NÃO PODE FALTAR

MODELAGEM DE CLASSES

Maurício Acconcia Dias

DIAGRAMA DE CLASSES E OBJETOS

Utilizados para modelar a estrutura do sistema a ser desenvolvido, considerando o nível de classes e interfaces, também mostra o relacionamento entre os componentes do sistema.

Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

PRATICAR PARA APRENDER

Seja bem-vindo à seção que trata do diagrama de classes. Nela iremos trabalhar um dos diagramas mais utilizados da linguagem UML. Ao modelar um problema do mundo real, como um sistema de vendas, um controle de estoque ou um sistema bancário, é mais intuitivo utilizar uma linguagem orientada a objetos visto que a abstração dos objetos do mundo real facilita o entendimento da modelagem de sistemas, a qual pode ser feita de maneira mais simples em relação ao que estamos acostumados a encontrar no nosso dia a dia.

Um exemplo de problema do mundo real que podemos abordar é o de um sistema de controle bancário. Bancos geralmente possuem sistemas que devem considerar tanto a parte de dados dos clientes e de suas contas quanto a hierarquia da distribuição das agências por cidades. Além disso, existem os diversos serviços oferecidos por essas instituições, como contas e empréstimos.

É importante destacar que a abstração das linguagens orientadas a objetos inicia-se com as classes, que nada mais são do que a abstração dos objetos do mundo real. Essas classes são “materializadas” por meio de um software. Portanto, é possível criar a classe banco e depois especificá-la para cada parte em seu sistema. Por exemplo, pode-se criar uma classe extrato, que armazena diversos modelos, como conta corrente e poupança, e que controla suas quantidades, sua data de entrada e saída, entre outras informações.

Dessa forma, pode-se observar que as classes e objetos estão presentes na modelagem de vários sistemas presentes no mercado. Sendo assim, entender como criar o diagrama de classes é muito importante, pois permite a visualização do sistema em relação à sua estrutura.

O diagrama de classes também se relaciona diretamente com o diagrama de objetos, que irá apresentar uma instância específica do diagrama de classes em um determinado momento da execução o software.

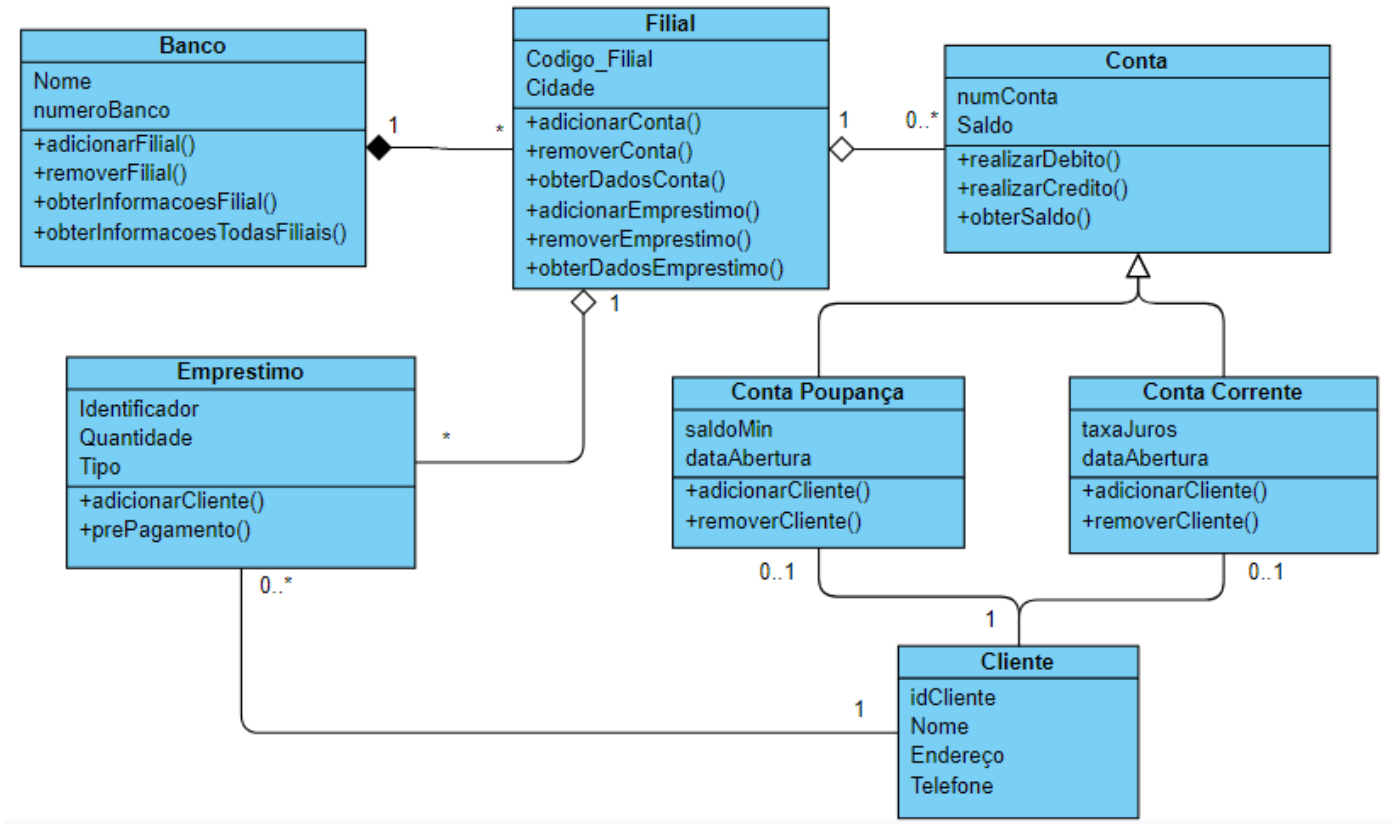
Imagine agora que, em um sistema bancário, temos uma classe inicial responsável por criar o banco. Além dessa classe outras são necessárias, como a que lida com as filiais do banco e suas agências; as responsáveis por modelar os tipos de contas oferecidas pelo banco e as que modelam clientes e serviços.

