



Desenvolvimento de Sistemas

UML: conceitos, diagramas de classes, diagramas de pacotes, diagramas de casos de uso, diagramas de sequência, diagramas de atividades

Conceitos

Você já está se ambientando à programação orientada a objetos e já tem uma noção de como funcionam algumas estruturas importantes acerca das UMLs (*unified modeling language*). O intuito então é expandir esse conhecimento e poder olhar a fundo tudo que essa estrutura oferece.

Quando se pensa em desenvolvimento de sistemas, é necessário considerar algumas questões importantes. Geralmente, a palavra “desenvolvimento” traz à mente pensamentos relacionados a códigos e linguagens, mas é importante sempre ter a completa noção de que **desenvolver sistemas** envolve processos **antes** e **depois** dos códigos.

É preciso, então, explorar alguns importantes conceitos sobre tudo que vem **antes** de colocar as mãos nas primeiras linhas de código e pensar em como facilitar não só o próprio trabalho como também o da equipe que desenvolverá o sistema.

A UML **não é uma linguagem de programação**, e sim uma forma de melhorar e exibir com mais detalhes a abstração e as classes de um sistema, bem como os objetos que poderão vir a ser gerados.

Imagine que a UML serve como um guia visual para os desenvolvedores dos sistemas, isto é, uma **planta baixa** de um sistema, na qual, por tabelas, setas e linhas, estão definidas as classes que precisarão ser desenvolvidas e a forma como elas conversarão entre si.

Desde a **lógica**, muito se tratou da **abstração** e de como traduzir os algoritmos, e, neste conhecimento, sobre **orientação a objetos**, trouxe-se a ideia de aproximar tais algoritmos do mundo real. Com a UML, agora isso poderá se tornar um pouco mais visual, facilitando até o processo de abstração, o qual será trazido para dentro de um sistema e compartilhado com a equipe.

É importante lembrar que a UML tem como base a orientação a objetos e, mediante símbolos e diagramas, forma a “planta baixa” do sistema.

Diagramas

Os diagramas são a alma das UMLs. É com eles que se começa a ter uma visualização do escopo e do trabalho para desenvolver o sistema. Ainda, nesse mesmo contexto, é preciso pensar que o trabalho não envolve só um tipo de diagrama, e sim vários.

Cada um desses diagramas exemplificará para o desenvolvedor e até mesmo para o cliente como cada função do sistema estará funcionando. A diferença (e importância) é que os diagramas serão sempre visuais e terão uma interpretação um pouco mais lúdica em vez de serem código puro.

Quando se trata dos diagramas, é necessário entender também que, além de eles trazerem o auxílio visual, também proporcionam um entendimento mais profundo do sistema como um todo, demonstrando ainda que **cada diagrama tem sua função dentro de um projeto**.

Os diagramas mostrarão funções entre **classes** e **relacionamentos** destas, **usuários** e comportamento destes quando utilizam cada uma das funções que o sistema proporciona, **separação entre funções específicas e demonstração destas**

com mais detalhes, entre outras diversas funções que fazem a UML ter um papel importante na vida de um desenvolvedor de sistemas.

Então, por qual diagrama começar?

O ideal é ter uma noção bem maior e uma boa documentação dos processos e do **escopo do projeto** primeiramente. Considerando que os diagramas ajudarão tanto os desenvolvedores quanto os clientes a visualizarem o fluxo de trabalho, é importante ter todos os detalhes bem explanados e fáceis de serem lidos.

Este material explora os cinco diagramas mais utilizados e que são, de fato, de extrema importância para o desenvolvedor de sistemas. O processo ideal para iniciar todos já têm em mente, mas agora é o momento de entender a complexidade que pode ser atingida antes mesmo de iniciar as primeiras linhas de código.

Diagrama de classes

Conversa diretamente com a orientação a objetos e também com a forma como as classes dos sistemas se relacionam com outras, podendo determinar a complexidade e o tamanho de qualquer aplicação, além de mostrar os atributos e os métodos de cada classe.

Diagrama de caso de uso

Descreve as funcionalidades do sistema como um todo, só que pelo ponto de vista dos usuários. É importante ressaltar que os usuários não são necessariamente só usuários externos; eles podem ser (e muitas vezes são) funcionários de uma empresa com diferentes acessos.

Diagrama de pacotes



Engloba os outros dois diagramas já apresentados e é um grande auxiliar para organizar justamente o que é produzido pelas **classes** e pelos **casos de uso**. Ele separa as funcionalidades em pacotes e ajuda a manter uma organização desses demais diagramas para facilitar a busca de informações.

Diagrama de sequência

Funciona como uma forma mais direta de explicar e demonstrar as interações entre os objetos, as quais ocorrerão de forma sequencial. Ele trabalha diretamente com os **métodos** descritos nos **diagramas de classes** para exemplificar a sequência de passos para a organização de uma ação.

Diagrama de atividades

Funciona como um fluxograma, sendo um pouco mais visual que os outros. Ele mostra o controle das atividades de um sistema, e, conforme vão sendo executadas, elas acabam passando por um processo que envolve outras **atividades**.

Todos os diagramas mencionados acabam utilizando símbolos para conectar seus fundamentos com os outros diagramas, o que deve ser bem exemplificado para que torne a questão visual correta. Esses símbolos auxiliam na ligação e na conexão entre os objetos dos diagramas e precisarão estar sempre presentes.

Diagrama de classes

É, sem dúvida, o grande pilar da programação orientada a objetos e um dos grandes responsáveis por haver uma visualização geral dos sistemas e dos aplicativos a serem desenvolvidos.

Quando se trabalha com os diagramas de classes, devem-se considerar algumas questões bem importantes. Apenas para lembrar as **classes**, todas têm **atributos** e **métodos**. Então esses diagramas são os utilizados para mostrar não só as classes bem montadas, mas também os relacionamentos entre elas.

