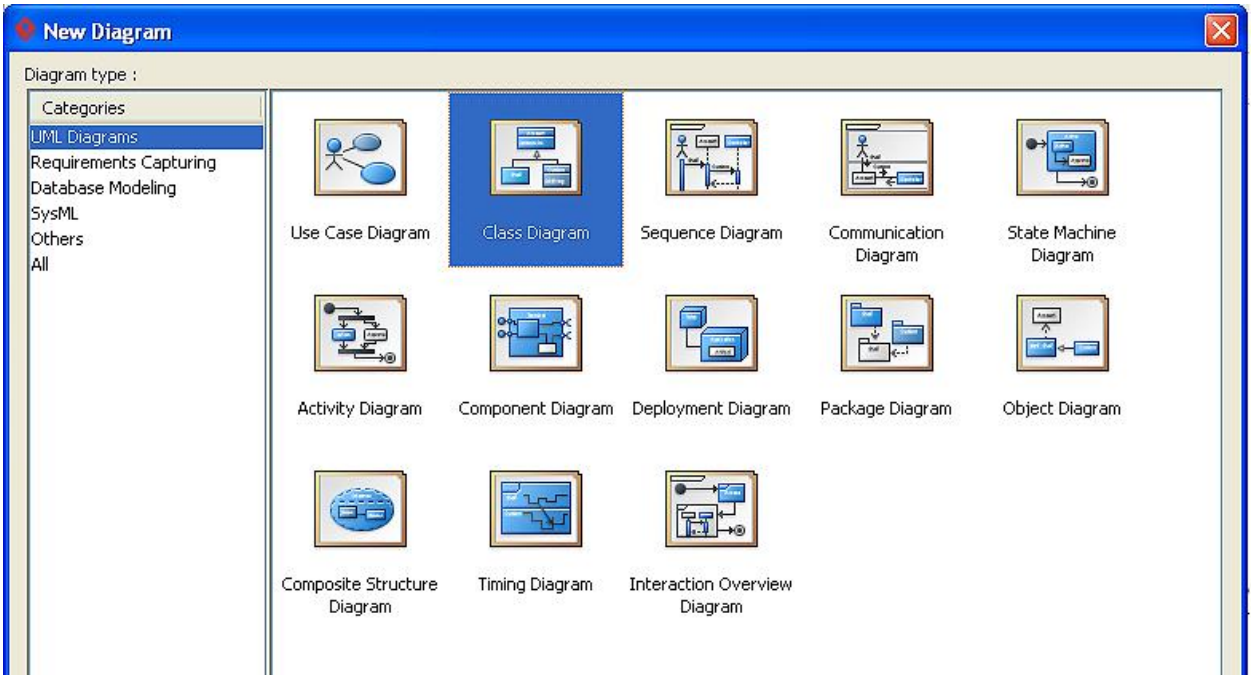
UNIDADE 2 – DIAGRAMAS BÁSICOS DA UML.
MÓDULO 1 – DIAGRAMA DE CLASSES (ATRIBUTOS E MÉTODOS)

01

1 - ATRIBUTOS

Olá, seja bem-vindo a mais uma Unidade estudo. Você continuará a aprender conceitos de Orientação a Objetos e diagramação UML. Trataremos agora dos atributos e métodos de uma classe.



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

Após identificarmos as classes, é hora de considerarmos as suas respectivas propriedades (atributos). Os atributos referem-se às informações que podem ser passadas e/ou recebidas para uma classe antes da execução de seus métodos. Normalmente, um projeto de *software* pode utilizar de duas técnicas de operação de métodos em uma classe, e saber diferenciá-los é importantíssimo para a boa construção de classes.

02

1.1. Uma classe que possui atributos

A forma mais tradicional de codificação é aquela que utilizamos atributos para guardar e recuperar as informações de uma classe. Imagine, por exemplo, uma classe “Pessoas” que mantém somente duas informações: “Nome” e “Endereço” da pessoa; para isso, a classe precisaria possuir dois atributos para

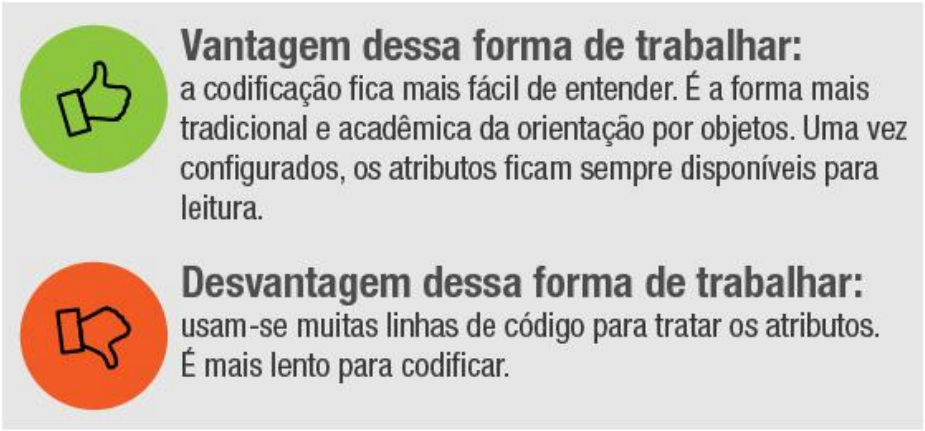
armazenar essas informações. A classe também precisaria de dois métodos, um para gravar os dados da pessoa e outro para ler os dados. Para poder executar a operação de gravar o nome e o endereço da pessoa, primeiramente seria necessário configurar os respectivos atributos, só então chamar o método. Algo assim:

```
Pessoa.Nome = “Maria do Rosário”
Pessoa.Endereco = “Rua das Flores, número 80”
Pessoa.gravarDados ()
```

Ou seja, as informações acerca da pessoa foram passadas por meio de atributos, depois, a operação de salvar as informações pegaria o nome e o endereço dos parâmetros fornecidos. O mesmo poderia ser feito para recuperar os dados da pessoa. Poderia existir um método de leitura dos dados que funcionasse assim:

```
Pessoa.lerDados ()
Nome = Pessoa.Nome
Endereco = Pessoa.Endereco
```

Note que primeiro foi executado o método e, depois, operações para recuperar as informações dos respectivos atributos.



03

1.2. Uma classe que não possui atributos

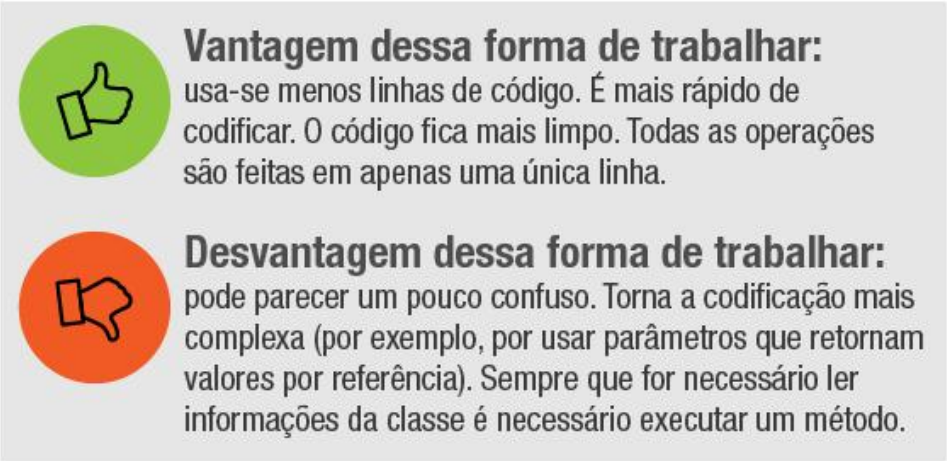
Uma outra forma de se trabalhar é onde você cria uma classe sem atributos, todas as informações enviadas ou recebidas da classe são feitas por meio dos parâmetros dos métodos da classe. Imagine, por exemplo, a mesma classe “Pessoa”, mas agora configurada de outra maneira. Essa classe poderia ter dois métodos construídos, um para gravar os dados da pessoa e outro para ler os dados da pessoa. Você poderia executar a operação de gravar o nome da pessoa assim:

```
Pessoa.gravarDados (“Maria do Rosário”, “Rua das Flores, número 80”)
```

Ou seja, as informações acerca da pessoa foram passadas por meio de parâmetros. O mesmo poderia ser feito para recuperar os dados da pessoa. Poderia existir um método que funcionasse assim:

```
Pessoa.lerDados (nome, endereco)
```

Esse método receberia nas variáveis “nome” e “endereco” os valores desejados.



Escolher entre uma ou outra opção é uma questão de modelo de desenvolvimento a ser discutido com a equipe de projeto.

04

1.3. Identificando atributos

Encontrar os atributos de uma classe normalmente não é tarefa difícil, uma das formas mais comuns é analisar como você descreveria um objeto daquele tipo de classe e como você poderia diferenciá-lo dos demais. Por exemplo, se a cor de um objeto faz sentido ou parece ser importante para diferenciá-lo de outro, então a cor poderá ser um atributo. Se o peso é importante, então este será outro atributo.

Num contexto de um sistema de informação, não devemos levar em conta todos os atributos possíveis e imagináveis de uma classe ou um objeto, mas somente aqueles que fazem sentido para o sistema.

Por exemplo: suponha que você esteja trabalhando na construção de um sistema para gerenciar as vendas de uma livraria, e que você identificou que “Cliente” é uma classe importante. É muito provável que os atributos mais importantes, relevantes e úteis que essa classe pode ter nesse contexto são: nome do cliente, endereço, CPF, identidade, e assim por diante. Outros atributos como cor dos olhos, altura, peso, cor dos cabelos, e outros, parecem ser totalmente inúteis e sem sentido para esse sistema; portanto, não devem fazer parte da sua modelagem.

05

1.4. Tipos de atributo

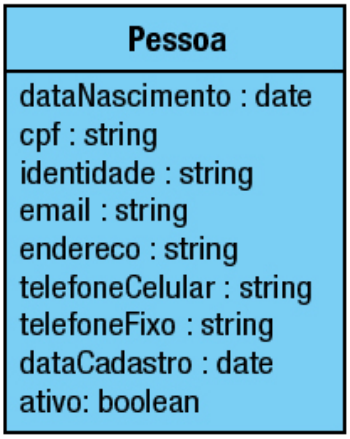
Após documentar os atributos das classes, é importante identificar seus tipos. Os tipos estão relacionados a quais tipos de informação aquele atributo pode armazenar. Por padrão, todos os tipos são declarados com a inicial em letra minúscula, seguindo esta sintaxe:

```
atributo: Tipo
```

Os tipos básicos comuns de tipos de atributo são:

- **Integer** (inteiro);
- **Float** (fracionados);
- **Double** (duplo);
- **Boolean** (booleano);
- **Date** (data);
- **String** (texto).

Dessa forma, para uma classe “pessoa” poderíamos ter, por exemplo, os seguintes atributos:



Exemplo de diagramação de tipos de atributos em uma classe

Integer
Para números inteiros, positivos e negativos, como: -30; 44; 0.

Float
Para números que possuem casas decimais, como: -33,453; R$ 44,30.

Double
Para números muito grandes ou muito pequenos, como: 0,20394890283098094; 1.234.567.890,01.

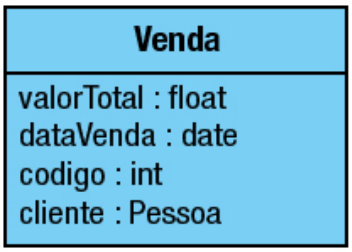
Boolean
Para valores binários como verdadeiro/falso ou 0 e 1.

Date
Para datas e datas com horas, como “01/01/2015”, “31/12/2014 12:00:00”.

String
Para um ou mais caracteres, como “José Pereira”, “P”, “+55 (61) 8429-3793”, “456.354.365/87”, “jose@gmail.com”.

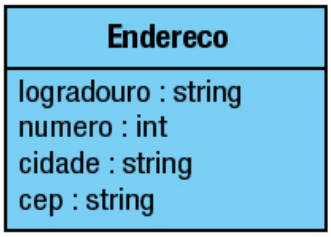
06

Algumas vezes, os atributos de um objeto podem ter sido obtidos em outra classe (ou referenciar um objeto em outra classe). Você pode representar essa situação utilizando o nome da classe no lugar do tipo de atributo. Por exemplo, em uma venda, você pode dizer que “cliente” não é um atributo simples, mas sim um objeto do tipo “Pessoa”. Dessa forma, nesse exemplo hipotético, poderíamos ter uma classe “Venda” com uma estrutura semelhante à:



Exemplo de diagramação de classe usando uma outra classe como tipo de atributo.

Outras vezes faz sentido organizar dados semelhantes e afins e uma classe. Isso acontece, por exemplo, com endereço. Normalmente, o endereço é composto por uma série de informações (logradouro, número, cidade, UF, CEP). Dessa forma, poderíamos tem uma classe “Endereco” com a seguinte estrutura:

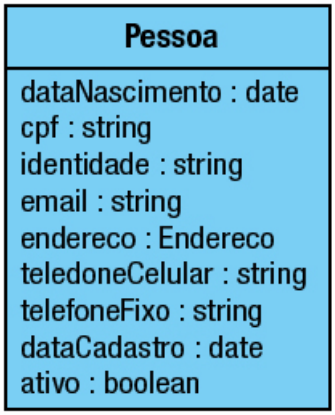


Exemplo de classe Endereço.

07

Feito isso, substituiríamos o tipo “string” do atributo “endereco” da classe pessoa, pela classe “Endereco”.

Dessa forma, a nova estrutura da classe “Pessoa” seria:



Exemplo da classe Pessoa utilizando o tipo do atributo da classe Endereco

Para evitar uma confusão entre o nome do atributo e o nome da classe, é muito comum que o nome do atributo seja alterado. Para o exemplo acima, poderíamos trocar o nome do atributo “endereco” para “enderecoPessoa”, ficando registrado da seguinte forma na classe:

```
enderecoPessoa = Endereco
```

08

1.5 Valor padrão

Quando o seu sistema entra em operação, espaços em memória são criados para armazenar os conteúdos dos objetos criados.

Se não especificados durante a criação, esses espaços conterão valores nulos. Valores nulos poder ser problemáticos em algumas situações, como em cálculos matemáticos ou em junção de strings (concatenação). Por isso, é boa prática de programação definir valores padrão para esses atributos. Normalmente utiliza-se 0 (zero) ou false (falso) para booleano, 0 (zero) para números, ‘’ (em branco) para strings, e a data atual ou uma data fixa para datas.

Para representar o valor padrão na definição de um atributo, utiliza-se o sinal de “=” (igual), e em seguida representa-se o valor padrão, seguindo esta sintaxe:

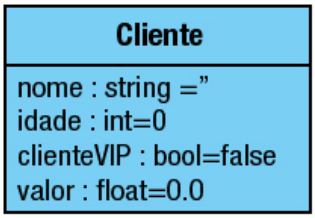
```
atributo: Tipo = ValorPadrão
```

Exemplos:

- `nome: String = ‘’`
- `idade: int = 0`
- `salario: float = 0,0`

```
• possuiFilhos: bool = False
```

Veja um exemplo hipotético de valor padrão na classe Cliente:



Exemplo da classe Cliente com valores padrão para os atributos.

09

1.6. Multiplicidade

Numa situação padrão, cada objeto tem um valor de atributo para cada atributo criado. Mas em situações especiais, você pode desejar que um objeto seja capaz de armazenar mais de um valor para o mesmo atributo. A notação para essa representação é:

```
Tipo do Atributo [Multiplicidade]
```

Por exemplo, para uma classe do tipo “pessoa”, você poderia permitir que o sistema armazenasse dois números de telefone para a mesma pessoa. Para resolver essa questão, não é necessário criar dois atributos, mas sim dizer que o atributo pode conter dois valores diferentes.

Dessa forma, o atributo telefone, seria representado assim: `telefone:String [2]`.