É possível, nesse caso, modelar um sistema de forma que a estrutura seja feita por classes mais simples e que a especificação seja realizada por subclasses herdeiras das características gerais. Os relacionamentos entre todas as classes citadas também podem ser explorados, por exemplo composições e agregações gerando um diagrama que modela situações reais ocorridas em uma agência bancária.

Você agora é um profissional que já apresenta diversas habilidades em UML. Ao ser contratado por uma empresa, foi alocado na equipe de engenharia de software e, devido à sua experiência com a linguagem UML, ficou responsável por realizar a modelagem de sistemas. A questão é que, para que seja testado seu nível de entendimento a partir de um diagrama de classes, foi-lhe mostrado um diagrama de um problema real desenvolvido pela empresa. Esse problema foi modelado a

partir do diagrama de casos de uso e sua equipe irá desenvolver o software. Como você está chegando nesse momento, deve demonstrar ao líder de sua equipe que é capaz de entender o diagrama e as relações apresentadas nele.

Figura 2.12 | Diagrama de classes de um banco



Fonte: elaborada pelo autor.

Com base no diagrama apresentado, explicite as classes e as relações existentes, evidenciando os detalhes existentes entre elas, os atributos e os relacionamentos apresentados na modelagem.

Agora, com os conhecimentos sobre os diagramas de classes e objetos, você será capaz de modelar grande parte dos sistemas exigidos pelo mercado utilizando a linguagem UML. Preparado? Então, bons estudos.

CONCEITO-CHAVE

Caro aluno, agora chegamos à seção de diagrama de classes e objetos da UML. Tais conhecimentos são fundamentais para a modelagem e, posteriormente, para o desenvolvimento de softwares. Para o completo entendimento desse conteúdo, é fundamental compreender o que são classes e objetos e quais são suas relações.

Um dos aspectos fundamentais para o uso do paradigma orientado a objetos é a necessidade constante de reutilização de código (DEITEL; DEITEL, 2016). Ao se programar de forma estruturada, por meio da construção de funções, existe uma separação entre os dados e seu processamento, ou seja, uma função recebe dados

de qualquer parte do programa e esses dados são processados. Todavia, ao se utilizar o paradigma orientado a objetos, os dados são integrados ao seu processamento, possibilitando um maior controle sobre eles e seu processamento.

CONCEITOS BÁSICOS DE ORIENTAÇÃO A OBJETOS

Existem alguns conceitos básicos do paradigma de orientação a objetos que devem estar claros para você entender os diagramas que serão apresentados. São eles:

- **Atributos:** são variáveis que armazenarão as características do objeto que se está modelando. Por exemplo, ao modelar um objeto carro, podemos pensar nas seguintes características (atributos): ano, modelo, cor, número de portas, manual, automático.
- **Métodos:** são as ações/comportamentos que podem ser desempenhados pelo objeto que se está modelando. No caso do objeto carro, podemos citar: acelerar, frear, mudar a marcha, dar a partida, desligar.
- **Classes:** representam o elemento abstrato de um conjunto de objetos; em outras palavras, uma classe possui toda a especificação do objeto, como as características (atributos) e as ações/comportamentos (métodos) dele. O carro, que utilizamos como exemplo até aqui, pode ter os atributos cor, ano, modelo, capacidade do motor; além disso, pode conter os métodos: acelerar, frear, dar a partida e desligar. Nesse caso, estamos lidando com a classe carro.
- **Objetos:** o objeto carro seria uma instância da classe carro, ou seja, um carro azul do ano de 1998 modelo GT2 com motor V8 de 4.0 litros. Este carro especificamente descrito é um objeto da classe carro com os atributos descritos.

EXEMPLIFICANDO

Para entender melhor os conceitos atributos, métodos, classes e objetos, vamos pensar no caso da bicicleta. Ao criar uma classe bicicleta, temos como atributos:

- Ano.
- Marca.
- Modelo.
- Aro.
- Tipo de freio.

E como métodos podemos pensar em:

- Pedalar para frente.
- Pedalar para trás.
- Virar.
- Frear.

Portanto, esta seria a classe bicicleta. Agora, para exemplificar a instância dessa classe, vamos pensar em bicicletas específicas:

- Ano: 2018; marca: trovão; modelo: sx45; aro: 18; freio: disco.
- Ano: 1963; marca: bike; modelo: a23; aro: 20; freio: normal.

Os atributos de cada uma das bicicletas são específicos, porém elas possuem os mesmos métodos já que são bicicletas.

Herança também chamada de generalização é um dos relacionamentos entre classes e pilares da orientação à objetos, a qual indica que uma classe é criada a partir de uma classe existente apenas adicionando atributos e métodos à sua definição original (DEITEL & DEITEL, 2016). Nesse sentido, pode-se afirmar que quando uma classe herda de outra, estamos definindo que a nova classe terá tudo que a anterior (ou superclasse) tinha em adição ao quer for definido na nova classe (ou subclasse).

EXEMPLIFICANDO

Para exemplificar o conceito de herança, vamos definir a classe bicicleta_motorizada a partir da classe bicicleta anteriormente definida, que possui os atributos:

- Ano, marca, modelo, aro, tipo de freio.

E os métodos:

- Pedalar para frente, pedalar para trás, virar e frear.

É possível definir que uma bicicleta motorizada tenha todas as características de uma bicicleta comum e, além disso, tenha como atributo adicional o motor e como método adicional acelerar. Sendo assim, em vez de definir toda uma nova classe bicicleta_motorizada com todos esses atributos e métodos, é definida a classe nova apenas com um atributo (motor) e um método (acelerar), herdando todos os outros métodos da classe bicicleta.

Outra vantagem da herança é que tudo que for melhorado, modificado ou corrigido na superclasse impacta diretamente na subclasse. Isso remove a necessidade de correção em diversas partes de um mesmo código tornando o desenvolvimento mais rápido e simples.

Com os conceitos de orientação a objetos revisados, iniciamos os diagramas de classes, os quais são elementos importantes para o processo de modelagem de sistemas.

DIAGRAMA DE CLASSES

O diagrama de classes da UML é um diagrama estrutural, que tem como objetivo principal ilustrar graficamente a estrutura do software, em níveis mais e menos abrangentes. Além disso, o diagrama de classes mostra como se dá a interligação entre os componentes da estrutura do sistema (UML-DIAGRAMS, 2016).

