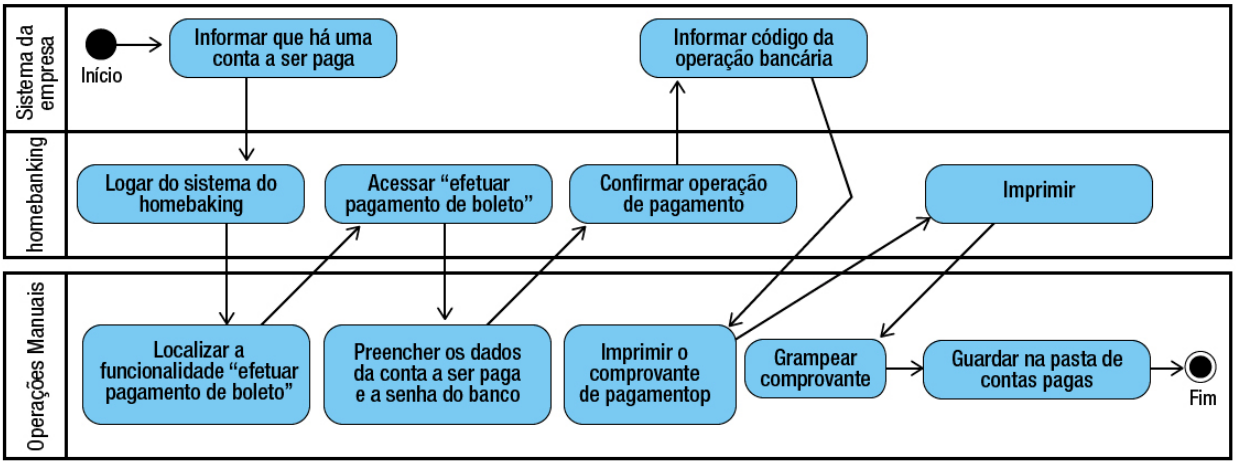
c. outra para as atividades manuais do operador do sistema.

Como essas informações tratam de ambientes distintos (dois sistemas distintos e um conjunto de operações manuais), o correto é utilizar piscinas (e não raias).

Feito isso, precisamos detalhar todas as operações necessárias (as ações):

- Acessar o site *homebanking* do banco;
- Localizar no menu de opções a funcionalidade “efetuar pagamento de boleto”;
- Clicar na funcionalidade;
- Preencher os dados de pagamento: data de vencimento, código de barras, valor, desconto (se houver);
- Digitar a senha de pagamento;
- Confirmar a operação;
- Digitar o código da operação bancária no sistema da empresa.
- Imprimir o comprovante de pagamento;
- Grampear o comprovante ao boleto físico;
- Guardar o comprovante e o boleto na pasta de contas pagas da empresa.

Veja como seria a representação do cenário acima com três piscinas (uma, para o sistema da empresa, outra, para o **homebanking** e outro, para as operações manuais).



Exemplo de diagrama de atividades representando várias “piscinas”.

Nesse caso, observe que a documentação extrapola o sistema de informação e detalha o próprio trabalho manual que é feito na empresa para efetuar o pagamento da conta. Observe que a documentação UML pode ter sua fronteira definida não só no sistema que estamos programando, mas percorrendo todas as ações que as pessoas têm de fazer para tudo funcionar a contento.

17

3.7. Concorrência

Concorrência (ou paralelismo) refere-se à possibilidade de realizar várias ações ao mesmo tempo.

Essa questão é especialmente útil quando um determinado processo precisa realizar várias ações logicamente independentes entre si, em outras palavras, elas podem ser executadas tão logo exista disponibilidade para o sistema executá-las, elas não precisam ocorrer em sequência.

Suponha, por exemplo, que um sistema que você crie irá efetuar pesquisas por passagens aéreas em diversas companhias aéreas. Você não precisa pesquisar a companhia A primeiro para depois pesquisar a companhia B, depois a C, depois a D e assim sucessivamente. Você pode disparar quatro consultas ao mesmo tempo aos sistemas das quatro companhias, e, à medida que os resultados vão sendo processados e comunicados ao seu sistema, você poderá apresentá-los ao seu usuário.

A representação de uma concorrência é a de um "garfo". Para posterior junção, utiliza-se a função de sincronização. Veja para o cenário descrito no exemplo acima, poderíamos ter uma representação simplificada assim:

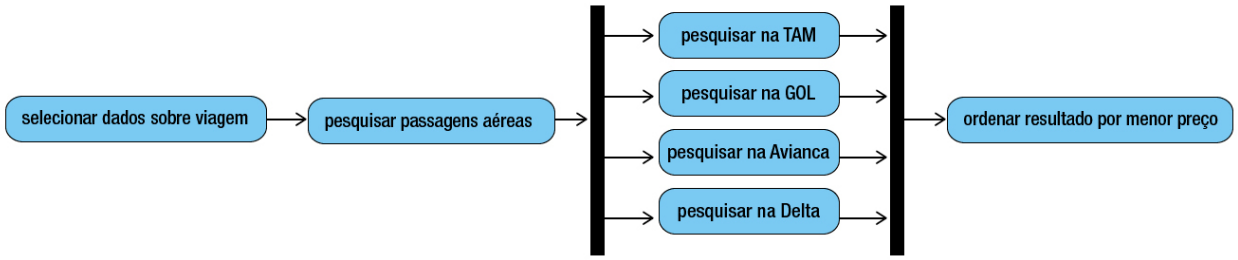


Diagrama de atividades representando uma concorrência.

Identificar uma oportunidade de concorrência não significa a definição de um requisito de concorrência, mas tão somente uma melhor forma de se fazer um determinado trabalho. A definição do requisito deve ser feita por meio de documento específico ou caso de uso, ambos provindos de entrevistas com os usuários da aplicação (ou demandantes da aplicação).

18

4 - Exercício prático

A melhor forma de aprender a diagramar UML é executar exercícios práticos. Para esses exercícios que faremos agora, você não precisa nem de um *software* de UML, basta lápis e papel.