A UML permite uma série de representações distintas (com resultados distintos). Veja os exemplos abaixo para você compreender as diversas possibilidades:

Exemplo de representação	Significado
1	Exatamente 1 atributo (é o padrão e não precisa ser escrito).
2	Exatamente 2 atributos.
1..3	De 1 até 3 (inclusive) atributos.
3,5	Ou 3 ou 5 atributos.
1..*	No mínimo um, no máximo infinitos atributos.
*****	No mínimo zero, no máximo infinitos atributos (é o mesmo que 0..*)
0..1	Ou zero ou 1 atributo.

10

Quando um atributo for opcional, ele deve incluir a opção 0 na definição da sua multiplicidade, da mesma forma, se ele for obrigatório – e poder conter mais de um valor – deve possuir a opção 1 na representação. Exemplos:

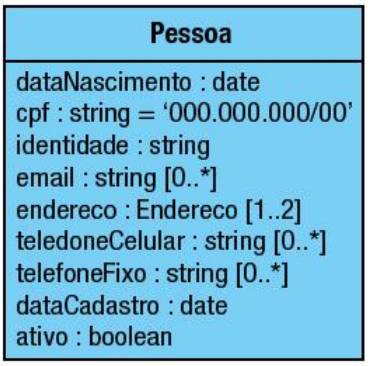
```
nomeDosPais:String [0..2].
numeroTelefone:Telefone [1..*]
```

Para atributos com mais de um valor, você pode representar os valore padrão colocando-os entre parênteses e separando-os por vírgulas.

Exemplos:

```
numeroTelefone:Telefone [2] (’00 00 0000-0000’, ’00 00 0000-0000’)
cidade:String[3] (‘Brasília’, ‘São Paulo’, ‘Rio de Janeiro’)
```

Veja como poderíamos evoluir a classe Pessoa incluindo algumas multiplicidades e valores padrão:



Exemplo da classe Pessoa com valores padrão e multiplicidade.

11

1.7. Ordenação e unicidade

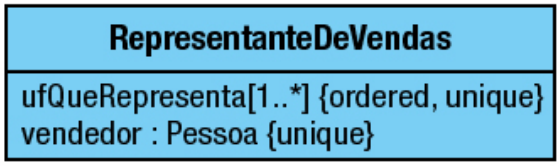
Há ainda duas informações importantes que podemos dar a um atributo, elas são a **ordenação** e a **unicidade**.

Dizer que um determinado atributo é ordenado, significa que todas as informações daquele item estarão em ordem alfabética. Para expressar essa característica, após a nomenclatura da multiplicidade, deve ser inserida a expressão `{ordered}`.

Também podemos dizer que aquele conjunto de informações do atributo não possam ser repetidas, ou seja, devem ser únicas. Para expressar essa característica, após a nomenclatura da multiplicidade, deve ser inserida a expressão `{unique}`.

Vamos ver um exemplo prático. Suponha que você possua uma classe de nome "RepresentanteDeVendas" e que esse representante de vendas atue em determinados estados. Você pode dizer que as unidades da federação em que ele atua estão ordenadas e são únicas. E que cada vendedor também é único (não pode haver dois vendedores com o mesmo nome).

Essa questão seria presentada da seguinte maneira:



Exemplo de classe com identificação de atributos únicos e ordenados.

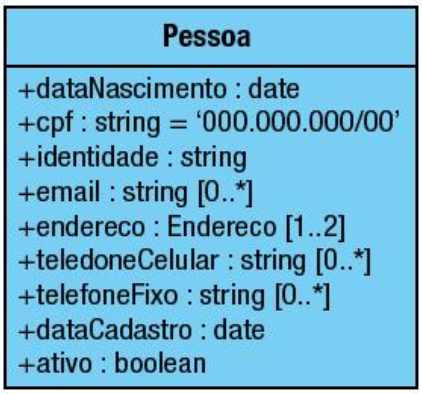
12

1.8. Visibilidade

A visibilidade é aplicada aos atributos e operações em uma classe. A **visibilidade** refere-se ao **escopo** de acesso permitido para um membro de uma classe. O escopo refere-se a regiões específicas dentro do sistema como um todo. As opções de símbolos de visibilidade são:

Símbolo	Visibilidade	Descrição
-	Privado	Só é possível acessar de dentro de uma classe. Geralmente, ao utilizarmos o conceito do encapsulamento, tudo que é encapsulado é privado.
~	Pacote	É possível acessar por qualquer classe que esteja dentro do mesmo pacote.
+	Público	É acessível por todos dentro do sistema.
#	Protegido	É acessível dentro de uma árvore de herança. Exemplo, uma classe A dentro de um pacote 1 e uma classe B que é herança de A em um pacote 2, neste caso, A será visível para B.

Dessa forma, aplicando os conceitos de visibilidade para a classe venda do exemplo anterior, teríamos:



Exemplo da classe Pessoa com atributos de visibilidade

13

2 - OPERAÇÕES

2.1. Identificando Operações

A partir do momento em que você estipula os atributos de cada classe, é chegada a hora de identificar os seus comportamentos. A UML utiliza o termo "operações" para se referir aos possíveis comportamentos de uma classe que os objetos dessa classe podem realizar. Para realizar tal tarefa, você deve se perguntar aos objetos daquela classe o que eles fazem, ou o que faz com que o estado deles mude.

O diagrama de atividades e a descrição dos casos de uso são duas fontes muito importantes de operações possíveis em um sistema.

A sintaxe padrão para descrever esses comportamentos é:

```
nomeDaOperacao (opcional
listaDeArgumentos): TipoDeRetorno
```

Dessa forma, como um exemplo hipotético, para que um objeto "Pessoa" efetue a compra de livros e confirme se ela conseguiu comprar os livros, a operação poderia se parecer algo como:

```
venderLivro (livros:Livro, valorTotal:float): VendaConcluida
```

Normalmente, sempre que uma operação é executada, o objeto "chamador" recebe algum retorno, mesmo que seja apenas uma confirmação do tipo verdadeiro/falso de que a operação ocorreu com sucesso. Entretanto, se nada é retornado ao objeto "chamador", você pode utilizar NULL ou omitir o tipo de retorno.

14

2.2. Nomeando operações e argumentos

A regra para definir os nomes para as operações é a mesma que utilizamos para os atributos (inicie com letra minúscula, elimine os espaços em branco, coloque as demais palavras com iniciais maiúsculas e depois insira parênteses para declarar que é uma operação – e não um atributo). Sendo assim, temos a seguinte sintaxe padrão:

```
nomeDaOperacao ()
```

Entretanto, ao invés de substantivos, operações devem possuir verbos. Para o idioma português, a maioria das organizações opta por verbos sempre no infinitivo [exemplos: `comprarLivro()`, `venderProduto()`, `cadastrarCliente()`, `consultarPessoa()`, `imprimirRelatorioGestaoResumido()`, `reservarDiariaHotel()`, `cancelarVendaRealizada()`].

Algumas situações permitem observar a questão por dois pontos de vista das partes (objetos) que fazem parte da ação. A boa prática nos recomenda que utilizemos sempre a visão mais significativa para o contexto, que normalmente é a do ator principal que realiza a operação.

Quando uma pessoa opera uma máquina automática de ingressos de cinema, a pessoa está realizando uma **compra**. Já a empresa está realizando uma **venda**. Então, que nome dar à operação, `comprar()` ou `vender()`? Nesse exemplo onde o escopo é apenas o da operação da máquina e que o ator principal é o cliente, parece-nos que o melhor nome é `comprar()`.

15

Por outro lado, em um contexto maior do sistema, essa ideia pode parecer confusa. Veja que num grande sistema que gerencie todas as operações da empresa de cinema, é bem provável que um funcionário da empresa precise comprar produtos de outros fornecedores para serem vendidos na empresa, por exemplo, pipoca. Nesse cenário, o nome da operação que o funcionário faz é comprar() também. Assim, será que o cliente também realiza o método comprar() para que ele possa comprar a pipoca pronta? Viu como isso pode parecer confuso.

Para deixar essas questões um pouco mais claras, devemos enxergar a empresa como algo no meio do caminho entre grandes fornecedores e seus clientes finais. Os produtos seguem o fluxo vindo dos fornecedores, passando pela empresa e sendo entregues aos clientes. Algo que se pareça com esse desenho:

Fornecedores -> Empresa (Cinema) -> Clientes

Mantendo essa ideia, a empresa Cinema sempre compra dos seus fornecedores e sempre vende para seus clientes. Assim, ao nomear a operação do cliente para adquirir bilhetes, essa operação deve se chamar `venderBilhetes()` para ficar bem claro.

16

2.3. Passando argumentos

Ao realizar uma operação, esperamos algo como resposta da operação. Entretanto, antes que algo possa ser retornado, alguma informação normalmente é repassada para este objeto. Essas informações são necessárias para que o objeto possa processar (executar) as operações atribuídas a ele.

Num exemplo hipotético de um sistema para uma livraria, para que uma venda de livros possa ser concluída, provavelmente seria necessário passar para o objeto Venda as seguintes informações:

- Dados do cliente que quer fazer a compra.
- Dados do(s) livro(s) que ele quer comprar.
- Dados da forma de pagamento (como dados do cartão de crédito ou informação que será pago com dinheiro ou cheque).

Todas essas informações devem ser passadas como argumentos para a operação. Dessa forma, uma possibilidade da nomenclatura completa da operação poderia ser algo como:

Classe VendaLivros com os parâmetros necessários para a operação

Neste exemplo, observe que a operação **venderLivro** possui três argumentos:

- o primeiro refere-se as informações do cliente,
- o segundo sobre o(s) livro(s) que ele queira comprar (um ou vários) e
- o último sobre a forma de pagamento.

Note que na medida em que todas as informações da modelagem são incluídas na documentação, os diagramas começam a ficar complexos e grandes demais. A maioria das ferramentas de UML permite configurar a visibilidade do que é representado no diagrama, a fim de deixar a visão mais simples. Mas não se preocupe, as devidas configurações de atributos (como multiplicidade, tipos etc.) serão criadas no código fonte, quando solicitado pela ferramenta.

17

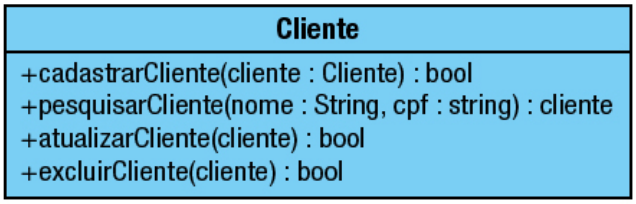
2.4. Operações de CRUD

Normalmente objetos que são mantidos em banco de dados oferecem as quatro funcionalidades relacionadas aos comandos SQL:

- **C** – **Cadastrar** (equivale ao comando Insert).
- **R** – **Report** (equivale ao Select) Selecionar e reportar.
- **U** – **Update** – Atualizar.
- **D** – **Delete** – Apagar.

Dessa forma, a grande maioria dos objetos que forem mantidos em bancos de dados terão esses quatro métodos que definem as operações de SQL em suas classes (além de outras operações mais específicas).

Exemplo:



Exemplo das quatro operações básicas de uma classe que mantém dados em bancos de dados

Cadastrar
Para essa operação passamos os dados a serem cadastrados como atributo e ela retorna um V/F para confirmar se a operação ocorreu com sucesso.

Report
Selecionar e reportar. Para essa operação passamos um parâmetro a ser pesquisado (como o nome ou o CPF de uma pessoa) e ela retorna o objeto que foi encontrado.

Update
Para essa operação passamos os dados a serem atualizados (normalmente o próprio objeto) como atributo e ela retorna um V/F para confirmar se a operação ocorreu com sucesso.

Delete
Para essa operação passamos algum dado que identifique o objeto a ser excluído (normalmente o código no banco ou o próprio objeto) e ela retorna um V/F para confirmar se a operação ocorreu com sucesso.

18

2.5. Direção do argumento

Até agora falamos que os argumentos são passados para o objeto, ou seja, eles **entram** no objeto. Essa é a direção padrão, onde as informações são identificadas pelo objeto "chamador" e enviadas para o objeto "executado". Para essa configuração padrão, dizemos que a direção é "in" (entrada).

Entretanto há outra possibilidade. Os argumentos podem ser passados vazios para o objeto e dentro do método as informações são criadas e armazenadas nos argumentos. Feito isso, o objeto "chamador" pode, então, utilizar essas informações. Dessa forma, os argumentos não entram no objeto executado, mas sim o oposto, as informações saem do objeto chamado. A ideia aqui é que ao invés de um objeto retornar apenas um valor (o que acontece quando usamos o atributo de retorno no final da chamada), ele pode retornar vários valores. Para essa configuração, mais rara, mas ainda comum em programação, declaramos a direção como"out" (saída).

A forma padrão para citar a direção é incluir as opções "in" ou"out" antes de declarar o argumento. Para o nosso exemplo anterior, a nomenclatura completa seria ajustada para:

```
venderLivro (in cliente:Pessoa, in livrosAComprar:Livros[1..*], in
pagamentoEscolhido:FormaPagamento): VendaConcluida
```

Para entender melhor a questão da direção do tipo "out", vamos pensar num exemplo hipotético. Suponha que você queira consultar todos os livros e todos os autores relacionados a uma determinada editora. Perceba que neste cenário poderia haver um objeto que retornasse dois conjuntos distintos de informação: livros e autores. Não seria possível retornar esses dois objetos como um retorno normal de um objeto, mas poderíamos utilizar então do artifício proposto (argumento do tipo "out"). Para isso, passaríamos como entrada os dados da editora, e o método retornaria os dois conjuntos de informação que solicitamos.

19

Uma suposta sintaxe para esse objeto poderia ser algo como:

```
editoraConsultar(in editora:Editora, out livrosDisponiveis:Livros [1..*],
out autores:Pessoa[1..*]): ConsultaOk
```

Observe nesse exemplo que:

- Passamos para o objeto a editora que queremos pesquisar;
- Receberemos em "livrosDisponiveis" os livros que essa editora comercializa;
- Receberemos em "autores" os autores relacionados a ela;
- Receberemos em "ConsultaOk" um verdadeiro ou falso dizendo se a operação foi concluída com sucesso.

É possível juntar os dois conceitos, utilizando o que é chamado de “in out”. Isso ocorre quando precisamos passar argumentos para método, modifica-los dentro do método, e posteriormente retorná-los.

Exemplo: para que o método abaixo possa passar, modificar e retornar dois objetos como resposta, optou-se por passa-los como referência na linha de argumentos.

```
consultarCliente (in out cliente:Cliente, in out movimentação:Movimentacao):Ok
```

20

RESUMO

Neste módulo, aprendemos que:

a) Atributos referem-se às informações que podem ser passadas e/ou recebidas para uma classe antes da execução de seus métodos.
b) Há duas formas de se definir classes: usando atributos ou passando os atributos como parâmetros dos métodos.
c) Os atributos de uma classe devem ser apenas aqueles que fazem sentido (e possuem uso) para o sistema.
d) Que os atributos podem ser de vários tipos diferentes: inteiros, fracionados, booleanos, data, caracteres e até mesmo outras classes.
e) Que uma classe pode receber vários valores no mesmo atributo, usando para isso a multiplicidade.
f) Que podemos descrever valores padrão, utilizando parênteses; e indicadores de ordenação e unicidade, utilizando as expressões {ordered} e {unique} respectivamente.
g) Que podemos trabalhar com a visibilidade dos atributos e métodos utilizando os símbolos “-“, “~”, “+” e “#”.