Para compreender o diagrama de classes, é preciso entender os relacionamentos entre as classes que poderão ser representadas no diagrama de classes e objetos.

RELACIONAMENTO DO TIPO ENCAPSULAMENTO

Um importante tipo de relacionamento é o encapsulamento, que tem como objetivo garantir a proteção e o controle de acesso dos atributos e métodos de uma classe, estabelecendo diferentes níveis de acesso aos usuários de uma classe

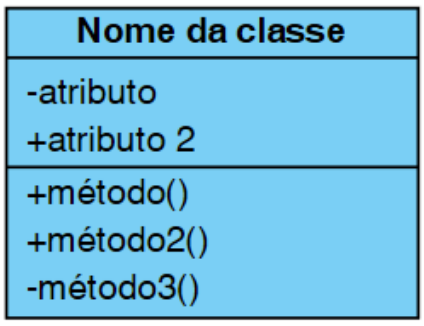
para que apenas usuários autorizados tenham visualização, ação, modificação de algum atributo ou método da classe (DEITEL; DEITEL, 2016).

Para encapsular informações em uma classe, são utilizados os operadores de escopo em atributos e métodos, que são:

- ***Public:*** nível de acesso mais permissivo. O método ou atributo da classe é livre para qualquer usuário manusear. É utilizada a notação (+) para representar atributos e métodos públicos.
- ***Private:*** nível de acesso mais protegido (restritivo). Apenas usuários definidos têm acesso. É utilizada a notação (-) para indicar atributos e métodos privados.
- ***Protected:*** nível intermediário (entre *public* e *private*). Permite acesso externo restrito e os atributos e métodos podem ser públicos dentro de uma classe específica. Envolve conceitos de herança abordados a seguir. A notação (#) é utilizada para indicar atributos e métodos protegidos.

Em geral, um diagrama de classes possui três informações: o **nome** da classe, os **atributos** (características) e os **métodos** (comportamento, operações). Os atributos são mostrados na primeira partição e os métodos na segunda, como mostrado na Figura 2.13 a seguir.

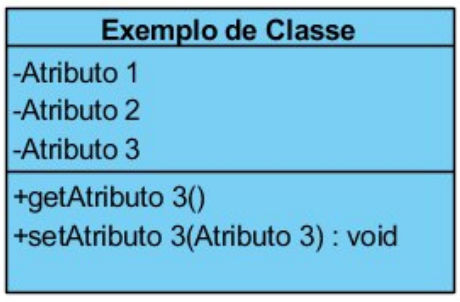
Figura 2.13 | Notação diagrama de classes



Fonte: elaborada pelo autor.

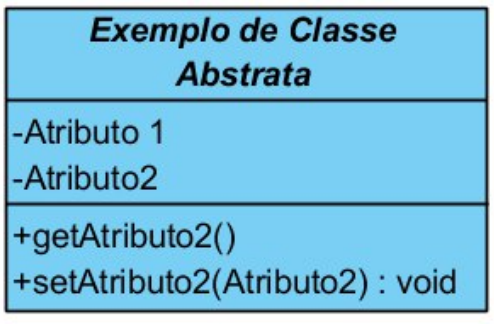
Os diagramas de classes UML apresentam dois tipos básicos de classe: as **classes comuns** (Figura 2.14) e as **classes abstratas** (Figura 2.15). As comuns possuem o nome da classe, os atributos e os métodos separados por um divisor; já as abstratas são representadas da mesma maneira, porém com o nome da classe em itálico (Figura 2.15). Cada classe possui seus próprios atributos e métodos.

Figura 2.14 | Exemplo de classe



Fonte: elaborada pelo autor.

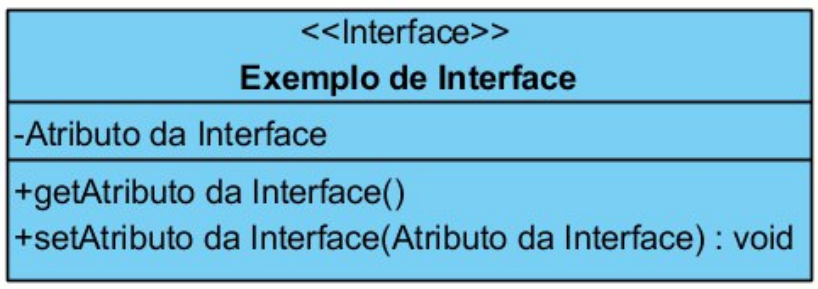
Figura 2.15 | Exemplo de classe abstrata



Fonte: elaborada pelo autor.

Outro componente presente nos diagramas de classe são as **interfaces**. Elas apresentam um conjunto de operações e elementos públicos, que devem ser implementados por outras classes (UML-DIAGRAMS, 2016). Esse procedimento faz com que as informações de implementação fiquem protegidas em outras classes privadas ou protegidas enquanto a interface é acessada. Nos diagramas de classes UML, as interfaces são representadas de maneira similar às classes, porém possuem um identificador, como apresentado na Figura 2.16.

Figura 2.16 | Exemplo de interface



Fonte: elaborada pelo autor.

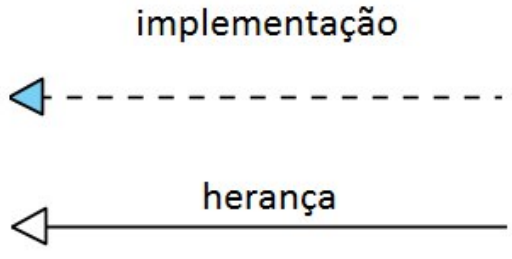
Como uma classe abstrata ou interface não instanciam objetos, é necessário que alguma outra classe implemente o que está descrito na interface ou que alguma classe herde os atributos e métodos da classe abstrata e implemente seu comportamento.

TIPOS DE RELACIONAMENTO

Outro ponto importante são os tipos de relacionamentos entre as classes. A **generalização** (herança) é representada por uma seta contínua, que sai da subclasse e vai em direção à superclasse, enquanto que a **implementação**, também chamada de realização, é representada por uma seta pontilhada, que sai da classe que implementa e aponta para a classe implementada. Ambas estão representadas na Figura 2.17.