4.1. Máquina automática de venda de ingresso de cinema

No módulo passado já apresentamos um diagrama de venda de ingressos de cinema. Você se lembra daquele diagrama que utilizamos para exemplificar os cenários do caso de uso? Para relembrá-lo, representamos novamente o diagrama simplificado abaixo:

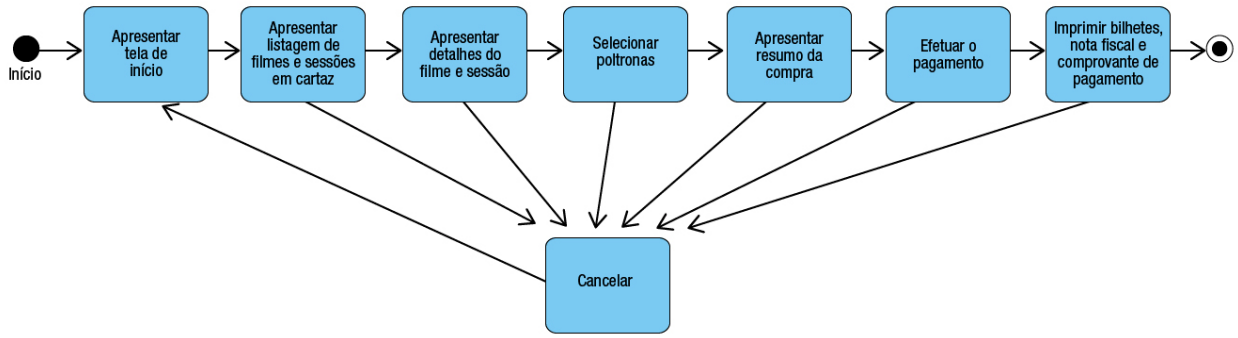


Diagrama simplificado de atividades representando o processo de venda de ingresso de cinema.

19

Entretanto, pensando em uma representação completa do diagrama, pode ser necessário que especifiquemos todos os símbolos UML. Dessa forma, veja como ficaria uma revisão das primeiras três atividades apenas:

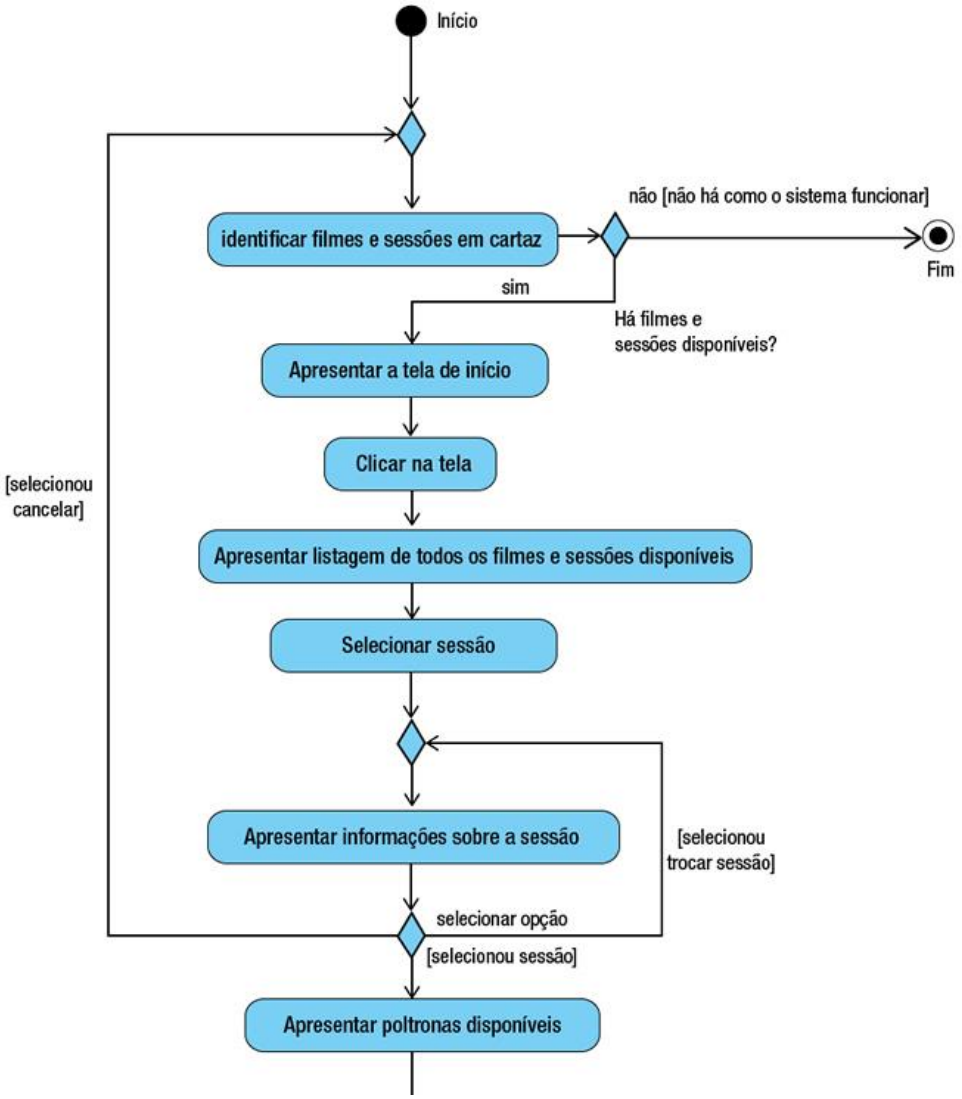


Diagrama revisado de atividades representando apenas as três primeiras atividades de venda de ingresso de cinema
(observação: devido ao fato de o diagrama completo ficar muito grande, optamos por representar apenas o detalhamento das três primeiras atividades).

A representação completa só precisa ser assim detalhada em dois casos:

a. quando a ferramenta UML é responsável por gerar código fonte do sequenciamento das atividades;

b. quando as atividades são tão críticas e/ou complexas que merecem esse detalhamento.

Caso o objetivo da documentação seja restrito à comunicação e informação do funcionamento do projeto para o desenvolvedor de sistemas (e futuramente ao testador do sistema), provavelmente, um diagrama resumido (como o primeiro) será suficiente.