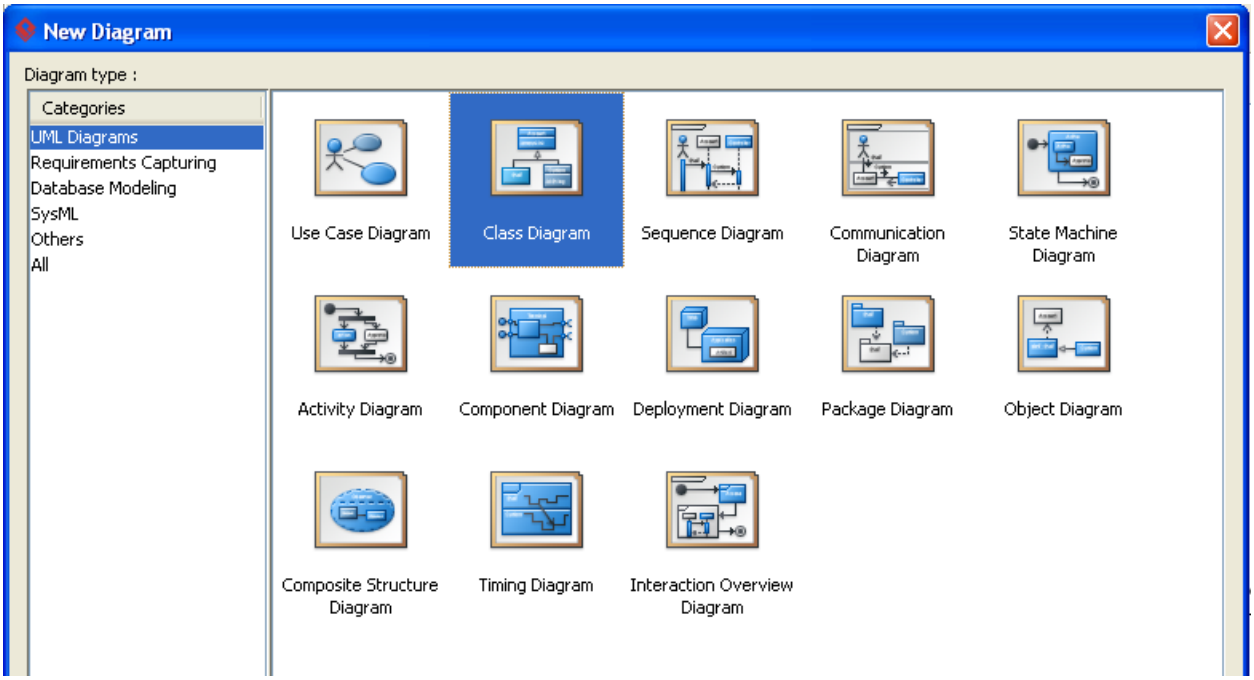
Que parâmetros podem retornar informações por referência (utilizando a sinalização ‘out’).

UNIDADE 2 – DIAGRAMAS BÁSICOS DA UML.
MÓDULO 2 – DIAGRAMA DE CLASSES (ASSOCIAÇÕES)

01

1- OBJETIVO E FUNÇÃO DE UMA ASSOCIAÇÃO

Neste módulo, continuaremos a aprender conceitos de Orientação a Objetos e diagramação UML. Especificamente, neste módulo trataremos dos tipos de relacionamentos entre as classes.



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

Um aplicativo de *software* requer um conjunto de recursos. Para usá-los você precisa descrever cada recurso usando uma definição de classe. As classes descrevem os tipos de recursos, as suas funções e as funcionalidades que elas oferecem, incluindo atributos e operações.

Porém, para coordenar a interação desses recursos, você precisa explicar como eles se comunicam. Para trocar informações entre si, eles precisam estar cientes um do outro. Assim como as pessoas usam vários meios para se comunicar, como telefonemas, correio ou e-mail, os objetos também precisam definir um canal de informação, ou seja, um tipo de relacionamento. A UML provê três tipos diferentes de relações:

- associação,
- generalização,
- dependência.

Vamos rever esses três conceitos.

02

Uma **associação** é uma relação semântica entre dois elementos do modelo. Em um diagrama de classes, uma associação define as regras que se aplicam às relações entre os objetos definidos pelas classes participantes.

Cada associação inclui as regras para estabelecer e manter a integridade dos relacionamentos, como os relacionamentos são criados e usados pelo aplicativo. A associação é usada para unir, temporariamente, objetos distintos e completos para um uso particular.

O mesmo conceito pode ser refinado para explicar os objetos compostos de outros objetos. Este tipo de associação, chamada **agregação**, facilita muito o uso de configurações complexas de objetos. A agregação também pode ser refinada para modelar conjuntos onde as partes têm uma associação mais restrita com a montagem. Neste refinamento de agregação, chamado **composição**, a vida das peças do conjunto depende inteiramente de sua participação no sistema. A associação e a composição são usadas para objetos cuja união forma outro objeto, resultado da soma das partes que o compõe (exemplo: um motor é formado pela associação de várias peças).

A relação de **dependência** não requer uma comunicação direta. Nessa relação, um objeto se baseia no fato de que outro objeto existe e está fazendo, ou fez, o seu trabalho.

Por exemplo, uma transação de negócios pode depender do sistema de segurança para garantir que nenhuma pessoa não autorizada pode acessar a transação comercial. A implementação desta relação é tratada no fluxo de trabalho, a tecnologia, ou outras opções de design no aplicativo.

03

A generalização é usada no contexto de herança.

Um relacionamento de **generalização** é muito diferente de uma associação. Na verdade, a generalização não requer qualquer das regras estabelecidas para as associações. Em vez disso, a generalização define a organização de informações sobre os objetos que compartilham o mesmo significado semântico, mas que variam em suas características individuais.

Por exemplo, o termo "carro" refere-se a uma ampla variedade de veículos. Para um fabricante de automóveis, é importante diferenciar os carros com base em suas características únicas. Então, um fabricante pode estabelecer distinções entre carros de passeio, SUVs, caminhões, e assim por diante, com base em suas características diferenciadoras.

Este módulo abrange todos os três tipos de relacionamentos de associação (incluindo agregação e composição), dependência e de generalização e fornece exemplos dos recursos usados para definir completamente cada tipo de relacionamento.

04

A finalidade de uma associação é estabelecer a razão pela qual duas classes de objetos precisam saber uma sobre a outra, incluindo as normas que regem o relacionamento.

Por exemplo, um espetáculo teatral pode ser realizado em um teatro. O espetáculo teatral precisa saber qual será o teatro escolhido, e o teatro escolhido precisa saber que tipo de espetáculo está acontecendo dentro dele. Estas são duas perspectivas sobre a mesma associação.

Algumas associações, como o espetáculo e o teatro, são muito simples. Outras associações podem ser muitíssimo mais complexas. Por exemplo, uma pessoa pode ter muitas razões diferentes para saber sobre um carro: ele pode possuir um carro, dirigir um carro, vender o carro, ou lavar um carro, e há muitas outras possibilidades. Cada razão define uma associação: uma razão pela qual um tipo de objeto precisa saber sobre o outro tipo de objeto.

Independentemente da complexidade, cada associação tem de definir suas próprias regras para estabelecer e manter a integridade do relacionamento. Estas regras incluem:

- Uma maneira de identificar com exclusividade e significativamente a associação.
- O número de objetos que podem participar na associação.
- As restrições sobre os objetos que estão autorizados a participar da associação.
- O papel que cada tipo de objeto desempenha quando participa na associação.
- Um meio para identificar quais objetos podem ter acesso através da associação.

Informações sobre a associação, como: quando começou, os termos da associação, quando terminou, e seu status atual.

Juntas, essas regras definem como os objetos em uma aplicação podem entrar em contato e colaborar uns com os outros.

05

2- MODELANDO UMA ASSOCIAÇÃO

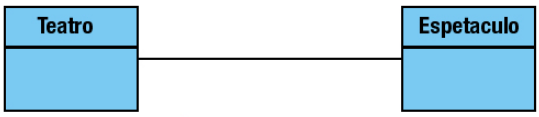
A modelagem de uma associação começa por identificar as classes participantes. Nos primeiros exemplos discutiremos o uso de duas classes, chamada de associação binária, uma vez que este é o tipo mais comum de associação. Nós vamos cobrir mais adiante associações que utilizam mais de duas classes participantes, chamadas associações “n-árias”.

2.1 - Notação de associação binária

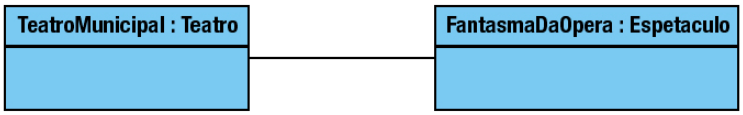
Em um diagrama de classes, uma associação binária documenta as regras que regem uma relação entre duas classes de objetos.

A associação é uma regra que explica o que é permitido. Em um diagrama de objeto, uma relação real é chamada de “link”. Uma associação é uma regra. Um link é um fato.

A figura abaixo mostra uma associação que liga a classe Teatro à classe Espetáculo. A classe Teatro define o que é um objeto teatro e o que ele pode fazer. A classe Espetáculo define o que é um espetáculo e o que ele pode fazer. A associação define um único tipo de relação que pode ser estabelecida entre os diferentes locais e espetáculos, uma razão pela qual esses tipos de objetos precisam se comunicar: um Espetáculo precisa acontecer em um Teatro, como exemplo, o espetáculo “Fantasma da Ópera” será apresentado no teatro “Teatro Municipal”.



Associação binária entre duas classes.



Link binário entre dois objetos.

Lembre-se: **Associação** é um tipo de relacionamento. Um **link** é uma instância ou implementação de uma associação.

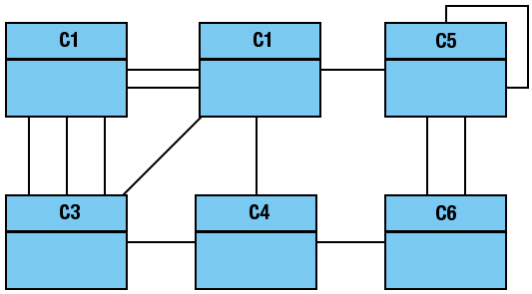
O objeto “`TeatroMunicipal: Teatro`” na figura acima descreve um objeto Teatro de nome TeatroMunicipal que esteja em conformidade com a definição de classe Teatro. O objeto “`FantasmaDaOpera: Espetaculo`” descreve um objeto Espetaculo chamado FastasmaDaOpera que está em conformidade com a definição da classe Espetaculo. O link define uma única relação entre o teatro TeatroMunicipal e o espetáculo FantasmaDaOpera.

06

No entanto, uma definição completa de associação é constituída por três partes: uma linha de ligação entre as classes e duas extremidades de associação. A linha de associação e seu respectivo nome definem a identidade e a finalidade da relação. As terminações da associação (extremidades) definem as regras sobre como os objetos das classes em cada extremidade podem participar. Essas extremidades possuem atributos como multiplicidade, restrições e papéis.

A associação diz ser binária porque ela liga duas classes somente. É possível que um diagrama contemple centenas de classes, todas elas com associações binárias. Quando a associação liga duas classes é chamada de binária, não importando quantas classes existam no modelo nem quantas associações existam na classe.

O exemplo hipotético abaixo, por mais estranho que possa parecer, apresenta apenas associações binárias:



Exemplo de várias classes associadas binariamente

07

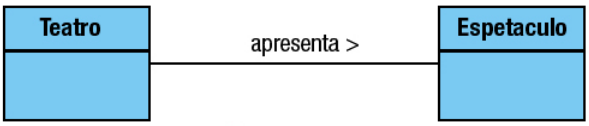
2.2 Nomeando a associação

O nome de uma associação exprime ao leitor a intenção do relacionamento. Cada associação representa um investimento de tempo e esforço para estabelecer a relação, preservar a sua integridade e garantir o seu uso adequado. Quando o nome estiver vago, introduz confusão e erros, aumentando custos e reduzindo a eficácia do processo de modelagem.

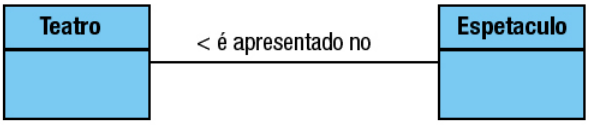
A maneira usual para citar uma associação é com um verbo no **presente do indicativo** (utilizando a terceira pessoa do singular). A figura a seguir mostra a mesma associação modelada com dois nomes diferentes:

a) usando o verbo na **voz ativa** "apresenta" (no sentido de que um teatro apresenta um espetáculo), e
b) usando a **voz passiva** "é apresentado no" (no sentido de que um espetáculo é apresentado em um teatro).

A posição do nome não é crítica. O nome só precisa estar próximo ao meio da linha, entre as duas classes. Mantenha-o perto o suficiente da linha de associação, de modo que o leitor do diagrama entenda claramente o que o nome se refere. Além disso, mantenha o nome distante das extremidades da associação, pois lá você estará adicionando um monte de informações. Apesar de o nome do relacionamento não aparecer no código fonte, a documentação dele aumenta o entendimento do objetivo do relacionamento entre as classes.



Nome da associação utilizando o verbo na voz ativa.



Nome da associação utilizando o verbo na voz passiva.

Uma boa prática utilizada pelas equipes de TI é evitar o uso da voz passiva. Não é errado, mas evite se puder.

08

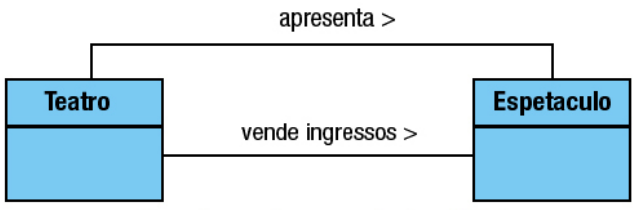
A **figura** anterior também introduziu um dispositivo útil chamado **indicador direcional**. Pelo menos na cultura ocidental, tendemos a ler da esquerda para a direita, de modo que quando o desenho nos obriga a rotular algo cujo sentido seja da direita para esquerda, precisamos de uma maneira de garantir o entendimento correto ao leitor. A solução para este problema é muito simples: coloque um indicador direcional, como uma seta ou os símbolos de “<” e “>”, ao lado do nome da associação para mostrar ao leitor como interpretar o significado do nome da associação.

Nas figuras anteriores, um indicador de direção foi colocado ao lado de ambos os nomes das associações para maior clareza. Muitas vezes o indicador de direção só é usado quando a ausência dele pode causar uma má interpretação. Em suma, garanta sempre a boa qualidade da documentação.

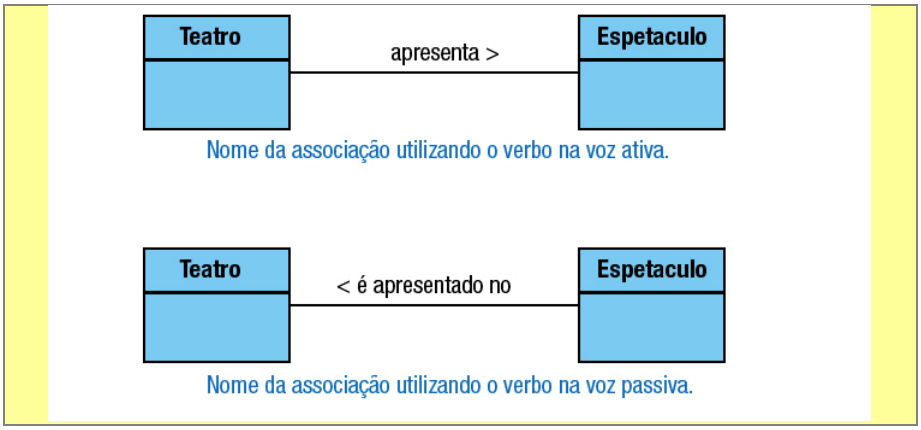
Em algumas ferramentas o indicador direcional é simplesmente uma característica de cada nome da associação que pode ser ligado ou desligado. Em outras ferramentas que você definir a direção dentro da especificação associação. Em outras ferramentas ainda, você pode ter que realmente desenhar o símbolo.

O nome da associação torna-se ainda mais importante quando duas classes têm mais de uma razão para colaborar. Poderíamos imaginar, por exemplo, que um Teatro poderia apenas vender ingressos de um determinado Espetáculo que estaria sendo apresentado e outro Teatro. Em outros momentos, um Teatro pode ser tanto o anfitrião e patrocinador. A figura abaixo usa duas associações para representar

o fato de que as regras para a realização de um Espetáculo são diferentes das regras para vender ingressos de um Espetáculo. As duas associações precisam ser definidas e mantidas separadamente.