Ver anotações

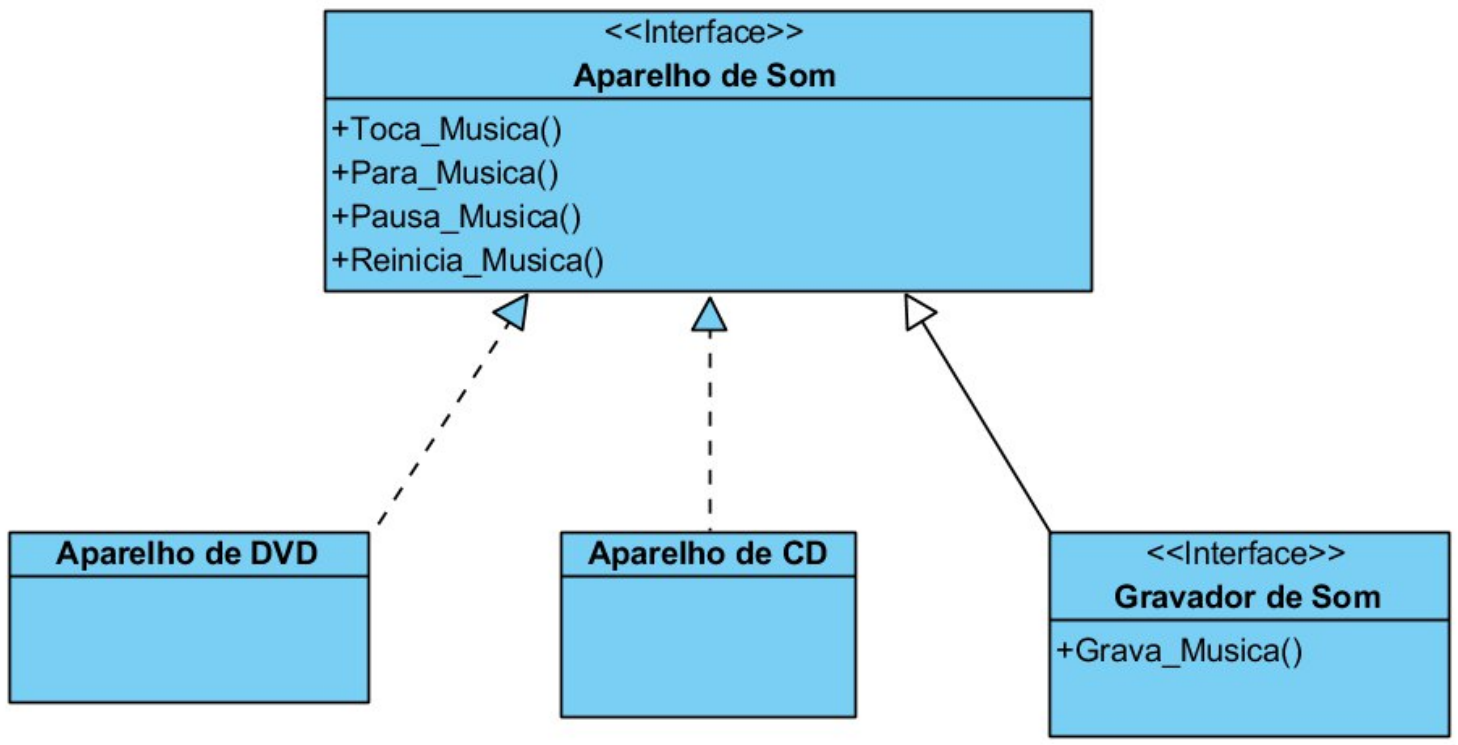
Figura 2.17 | Relacionamentos de implementação e herança



Fonte: elaborada pelo autor.

Para melhor compreensão, observe o exemplo de um diagrama de classes simples apresentado na Figura 2.18.

Figura 2.18 | Diagrama de classes para um sistema de música



Fonte: elaborada pelo autor.

Na Figura 2.18, é apresentado um diagrama de classes para um sistema de música. Nela existe uma interface que apresenta as funções básicas que devem estar disponíveis para o sistema: tocar, parar, pausar e reiniciar uma música. As classes aparelho de DVD e aparelho de CD implementam a interface, cada uma para uma mídia diferente. A herança nesse caso é representada entre a classe aparelho de som e gravador de som, já que se entende um gravador como um aparelho de som

que possui a função de gravar. Observe que nenhuma classe implementa o gravador ainda, porém iremos adicionar funcionalidades a esse diagrama ao longo da seção.

É importante destacar que existem diferentes perspectivas, as quais estão relacionadas à quantidade de detalhes e os tipos de relacionamentos entre as classes. As três perspectivas que podem ser apresentadas são as seguintes:

- **Conceitual:** que define um relacionamento entre entidades conceituais, conceito no domínio.
- **Especificação:** que define funções/responsabilidades entre duas classes. Existem métodos que tratam essas operações/responsabilidades. Esta perspectiva é utilizada para entendimento da arquitetura em nível de interface.
- **Implementação:** é a mais completa, pois possui diversos detalhes, como visibilidade de atributos e métodos, parâmetros, tipos de atributos e retornos. Além disso, há a possibilidade de visualizar a navegabilidade (seta no fim do relacionamento) na representação gráfica e as responsabilidades.

É importante conhecer as todas as perspectivas que podem ser utilizadas em diferentes etapas do desenvolvimento do software mesmo sendo a implementação a perspectiva que envolve mais elementos e mais detalhes.