RESUMO

Neste módulo, aprendemos que:

a. O diagrama de atividades da UML 1.4 é muito semelhante a um fluxograma clássico. Ele pode ser aplicado a qualquer processo, grande ou pequeno. Três aplicações comuns de diagramas de atividades são para explicar o fluxo de trabalho (uma série de casos de uso), para explicar um único caso de uso, e para explicar um método (a sequência de ações realizadas pelo método).

b. O diagrama de atividades representa uma tarefa como um retângulo com os cantos arredondados, contendo uma descrição de texto de forma livre da tarefa. A transição de uma atividade para a próxima é mostrada como uma seta. A notação prevê pontos de início e fim, usando um pequeno círculo pintado e um círculo pintado com um anel externo, respectivamente.

c. As decisões são representadas como um pequeno losango. Cada transição de saída deve ser marcada com uma condição de guarda (com texto entre colchetes), e as condições devem ser mutuamente exclusivas. O losango também pode ser usado para representar um ponto de fusão, unindo dois ou mais percursos alternativos na sequência.

d. As condições de guarda também podem ser usadas em transições deixando uma atividade, em que o resultado da atividade fornece todas as informações necessárias para cumprir uma das condições.

e. A concorrência refere-se às múltiplas atividades ocorrendo simultaneamente. Uma barra do tipo “garfo” representa uma transição de entrada para iniciar várias transições. A barra de sincronização mostra várias transições chegando ao fim e uma nova transição combinando-as.

f. Muitas vezes, uma explicação textual pode ser um pouco ambígua para a definição de um processo complexo. O diagrama de atividades normalmente é utilizado como uma ferramenta para facilitar a comunicação das explicações textuais.

g. Para traduzir a descrição do usuário em um diagrama de atividades, isole cada tarefa como uma atividade. Indique a sequência das tarefas desenhando uma seta de transição de uma atividade para a próxima atividade na sequência.

h. Para modelar as decisões no processo, você tem duas opções. Uma decisão que resulta da conclusão de uma atividade é desenhada com condições de guarda (cada transição para fora da atividade é marcada com uma única – e mutuamente exclusiva – expressão condicional entre colchetes). Para representar uma decisão que não é o resultado de uma atividade específica, utilize o ícone de losango. Cada transição para fora do ponto de decisão do losango também é marcada com uma expressão condicional única entre colchetes (exemplo: [sim] e [não]).

i. Quando o fluxo lógico do processo precisa retornar a um ponto anterior no fluxo, use o ícone de losango para representar um ponto de fusão. Pode haver duas ou mais setas que entram no ponto de fusão, mas apenas uma pode sair.

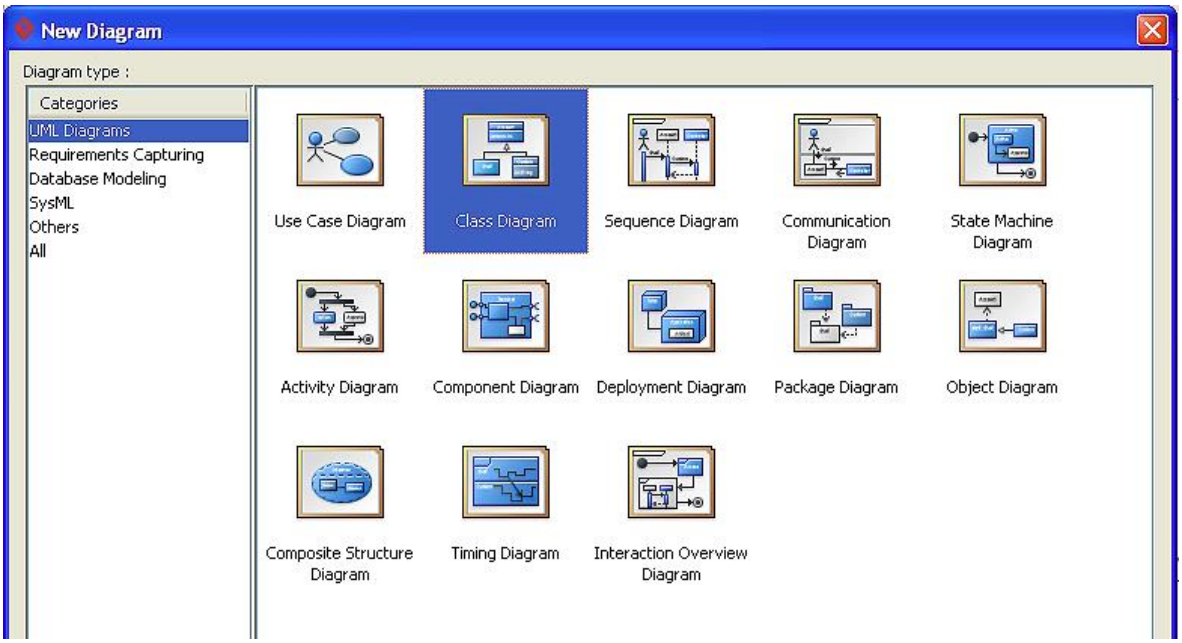
UNIDADE 1 – CONCEITOS INICIAIS SOBRE UML
MÓDULO 3 – DIAGRAMA DE CLASSES – CONCEITOS

01

1 - RECONHECENDO CLASSES E OBJETOS

Nesta etapa você aprenderá os conceitos mais comuns e utilizados no dia a dia no desenvolvimento de aplicações orientadas a objetos. Tanto atuando como um modelador de sistemas quanto como um programador, nós iremos familiarizá-los com os conceitos de objetos, classes, associação, generalização, agregação e pacotes. Você verá detalhes importantes sobre a notação de modelagem de objetos por meio da UML e verá dicas importantes e boas práticas. Você também poderá identificar os problemas mais comuns que deverá evitar.

O início de um projeto de *software* é também o momento mais importante do projeto. Saber utilizar bem o tempo gasto nessa fase é primordial para obter uma boa concepção do projeto. Nesse momento você deverá ser capaz de identificar os atributos das classes, as operações que elas poderão realizar e quais partes devem permanecer privadas ou expostas (compartilhadas).



Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de classes.

02

O diagrama de classes é sem dúvida o diagrama mais utilizado na UML. A ideia aqui não é qualificar o diagrama de classes como “o diagrama mais importante da UML”, todos os diagramas são importantes, mas o diagrama de classes é aquele que, em modelagens mais simples, é o escolhido como o mais desejável pelas equipes. Portanto, dê uma atenção especial para este diagrama (ele contempla boa parte do nosso estudo), pois certamente será o que você mais irá modelar e analisar no seu dia a dia.