Exemplo com duas associações distintas.
Cada associação representa um conjunto separado de regras.



09

2.3 Terminações de uma associação

Para definir esses papéis, a UML trata cada extremidade da associação como uma entidade separada e distinta, com suas próprias regras. Ou seja, a participação de um Teatro na associação "apresenta >" é diferente de participação do Espetáculo na associação "apresenta >".

Cada extremidade da associação precisa explicar, por exemplo, qual o papel que o objeto nesse final desempenha na relação, como muitos objetos desse tipo podem participar no relacionamento, se podem existir muitos objetos participantes e se eles têm que ser apresentados em alguma ordem.

A extremidade da associação também especifica se há alguma característica do objeto que poderia ser usada para acessá-lo, e se o objeto em uma das extremidades pode até mesmo acessar o(s) objeto(s) na outra extremidade.

Cada extremidade da associação inclui algumas ou todas estas **características**:

- Papéis.
- Multiplicidade.
- Ordenação.

- Restrições.
- Qualificadores.
- Navegabilidade.
- Especificador de Interface.
- Visibilidade.
- Capacidade de mudar (ou ser constante).

Os subitens seguintes abordam os principais recursos e suas notações.

10

2.3.1 Papéis (ou funções)

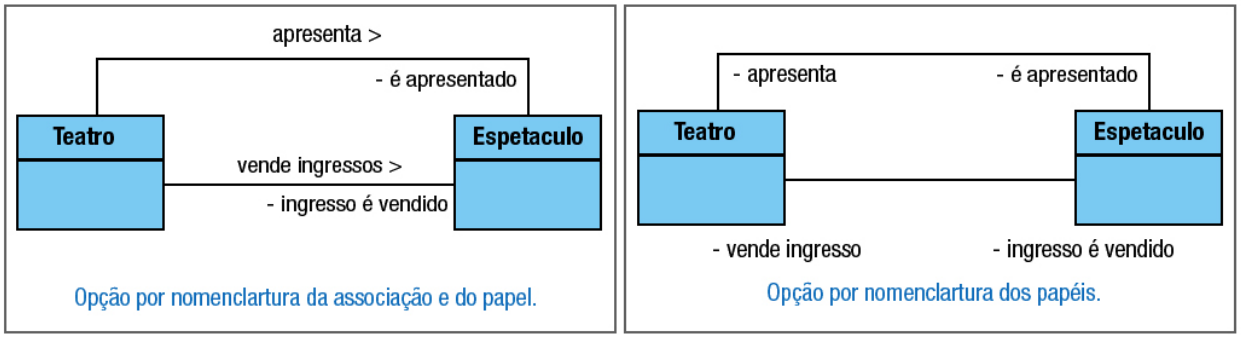
Um nome de papel (ou função) explica como um objeto participa do relacionamento. Ao contrário do nome da associação, o nome da função no final da associação pode gerar o código. Cada objeto tem de manter uma referência para o objeto ou objetos associados. A referência é realizada em um valor de atributo dentro do objeto. Quando só existe uma associação então existe apenas um atributo mantendo uma referência. No código gerado, o atributo será nomeado com o nome da função do objeto referenciado.

Por exemplo, o espetáculo Fantasma da Ópera é apresentado no Teatro Municipal. O objeto "`FantasmaDaOpera`" contém um atributo com uma referência para o objeto "`TeatroMunicipal`". Até agora, não importa que nome daremos ao atributo, porque há apenas uma referência. É possível um espetáculo ser apresentado em um único teatro, mas ter seus ingressos à venda em vários outros teatros. Isto significa que existem duas referências possíveis para um Teatro no mesmo objeto Espetáculo.

Agora precisamos ser capazes de dizer a diferença entre **apresentar** e **vender** ingressos. Para representar duas referências distintas, os nomes de papéis são muitas vezes utilizados para nomear os atributos que mantêm as referências.

11

Os nomes dos papéis descrevem a associação em termos de como cada tipo de objeto (Teatro e Espetáculo, no nosso exemplo) participam da associação. Já que tanto o nome da associação quanto o nome do papel ajudam a descrever a natureza da relação, os nomes dos papéis podem ser usados com ou no lugar do nome da associação. A figura abaixo apresenta alternativas para as relações de Teatro e de Espetáculo. O modelo da esquerda usa um nome de associação e papéis apenas nas extremidades do Espetáculo. O modelo da direita usa apenas os papéis em ambas as extremidades.



Coloque o nome do papel no final da linha de associação e ao lado da classe que ele descreve. A posição exata não é crítica. Basta mantê-lo perto do fim da associação, perto da classe. Existem outros elementos do modelo que terão de caber no final de associação, por isso não se surpreenda se você tiver que mover as coisas ao redor do diagrama para facilitar a leitura.

12

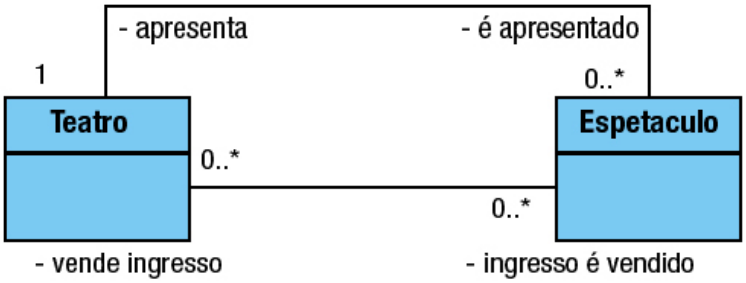
2.3.2 Visibilidade

A figura anterior ilustra também a notação de visibilidade para nomes de funções. O termo “visibilidade” já foi tratado no módulo anterior, e aqui nós estamos usando os mesmos conceitos, indicando que a visibilidade refere-se a quem pode acessar o nome da função. Ao observar a figura anterior, a classe Teatro tem uma referência a um nome de função "- apresenta". O sinal de menos (-) na frente do nome da função refere-se ao símbolo UML para a visibilidade privada (Lembra-se dos símbolos “+, -, ~e #”?). Isto significa que um Espetáculo contém um atributo privado, a referência ao Teatro que desempenha o papel de anfitrião do Espetáculo. Se você quiser ter acesso ao valor do atributo, você precisará solicitá-lo por meio de uma operação que tenha uma visibilidade de algo diferente de acesso privado.

2.3.3 Multiplicidade

A multiplicidade refere-se ao número válido de objetos que podem estar relacionados às regras da associação.

A multiplicidade pode expressar uma gama de valores, um valor específico, um intervalo sem limite, ou um conjunto de valores discretos. Para uma explicação completa da multiplicidade, consulte o módulo anterior. No exemplo abaixo podemos ver que um teatro pode apresentar nenhum ou vários espetáculos, que um espetáculo pode ser apresentado em apenas um teatro, que um teatro pode vender ingressos de nenhum ou vários espetáculos, e que os ingressos de um determinado espetáculo podem estar à venda em nenhum ou vários teatros.



Representação da multiplicidade entre as classes.

13

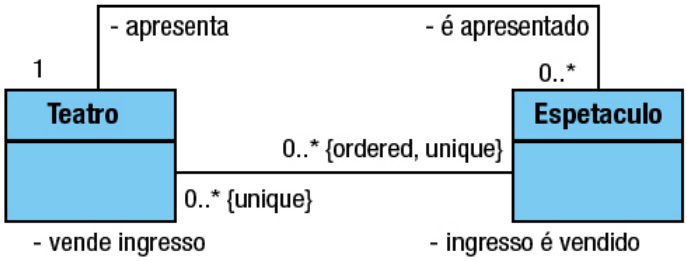
2.3.4 Ordenação e Unicidade

Lembrando que a multiplicidade permite mais de um objeto, a UML permite declarar que os vários objetos estão ordenados (alfabeticamente, por exemplo).

A ordenação é declarada por um valor booleano (v ou f) que indica que os objetos do grupo precisam ser organizados em uma sequência.

A representação da ordenação já foi discutida em momento anterior. Da mesma forma, ela aqui é representada pela palavra-chave "{ordened}" no final da associação. O mesmo se refere à unicidade.

A unicidade indica que cada item deve ser único (não pode ser repetido). Utilizamos a palavra-chave "{unique}" para representar a unicidade.



Representação de ordenação e unicidade

A figura acima representa o fato de que, quando um Teatro recebe um número de Espetáculos, a lista de Espetáculos necessita de ser dispostas numa sequência especificada. Também define que não pode haver espetáculos com mesmo nome num mesmo teatro.

14

2.3.5 Restrições

Uma restrição define uma regra restritiva que necessita de ser aplicada sobre um elemento de modelagem para assegurar a sua integridade durante a vida útil do sistema. Por exemplo, num determinado contexto de sistema, poderia haver uma restrição quanto ao custo mínimo de um ingresso. Para isso, utilizamos o mesmo espaço delimitado para ordenação e unicidade para informar essa condição. Um exemplo hipotético seria de que o valor mínimo do ingresso fosse maior que R$ 100,00. Dessa forma, a representação completa da restrição seria ajustada para:

```
{ordered, unique, valorMinimoIngresso > 100}
```

Outra possibilidade seria indicar qual atributo define a ordenação, para isso, utilizamos a expressão “ordered by”. Um exemplo hipotético seria dizer que o espetáculo é ordenado pelo seu nome. Dessa forma, a representação completa da restrição seria ajustada para:

```
{ordered, unique, valorMinimoIngresso > 100, ordered by nomeEspetaculo}
```

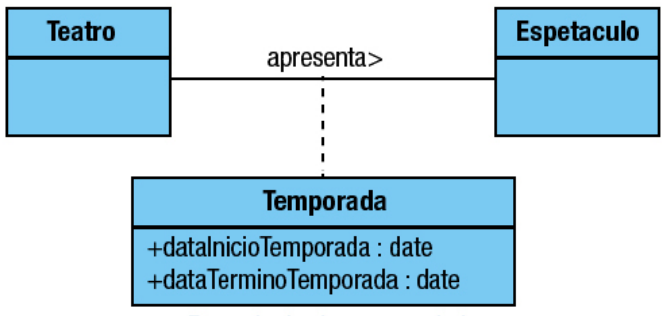
15

3- Modelando uma classe de associação

Muitas vezes, ao associar duas ou mais classes, precisamos de informações e funcionalidades adicionais às que já existem na natureza particular de cada classe. Por exemplo, ao associar um espetáculo a um teatro, em nenhuma dessas duas classes teríamos informações de quando começa e termina a temporada desse espetáculo nesse teatro.

Para resolver essa necessidade, inserimos uma nova classe entre as duas para referenciar novas informações resultantes do relacionamento inicial.

Veja como seria a representação desse exemplo citado:



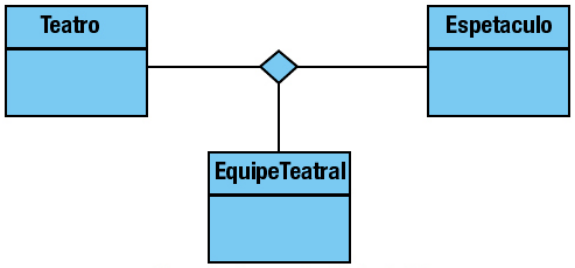
Exemplo de classe associativa

16

4- Modelando associações "n-árias"

É possível em alguns casos a necessidade de associar mais de duas classes ao mesmo tempo. Esse tipo de associação é chamada de "n-ária". Para representar uma associação desse tipo, utilizamos um losango para unir as classes associadas.

Um exemplo de como representar esse tipo de associação é mostrado abaixo:



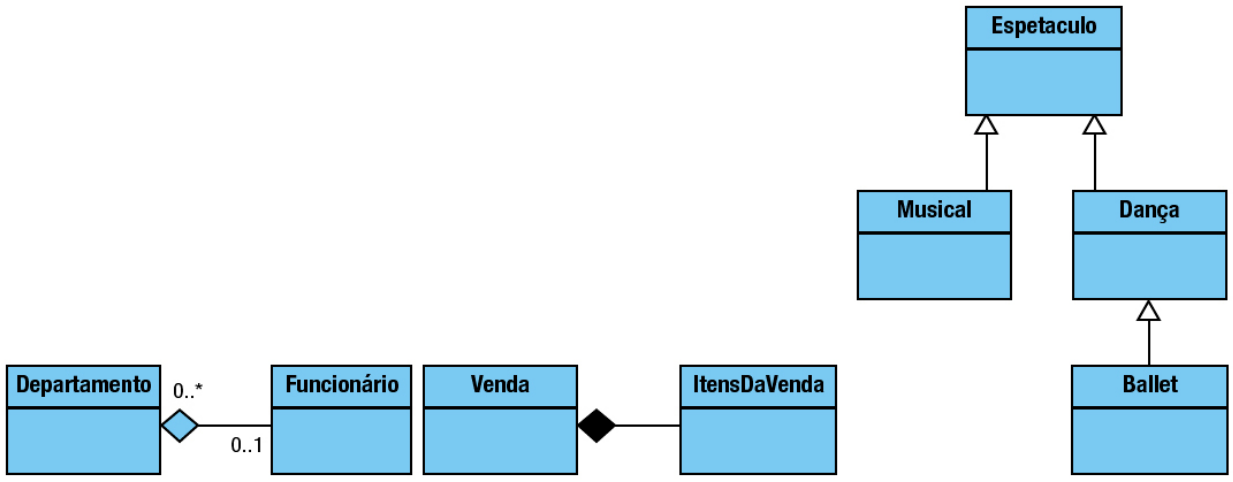
Exemplo de associação "n-ária".

17

5- Modelando a agregação, composição e generalização

Os elementos que descrevem a agregação, composição e generalização já foram tratadas na unidade 01, módulo 04.

Reveja esse módulo caso você ainda tenha dúvida quando aos símbolos abaixo:



Exemplo de agregação, composição e generalização.

18

RESUMO

Neste módulo, aprendemos que:

a) Associações e links: tipo de ligação mais simples de relacionamento que suporta a comunicação entre objetos. Dois objetos estão conscientes um do outro porque eles possuem referências entre si. Uma associação define um tipo de ligação que pode existir entre os tipos de objetos. Um link é um exemplo de uma associação da mesma forma que um objeto é uma instância de uma classe.

b) Multiplicidade: cada extremidade de uma associação deve definir o número de objetos que podem participar na associação. A multiplicidade pode ser expressa como um intervalo (mínimo e máximo) ou uma lista de valores (valor 1, valor 2, ...). Um asterisco (*) usado por si só designa zero ou mais objetos. Um asterisco usado como o valor máximo numa gama significa que não há limite superior.

c) Restrições: define uma regra que deve ser aplicada para a associação para ser válida. A restrição é colocada no final de uma associação de ditar as condições que devem ser satisfeitas antes que os objetos possam participar do relacionamento. As restrições estão entre chaves ({}).

d) Os nomes de função: explica como um objeto participa de um link. O nome do papel é colocado na extremidade de uma associação próxima com a classe que desempenha o papel.

e) Classe de associação: encapsula informações sobre uma associação; por exemplo, quando a relação começou, terminou, termos e condições, e status. Uma classe de associação é necessária porque completa uma associação com informações e/ou operações. A classe de associação está ligada a uma associação por uma linha tracejada.

f) Agregação: tipo de associação em que uma classe define os objetos que participam como peças em uma montagem ou configuração e objetos de outra classe que representam o todo, o conjunto inteiro.

g) Composição: tipo de agregação em que um objeto membro só pode ser um objeto parte de no máximo um objeto agregador (e tem seu ciclo de vida controlado por ele).

h) Generalização, superclasse e subclasse: a generalização fornece um meio para organizar as semelhanças e diferenças entre um conjunto de objetos que compartilham a mesma finalidade. A generalização é um relacionamento entre as classes em que uma classe chamada de superclasse, contém recursos compartilhados por todos os objetos da classe, e as outras classes, chamadas de subclasses, contêm apenas as características que as tornam um subconjunto de objetos diferente de todos os outros objetos representados pela

superclasse. A generalização é modelada como uma linha entre as classes com um triângulo fechado e oco no final perto da superclasse.

i) Especialização: descreve o processo de identificação das características que tornam objetos únicos dentro de uma superclasse. "Uma especialização" refere-se a uma subclasse indivíduo.

j) Hereditariedade: a herança descreve o fato de uma subclasse ter acesso aos recursos de uma superclasse no momento em que ela é usada para instanciar um objeto. A herança pode ser comprometida pela utilização de restrições de visibilidade.