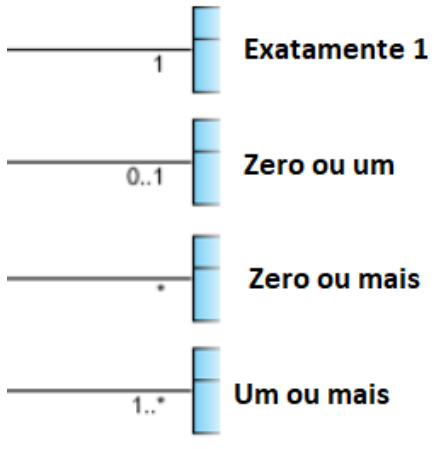
RELACIONAMENTO DE MULTIPLICIDADE

Ainda sobre os diagramas de classes, existem outros relacionamentos que devem ser descritos. O relacionamento “papel” descreve a ligação entre as classes, facilitando o entendimento. Além disso, é importante destacar o relacionamento de multiplicidade entre as classes, que diz respeito à quantidade de instâncias de uma classe associada com um ou mais instâncias de outra classe.

- ***n..m* –** significa *n* para *m* instâncias entre classes, ou seja, *n* classes A se relacionam com *m* classes B.
- ***n..** -** quando se utiliza o asterisco (em qualquer um dos lados da expressão *n..m*) indica-se “*muitos*”, ou seja, muitas instâncias da classe.
- **1 –** quando se utiliza o número 1, quer-se dizer exatamente uma instância associada.
- **1..* –** esta relação, em que temos o número *1..**, apresenta o conceito de um ou mais instâncias que podem estar relacionadas.

As notações dos tipos de relacionamento multiplicidade são ilustrados na Figura 2.19 a seguir.

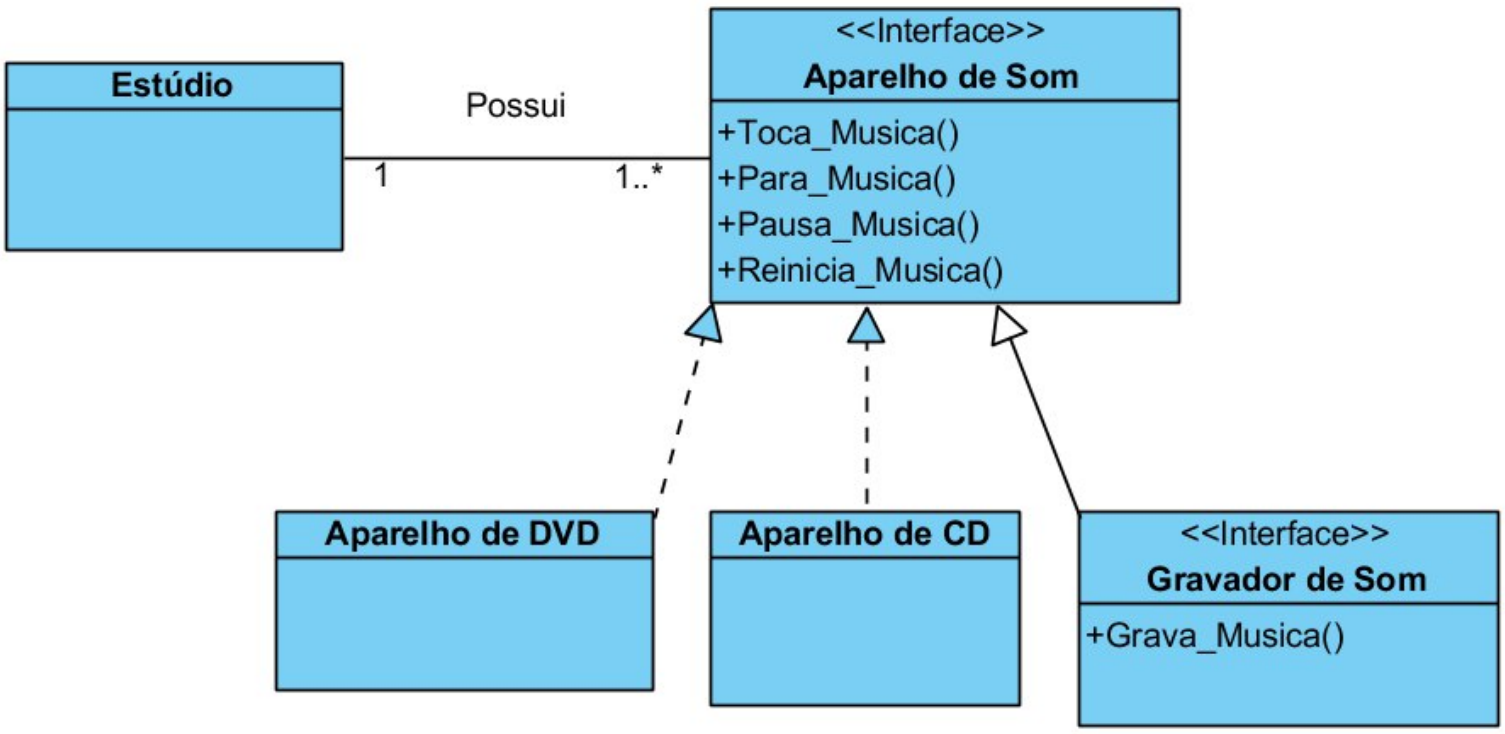
Figura 2.19 | Notação de multiplicidade



Fonte: adaptada de Visual Paradigm (c2020).

Para exemplificar a multiplicidade, o diagrama do sistema de som, apresentado anteriormente, foi modificado e é apresentado na Figura 2.20. Neste caso foi adicionada a classe estúdio e os relacionamentos de multiplicidade, que representam que um estúdio (1) possui um ou mais (1..*) aparelhos de som.

Figura 2.20 | Exemplo de diagrama de classe com relacionamento multiplicidade



Fonte: elaborada pelo autor.

Outros dois relacionamentos que podem ser representados em diagramas de classe são: **agregação** e **composição**.