Para começar, veja um problema simples:

Em um primeiro momento do sistema, considere que existirão **duas** classes, pensando em um sistema de uma faculdade. Inicia-se com o “professor”. Como **atributos**, ele deve ter nome, número de funcionário, anos de serviço e número de turmas. Claro que todo sistema de faculdade também tem alunos. O “aluno” deve ter nome, número de matrícula e média geral. Além disso, ele precisa conseguir verificar a possibilidade de se matricular e a quantidade de horas complementares que já cumpriu.

Então, seguindo o padrão de diagramas de classes, veja como montar individualmente cada uma:

Professor
- nome: String
- numFuncionario: int
- anosServico: int
- numTurmas: int

Aluno
- nome: String
- numMatricula: int
- mediaGeral: double
+ gets e sets()
+ podeMatricular(): bool
+ verificarHorasComplementares():int

Até o momento, o processo está simples e familiar, certo?



É importante lembrar também a estrutura de um diagrama de classes: na **seção superior**, está o nome da classe; na **seção central**, estão os atributos, com suas visibilidades e seus tipos; e, por fim, na **seção inferior**, estão os métodos, com seus parâmetros (caso sejam necessários), sua visibilidade e seu retorno.

Contudo, você precisa ter em mente que trabalhará, sempre que possível, com diversas classes em um programa mais complexo, e isso gerará um diagrama cada vez maior e também com mais conexões.

Antes de continuar com o exemplo, vale a pena conhecer os símbolos e legendá-los, para que, em uma construção maior, fique claro não só o que está sendo feito, mas também a importância do processo.

Símbolos

Dependência

É extremamente simples, mas acaba trazendo uma visualização maior de como algumas classes acabam se comunicando entre si dentro do projeto e do aplicativo. Suponha que, no sistema da faculdade, tenha o Portal do Aluno e o Portal do Professor, ambos com *layouts* e visualizações diferentes. Então, os símbolos e o diagrama seriam estes:

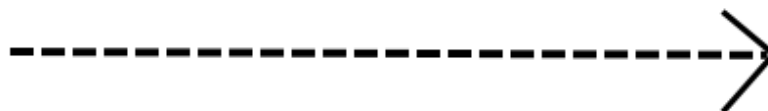


Figura 1 – Símbolo de dependência

Fonte: Senac EAD (2022)

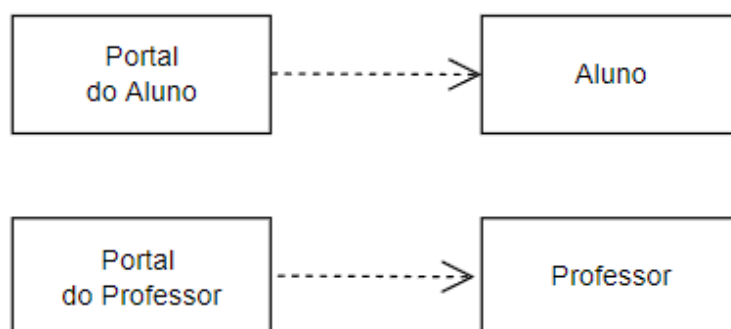


Figura 2 – Exemplo do uso de dependência

Fonte: Senac EAD (2022)

É importante pensar da seguinte forma: “Portal do Aluno” depende de “Aluno”; e “Portal do Professor” depende de “Professor”.

Agregação

Funciona de forma simples, mas indica um relacionamento importante entre uma e mais classes. Utiliza-se a agregação quando **as informações contidas em um dos objetos** precisam ser complementadas por outra classe. O símbolo de agregação é este:

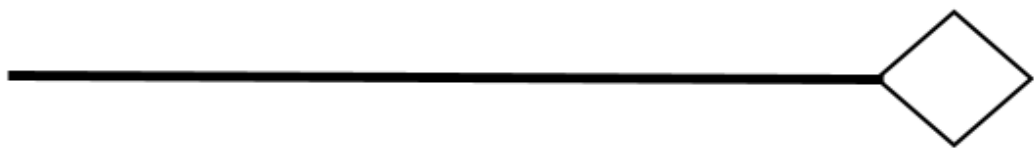


Figura 3 – Símbolo de agregação

Fonte: Senac EAD (2022)

Agora pense no uso da agregação em um sistema *web* utilizado por uma loja para vender livros. O usuário pode adicionar livros ao carrinho e depois fechar a compra retornando o valor final de tudo:

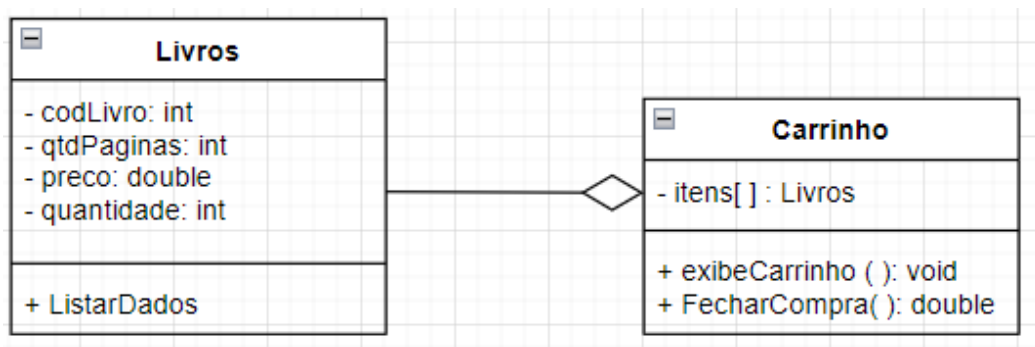


Figura 4 – Exemplo do uso da agregação

Fonte: Senac EAD (2022)

Composição

É similar à agregação, até mesmo no seu símbolo, mas contém uma diferença bem grande na hora de criar a lógica por trás disso. A composição implica um relacionamento no qual a classe filha não pode existir sem a classe pai. O símbolo de composição é o seguinte:

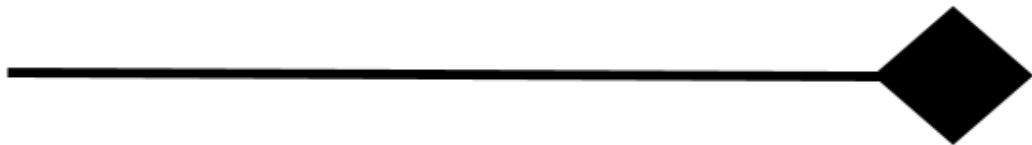


Figura 5 – Símbolo de composição

Fonte: Senac EAD (2022)

Exemplificando o uso de composição de uma forma mais abstrata, pense em uma casa. Uma casa tem quartos, correto? Porém, pensando na lógica por trás da composição, os diagramas ficariam assim:

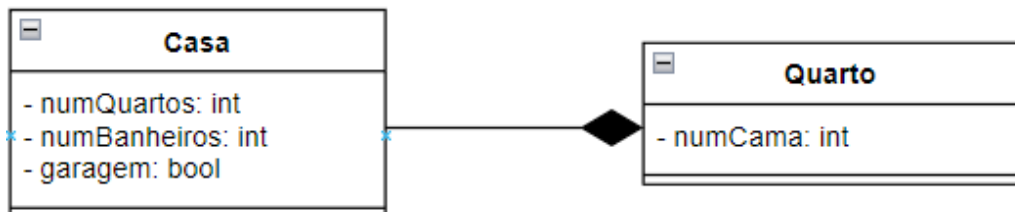


Figura 6 – Exemplo do uso de composição

Fonte: Senac EAD (2022)

A agregação e a composição são muito similares. Então, para resumir e trazer tais conceitos de maneira mais simples, veja ambas aplicadas aos exemplos anteriores:

- ◆ **Agregação:** relacionamento em que a classe filha pode existir independentemente da classe pai, isto é, **Livro** pode existir sem **Carrinho**.
- ◆ **Composição:** relacionamento em que a classe filha não pode existir sem a classe pai. **Quarto** não pode existir sem **Casa**.

Na prática

Você foi designado para criar um sistema que cadastre pacientes de uma clínica. É de extrema importância que, nesse sistema, constem os dados dos pacientes, tais como nome, CPF (cadastro de pessoa física), telefone de emergência, data de nascimento, alergias etc. Além disso, foi solicitado também o cadastro de remédios, com seus respectivos dados: nome, número de registro, quantidade em estoque, entre outras informações.

Pense que esses remédios são usados (ou não) pelos pacientes da clínica. Pode haver, assim, um relacionamento entre essas classes (paciente e remédio), possibilitando a criação de novos atributos e métodos.

Diagrama de caso de uso

É um modelo bem diferente do diagrama de classes. É no diagrama de caso de uso que se consegue separar **usuário por usuário**, bem como os papéis que cada um deles desempenhará dentro do sistema. Cabe lembrar que, dentro de um sistema, na maioria das vezes, existirão usuários com diferentes níveis de acesso, e os casos de uso servem justamente para mostrar isso.

Os diagramas de caso de uso compreendem quatro tipos de símbolo:

Atores

Os atores serão usados para mostrar qualquer entidade que tenha um papel dentro do sistema. Pode ser uma pessoa, uma organização, um sistema externo ou qualquer coisa que venha a ter uma função com acessos específicos do sistema.



Figura 7 – Símbolo de ator

Fonte: Senac EAD (2022)

Caso de uso

O caso de uso representa uma função ou uma ação feita dentro do sistema. Sempre se usa um símbolo oval para descrever essa ação:

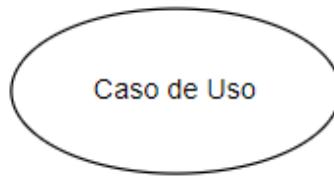


Figura 8 – Símbolo de caso de uso

Fonte: Senac EAD (2022)

Sistema

O símbolo de sistema exemplifica todo o sistema. É representado por um retângulo, no qual, na parte de dentro, ficarão os **casos de uso** e, na parte de fora, os **atores**:

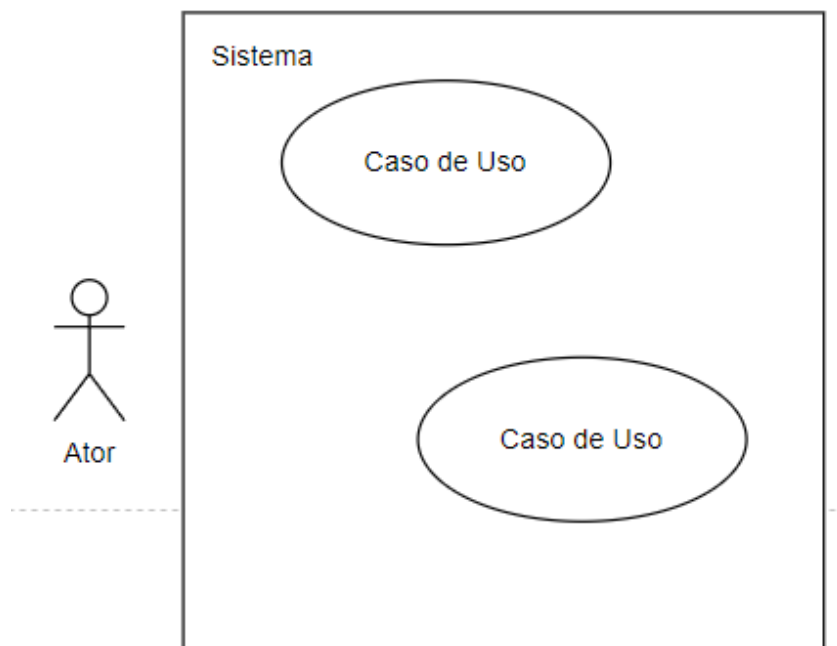


Figura 9 – Símbolo de sistema com ator e caso de uso

Fonte: Senac EAD (2022)

Relacionamentos

Como existem atores e casos de uso, é preciso haver os relacionamentos entre eles. Esses relacionamentos são indicados com setas e com duas palavras: *include* e *extend*.