UNIDADE 2 – DIAGRAMAS BÁSICOS DA UML.
MÓDULO 3 – DIAGRAMA DE SEQUÊNCIA

01

1- MODELANDO UM DIAGRAMA DE SEQUÊNCIA

Olá, iniciamos agora mais uma etapa do nosso estudo, em que trataremos dos diagramas de sequência.

Já aprendemos até agora que os diagramas UML baseiam-se em três temas gerais:

- diagramas estruturais,
- diagramas comportamentais e
- diagramas de gerenciamento de modelo.

Os diagramas estruturais (classe, objeto, componente, e assim por diante) representam a forma como os objetos são definidos e como eles estão relacionados entre si. Eles **não** representam como os objetos se comportam quando você os executa em uma aplicação. Em contraste, os diagramas comportamentais representam a forma como os objetos funcionam usando a estrutura já definida nos outros esquemas. Este ponto de vista dinâmico contém diagramas projetados especificamente para modelar como os objetos se comunicam, a fim de realizar tarefas dentro do funcionamento do sistema. Eles podem representar como o sistema responde às ações dos usuários, como ele mantém a integridade interna, como os dados são movidos entre a interação do usuário e o banco de dados, como os objetos são criados e manipulados, e muito mais.

Tendo em vista que o comportamento do sistema pode ser grande e complexo, os diagramas de comportamento tendem a olhar para pedaços pequenos, discretos do sistema, tais como cenários ou operações individuais.

Você não precisa representar todos os cenários e comportamentos possíveis do sistema em diagramas comportamentais, pois nem todos os comportamentos são complexos o suficiente para justificar uma explicação visual da comunicação necessária para realizá-los.

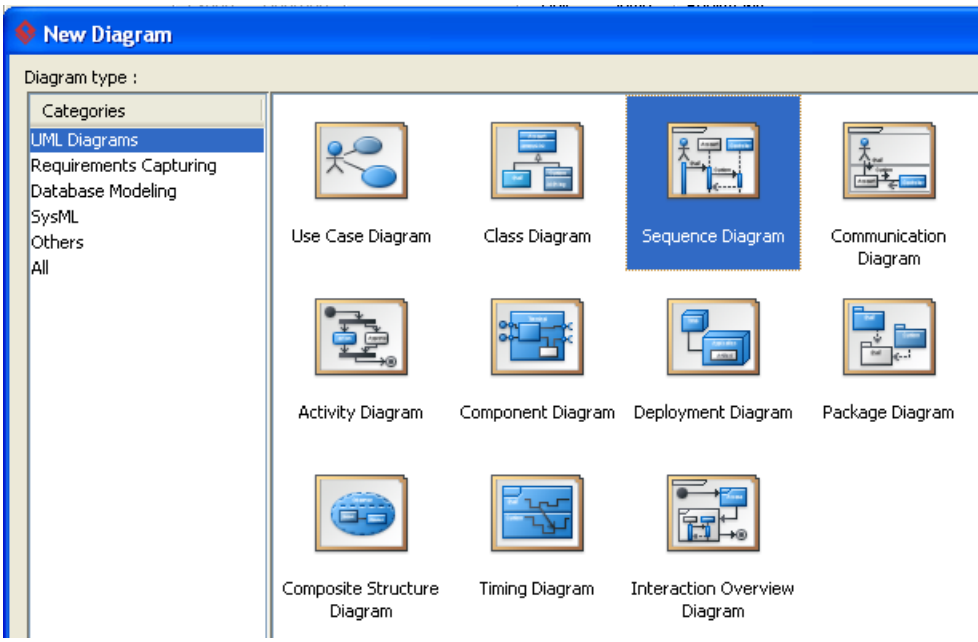
Mesmo assim, o diagrama de classes e os diversos diagramas comportamentais são os mais usados, justamente porque eles revelam as características específicas necessárias para gerar o código.

02

Há uma pequena alteração na nomenclatura dos diagramas e a criação de novos diagramas entre as versões 1.4 e 2.0 da UML. Veja o resumo no quadro a seguir:

UML 1.4	UML 2.0
Diagrama de sequência	Diagrama de sequência (não mudou)
Diagrama de colaboração	Diagrama de Comunicação
Diagrama de estados	Máquina de estados
(Não existia)	Diagrama de temporalidade
(Não existia)	

Este módulo trata exclusivamente do **diagrama de sequência**.



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

Os diagramas de sequência e de colaboração são mais frequentemente utilizados para ilustrar as interações entre objetos. Como tal, ambos modelam os objetos e mensagens entre os objetos.

03

O diagrama de sequência usa uma visualização orientada pelo tempo. Para cada objeto do diagrama, é utilizado um conjunto de ícones de objetos e prazos associados, chamado de linha de tempo do objeto. Já o diagrama de colaboração apresenta uma visualização orientada a estrutura.

Você pode pensar que é estranho que a UML tenha dois diagramas que fazem a mesma coisa. De certa forma, você está certo. A razão é que eles vêm de diferentes metodologias e cada um oferece uma perspectiva ligeiramente diferente, que pode ser bastante valiosa (os dois diagramas serão comparados mais adiante).

A característica comum de ambos os diagramas, é a capacidade de representar **interações**. Interações nos mostram como os objetos enviam mensagens uns aos outros. Quando um objeto quer enviar uma mensagem a outro objeto, o segundo objeto tem que ter uma forma para receber a mensagem. A mensagem tem que corresponder a uma interface fornecida pelo segundo objeto. É como receber um telefonema. A pessoa que recebe a chamada tem de possuir um aparelho telefônico com microfone e alto-falante, a fim de receber a chamada. O aparelho telefônico é a interface. Assim, as interações, as mensagens enviadas entre os objetos, revelam os requisitos de interface. A interface é literalmente uma operação.

04

Este emparelhamento de mensagens e interfaces é útil para construir e testar o seu modelo por duas razões:

1. Ele pode apontar para os elementos existentes no modelo. Se um objeto precisa enviar uma mensagem para um objeto de destino, você deve verificar se o segundo objeto já fornece a interface necessária. Ao usar uma ferramenta de modelagem, você faz isso analisando a lista das interfaces definidas para o objeto de destino e, em seguida, escolhendo a interface correta para aquela interação. Em uma ferramenta de modelagem, você pode fazer isso a partir do diagrama de sequência ou de colaboração, ou verificando a especificação da classe correspondente ao objeto no diagrama de classes. **Saiba+**

2. Ele pode auxiliar na descoberta da necessidade de novos elementos do modelo. Se você achar que precisa enviar uma mensagem para um objeto de destino, mas esse objeto não tem uma interface correspondente, então você descobriu um novo requisito de interface. Da mesma forma, se você ainda não tem um objeto de destino apropriado para assumir a responsabilidade de receber e responder à mensagem, então você descobriu um requisito para um novo tipo de objeto, uma nova definição de classe.

Os diagramas de sequência e colaboração são baseados em requisitos de mensagens. Requisitos de mensagens vêm do fato de que os objetos são necessários para realizar comportamentos. Então, onde você encontra os comportamentos que você precisa para modelar? Em muitos projetos, os casos de uso são criados em primeiro lugar porque eles modelam o comportamento do sistema esperado pelos usuários. Esses comportamentos fornecem a base para a construção de cenários. Cada cenário descreve em forma de texto como o sistema deve se comportar sob um conjunto específico de circunstâncias, tais como o cliente compra um ingresso para o cinema na máquina de venda automática. Os diagramas de sequência e de colaboração fornecem um caminho a partir das descrições textuais de comportamentos nos cenários para as operações (interfaces) necessárias pelo diagrama de classe, a fim de gerar o código.

Saiba+
Esta capacidade de escolher as interfaces existentes faz a adição de novos recursos ser muito rápida e fácil, facilitando o trabalho de desenvolvimento em fases subsequentes do projeto e atividades de manutenção posteriores.

05

2- MODELANDO UMA LINHA DE VIDA DE UM OBJETO

Enquanto as classes definem os tipos de comportamentos que os objetos podem executar, todo o comportamento no sistema orientado a objetos é feito por objetos, e não por classes. Os objetos assumem a responsabilidade de edição e armazenamento de dados, respondendo a consultas, protegendo o sistema, e muito mais. Os objetos trabalham em conjunto para executar essas tarefas, comunicando uns com os outros. Examinando como objetos específicos trabalham sob circunstâncias específicas revela a natureza exata da comunicação.

Consequentemente, diagramas de sequência (e colaboração) são frequentemente modelados no nível do objeto, em vez de o nível de classe. Esta abordagem também suporta cenários em que vários objetos da mesma classe trabalham juntos como, por exemplo, quando um objeto empregado colabora com outro objeto empregado.

Neste cenário, também é possível utilizar o diagrama de sequência para modelar fatos nas formas de dados de teste e exemplos. O diagrama de sequência usa dois elementos fundamentais:

- a notação de **linha de vida** dos objetos e
- mensagens ou estímulos.

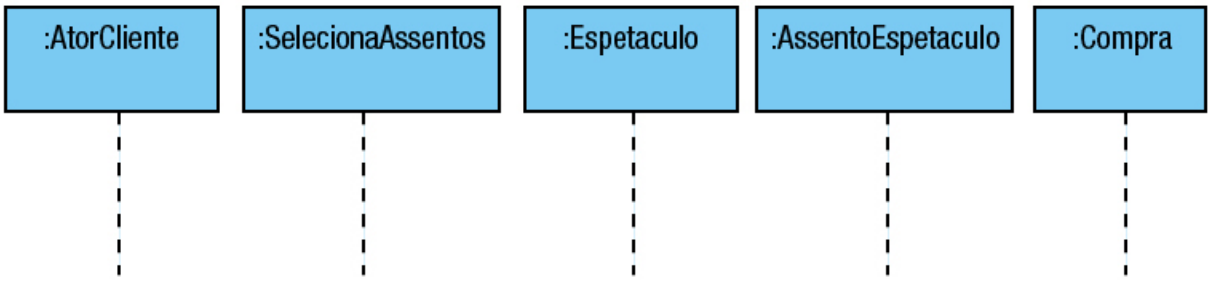
Tenha em mente que a **linha de vida** representa o momento inicial onde o objeto é criado (na parte superior da linha) até o momento em que o objeto é finalizado (na parte inferior). Em situações intermediárias, o objeto pode ser cancelado e assim ter seu ciclo de vida encerrado.

Linha de vida
O termo linha de vida vem do termo em inglês “Lifeline”. Essa expressão foi traduzida para o português de diversas formas. É possível que você encontre em outras literaturas sinônimos do termo “linha de vida”, como: Linha de fluxo, ciclo de vida, sequenciamento, linha de sequência, ciclo de vida e outros.

06

A notação de linha de vida do objeto combina o ícone do objeto e uma espécie de cronograma. A notação do objeto é o mesmo usado no diagrama de objetos: um retângulo que contém o nome do objeto. Lembrando que a forma tradicional de representar a nomenclatura de um objeto é `objeto:classe`, podemos suprimir o nome do objeto, criando uma notação anônima do tipo `:classe`. Essa notação é muito útil para criar conceitos genéricos de um mesmo cenário.

A figura abaixo esboça um cenário em que um cliente seleciona lugares para comprar ingressos em uma apresentação de teatro. Veja como os cinco objetos participantes estão alinhados na parte superior do diagrama. Este é o local padrão para todos os objetos em um diagrama de sequência. O posicionamento de um objeto na parte superior do diagrama indica que o objeto já existe desde o início do cenário. Isto implica que o posicionamento de um objeto em qualquer outro local mais abaixo indica a criação do objeto durante o cenário.



Exemplo de cinco objetos com suas linhas de vida

A notação do diagrama contempla o **retângulo** para representar o nome do objeto e a **linha pontilhada** vertical para representar a respectiva linha de vida. O período de tempo representado pela linha de tempo depende do comprimento da interação que está modelando, portanto, as linhas pontilhadas podem ser curtas ou bem compridas.

07

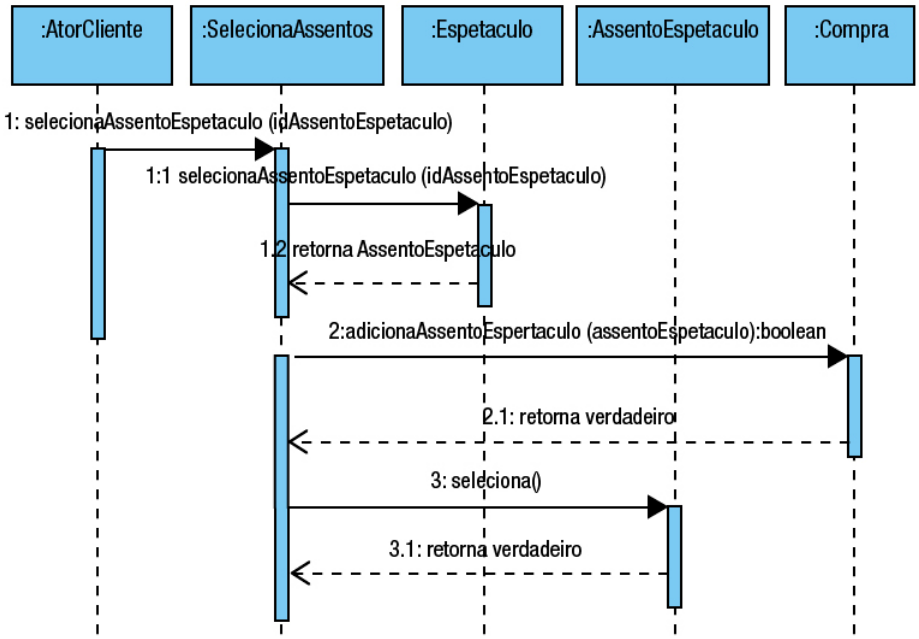
3- MODELAGEM DE UMA MENSAGEM OU ESTÍMULO

Uma **mensagem** é a especificação formal de um estímulo. Uma mensagem representa algum tipo de comunicação entre objetos.

Como tal, ela pode invocar uma operação, levantar um sinal, criar/destruir objetos de destino. Um estímulo é uma instância de uma mensagem. A mensagem explica os papéis do remetente e do receptor do estímulo, bem como o procedimento que gera o estímulo. Para fins práticos, podemos pensar em uma mensagem e um estímulo como a mesma coisa, isto é, como uma unidade de comunicação entre objetos, por isso usaremos somente o termo **mensagem** neste momento (mais adiante apresentaremos as diferenças entre os dois termos).

Uma mensagem é modelada como uma seta. O tipo de seta descreve visualmente o tipo de mensagem. O tipo mais comum de seta é uma linha sólida com uma seta preenchida, o que representa uma mensagem que requer uma resposta, chamada de **mensagem simples** ou **síncrona**.

A seta tracejada com uma ponta de seta em forma de "→" e "←" representa uma resposta ou retorno. As mensagens são colocadas na horizontal entre as linhas de vida do objeto, conforme mostrado na figura abaixo. O posicionamento horizontal indica que a transmissão da mensagem é considerada instantânea, isto é, a mensagem é enviada e recebida no mesmo instante. A posição vertical relativa nas linhas de tempo representa a ordem na qual as mensagens devem acontecer. Este padrão significa que você pode ler o diagrama do começo ao fim, lendo as mensagens de cima para baixo.