As classes formam a base do diagrama de classes. Para modelar esse tipo de diagrama, você precisa estar bem seguro sobre a diferença entre classes e objetos. Uma classe é uma definição de um recurso.

A classe descreve as características de uma entidade e como ela pode ser usada. Em contraste, um **objeto é uma entidade** identificável de forma específica, que está de acordo com as regras definidas pela classe.

Em termos de *software*, o código é escrito como um conjunto de classes e referências a comportamentos definidos pelas classes. A informação criada e manipulada pelos usuários pertence a objetos que estejam em conformidade com as descrições de classe. Os objetos são representados por linhas em um banco de dados, registros em arquivos, ou áreas de memória em um computador.

Um exemplo prático de um *software* que faça o gerenciamento de vendas de uma padaria:

- Nas classes você informará quais atributos um produto pode ter, como nome, valor, descrição, marca, modelo, quantidade em estoque. Sobre a classe venda, você dirá que ela possui produtos associados, que possui data, valor total, cliente etc.
- Nos objetos estará o conteúdo de uma venda real. Um objeto de venda pode incluir os produtos pão, leite e queijo, no valor total de R$ 12,00. O objeto leite conterá a informação de quantos litros de leite o cliente está comprando e o valor unitário de cada litro.

03

1.1. Classe

Uma classe representa as características, estruturas e comportamentos de uma espécie comum.

Quando pensamos em um ser humano como uma “pessoa”, identificamos que “pessoa” é uma classe que possui vários atributos como nome, idade, sexo, altura, peso, nacionalidade, estado civil, e outros. A classe “pessoa” também possui comportamentos comuns, toda pessoa pode andar, dormir, comer, sentar-se. Essas características e comportamentos comuns é o que tipifica uma pessoa. Em se tratando de sistemas de informação, é essencial que sejam identificados todos os “tipos de item” que o sistema irá tratar. Pense por exemplo em uma livraria, tipos de itens comuns para um sistema de gerenciamento de uma livraria seriam: livros, autores, editoras, clientes, fornecedores, vendas, compras, e outros.

1.2. Objeto

Um objeto é uma classe materializada, uma classe que existe e que tem identidade própria.

Quando pensamos em “pessoa”, não pensamos em alguém específico, mas quando falamos em “Pedro Álvares Cabral”, aí sim sabemos exatamente quem ele é. No conceito de um sistema para gerenciamento de uma livraria, seriam exemplos de objetos: o livro de culinária “cozinhe feliz”, o autor “João Cruz Neto”, a editora “Brasil e-books”, o cliente “João Feliciano”, o fornecedor “gráfica bandeirante”, e outros.

Objetos são criados a partir de classes predefinidas. Quando pensamos em um sistema de informação, precisamos identificar qual é a realidade daquela empresa que utilizará o sistema, e assim conseguir identificar as classes que o sistema deverá possuir. Tudo aquilo que existir como objeto no mundo real da empresa, poderá existir como um objeto no *software* que você irá construir.

04

Uma das formas mais comuns de você identificar os objetos e classes de um futuro sistema de informação é escrever todos os procedimentos que o sistema deve fazer. Feito isso, identifique todos os substantivos descritos e avalie se:

a. Ele representa uma coisa em particular (um objeto) ou um conjunto de coisas semelhantes (uma classe);

b. Ele é parte do problema a ser resolvido (é escopo do projeto);

c. Ele não é parte de um detalhe de implementação (uma característica de um objeto);

d. Ele não é um evento ou uma ocorrência;

e. Ele não é uma propriedade (uma característica) de uma coisa.

Após essa análise, é bem provável que você consiga identificar as classes e objetos do sistema. Veja, no pequeno exemplo da realidade de uma livraria, como poderíamos hipoteticamente identificar alguns objetos e classes:

*“O **cliente** vem à loja e procurar por um ou mais **livros** de seu interesse. Ele pesquisa o preço dos livros por meio da **etiqueta** afixada em cada livro. Se houver interesse, ele pode comprar um ou mais livros, pagando em **dinheiro**, **cartão** ou **cheque**.”*

Neste pequeno exemplo conseguimos identificar os seguintes objetos e classes:

• Cliente

Classe que define (generaliza) as características das pessoas que vão à livraria comprar livros.

• Livros

Classe que define (generaliza) as características dos produtos que a livraria vende.

• Etiqueta

Classe que possui algumas das informações dos livros (como código de barras e preço). Possui uma associação com a classe livro.

• Dinheiro, cartão e cheque

Podem ser entendidos como classes isoladas ou generalizados como uma classe de nome “forma de pagamento”.

05

Outras informações que também conseguimos identificar:

- **Procurar** – uma atividade que futuramente será um método de alguma classe.
- **Preço** – é um atributo de um livro.
- **Venda** – classe subentendida que existe na operação de “comprar”, possui associação com as classes cliente e livros.
- **Cliente e vendedor** – atores dos casos de uso pertinentes ao cenário.
- **Pesquisar** livros, comprar livros, efetuar pagamento, pagar com dinheiro, pagar com cartão, pagar com cheque – casos de uso do cenário.

Dessa forma, as informações que encontramos ao analisar um cenário são caracterizadas por:

Tipo de informação	Exemplo	Parece ser...
Um conjunto de coisas	Livro, Cliente	Uma Classe
Um nome próprio	Livro “Cozinhe Feliz”, Cliente “Pedro Afonso Neto”	Um objeto
Uma propriedade de alguma coisa	Preço, Quantidade em estoque	Um atributo
Um valor ou um dado	R$ 47,00; 44 unidades; “Cozinhe Feliz”; “Pedro Afonso Neto”	O valor de um atributo

Uma condição de uma coisa	Em falta, Em estoque	Um estado
Uma ocorrência, evento ou tempo	Cliente chega à loja; o telefone toca; o cliente pesquisa o preço	Uma operação
Parte da implementação	Banco de dados, tabela	Deixe para o posterior momento do designer da aplicação
Um papel	Cliente	Deixe para o diagrama de casos de uso
Uma atividade	Comprar	Será utilizando como um método em uma classe e também no diagrama de atividades.
Um caso de uso	Pagar com cartão	Será utilizado como um método em uma classe, um item em um diagrama de atividades e um caso de uso.