O termo “*include*” é utilizado sempre que um ator precisa usar um dos casos de uso, ou seja, quando este não for opcional e precisar sempre estar presente. Já o termo “*extend*” é usado quando é preciso adicionar um novo caso de uso a um já existente. O *extend* não é obrigatório, mas geralmente uma condição (verdadeira ou falsa) pode dar o gatilho para ele acontecer.



Figura 10 – Símbolo de **Extends** e **Include**

Fonte: Senac EAD (2022)

As setas de *extend* e *include* são bem similares; o que as diferencia entre si sempre é a palavra escrita juntamente à linha.

Conhecidos os símbolos, pode-se elaborar o diagrama de caso de uso. Pense no seguinte sistema de forma bem simples: em um sistema bancário, há um cliente e o funcionário do banco. O cliente pode abrir uma conta, depositar dinheiro e, dependendo de algumas condições, ganhar um bônus. O funcionário do banco também pode abrir uma conta.

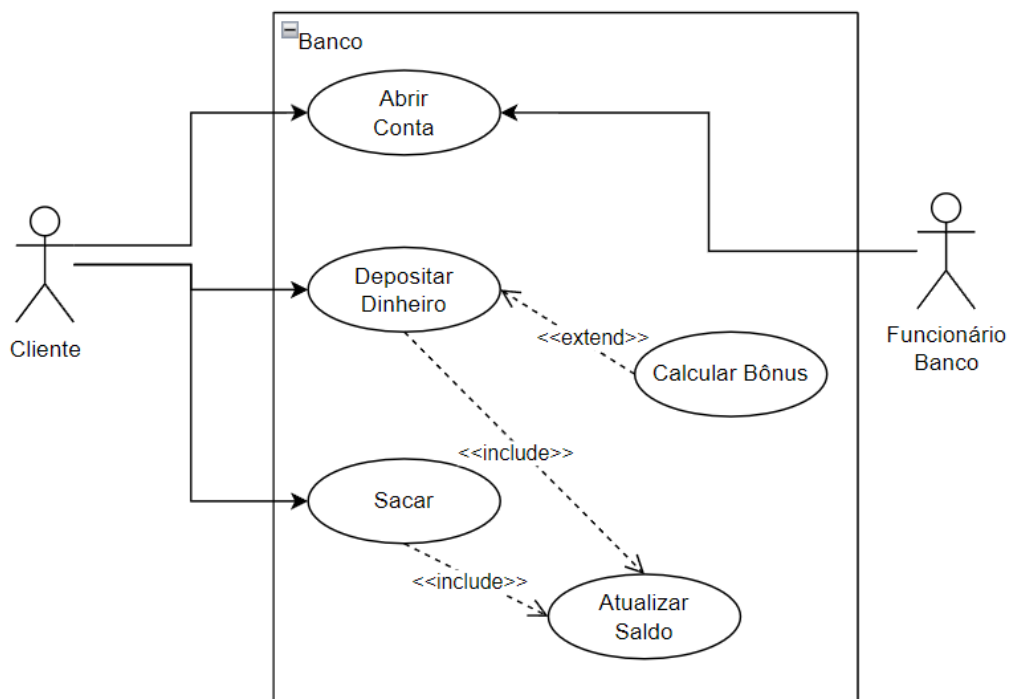


Figura 11 – Exemplo de diagrama de caso uso

Fonte: Senac EAD (2022)

Note que, como funções principais, o cliente pode **Abrir Conta**, **Depositar Dinheiro** e **Sacar**. Porém, para que o saldo seja atualizado, é preciso que as outras operações sejam feitas. Por isso é utilizado o *include* no caso, mostrando que a função **Atualizar Saldo** precisa dos demais casos de uso. Já **Calcular Bônus** traz um *extend*, pois existe uma condição para que esse cálculo seja aplicado. Obviamente, essa condição é definida pelo desenvolvedor (o bônus pode ser aplicado caso o usuário tenha mais de R\$ 10 mil na conta ou mais de 60 anos de idade).

Na prática

Você está desenvolvendo um sistema para um prédio com diversos inquilinos. O síndico deve ter um acesso privilegiado ao sistema, pois precisa ter funções e acessos diferentes, como preparar os gastos com o condomínio

mensalmente, lançar despesas de cada inquilino, calcular o total etc. É importante também que exista uma transparência desses gastos.

Pense bem em quais serão os atores e em quais serão de fato os casos de uso do síndico e dos inquilinos.

Diagrama de pacotes

É um dos diagramas visualmente mais simples, porém é utilizado geralmente em projetos de média a grande escala, isto é, quando se tem uma grande quantidade de diagramas de caso de uso e diagramas de classes.

Você pode e deve utilizar o diagrama de pacotes para organizar e visualizar melhor o projeto de uma forma macro. Confira algumas das vantagens em utilizá-lo:



- ◆ Proporciona uma visão estrutural da hierarquia de outros elementos de UML dentro do sistema
- ◆ Simplifica os diagramas de classes sem precisar ficar exibindo os atributos e os métodos para melhorar a visualização
- ◆ Demonstra que, quanto maior for o projeto, mais necessário se torna o uso de diagrama

O símbolo do diagrama de pacotes é bem simples:

- ◆ **Pacote:** agrupa as classes e os seus relacionamentos. É representado por um retângulo com uma etiqueta acima contendo o nome:

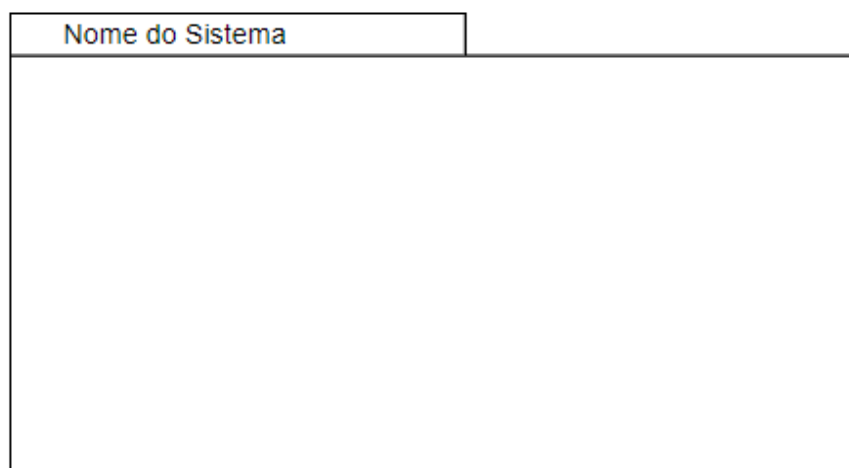


Figura 12 – Símbolo de pacote

Fonte: Senac EAD (2022)

Dentro de pacote serão colocadas as classes e feitas as ligações entre elas. Esse processo é extremamente necessário para se ter uma visão macro dos sistemas, além de proporcionar uma estrutura mais organizada, que ajuda os desenvolvedores a verem o tamanho do sistema.

Pensando ainda no sistema bancário, é importante organizar algumas classes dentro desse pacote e criar os relacionamentos entre elas:

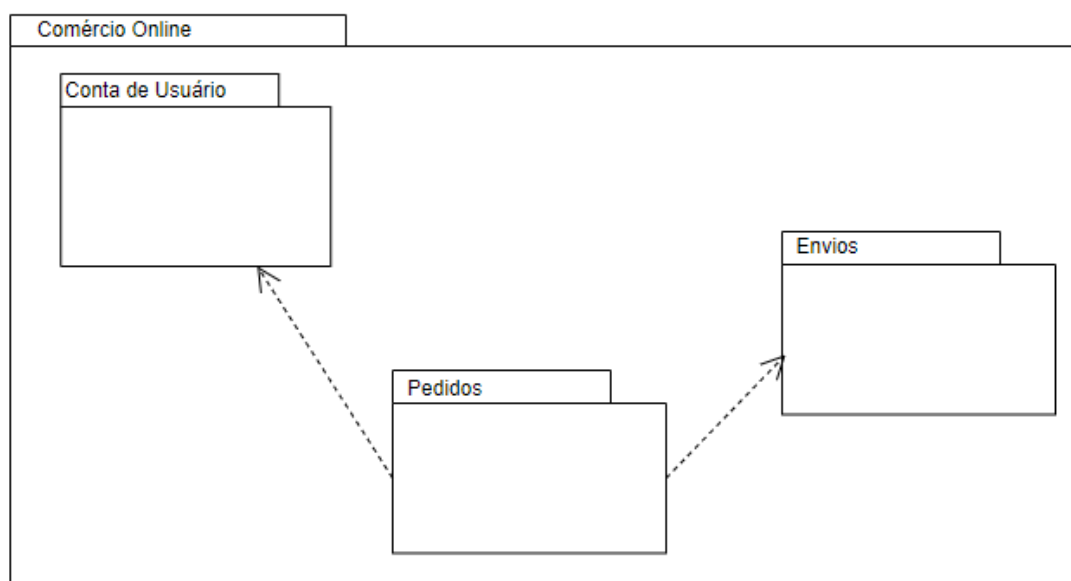


Figura 13 – Exemplo de diagrama de pacotes em um sistema de comércio *on-line*

Fonte: Senac EAD (2022)

Simple, não é? Como você pôde ver, as classes estão ocultas, apenas mostrando os pacotes principais que estarão dentro desse sistema. Isso porque elas não precisam estar visíveis nesse tipo de diagrama.

Outra função muito importante dos diagramas de pacotes é a possibilidade de criar um pacote que agrupa outros pacotes. Como dito, tudo o que está sendo mostrado neste conteúdo é usado em projetos de larga escala e facilita muito o entendimento do sistema como um todo.

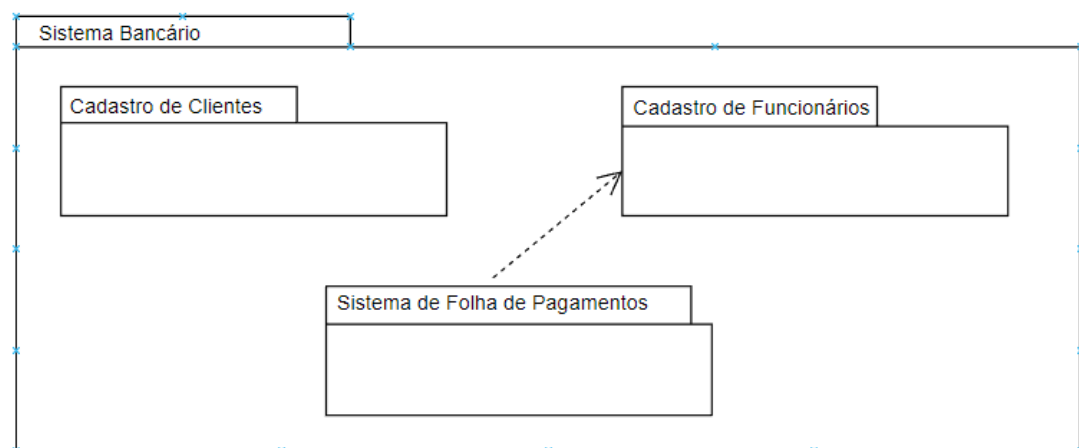



Figura 14 – Exemplo de um diagrama de pacotes com vários pacotes

Fonte: Senac EAD (2022)

Que tal elaborar agora o diagrama de pacotes, expandindo o comércio *on-line* exemplificado?

Na prática

No pacote de comércio *on-line*, apenas foram previstos três pacotes básicos. Porém, você foi escolhido para criar um diagrama de pacotes separando agora os tipos de compras que podem haver. Então, crie pacotes de **Compras Internet**, **Compras *mobile*** e **Compras via telefone**.