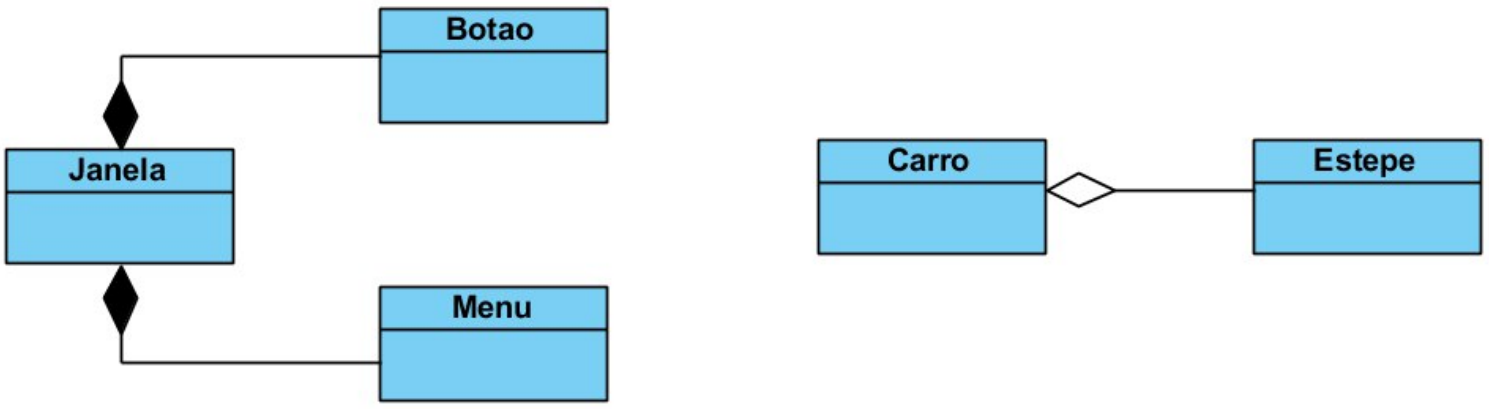
A agregação é considerada um caso especial de associação; ela representa o fato de que uma das classes do relacionamento é uma parte de ou está contida em outra classe. A associação agregação é do tipo todo-parte quando um objeto é parte de outro; porém, é preciso salientar que a parte existe independentemente do todo. Sendo assim, é considerada como um relacionamento fraco. Por exemplo, se uma classe 2 é parte da classe 1, ambas as classes existem separadamente.

A composição é uma variação da agregação que apresenta um vínculo mais forte entre as associações. No caso, objetos-parte devem obrigatoriamente pertencer ao objeto-todo. Logo, na composição, o todo não existe sem as partes e as partes não existem sem o todo.

A Figura 2.21 ilustra um exemplo com os relacionamentos agregação e composição. A imagem à esquerda é um exemplo de composição: o menu e os botões não existem sem as janelas para acomodá-los. Portanto, a composição implica uma relação parte-todo configurando uma relação mais forte no sentido de que uma classe não existe sem a outra. A notação da composição é um losango preenchido ao lado do todo.

No caso da agregação, relação à direita na Figura 2.21, é apresentado um carro e seu estepe. Um carro pode existir sem o estepe ainda que possa apresentar problemas ao usuário caso um pneu fure. O relacionamento de agregação é representado por um losango sem preenchimento na extremidade da classe que contém os objetos-todo.

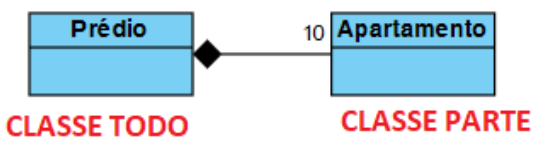
Figura 2.21 | Exemplos de composição e agregação



Fonte: elaborada pelo autor.

Vamos agora apresentar um outro exemplo de composição que envolve multiplicidade. Na Figura 2.22, temos um exemplo de composição entre prédio e apartamentos. Assim, temos duas classes: prédio e apartamento. O prédio é considerado o todo, ou agregada, e a classe apartamento é parte da classe prédio.

Figura 2.22 | Exemplos de composição e agregação



Fonte: elaborada pelo autor.

No exemplo, pode-se observar que os apartamentos apenas existem se o prédio existe também. Já a multiplicidade apresentada no diagrama representa que é necessário, obrigatoriamente, um prédio possuir dez apartamentos.

ASSIMILE

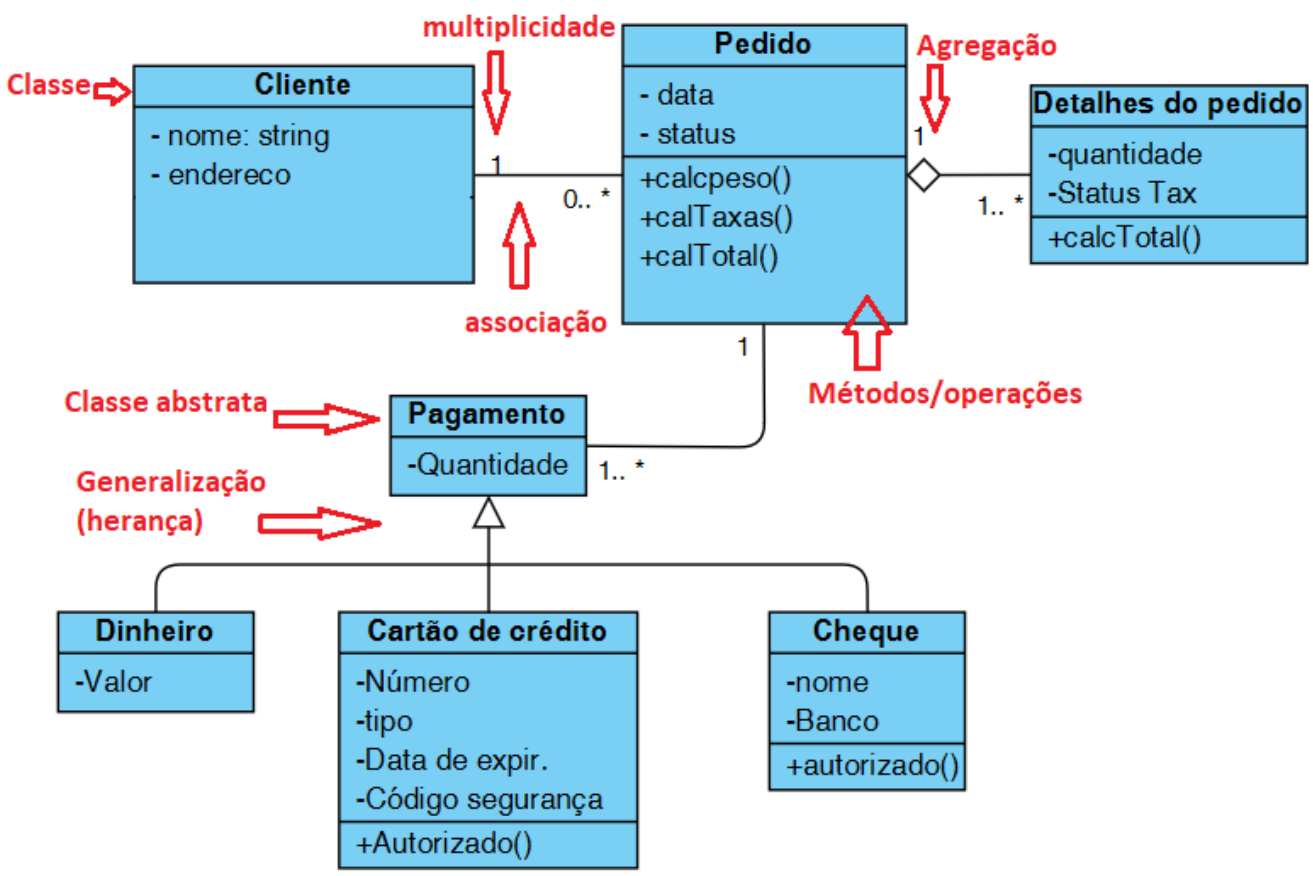
Importa destacar os conceitos de agregação e composição relacionados às necessidades de representação do diagrama de classes. Quando, na modelagem, existe uma relação de todo-parte entre duas classes, é necessário definir se o relacionamento é do tipo agregação ou composição. Na agregação, o todo e a parte existem independentemente e, no caso da composição, a parte depende do todo para existir. Algumas palavras utilizadas para reconhecer uma agregação são as seguintes: "consiste em", "contém", "é parte de". É importante lembrar que essas relações são abstrações, ou seja, quando implementamos uma relação de composição,