06

Veja que ao analisar um **cenário**, identificamos ao mesmo tempo informações de **classes**, **casos de uso** e **atividades**. Nossos modelos serão criados e evoluídos na medida em que as informações vão ficando cada vez mais detalhadas e precisas.

Crie uma lista dos nomes que fazem sentido para o contexto. Seja generoso, se algo parece ser duvidoso, adicione à lista da mesma forma (depois você eliminará redundâncias e itens que não serão escopo, em outros casos, essas informações poderão vir a tornarem-se atributos, eventos e outros). Após identificar os objetos, procure inferir quais classes criam e definem esses objetos.

A chave para identificar essas informações é analisar completamente a descrição do negócio a ser resolvido pelo sistema (ou que já existe na organização). Se o processo de trabalho ainda não existe, crie um (ou peça para algum especialista no assunto criar um).

Na maioria das vezes, a descrição do comportamento do sistema é a melhor forma de organizar o conjunto de entidades (chamada de atores), validando seus objetivos individuais (chamados casos de uso) que serão invocados pelo sistema.

Trataremos de atores e casos de uso mais à frente, ainda nesta disciplina.

07

2 - NOMEANDO CLASSES E OBJETOS

Após identificar as classes e objetos que você quer no seu sistema, é hora de definir como você irá chamá-los. Todo projeto pode conter suas próprias regras de como nomear classes, mas os nomes que você irá definir também devem seguir alguns padrões e boas práticas do mercado. Veja algumas regras básicas que a maioria dos projetos de *software* utilizam para nomear classes:

- Use sempre substantivos (verbos no infinitivo só aparecem em métodos);
- Use sempre o singular (use o plural apenas em objetos, nunca em classes);
- Evite pronomes, adjetivos, advérbios, artigos e preposições;
- Use as iniciais em maiúsculo;
- Não use espaços, caracteres acentuados ou cedilha;

Seguindo essas regras, podemos imaginar alguns exemplos hipotéticos de classes para um sistema de biblioteca: PedidoCompra, Venda, Livro, Cliente, Vendedor.

Levando em consideração que o objetivo de um bom nome é prover uma informação rápida e precisa, evite qualquer coisa que possa confundir ou deixar a interpretação lenta. Uma boa política é evitar qualquer tipo de abreviação e palavras com duplo sentido (homônimos).

08

2.1 Encapsulamento (ou encapsulação)

O encapsulamento é uma forma de organizar os vários tipos de informação e comportamento descrito anteriormente para que os objetos possam ser utilizados da forma mais eficiente e eficaz possível. A encapsulação afirma que ao projetar um objeto que deve separar o que sabemos sobre o objeto de acordo com dois conceitos:

- As informações mínimas necessárias para usar o objeto.
- As informações necessárias para fazer funcionar o objeto corretamente.

Estes dois conceitos nos orientam a olhar para o objeto a partir de duas perspectivas: uma visão externa: "O que eu posso fazer com este objeto?" e uma visão interna: "Como é que esta coisa funciona?"

Encapsular está ligado à ideia de proteger e esconder dentro da classe aquilo que quem a chama não precisa saber (por exemplo, como ela funciona) e divulgar, mostrar, deixar expostas as informações

(atributos) e ações (métodos) necessárias para poder utilizar a classe.

09

Encapsular possui os seguintes **objetivos**:

- Juntar as partes afins e semelhantes de forma a criar um contexto completo e funcional ao objeto;
- Proteger a funcionalidade interna.

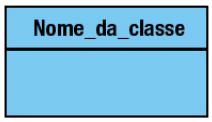
Veja por exemplo essa situação hipotética: suponha que você esteja construindo um carro e você compra um motor pronto de um fabricante. Você não precisa saber como funciona os cilindros, como o cabeçote está fixado, como os pistões rodarão o virabrequim, pois todos esses itens estão "encapsulados" dentro do motor. O que você precisa saber para usar o motor é: por onde colocar o combustível, como controlar a potência do motor, por onde será fixado ao carro, por onde será conectado ao sistema de câmbio e embreagem. Esses sãos os métodos e propriedades do motor que ficam expostos a você.

Na prática, você poderia ter hipoteticamente o seguinte cenário: suponha que você esteja criando uma aplicação que toque músicas MP3. Você pode criar uma classe que exponha um local para ser informado o local do arquivo MP3 e funções para tocar, parar, adiantar, voltar e pausar. Quem chama essa classe não precisa saber qual é o código fonte que você utilizou para fazer a música tocar, essa parte está protegida (encapsulada); quem chama essa classe, só precisa saber como enviar o arquivo MP3 (atributo) e como realizar as operações citadas (métodos), essa é a parte pública da classe.

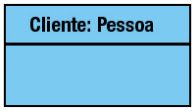
10

2.2 Diagramando uma classe

Falamos anteriormente que a UML é uma técnica de diagramação. Neste sentido, cada componente deve ser representado como um desenho geométrico. Em se tratando de classes e objetos, o desenho geométrico que representa esses componentes é o retângulo, onde, dentro dele, colocamos o nome da classe ou do objeto. O *layout* padrão é:



Para diferenciar objetos de classes, a regra estabelece que objetos devam ser escritos com o nome do objeto, seguindo por dois pontos, e depois o nome da classe. Nesta ideia, para representar o objeto cliente da classe Pessoa, teríamos algo como:



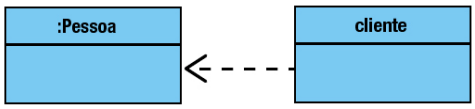
Se o texto do retângulo fosse apenas “cliente”, não saberíamos dizer, a princípio, a qual classe ele pertence.

11

2.3 Utilizando setas para indicar a classe do objeto

A UML permite mais de uma maneira de representar a mesma informação. Isso significa que você deve utilizar aquela mais adequada para sua necessidade. A redundância muitas vezes aprimora a qualidade da comunicação, entretanto, traz consigo um diagrama mais complexo.

A UML possui uma outra forma de representar a relação de classe-objeto, ela pode ser vista pela representação de dois retângulos interligados por uma seta pontilhada que liga o objeto à classe. Para o exemplo anterior, a relação cliente-Pessoa poderia ser representada assim:



Tenha em mente que ao utilizar essa notação, o seu diagrama conterá o dobro de retângulos (e as setas) do que a forma anterior.