Fique Atento!

Exemplo de mensagens colocadas sobre as linhas de vida dos objetos para modelar a interações entre os objetos

A colocação das setas na linha do tempo não implica qualquer medição de tempo específico (horas, minutos ou segundo). O que é representado é apenas a ordem de sequência entre elas.

08

A interpretação da leitura do **diagrama anterior** é a seguinte:

1. O cliente escolhe o assento do espetáculo que ele quer comprar, provavelmente a partir de uma interface de usuário, passando o identificador do assento específico para aquele espetáculo para um objeto de controle, uma instância de SelecionaAssentos. O objeto de controle é, basicamente, um manipulador de eventos nesse cenário.

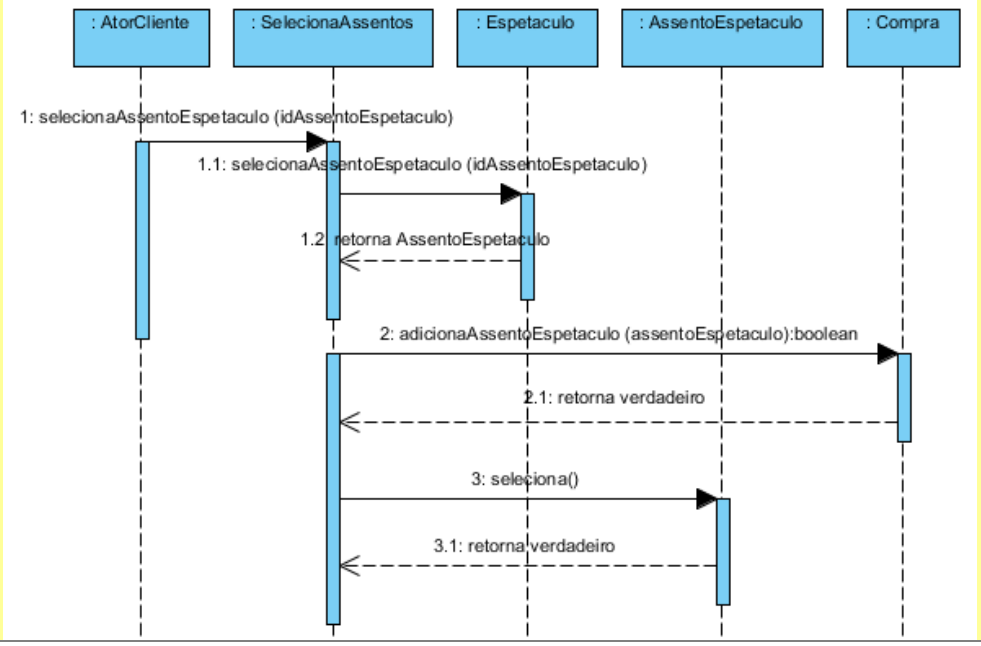
2. O objeto de controle, SelecionaAssentos, pede ao objeto Espetaculo uma referência para o assento do espetáculo que coincide com o identificador fornecido.

3. O objeto Espetaculo retorna uma referência a um objeto AssentoEspetaculo baseado no idAssentoEspetaculo passado para ele.
4. O objeto SelecionaAssentos informa o objeto Compra para adicionar o assento selecionado no pedido de compra do cliente.
5. O objeto Compra retorna um valor booleano Verdadeiro dizendo que tudo funcionou corretamente.
6. O objeto de controle SelecionaAssentos, então, diz para o objeto AssentoEspetaculo que aquele assento foi selecionado com sucesso. Isso faz com que aquele assento mude seu status de "disponível" para "vendido".
7. O objeto AssentoEspetaculo retorna uma mensagem de que tudo ocorreu bem, ou seja, um ok.

Construir o diagrama de sequência é mais fácil se você tiver concluído, pelo menos, um primeiro esboço o modelo de caso de uso (o diagrama e as descrições de casos de uso associadas) e o diagrama de classes. A partir desses dois recursos, você pode encontrar conjuntos de interações (cenários) e um conjunto de objetos candidatos a assumir a responsabilidade por essas interações.

Os números de sequência no início de cada uma das mensagens são opcionais, mas eles são muito úteis quando você precisa discutir o diagrama ou fazer alterações. Os números também são úteis quando se trabalha com vários cenários que usam algumas das mesmas mensagens. **Veja um exemplo.**

Diagrama anterior



Exemplo

Por exemplo, se os sete primeiros passos do cenário A são os mesmos no cenário B, então o cenário B poderia simplesmente referir aos sete primeiros passos do cenário A por meio de um ícone de comentário, ao invés de redesenhá-los. Felizmente, muitas ferramentas de modelagem automatizam a numeração para você.

09

4- TIPOS DE MENSAGEM

Para entender os vários tipos de mensagens e notações que podem ser usados em um diagrama de sequência você precisa de algumas definições básicas. Estímulo, mensagem, operação e sinal compõem o vocabulário para trabalhar com as interações entre objetos nos diagramas de sequência e de colaboração.

4.1- Estímulo

Um estímulo é um item de comunicação entre dois objetos.

O estímulo apresenta as seguintes **características**:

- É associado tanto com o objeto emissor quanto o objeto receptor.
- Normalmente viaja através de um link. No entanto, quando o estímulo é enviado e recebido pelo mesmo objeto, não há nenhuma ligação. Este tipo de estímulo é chamado de autorreferência.
- Um estímulo pode invocar uma operação, emitir um sinal, ou até mesmo criar ou destruir um objeto.
- Um estímulo pode incluir parâmetros e/ou argumentos na forma de valores primitivos ou referências a objetos.
- Um estímulo está associado com o processo que faz com que ele seja enviado.

10

4.2 Mensagem

Uma mensagem é a especificação de um estímulo. A especificação inclui os papéis que o objeto remetente e o objeto receptor desempenham na interação, bem como o procedimento que despacha o estímulo.

Então, tecnicamente falando, um estímulo é uma instância de uma mensagem. Nas partes da especificação UML que cobrem diagramas de sequência e de colaboração, os dois termos são quase sempre usados como sinônimos.

Exemplo: quando desenhamos duas classes A e B associadas e informamos que A chama um método de B, estamos falando que nesse momento definimos uma mensagem. Quando o objeto “a” da classe “A” chama um método do objeto “b” da classe “B” com os parâmetros XYZ, estão, agora estamos falando que aqui temos um estímulo.

11

4.3 Sinal

Uma mensagem pode gerar um sinal. Isso significa que a mensagem pode ser apenas um alerta, como um *beep*, uma piscada em uma janela, uma tela de alerta, ou uma caixa de mensagem simples na tela do computador.

Ela não requer uma mensagem de retorno (no máximo um clique em um botão de ok ou o fechamento da janela de alerta). Um sinal é um tipo de classe associada a um evento que pode desencadear um procedimento dentro da classe de destino. Nesse contexto, normalmente não existe um retorno para a classe que a invocou.

Sinais são utilizados para informar um determinado acontecimento ao usuário, como o término de um processamento, a chegada de um e-mail, um alerta baseado em uma unidade de tempo. Sinais não são usados para controle de erros, mas eles podem ser usados para o controle de operações. **Exemplo.**

Veja este outro exemplo: suponha que no momento de concluir a venda do ingresso para o espetáculo teatral ocorra um problema com o cartão de crédito do cliente e ele fique impossibilitado de efetuar o pagamento. Neste cenário um sinal gerado pelo sistema de pagamento é retornado à aplicação (provavelmente algo como `pagamentoConfirmado() = Falso`) e com essa informação o sistema terá de desfazer a reserva do assento, liberando-o para outra venda. Observe que não houve um erro, mas sim um sinal de que a operação de pagamento não foi concluída com sucesso.

Os sinais são utilizados para controlar os caminhos alternativos de uma aplicação. Normalmente esperamos que todos os procedimentos ocorram com sucesso, mas nem sempre isso acontece: pagamentos que são cancelados, produtos em uma pré-venda que ficam com estoque indisponível, cadastros que ficam sem informações essenciais e obrigatórias e por isso os sistemas devem permitir um

controle de desfazer ou cancelar algo que começou a ser feito (como uma venda). Nesse sentido, é muito comum que o sinal requeira do sistema apenas um cancelamento dos objetos em memória (destruindo-os e voltando para a tela inicial do sistema ou do módulo) ou desfazendo operações em banco de dados (muitas vezes cancelando transações de banco de dados).

Exemplo
Suponha que seu programa esteja copiando um arquivo de um local para o outro, e que durante a cópia acabe o espaço livre no local de destino, impossibilitando a cópia. Neste momento, sua aplicação pode gerar um sinal de alerta, informando que não há espaço livre no destino e, consequentemente, cancelar a operação. Observe que não houve um erro interno, mas tão somente a impossibilidade de concluir a operação por conta da falta de um requisito da operação: espaço em disco suficiente.

12

4.4 Exceção

Uma “exceção” é um tipo especial de sinal. Lançar uma exceção significa o envio de uma mensagem contendo um objeto que descreve a condição de **erro.**

É algo muito mais brusco e inesperado do que o cenário esperado. Por exemplo, você cria um método que espera uma data válida e no momento da execução o método recebe uma data inválida. É impossível prosseguir com o sistema, e a operação deve ser cancelada. Normalmente, o código do sistema desvia o processamento normal para um processamento de exceção, onde o erro será tratado.

As exceções (e algumas vezes os sinais também) necessitam invocar ações que desfaçam um trabalho feito pela metade, como, por exemplo, desfazer uma transação no banco de dados que se iniciou. Uma condição de erro muitas vezes afeta o funcionamento normal do sistema e é comum vermos algumas das seguintes consequências:

- Volta para a tela de início do sistema;
- Requer que o usuário faça novo login na aplicação;
- Requer que seja instalado algum componente do sistema que esteja faltando ou desatualizado;
- Impossibilita, temporariamente, o uso do sistema;
- Desfaz transações feitas em banco de dados;
- Cancela operações / cadastros;
- Grava um log do erro ocorrido.

Podemos modelar, caso necessário, cenários de erro por meio de diagrama de sequência. **Saiba como.**

A maioria das linguagens de programação possui controles de erro, como, por exemplo, o try/catch do Java. Essas funcionalidades permitem que o programador crie procedimentos para o controle do erro, efetuando as ações necessárias para o sistema.

Saiba como
Basta criarmos as visões para esses cenários alternativos. Exemplo: veja o diagrama do capítulo 4 deste módulo. Observe que o passo 2.1 retornou verdadeiro, ou seja, a compra foi concluída. Mas o que aconteceria se a compra não tivesse sido concluída com sucesso? Teríamos que desfazer a reserva dos assentos selecionados pelo cliente. Então, para modelar esse cenário, bastaria criar outro diagrama, substituindo o passo 2.1 para falso e em seguida detalhar como a aplicação deveria se comportar.

13

4.5 Mensagem de autorreferência

Em uma mensagem de autorreferência, o emissor e o receptor da mensagem são o mesmo objeto.

Por exemplo, imagine que ocorra um problema com os atores da peça teatral e aquele espetáculo não vá mais acontecer no teatro. Para que o sistema lide com essa situação, um objeto de usuário precisaria enviar uma mensagem de "cancelar o espetáculo teatral" a um objeto de espetáculo de teatro. O evento deve mudar o seu próprio estado (cancelando o espetáculo teatral), o estado de todas as sessões do espetáculo (cancelando todas as sessões) e cancelando todos os assentos relacionados às sessões.

Deve ser capaz ainda de devolver o dinheiro dos ingressos já vendidos (outro cenário complementar).

Veja no diagrama abaixo uma possível representação desse cenário. Observe o passo 1.1, ele é uma autorreferência ao objeto que muda seu estado e dispara a próxima ação de cancelar sessões.

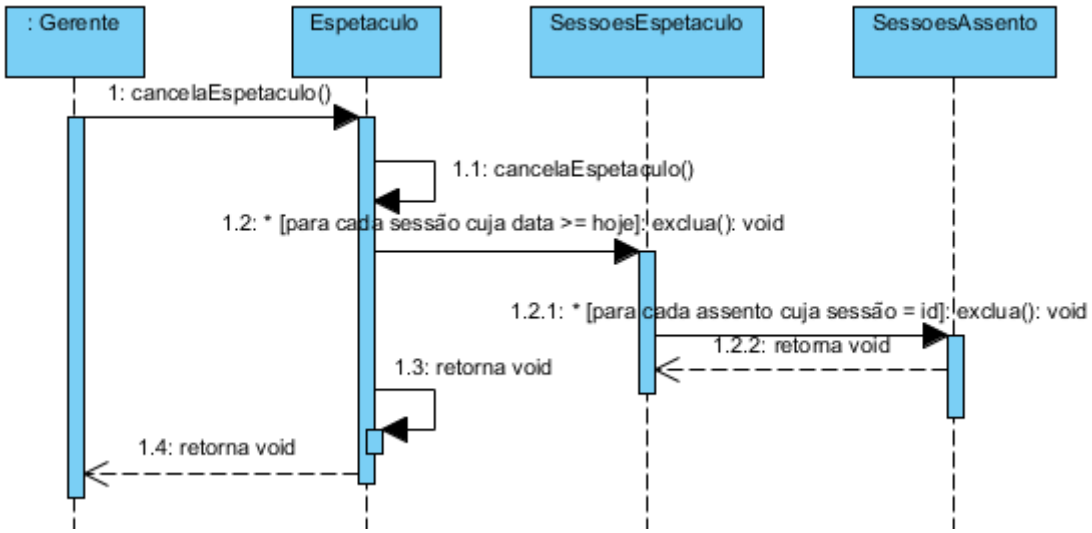


Diagrama exemplificando uma autorreferência.

14

4.6 Iteração

Uma iteração refere-se à necessidade de executar uma mensagem mais de uma vez dentro de um contexto.

Uma iteração é modelada por meio de um texto que se inicia com o asterisco (*) e em seguida descreve-se a condição que controla o número de repetições (de iterações). O texto entre colchetes define a regra de negócio de controle das iterações (exemplos: [para cada funcionário cadastrado], [para data da venda >= 01/01/2015], [para todos os assentos disponíveis]). Ao executar uma mensagem várias vezes implica realizar todo o caminho subordinado várias vezes também. Dessa forma, criamos laços de repetição que nos permitem controlar todo um conjunto de objetos e operações subordinadas.

No **diagrama** do item anterior você também poderá observar a existência de uma iteração no passo 1.2, onde há a expressão "* [para cada sessão cuja data >= hoje]: exclua(): void". Essa expressão informa ao programador a regra de negócio a ser executada nesse instante. O programador deve transformar esse texto em um algoritmo que realize a operação indicada. **Veja aqui** a tradução do que está informado nessa linha.

Note que dentro do laço que criamos no item 1.2 há outra estrutura de repetição no item 1.2.1. Temos dois laços criados um dentro do outro: para cada sessão futura, todos os assentos relacionados a cada uma das sessões serão cancelados.

Uma dica: lembrando que a autorreferência é algo que acontece dentro do objeto, a visibilidade que esse método deve ter dentro do objeto deve ser do tipo "privado" para assegurar que somente será acionado no contexto do objeto.

Diagrama anterior
Programação: inserir aqui a imagem do diagrama da tela anterior

Veja aqui
A tradução do que está informado na linha "* [para cada sessão cuja data >= hoje]: exclua(): void" é:

- O asterisco indica que ali ocorrerá uma interação, ou seja, a mensagem será enviada mais de uma vez dentro daquele contexto.
- Os dois pontos indicam a separação de cada operação a ser executada pelo algoritmo.
- [para cada sessão cuja data >= hoje] indica a regra de negócio: deve ser varrido todos os objetos instanciados e selecionados aqueles que satisfazem essa condição.

- exclua(): indica a operação de cancelamento da sessão que deve acontecer caso a sessão em questão satisfaça a condição anterior (data >= hoje).
- void: indica que nenhum retorno é dado ao método que o chamou.