Ver anotações

> não necessariamente seria impossível instanciar um objeto da parte sem ter instanciado o todo, porém conceitualmente isso não deve ser feito no código.

Para sintetizar os conceitos apresentados sobre diagrama de classes e os seus relacionamentos, a seguir é apresentado um exemplo de aplicação de um sistema de compras. A Figura 2.23 ilustra esse sistema no qual temos as classes cliente, pedido, detalhes do pedido, a classe abstrata pagamento e as classes dinheiro, cartão de crédito e cheque. Pode-se observar que temos as associações com multiplicidades, em que um cliente faz nenhum ou mais pedidos. Para cada pedido deve-se realizar os pagamentos, ação que está associada ao pedido, pago através de cartão, dinheiro ou cheque. Observe que temos três subclasses: dinheiro, cartão de crédito e cheque herdadas da superclasse chamada pagamento.

Figura 2.23 | Exemplo de diagrama de classes de um sistema de compras



Fonte: adaptada de Visual Paradigm (c2020).

MODIFICADORES DE ACESSO

Para encerrar o tópico sobre os elementos de um diagrama de classes, outro conceito importante é o dos modificadores de acesso, que foram introduzidos anteriormente nesta seção quando tratamos dos operadores de escopo em encapsulamento, na revisão de orientação a objetos. Agora vamos tratar deles no contexto de aplicação em um diagrama de classes.

Os modificadores de acesso são modelos de visibilidade para acesso às classes, atributos e métodos.

Em UML é possível representar itens com visibilidade *public*, *protected*, *private* e *package* (que apenas é visível em um determinado *namespace*). A Figura 2.24 apresenta um exemplo de classe com métodos de visibilidade diferentes.

Figura 2.24 | Classe com exemplos de visibilidade

Fonte: elaborada pelo autor.

A classe da Figura 2.24 apresenta um método configurado com cada tipo de visibilidade. O primeiro é *public*, indicado com um '+'; o segundo é do tipo *package*, indicado por um '~'; o terceiro é o *protected*, indicado por um '#' e o último é o *private*, indicado por um '-'. Portanto, ao analisar um diagrama, é necessário considerar como os atributos e métodos estão indicados quanto a sua visibilidade.

REFLITA

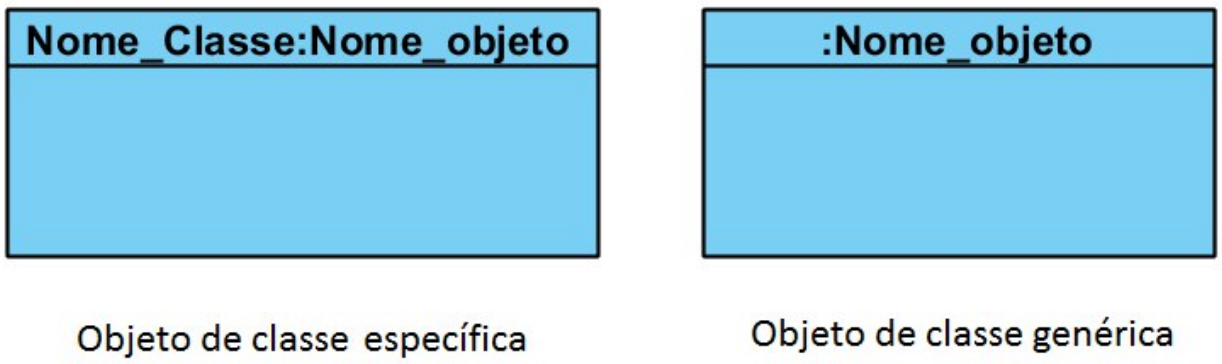
Sabe-se que as classes são "modelos" que podem ser instanciados em objetos. A modelagem normalmente ocorre criando-se as classes necessárias e todos os relacionamentos existentes. Porém, existem alguns problemas que podem aparecer ao se fazer a modelagem das classes. Caso não sejam analisados os objetos instanciados e suas relações, as classes podem apresentar problemas de implementação, que só irão aparecer quando o programa for executado. Você consegue imaginar esse tipo de problema? Tente pensar em algum que não apareceria em uma implementação de classes, mas que causaria erros na execução do programa ao instanciar os objetos e ao executar o programa.