12

3 - Definindo associações e links (ligações)

Identificar os objetos e classes individuais é apenas o começo de modelagem. Para que os objetos e classes de um *software* realizem o respectivo trabalho a ser feito, eles precisam trabalhar juntos de forma a representar o contexto do trabalho a ser feito.

Por exemplo, quando um cliente solicita ingressos de um filme (cinema), o sistema cria uma venda e adiciona ingressos para essa venda. Ainda, associa um filme, uma sessão, reserva poltronas e pode definir outras características do ingresso (como por exemplo valor promocional para estudante ou idosos). Os objetos de *software* que representam os clientes, a venda, a sessão do filme, as poltronas, e os ingressos precisam replicar exatamente as relações entre os objetos do mundo real.

Atenção quanto à nomenclatura da UML:

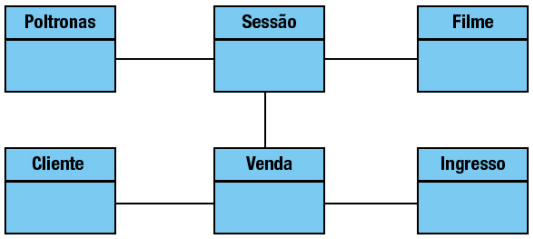
- um **link (ligação)** é uma relação entre dois objetos;
- uma **associação** é um relacionamento entre duas classes.

Uma associação é a definição de um tipo de link (ligação) da mesma forma que uma classe é a definição de um tipo de objeto. Assim como um objeto é uma instância de uma classe, um link é um exemplo de uma associação.

Associações (e links) podem assumir três formas diferentes: **associação**, **agregação** e **composição**. A forma mais simples, a associação, é uma relação ponto a ponto. Um objeto simplesmente sabe sobre outro objeto da mesma forma que uma pessoa pode saber sobre outra pessoa. Esse conhecimento é normalmente armazenado no objeto como uma referência, como uma pessoa que mantém um número de telefone ou endereço de outra pessoa que ele quer entrar em contato.

13

Uma **associação** é geralmente modelada como uma linha de chamada entre duas classes, conforme mostrado abaixo.



Exemplo de associação entre objetos.

No mundo real, podemos identificar as mesmas relações representadas no diagrama:

- Um cliente solicita uma venda de ingressos;
- A venda inclui, obviamente, um ou vários ingressos;
- A venda de ingressos refere-se a uma sessão de cinema;
- A sessão de cinema possui poltronas (disponíveis e ocupadas);
- A sessão de cinema representa o horário de um filme específico.

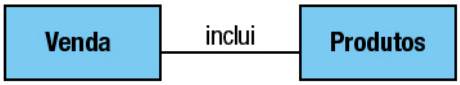
Essas associações representam que, no contexto apresentado, há uma relação entre elas para que a funcionalidade do sistema seja válida.

14

A **associação** é usada para juntar dois objetos distintos em um contexto: quando você junta pessoa à venda é apenas naquele contexto que existe essa associação.

A associação corresponde a relações temporárias e rápidas para um objetivo em comum de objetos distintos.

Uma associação pode ser nomeada para deixar o relacionamento ainda mais claro. É importante levar em consideração que devemos sempre usar a **voz ativa** (evitando a voz passiva). Dessa forma, como exemplo, as classes “Venda” e “Produtos” podem ser relacionadas pelo texto “inclui” (uma venda inclui produtos). O nome “são incluídos” (produtos são incluídos em uma venda) não é uma boa escolha.



Exemplo de nomenclatura de uma relação entre classes

Assim como no mundo real, as associações podem requerer regras específicas para controlar o relacionamento entre elas. Quem participa do relacionamento? Quantos objetos podem participar? Como os participantes controlam o comportamento do objeto? Em qual direção a associação deve ser ligada? Quando o relacionamento começa e termina? As regras são definidas e representadas por meio de multiplicidade, restrições, indicadores de direção, classes de associação papéis e comentários. Cada um deles será apresentado no devido tempo com a evolução dessa matéria.

15

3.1 Tipos de Associação

As associações podem ser refinadas em um modelo mais restritivo de relacionamento, chamado “agregação”. Uma agregação pode ainda ser refinada em um relacionamento ainda mais restritivo, denominado “composição”.

Associação	Agregação	Composição
• Os objetos sabem da existência do relacionamento que há entre si, dessa forma eles podem trabalhar juntos.	• A integridade da configuração é protegida. • Funcionam como um único objeto. • Um objeto controla todos os outros. • Toda agregação é um tipo de associação. • Toda agregação possui as características de uma associação e outras novas específicas da agregação.	• Cada parte só pode ser membro de um objeto agregado • Toda composição é um tipo de agregação. • Toda composição possui as características de uma agregação e outras novas específicas da composição.

16

3.2 Definindo agregações

A agregação é um tipo especial de associação usada para indicar que os objetos participantes não são apenas objetos independentes que sabem sobre o outro. Em vez disso, eles são montados ou

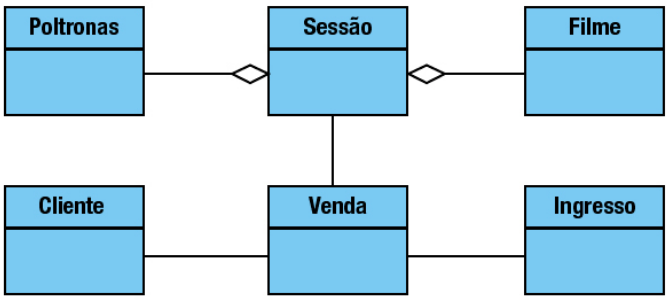
configurados em conjunto para criar um novo objeto, mais complexo.

Por exemplo, um automóvel é composto por várias partes: motor, carroceria, rodas etc. A associação que essas peças têm com o automóvel é do tipo agregação: “um automóvel agrega várias partes juntas, como uma carroceria, um motor, quatro rodas etc.”.

Para modelar a agregação em um diagrama de classes:

- Desenhe uma associação (linha) entre a classe que representa o membro ou parte do conjunto e da classe que representa a agregação, ou seja, o conjunto (o todo).
- Desenhe um losango na extremidade da associação que está ligado à classe agregadora. Exemplo: no relacionamento entre automóvel e motor, a classe automóvel possuiria um losango para representar que um automóvel inclui um motor.