Lembre-se de criar as dependências. Mesmo as compras sendo feitas de modos diferentes, o estoque e o pagamento serão os mesmos.

Diagrama de atividades

É um diagrama extremamente simples se comparado aos diagramas de sequência. Ele traz as atividades que serão realizadas pelo sistema em forma de fluxograma, facilitando a visibilidade não só para os desenvolvedores, mas também para os clientes, mostrando o processo de forma simples e não tão técnica. Em suma, traz muitas vantagens. São elas:



- ◆ Descreve o fluxo de atividades de um sistema, descrevendo uma a uma
- ◆ Mostra a sequência de uma atividade, uma após a outra
- ◆ Traz as condições para que uma atividade seja realizada ou não, criando mais caminhos dentro de uma simples atividade, prezando os detalhes

O diagrama de atividades conta ainda com diversos símbolos, e cada um deles é necessário para facilitar a leitura do diagrama. Veja:

Atividade

Contém geralmente uma curta descrição textual da atividade que é executada. É o símbolo principal do diagrama de atividades.



Figura 15 – Símbolo de atividade

Fonte: Senac EAD (2022)

Conector

Demonstra as conexões entre atividades. Depois que uma atividade é finalizada ou executada, é o conector que cria o “*link*” com a próxima atividade ou com o próximo passo do sistema.



Figura 16 – Símbolo de conector

Fonte: Senac EAD (2022)

Início de processo e fim de processo

Servem, como o próprio nome sugere, apenas para indicar o início e o fim de um processo. Na maioria das vezes, há apenas um início, mas pode haver diversos fins dependendo das atividades e das condições que o sistema impõe ao usuário.



Figura 17 – Símbolos de início (à esquerda) e de fim (à direita) de processo

Fonte: Senac EAD (2022)

Condição

É neste símbolo que as atividades começam a ter caminhos diferentes. Quando são definidas condições dentro do sistema (geralmente usando SE ou IF), sempre existirão um caminho verdadeiro e um falso, podendo retornar duas atividades diferentes. Durante o caminho, no qual se usam os conectores, coloca-se um texto acima deles para indicar o lado para o qual a condição está sendo levada.

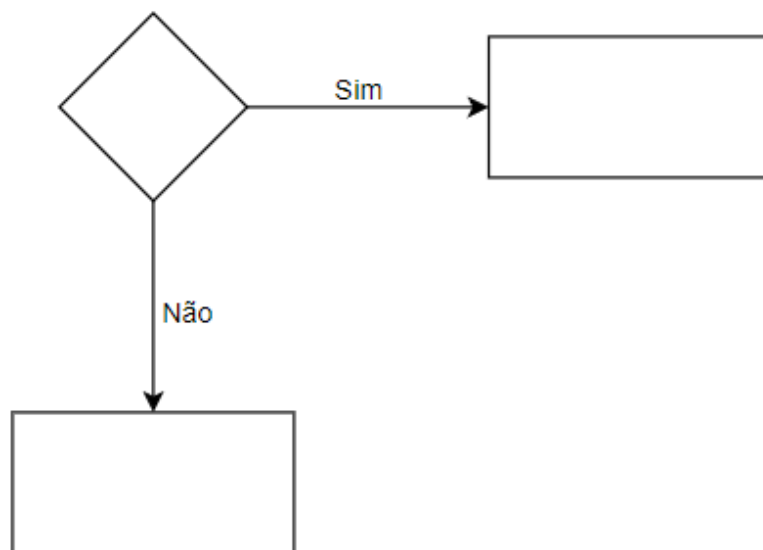


Figura 18 – Símbolo de condição

Fonte: Senac EAD (2022)

São muitos símbolos desta vez, mas a visualização acaba geralmente por ficar simples. O exemplo a seguir continua baseado em um sistema bancário, para facilitar a visualização. A atividade a ser realizada é a do uso de um **caixa eletrônico** simples para o usuário poder realizar diferentes atividades durante o processo:

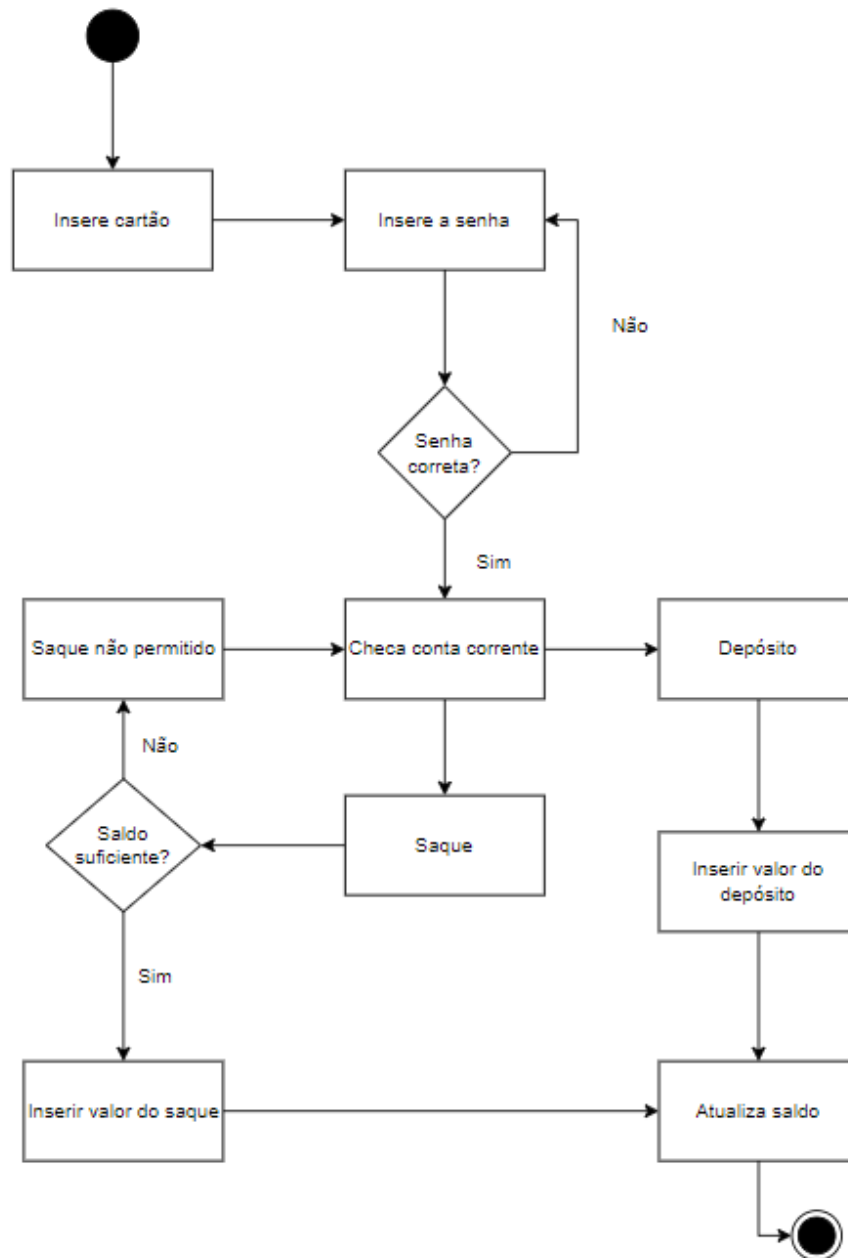


Figura 19 – Exemplo de diagrama de atividades para uma atividade em um sistema bancário

Fonte: Senac EAD (2022)

Como você pôde ver na figura 19, a criação do diagrama de atividades é bem direta e simples. Claro que é necessário levar em consideração o fato de que todas essas atividades estarão ativando, usando e atualizando os **métodos** e os **atributos** de todas as classes envolvidas nessa simples atividade.

Pensando no exemplo do sistema bancário, o atributo “saldo” do cliente do banco será buscado para permitir o saque e/ou o depósito – (**getSaldo()**), por exemplo –, e o método **AtualizaSaldo()** será usado antes de finalizar o processo.

Em atividades mais complexas, o diagrama de atividades ajuda muito na forma de desenvolvimento de cada atividade e na escolha dos atributos e dos métodos que o desenvolvedor precisará utilizar durante todo o processo do sistema.

Na prática

Você precisa construir um diagrama de atividades de um sistema escolar no qual os professores cadastrem as atividades para os alunos realizá-las. As atividades podem ser criadas do zero, podem atribuir uma nota ou podem até mesmo ser editadas.

Lembre-se das condições deste desafio, pois elas levarão para novas atividades. Todas devem estar previstas no diagrama.

Diagrama de sequência

É, sem dúvida, um dos mais complexos em termos de símbolos e de leitura, mas o grau de detalhes que ele traz para determinadas informações é essencial para ver justamente quais são as classes (e qual é a ordem) que se envolvem dentro de determinado processo.

Entre os vários benefícios do diagrama de sequência, estão:



- ◆ Representa, em detalhes, cada processo de um **diagrama de caso de uso**
- ◆ Mostra como objetos, classes e componentes interagem um com os outros até um processo estar completo
- ◆ Ajuda a planejar a funcionalidade de uma ou mais atividades que já existem e abre espaço para atividades futuras

Como dito, o diagrama de sequência, diferentemente dos demais, mostra toda a sequência de um processo. Logo, é importante saber que todas as classes e todos os processos envolvidos devem ser trazidos, sejam atores, sejam classes que interagirão com esses atores. Os símbolos desse diagrama são um pouco mais complexos. Veja:

Ator

Já conhecido do diagrama de caso de uso, o ator, no diagrama de sequência, serve para a mesma indicação visual:



Figura 20 – Símbolo de ator

Fonte: Senac EAD (2022)

Objeto

O objeto representa uma classe ou um objeto dentro do diagrama. Não é necessário mostrar atributos e métodos. O símbolo é um simples retângulo:

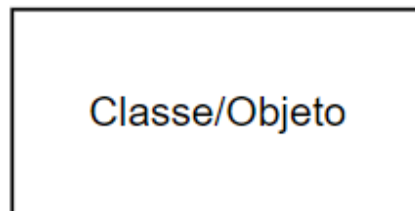


Figura 21 – Símbolo de classe ou objeto

Fonte: Senac EAD (2022)

Caixa de ativação

É o principal símbolo do diagrama de sequência. A caixa de ativação mostra o tempo em que estará ativo um objeto, uma classe ou um ator durante toda a sequência que está sendo executada. Dependendo das opções selecionadas, novas classes podem ser ativadas.

O símbolo consiste em um retângulo vertical, e geralmente se usam cores para representar cada uma das caixas:



Figura 22 – Símbolo de caixa de ativação

Fonte: Senac EAD (2022)

Mensagem síncrona

É o símbolo utilizado quando o sistema envia uma mensagem e deve aguardar a resposta ou o retorno para poder continuar. O diagrama deve, obrigatoriamente, executar a mensagem enviada e a resposta. Por exemplo: registrar uma venda em um sistema.



Figura 23 – Símbolo de mensagem síncrona

Fonte: Senac EAD (2022)

Mensagem assíncrona

É similar à mensagem síncrona. A mensagem assíncrona, porém, envia para o usuário um retorno sem depender de outra função ou de retorno similar. Por exemplo: uma operação para apresentar uma mensagem em um

monitor.