DIAGRAMA DE OBJETOS

Além do diagrama de classes, temos o diagrama de objetos; este é uma variação daquele, com uma notação semelhante à do diagrama de classes. O diagrama de objetos é utilizado para representar os objetos instanciados de uma classe. Ele utiliza notação de objetos para a própria construção.

Esse diagrama tem por objetivo representar os objetos e os relacionamentos entre eles e nada mais é do que um grafo de instâncias, incluindo os objetos e os valores de seus atributos (UML-DIAGRAMS, 2016). As representações de objetos são feitas de forma similar às de classes, porém com a diferença de que eles só possuem atributos, e o nome irá indicar a classe da qual foram instanciados ou então apresentará uma instância de qualquer classe. Esses dois exemplos são apresentados na Figura 2.25.

Figura 2.25 | Tipos de objetos do diagrama de objetos



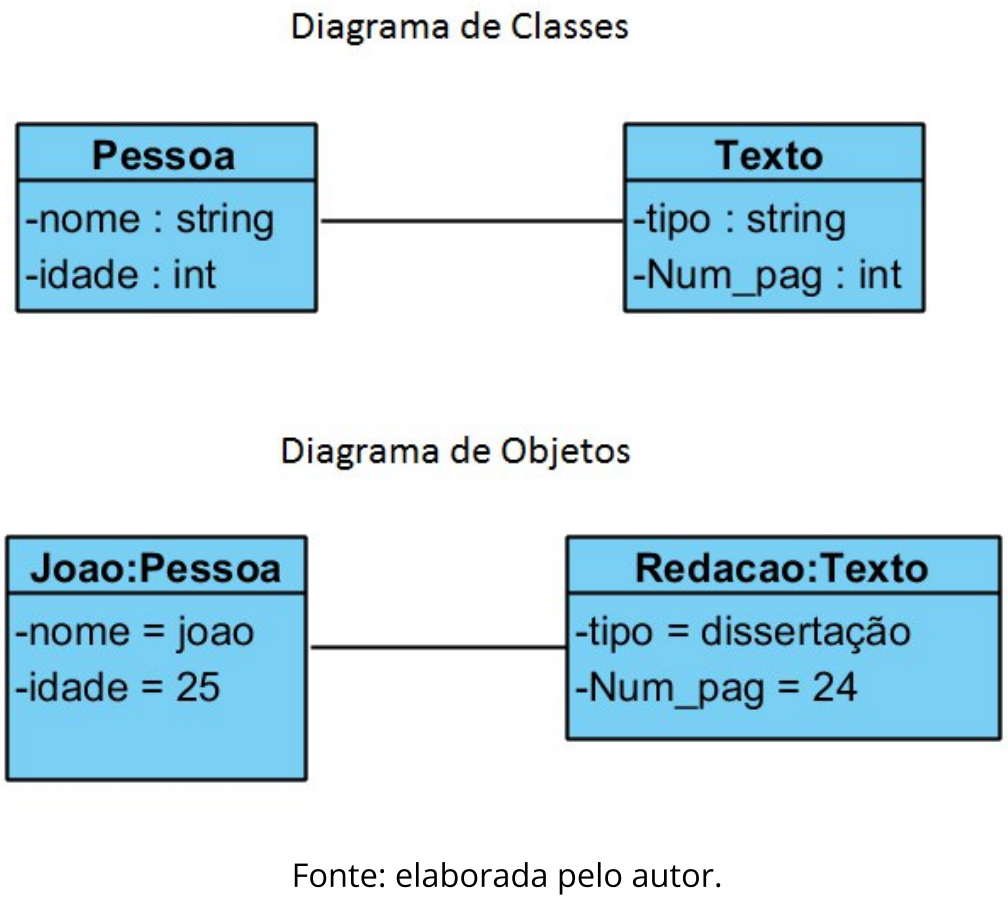
Fonte: elaborada pelo autor.

Além dos objetos, são representadas as associações e os valores de seus atributos.

É importante ressaltar que um diagrama de objetos deve ser gerado a partir de um diagrama de classes.

A Figura 2.26 ilustra um exemplo de diagrama de classes e um possível diagrama de objetos gerado a partir dele. Temos as classes pessoa e texto. A partir do diagrama de classes, geramos o diagrama de objetos.

Figura 2.26 | Exemplo de diagrama de classes e objetos



Fonte: elaborada pelo autor.

Observe que são apresentados apenas os nomes e os atributos no diagrama de objetos. Os valores dos objetos, obedecendo ao tipo de variável (int), também são apresentados. Nesse caso é possível entender como gerar um diagrama de objetos a partir de um diagrama de classes.

Caro aluno, nesta seção foram abordados alguns conceitos de orientação a objetos para que você conhecesse os diagramas de classes e objetos, bem como os principais componentes e relacionamentos. Além disso, tratou-se dos diagramas de objetos a partir de um determinado diagrama de classes. Agora você pode aplicar os seus conhecimentos para elaborar um diagrama de classes e de objetos para algum exemplo de aplicação do seu dia a dia. Então vamos lá? Bons estudos e mãos à obra!

REFERÊNCIAS

BELL, D. Fundamentos básicos de UML: O diagrama de classes: Uma introdução aos diagramas de estrutura em UML 2. IBM Developer Works, 2016. Disponível em: https://bit.ly/2DccZMC. Acesso em: 6 jul. 2020.

DEITEL, P.; DEITEL, H. Java: como programar. 10. ed. Campinas: Pearson Universidades, 2016. 968p.

RUBAUGH J.; JACOBSON, I.; BOOCH, G. The Unified Modeling Language Reference Manual. 2. ed. [S.l.]: Pearson Higher Education, 2004.

SOMMERVILLE, I. Engenharia de Software. 9. ed. Campinas: Pearson Universidades, 2011.

UML-DIAGRAMS. The Unified Modeling Language, [S.l.], 2016. Disponível em: https://bit.ly/3f7qM41. Acesso em: 6 jul. 2020.

UNHELKAR, B. Software engineering with UML. [S.l.]: CRC Press, 2018.

VISUAL PARADIGM. UML Class Diagram Tutorial. Visual Paradigm, Hong Kong, c2020. Disponível em: https://bit.ly/2QBc1N4 . Acesso em: 6 ago. 2020.