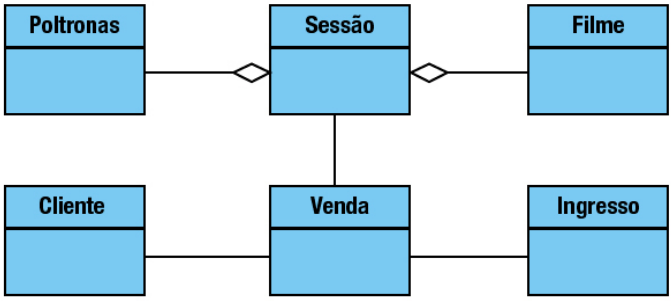
Voltando ao nosso exemplo anterior, da venda de ingressos de cinema, podemos perceber que há uma relação intensa entre a sessão de cinema e o filme. Não faz sentido o objeto sessão sem ele estar associado a um filme, da mesma forma, um filme precisa possuir sessões (disponíveis) para permitir uma venda de ingressos. A sessão inclui um filme como parte do seu conjunto. Da mesma forma, também há uma relação entre as poltronas e a sessão de cinema. Não faz sentido o objeto poltrona sem que esteja associado a uma sessão, assim, uma sessão precisa possuir poltronas (disponíveis) para permitir uma venda. Observe que a sessão é o centro da questão, ela possui características próprias (como os horários) e incorpora outras informações (outros objetos), no caso, poltronas e filme. Sob essa ótica, poderíamos refinar a associação existente entre sessão, poltronas e filme, criando agregações e evoluindo o diagrama para a representação abaixo:



Evolução do diagrama anterior utilizando agregações

17

Observe novamente o diagrama:



Evolução do diagrama utilizando agregações

Note que, com a agregação, o contexto torna-se um pouco mais significativo. Agora, o objeto sessão é o "chefe" que tem o poder de comandar poltronas e o filme. Dessa forma, o controle, além de simplificado (pois, ao invés de comandar esses três objetos para fazer a venda de ingressos, a classe venda só precisará comandar um), o contexto também fica protegido (a classe venda não precisa – nem pode – acionar poltronas e filmes individualmente, tudo poderá – e deverá – ser feito por meio da classe sessão).

Na prática, teríamos esses resultados:

- Se você quer saber quais poltronas estão disponíveis, pergunte à classe sessão e ela responderá.
- Se você quer reservar poltronas, solicite à classe sessão e ela responderá.
- Se você quer ver uma imagem ou informações sobre o filme, pergunte à classe sessão e ela responderá.
- Quer comprar ingresso e marcar a poltrona como reservada? Solicite à classe sessão e ela o fará.

18

3.3 Definindo a composição

A composição é utilizada para agregações em que o período de vida da parte depende do tempo de vida do objeto agregador. O objeto agregador tem controle sobre a criação e destruição dos objetos agregados. Em suma, o objeto membro não pode existir para além da montagem.

Desenhe esta forma mais forte de agregação simplesmente fazendo o losango da agregação se tornar sólido (preto).

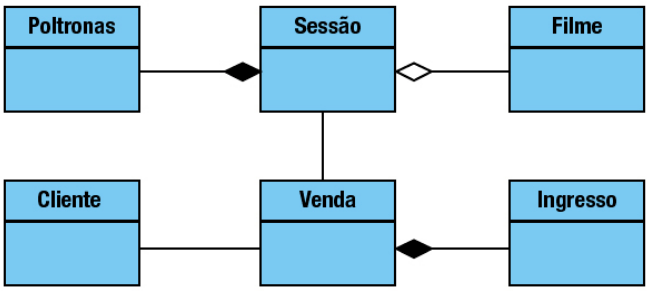
No nosso exemplo anterior, percebemos que o objeto ingresso não faz sentido se ele não estiver associado a uma venda. Não existe um ticket "em branco". Não existe um ingresso que não tenha uma sessão, poltrona, filme e venda associado. Dessa forma, o relacionamento do ingresso com a venda é mais forte que uma simples agregação, é uma associação.

E quanto aos relacionamentos anteriores? Será que eles poderiam ser uma composição também?

Vamos ver os dois casos anteriores:

- Um filme pode existir por si só, sem uma sessão associada? Sim, o filme pode ser um pré-lançamento, uma previsão, uma propaganda, então ele pode existir sim sem estar associado a uma sessão. Então, a relação entre sessão e filme não chega a ser uma composição.
- E poltronas, podem existir poltronas disponíveis e reservadas sem estarem associadas a uma sessão? De forma alguma, o conceito e uso da disponibilidade de poltronas só faz sentido quando associadas a uma sessão de filme. Se a sessão de filme fosse cancelada, todas as reservas e disponibilidades de poltronas deixariam de existir. Dessa forma, o relacionamento é tão intenso que deve ser evoluído para uma composição.

Percebemos então que nosso diagrama deve ser atualizado para:



Evolução do diagrama anterior utilizando composições

19

4 - Definindo herança e generalização

Criar uma generalização é o processo de organizar as características dos diferentes tipos de objetos que compartilham a mesma finalidade. Uma generalização é uma descrição das características comuns de um conjunto de objetos.

Usamos o processo de generalização para organizar grandes quantidades de informação. Caminhe por um supermercado e você encontra alimentos localizados em áreas de armazenamento de acordo com suas propriedades - produtos enlatados estão localizados em uma área, frutas e legumes em outro, grãos em outros. Todos esses itens são alimentos, mas são tipos diferentes de alimentos. Frases como "uma espécie de" ou "tipo de" são muitas vezes utilizados para descrever um relacionamento de generalização entre classes (por exemplo, uma maçã é um tipo de fruta que por sua vez é um tipo de alimento e assim por diante).

Você também pode ouvir esse tipo de relacionamento referido como **herança**. Muitas vezes os termos de generalização e herança são usados como sinônimos. Isso porque se uma maçã, por exemplo, é um tipo de fruta, em seguida, ela herda todas as propriedades de fruta. Da mesma forma, uma maçã é uma especialização da fruta porque ela herda todas as propriedades generalizadas de frutas e adiciona

algumas propriedades únicas que fazem maçãs especiais ou exclusivas dentro do maior grupo de frutas. No sentido inverso, eu poderia dizer que o conceito de "fruta" é uma generalização dos fatos que são verdadeiros para as melancias, maçãs, pêssegos, e todos os tipos de objetos do grupo.