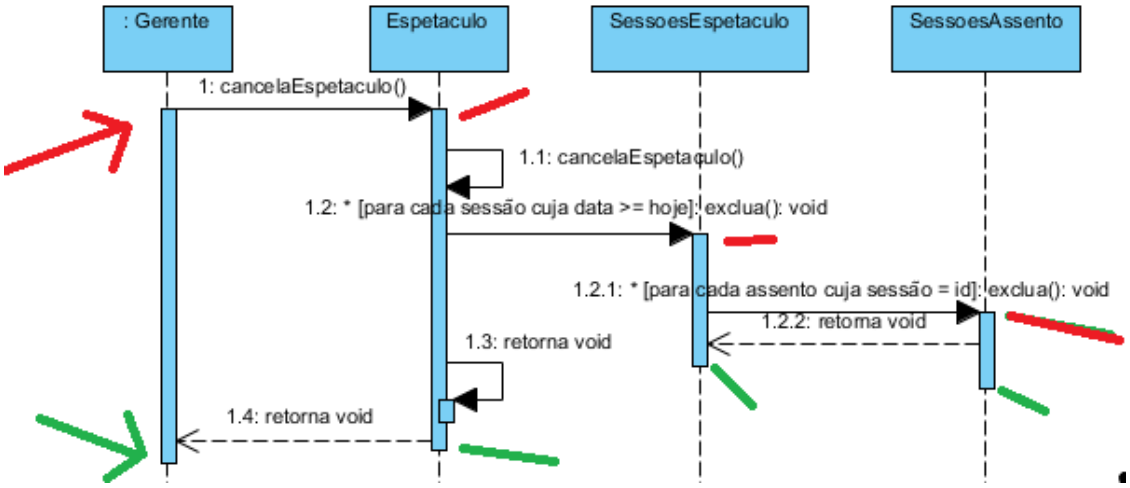
15

4.7 Modelando a ativação (foco de controle) e a desativação do componente

Não sei se você já percebeu, mas na linha de vida de cada objeto aparece um retângulo estreito que tem o topo próximo à primeira mensagem e o término próximo à última mensagem. Esses retângulos apresentam o **tempo** de ativação do objeto, onde ele recebe o foco até onde ele libera o foco. A parte de cima indica quando ele é ativado/recebe o foco de controle e a parte de baixo represente quando ele libera o controle.

Não confunda: não estamos falando aqui do ciclo de vida do objeto, que representa quando ele é instanciado até quando ele é destruído, mas sim do momento dentro da aplicação que ele possui o controle das operações.

No diagrama anterior, podemos observar que o controle do objeto Gerente inicia com a mensagem 1 e termina com a mensagem 1.4. Veja que ele tem um tempo de controle muito maior que o do objeto SessoesAssento, cujo foco inicia com a mensagem 1.2.1 e termina logo em seguida com a mensagem 1.2.2.



Indicação da ativação e desativação do componente. No diagrama acima, as linhas em vermelho apontam para o início do foco do objeto, enquanto que as linhas em verde apontam para o término.

RESUMO

Neste módulo, aprendemos que:

a) Os diagramas comportamentais, como os diagramas de sequência e de colaboração (UML 1.4) / comunicação (UML 2.0), representam a forma como os objetos funcionam usando a estrutura já definida nos outros esquemas.

b) O diagrama de sequência usa uma visualização orientada pelo tempo.

c) Iterações são trocas de mensagens entre objetos.

d) O diagrama de sequência usa dois elementos fundamentais: a notação de linha de vida dos objetos e mensagens ou estímulos.

e) A notação do diagrama contempla o retângulo para representar o nome do objeto e a linha pontilhada vertical para representar a respectiva linha de vida.

f) Uma mensagem representa algum tipo de comunicação entre objetos.

g) Uma mensagem: pode invocar uma operação, levantar um sinal, criar/destruir objetos de destino.

h) Uma mensagem é modelada como uma seta preenchida com linha sólida quando ela indica que precisa de uma resposta e uma seta aberta com linha pontilhada para indicar a resposta.

i) As mensagens são representadas sequencialmente de cima para baixo. Ainda, são numeradas de forma a montarem blocos de sequenciamento.

j) Estímulo é qualquer tipo de comunicação entre dois objetos.

k) Uma mensagem é a representação de uma ligação entre duas classes. Um estímulo representa a instanciação dessas classes em dois objetos.

l) Sinais são um tipo especial de mensagens que normalmente não recebem um retorno do usuário. Servem para indicar o término de um processamento ou a realização de um evento (como uma chegada de um e-mail, por exemplo).

m) Exceções são mensagens especiais que normalmente precisam de uma abordagem de controle de erro por parte do programador.

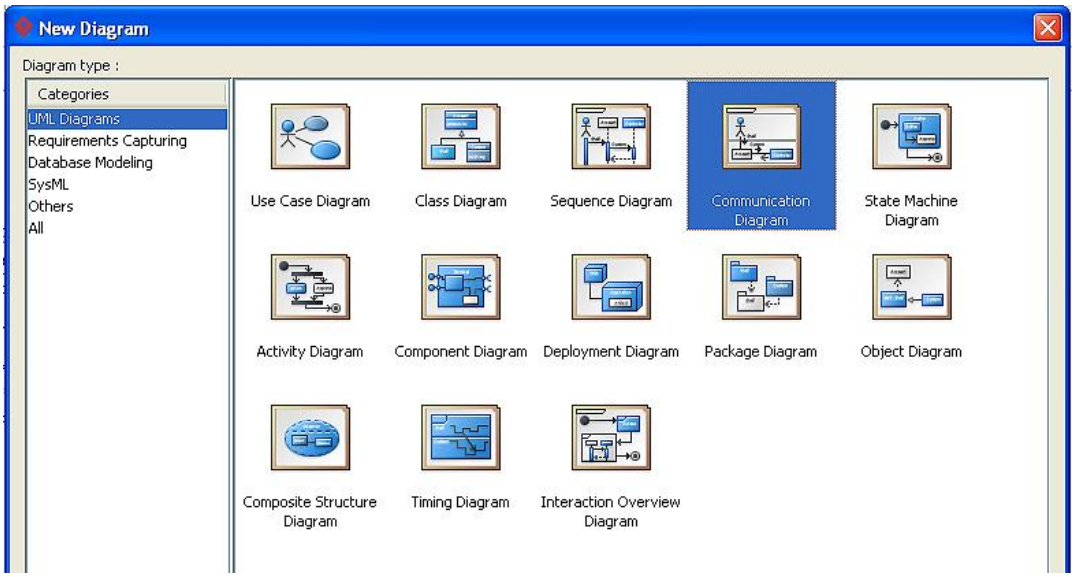
n) Iterações são mensagens que ocorrem mais de uma vez dentro de um objeto.

UNIDADE 2 – DIAGRAMAS BÁSICOS DA UML.
MÓDULO 4 – DIAGRAMA DE COMUNICAÇÃO

01

1- MODELANDO UM DIAGRAMA DE COMUNICAÇÃO

Neste módulo trataremos dos diagramas de comunicação.



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

O diagrama de colaboração (UML 1.4) ou comunicação (UML 2.0) oferece uma alternativa para o diagrama de sequência. Em vez de modelar mensagens ao longo do tempo como o diagrama de sequência, os modelos de diagrama de comunicação apresentam uma visão geral das mensagens relativas à estrutura do objeto.

O diagrama de comunicação usa essa abordagem para enfatizar o efeito dos objetos e suas ligações sobre o padrão de interações. A vantagem do diagrama de comunicação é que ele pode ajudá-lo a validar as associações entre as classes ou mesmo descobrir a necessidade de novas associações.

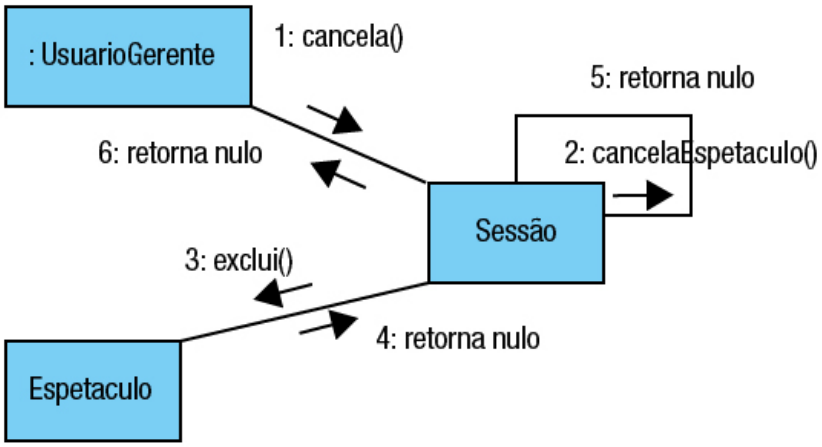
02

A figura abaixo contempla um diagrama de comunicação que representa uma mesma visão do diagrama de sequência que vimos anteriormente quando apresentamos o cenário de um gerente cancelando um espetáculo teatral no sistema.

Enquanto que, em um diagrama de sequência, os números de sequência são opcionais, em um diagrama de comunicação **os números são essenciais**, uma vez que não há outra maneira de determinar a ordem em que as mensagens são passadas.

Enquanto que, em um diagrama de sequência, os números de sequência são opcionais, em um diagrama de comunicação os números são essenciais, uma vez que não há outra maneira de determinar a ordem em que as mensagens são passadas.

Para ler o diagrama de comunicação, siga as mensagens numeradas para percorrer o cenário. Neste exemplo, as mensagens 1, 2, e 3 são mensagens síncronas (ficam aguardando uma resposta), a mensagem 2 é uma autorreferência, as mensagens de 4, 5 e 6 são os retornos.

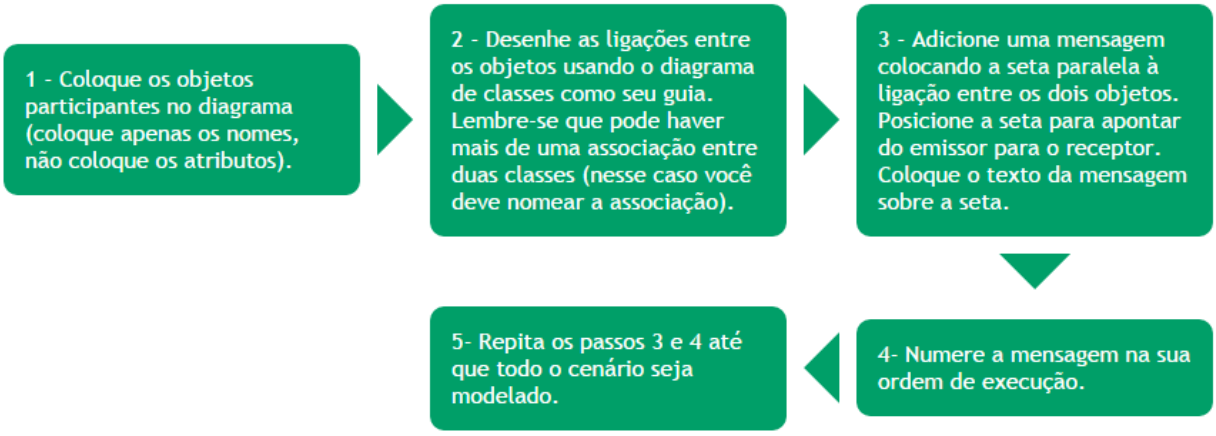


Exemplo de diagrama de comunicação

03

O diagrama de comunicação é construído a partir de um diagrama de objeto, como se segue:

Fique Atento!



Observação: veja que uma autorrelação é representada por um link (uma linha que entra e sai do mesmo objeto).

04

1.1. Modelagem de mensagens e estímulos

Observe no diagrama anterior que as mensagens são representadas por setas que vão do objeto emissor para o objeto receptor. Essas setas são comumente desenhadas paralelamente à linha de associação.

É comum haver muitas mensagens entre os objetos, consequentemente haverá muitas setas de mensagens representadas no diagrama. Para evitar um diagrama muito poluído, saiba separar as mensagens em cenários de tamanho apropriado. Por fim, não se esqueça de numerar as mensagens na ordem em que eles ocorrem.

O formato para especificar uma mensagem é o mesmo que no diagrama de sequência, no entanto, para minimizar a quantidade de informações no diagrama, utilizamos apenas o nome da operação e os atributos passados. Exemplos:

- `leEspetaculos (sessao)`
- `gravaEspetaculo (espetaculo)`
- `mostraConfirmacao`
- `retorna sessão`
- `retorna nulo`

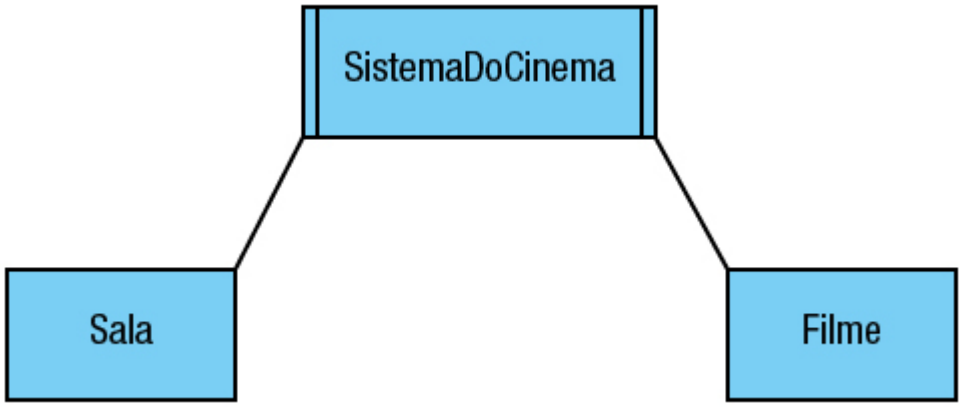
05

1.2. Objeto Ativo

Um objeto ativo refere-se a um objeto que inicia controles ou direciona outros objetos.

Compare isso com objetos passivos que simplesmente respondem às solicitações.

Na figura abaixo, o SistemaDoCinema representa um objeto ativo, enquanto os objetos Sala e Filme são passivos. Para identificar um objeto ativo, a borda do objeto é realçada. Além disso, a palavra-chave {ativo} pode ser colocada no ícone do objeto.



Exemplo de objeto ativo: SistemaDoCinema.

Em muitos casos, o objeto ativo é, na verdade, um conjunto de objetos, como é o caso com o objeto SistemaDoCinema acima. Se você quiser, você também pode expandir o objeto SistemaDoCinema transformando-o em um grande retângulo, e, em seguida, modelar os objetos que compõem o SistemaDoCinema dentro do retângulo.

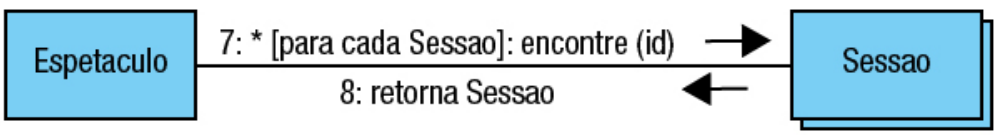
06

1.3. Objetos múltiplos (ou Multiobjetos)

Um objeto múltiplo é um conjunto (ou uma coleção) de objetos do mesmo tipo.

Quando você precisa realizar a mesma operação em todos os objetos de um conjunto, você pode representar o conjunto como um ícone de objeto em cima de outro ícone do objeto, isso é o suficiente para fazê-lo parecer como um de uma coleção de objetos.

O objeto de Sessão, ao lado direito da figura abaixo é um ícone de objeto múltiplo.



Exemplo de objeto múltiplo: Sessão.

Observe no diagrama acima que a mensagem 7 possui uma nomenclatura diferenciada. A fim de executar uma operação em todos os membros do conjunto, utilizou-se o asterisco "*" e a condição de pesquisa "[para cada Sessao]", após os dois pontos está o método que será executado "encontre" e o respectivo parâmetro passado "id". Na mensagem 8 vemos o retorno, que é um objeto do tipo Sessao.