Figura 24 – Símbolo de mensagem assíncrona

Fonte: Senac EAD (2022)

Explicados todos os símbolos, chegou o momento de construir o diagrama de sequência – com o diagrama de atividades, que mostra outras classes e outros objetos envolvidos – usando o mesmo exemplo de sistema bancário no qual o cliente solicita fazer um saque e ver a diferença de saldo:

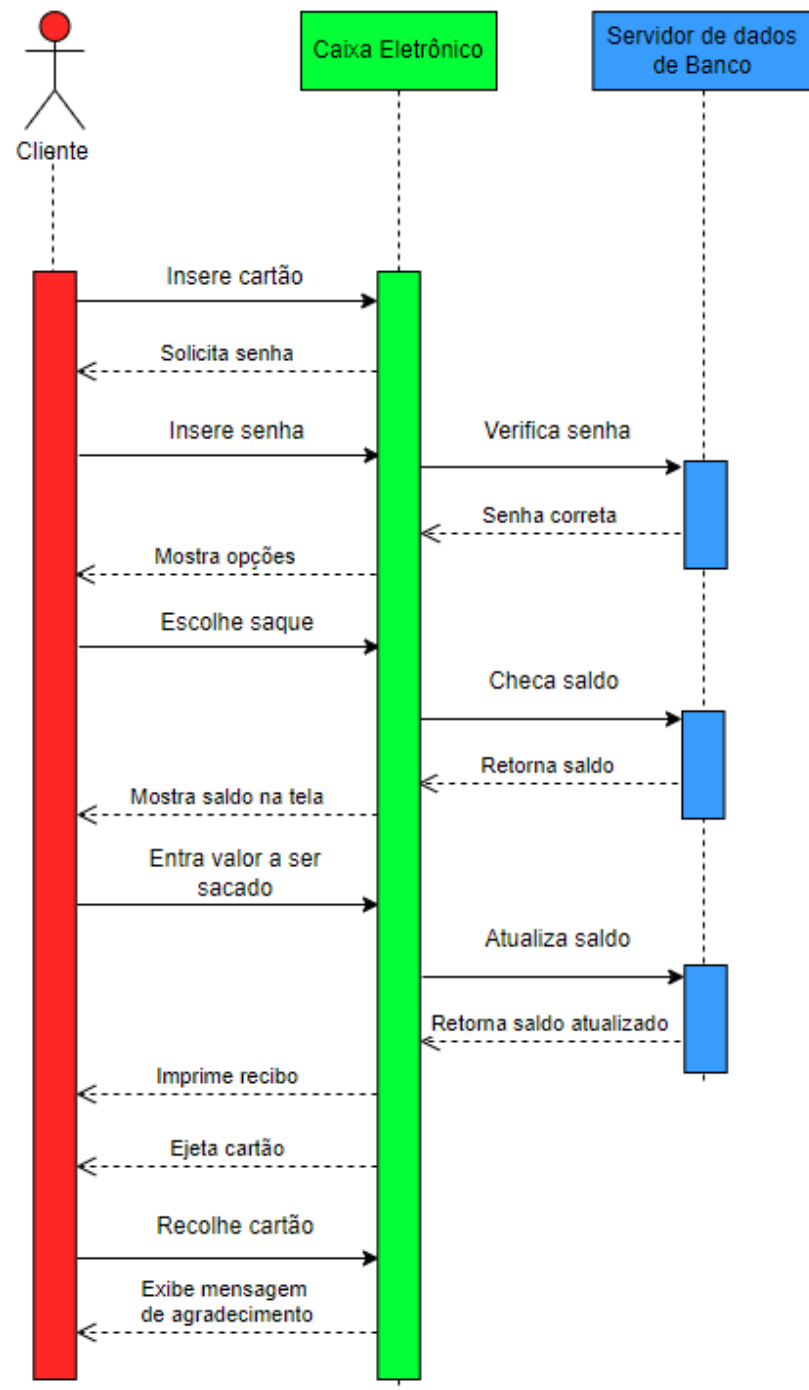


Figura 25 – Exemplo de diagrama de sequência para uma operação de saldo

Fonte: Senac EAD (2022)

Note que o cliente está interagindo o tempo inteiro com o caixa eletrônico, e este está interagindo com o servidor para verificar as atividades solicitadas pelo cliente. Porém, na maioria das vezes, apenas em

determinadas situações o servidor é chamado nessa sequência, pois se trata de uma classe a ser usada dependendo das atividades.

Tudo isso apresenta os processos de maneira mais visual. Embora seja um diagrama longo, é ele que detalha mais ainda cada passo dos **casos de uso** e a forma como os **atores** impactam o sistema.

Na prática

Uma agência de turismo contratou você para desenvolver um sistema de venda de passagens. Esse sistema consiste em um pedido do usuário que é checado por um funcionário, e o sistema da agência fará as reservas, os cálculos, enviará *e-mail* para o cliente etc. Você deve desenvolver um diagrama de sequência considerando todos os passos e os retornos possíveis.

Fique atento ao fato de que o cliente conversará com um funcionário, e o funcionário colocará as informações no sistema para verificar os dados.

Encerramento

Você chegou ao fim deste conhecimento. Os diversos diagramas abordados têm como função principal exemplificar e tornar visual tudo que os sistemas poderão fazer e trazer para os usuários.

Em uma empresa, o processo de construção de diagramas é necessário para documentar o projeto como um todo, auxiliando todos os membros da equipe. Lembre-se de que nem sempre (e idealmente) uma equipe é composta apenas de

desenvolvedores. Há artistas de UI (*user interface*) e UX (*user experience*), engenheiros de *softwares*, analistas de sistemas, cientistas da computação, entre outros, e todos eles devem ter acesso e ser capazes de ler os diagramas para estarem na mesma sintonia e até mesmo trazerem novas ideias.

O processo pode ser trabalhoso, mas, com certeza, é válido dentro de qualquer projeto. Essa fase de pré-produção se faz necessária e traz confiabilidade e profissionalismo para todos os sistemas que vierem a ser produzidos. Pense em um projeto de uma casa, por exemplo. Um engenheiro precisa ter a documentação e a planta da casa para saber por onde começar e como construir cada cômodo. Os diagramas são, assim, a “planta” do seu sistema, a qual, se bem-feita e bem produzida, pode poupar tempo e tornar o sistema mais coeso para todos os usuários e para a equipe envolvida.