20

A diferença entre os termos generalização e herança é apenas na leitura da relação. Veja o exemplo: uma banana herda as propriedades das frutas, uma fruta é a representação genérica de uma banana.

A generalização não é uma associação. Associações definem as regras de como os objetos podem se relacionar entre si. A generalização diz que certas classes contém um subconjunto das propriedades de uma classe superior. Nesse sentido, uma generalização não possui atributos e regras comuns das associações, como multiplicidade, papel, restrições.

Um exemplo bem simples e ilustrativo de herança e generalização seria o seguinte:

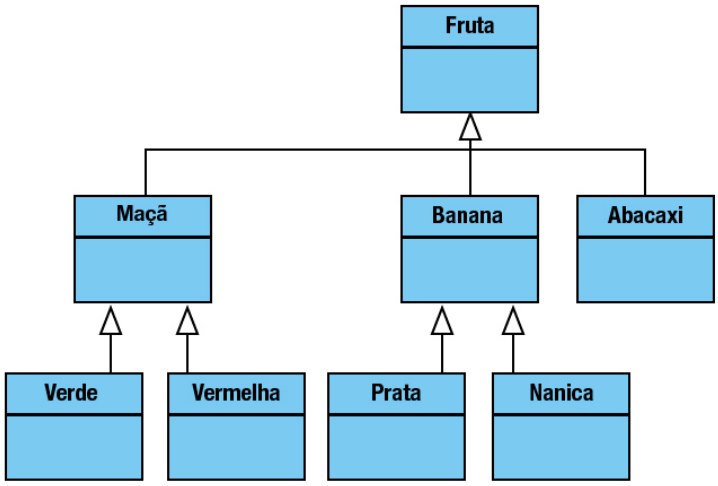


Diagrama de classes mostrando herança e generalização

21

5 - Representando Atributos e Operações

Você já sabe que uma classe possui um nome que a define. Também sabe que uma classe contém propriedades (**atributos**) e métodos (**operações**).

As propriedades são as informações da classe que são expostas a quem a chama, por sua vez, os métodos são as ações que ela possui para permitir que operações aconteçam dentro dela.

Temos então três informações básicas e precisamos poder representá-las em um diagrama de classes: o nome, os atributos e os métodos.

Lembrando que uma classe é representada por um retângulo, vamos dividir esse retângulo com duas linhas horizontais. Em cada compartimento criado, iremos escrever, respectivamente, o nome da classe, os atributos e as operações:

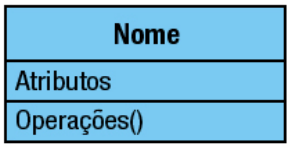
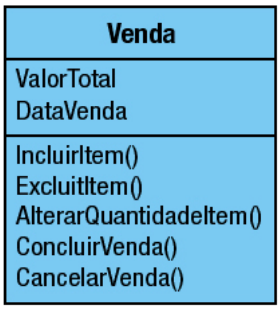


Diagrama representando a sintaxe dos itens que compõe uma classe

O espaço reservado para atributos e operações pode conter quantos itens forem necessários. Veja o exemplo abaixo:



Exemplo de classe com atributos e métodos.

Na próxima oportunidade iremos ver em detalhes as características dos atributos e dos métodos.

RESUMO

Neste módulo, aprendemos que:

a. Classes e objetos: Uma classe é uma definição de um tipo de entidade, da mesma forma que um "livro" define como deve ser "um livro de informática". Um objeto é uma abstração para uma única entidade, como "o meu livro de UML". A classe define regras. Um objeto define fatos. Um objeto é instanciado (criado) de acordo com as regras definidas pela sua classe.

b. Abstração: Uma abstração é uma representação de algo real, como o seu rosto em uma foto: aquilo que vemos na foto é uma representação de você, e não você (a pessoa real). Mesmo que haja uma quantidade quase infinita de informações sobre qualquer entidade real, uma abstração de software útil contém apenas a informação suficiente para apoiar a finalidade da aplicação de software.

c. Encapsulação: Descreve um modo para organizar a informação de uma abstração de modo que ele pode ser usado eficientemente em uma aplicação de software. A encapsulação as

informações que precisamos saber da lógica de funcionamento do objeto que não precisamos conhecer. Para usar um objeto, ele precisa expor a sua finalidade e interfaces. O objeto precisa conter os dados e comportamentos que satisfaçam os serviços oferecidos pelas interfaces.

d. Associações e links: Uma associação define um tipo de relacionamento, da mesma forma que uma classe define um tipo de entidade. Um link é uma abstração de uma relação específica que está em conformidade com as regras estabelecidas por uma associação, da mesma forma que um objeto é uma abstração de uma entidade que está em conformidade com as regras estabelecidas por uma classe.

e. Agregação: Agregação é um tipo de associação que afirma que um dos objetos participantes deve desempenhar o papel de controlador do outro objeto. Todas as interações com o par são dirigidas para o objeto de controle, o objeto agregador. O objeto agregador define como um objeto agregado deve se comportar. O objeto agregador propaga as instruções necessárias para o objeto agregado. As duas características que definem a agregação são: a) um objeto desempenha o papel de controlador e outro o papel subordinado; e b) dois (ou mais) objetos se comportam como um só.

f. Composição: Composição é um tipo de agregação, que afirma que um objeto membro pode existir se dentro do contexto da agregação. O tempo de vida do membro é dependente do tempo de vida do objeto que o compõe.

g. Generalização, especialização e herança: Generalizar é reunir todas as características comuns a partir de um conjunto de classes e defini-las em uma única classe. A generalização é a classe que contém os recursos compartilhados. Especializar é isolar as características de uma classe que só se aplica a um subconjunto do objeto dessa classe. A especialização é uma classe que contém os recursos exclusivos para o subconjunto de objetos. A herança é o princípio que permite uma especialização para ter acesso aos recursos em uma generalização. Um objeto instanciado por uma classe especializada é construído a partir dos recursos da classe especializada e de todas as suas características generalizadas.