07

2. Comparando diagramas de sequência e de comunicação

Os diagramas de Sequência e de Comunicação modelam o mesmo conjunto de elementos: mensagens e objetos. Na verdade, os dois diagramas são tão semelhantes que algumas ferramentas de modelagem fornecem um recurso para alternar entre os dois pontos de vista.

Ambos os diagramas permitem atribuir visualmente responsabilidades a objetos para o envio e recebimento de mensagens. Ao identificar um objeto como o receptor de uma mensagem, você está no efeito da atribuição de uma interface para esse objeto. A descrição da mensagem torna-se uma assinatura de operação no objeto de recebimento. O objeto de envio invoca a operação.

Todos os tipos de mensagens são suportados em ambos os diagramas.

Esses diagramas também são excelentes ferramentas para avaliação de acoplamento. O acoplamento é uma medida de qualidade que testa o grau de dependência entre os elementos do modelo.

A dependência é claramente vista na necessidade de comunicação entre objetos. Se você rever todos os diagramas em que um par de objetos participam, você pode ver quantas mensagens e qual o tipo de mensagens que eles usam para trabalhar em conjunto. Isso proporciona a oportunidade de avaliar a possibilidade de reduzir ou simplificar a comunicação e melhorar o design. Isto é muito difícil de fazer quando os únicos recursos que você tem são o diagrama de classes e o código.

Ao simplificar a quantidade de comunicações entre objetos, você provavelmente substituirá várias mensagens simples (com poucos atributos) por uma mensagem complexa (com muitos atributos).

08

Apesar de suas semelhanças, os diagramas também têm diferenças importantes. O diagrama de comunicação dá **prioridade ao mapeamento dos links** entre os objetos. Ou seja, o diagrama de

comunicação **desenha os objetos participantes** representando as mensagens paralelas às ligações entre objetos. Essa perspectiva ajuda a validar o diagrama de classe, fornecendo evidências da necessidade de cada associação, como meio de transmissão de mensagens. Em contraste, o diagrama de sequência não ilustra todos os links.

Isso destaca uma vantagem do diagrama de comunicação. Você não pode desenhar uma mensagem onde não há um link, simplesmente porque não há nenhum caminho físico através do qual a mensagem possa viajar. Em um diagrama de sequência não há nada que impeça você de desenhar uma seta entre dois objetos (mesmo não havendo nenhum link entre eles), mas isso iria modelar uma interação lógica que não poderia existir fisicamente no código fonte.

Isso quer dizer que é mais seguro e há menos chances de erro ao criar as mensagens em diagramas de comunicação.

Você também pode ver isso como uma vantagem do diagrama de sequência, em que o desenho de uma mensagem onde não há nenhuma ligação revela a necessidade de um novo link. Apenas certifique-se de que você realmente atualizará seu diagrama de classes ou você não será capaz de implementar a mensagem ilustrada no diagrama.

09

Outra vantagem do diagrama de sequência é a sua capacidade para **mostrar a criação e a destruição dos objetos**. Objetos recentemente criados podem ser colocados sobre a linha de vida do objeto no exato momento onde eles são criados. Um grande X no final de uma linha de tempo indica que o objeto já não está disponível para utilização. No diagrama de comunicação, ou o objeto está presente ou não. Não há maneira de indicar o momento de criação ou de término (no máximo temos mensagens que podem indicar o início do uso e o término do uso do objeto).

Os diagramas de sequência também têm a vantagem de **mostrar ativação do objeto**. Como o diagrama de comunicação não ilustra o tempo, é possível indicar explicitamente quando um objeto torna-se ativo ou inativo sem precisar interpretar os tipos de mensagens.

Levando em conta que o diagrama de classes é a fonte principal para a geração de código em desenvolvimento orientado a objeto, você precisa mapear o que você encontra de novo nos diagramas de sequência e de comunicação para criá-los no diagrama de classes.

Cada mensagem torna-se uma operação na classe do objeto que recebe a mensagem.

10

3 - EXEMPLO PRÁTICO

Vamos ver neste exemplo prático como um diagrama de comunicação surge a partir de um diagrama de classes. Para tal, utilizaremos como informação inicial o diagrama de classes abaixo que ilustra um cenário de uma pessoa comprando ingressos para assistir a um filme.

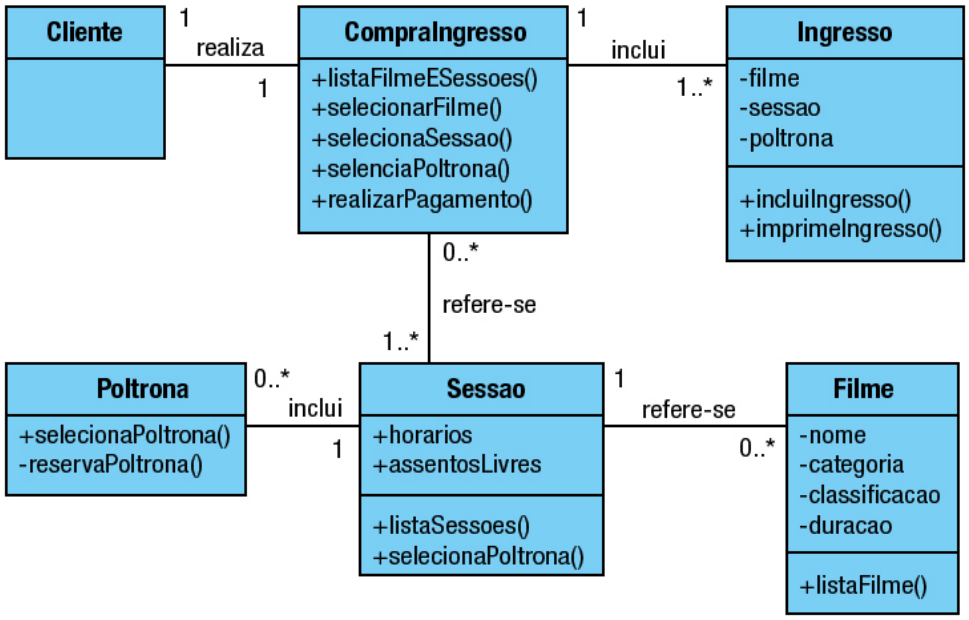


Diagrama de Classes do cenário de compra de ingressos de filme.

A sequência de atos que o cliente faz é:

1. O cliente inicia a compra de ingressos pesquisando os filmes e sessões disponíveis – método listaFilmeESessoes() da classe CompraIngresso que pesquisa os filmes disponíveis e sessões por meio da classe da classe Filme, método listaFilme(), e da classe Sessao, método listaSessoes().
2. O cliente seleciona o filme desejado - método selecionaFilme() da classe CompraIngresso
3. Seleciona então a sessão desejada – método selecionaSessao() da classe CompraIngresso;
4. As poltronas disponíveis são apresentadas na tela - método assentosLivres() da classe Sessão;
5. Seleciona uma ou mais poltronas para aquela sessão – método selecionaPoltrona() da classe Poltrona;
6. Pode ainda selecionar outros ingressos relativos a outros filmes/sessões (repetindo os passos 1 a 3);
7. Finaliza a compra efetuando o pagamento – método realizaPagamento() da classe CompraIngresso().

11

Para este cenário, teríamos o seguinte diagrama de sequência:

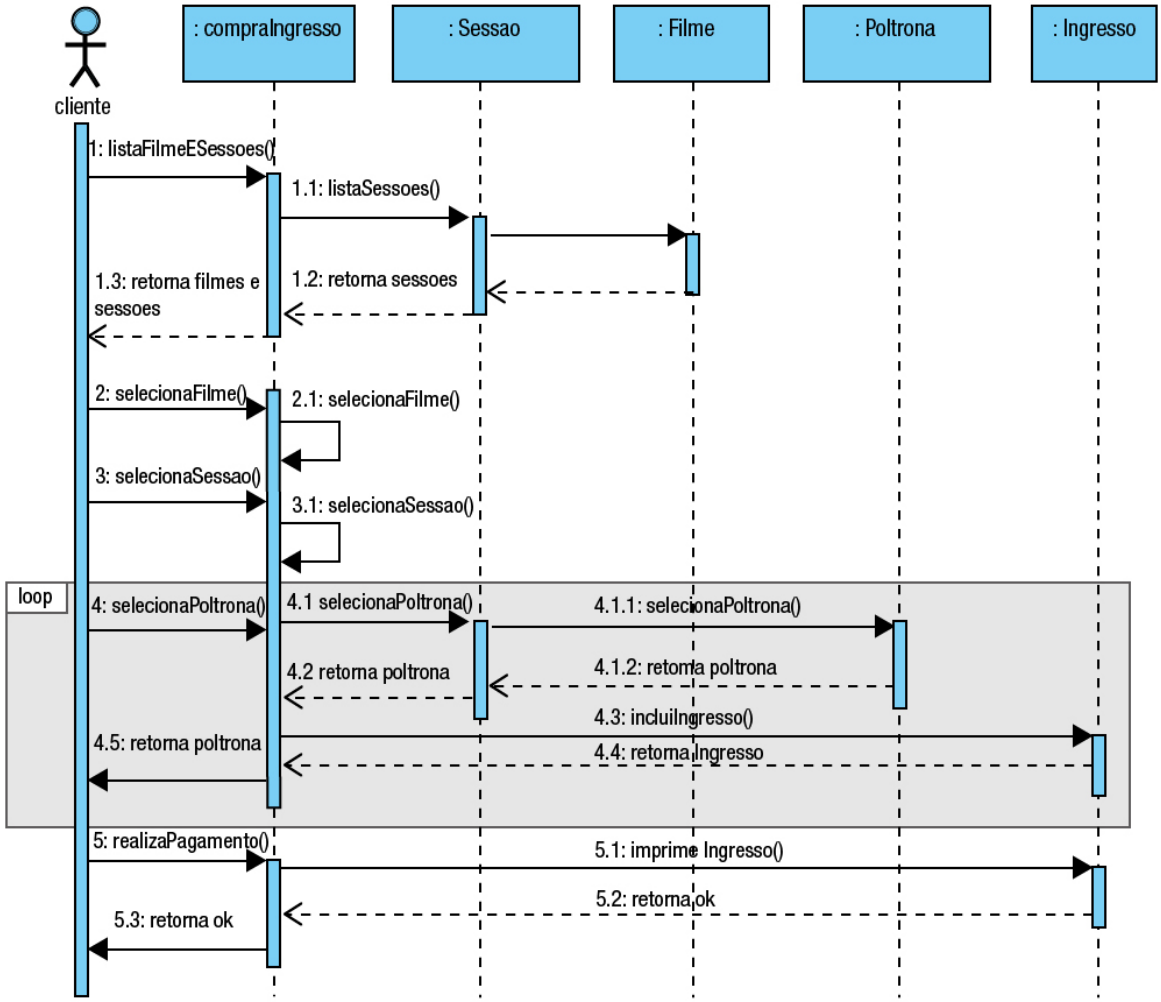


Diagrama de sequência que apresenta o cenário de compra de ingresso de cinema.

Podemos notar que o cenário é composto de cinco blocos de ações:

1. O primeiro, que lista os filmes em cartaz e as sessões disponíveis;
2. O segundo, que o cliente seleciona o filme desejado;
3. O terceiro, que o cliente seleciona a sessão desejada;
4. O quarto, que o cliente seleciona uma ou mais poltronas disponíveis para a sessão;
5. E o quinto, onde o cliente finaliza o pagamento e recebe os ingressos impressos.

12

Para não criar um diagrama muito poluído, vamos ver como seria o diagrama de comunicação para este mesmo cenário contemplando os cinco blocos de ações diagramados separadamente.

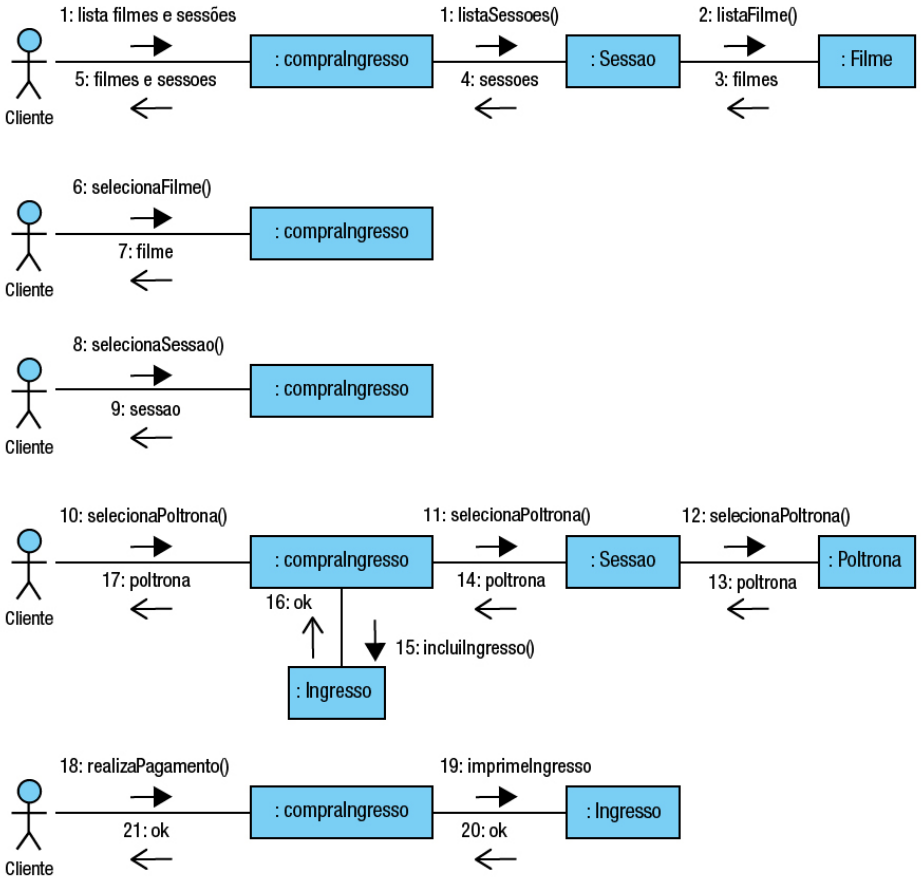


Diagrama de comunicação apresentando os cinco blocos de ações.

Compare agora os diagramas de sequência e de comunicação. Observe que as informações são praticamente as mesmas.

13

Resumo

Neste módulo, aprendemos que:

a. Os diagramas de sequência e de comunicação (chamado de diagrama de comunicação na UML 2.0) ilustram a comunicação entre objetos. O âmbito de aplicação das interações é tipicamente um cenário, mas os diagramas podem ser utilizados para modelar as interações em qualquer nível de abstração na concepção de um sistema.

b. Algumas mensagens são pré-definidas pela UML: síncrona (linha sólida com ponta de seta sólida), assíncrona (linha sólida com ponta de seta aberta), e retorno (linha tracejada com ponta de seta aberta).

c. O diagrama de comunicação modela as interações entre os objetos por meio dos links que proveem os caminhos de comunicação entre os objetos participantes. As mensagens devem ser numeradas para definir a ordem de execução.

d. Objetos e links: Objetos e links são representados com a mesma notação que em um diagrama de objeto.

e. Mensagens: usam a mesma sintaxe dos diagramas de sequência.

f. Objetos ativos: correspondem ao foco de objetos de controle em um diagrama de Sequência. Eles iniciam a interação e governam a sua execução.

g. Multiobjetos: Diagramas de interação pode modelar o uso de um conjunto de objetos usando um multiobjeto, um ícone que representa um conjunto em vez de uma única instância.

h. As informações descobertas sobre os diagramas de interação identificam e sugerem alterações no diagrama de classes. Conciliar os diagramas de interação com o diagrama de classe inclui o seguinte:

    i. Mensagens tornam-se operações.

    ii. Parâmetros e retornos tornam-se atributos ou operações para obter atributos derivados. A classe proprietária deve fornecer operações de acesso.

    iii. Condições e iterações tornam-se parte da lógica de implementação.