TEXTO DE APOIO

AULA 2

Desenvolvimento de Sistemas I

Professora Cláudia Rossi

EaD | Universidade Presbiteriana Mackenzie

Sumário

CLASSES E OBJETOS

INTRODUÇÃO

Aprenderemos alguns conceitos básicos de modelagem de classes que serão usados ao longo deste componente curricular. Para isso, é importante realizar uma leitura atenta de cada conceito e procurar relacioná-los com o que você estudou em programação orientada a objetos. É muito importante também que você associe a notação gráfica de representação de objeto e classe correspondente com o código-fonte na linguagem Java, por exemplo.

Vamos lá, agora é hora de aprender alguns conceitos relevantes, que são: objetos, classes, ligação entre objetos e associação entre classes.

OBJETO: UMA ABSTRAÇÃO DE ALGUMA COISA EM UM DOMÍNIO DE PROBLEMA

Um objeto é um conceito, uma abstração ou uma coisa com identidade que possui significado para uma aplicação. Os objetos normalmente parecem como nomes próprios ou referências específicas nas descrições de problemas e discussões com os usuários e clientes durante o processo de desenvolvimento de sistemas de software.

Todos os objetos possuem identidade e são distinguíveis. Nas figuras a seguir, você pode notar diferentes tipos de motocicletas que podem ser utilizadas para diferentes propósitos. Algumas pessoas utilizam para trabalho, outras utilizam para passeio. Mas todas são motocicletas e têm propriedades em comum.

Note os diferentes tipos de motos com diferentes características e propriedades e que são distinguíveis entre si.

Moto de trabalho

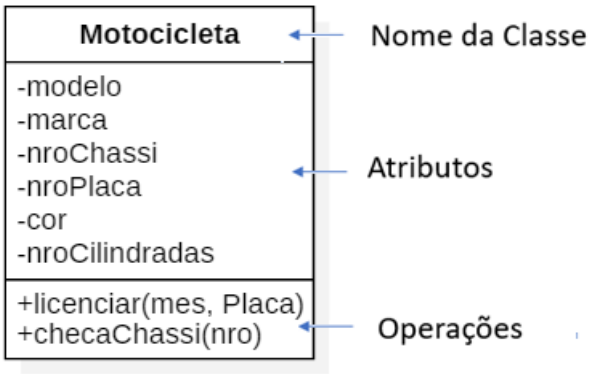
Moto de Passeio

O termo **identidade** significa que os objetos são inerentemente diferenciáveis por sua existência, e não por propriedades descritivas que eles possam ter.

CLASSE: UMA DESCRIÇÃO DE UM OU MAIS OBJETOS POR MEIO DE UM CONJUNTO UNIFORME DE CARACTERÍSTICAS E COMPORTAMENTOS

Um objeto é uma *instância ou ocorrência* de uma classe. Uma classe descreve um grupo de objetos com as mesmas características ou propriedades (atributos), comportamento (operações), tipos de relacionamento e semântica. Por exemplo, todas as motos das figuras anteriores pertencem à classe Motocicleta e têm as mesmas propriedades, que são: modelo, marca, número do chassi, placa, cor, número de cilindradas.

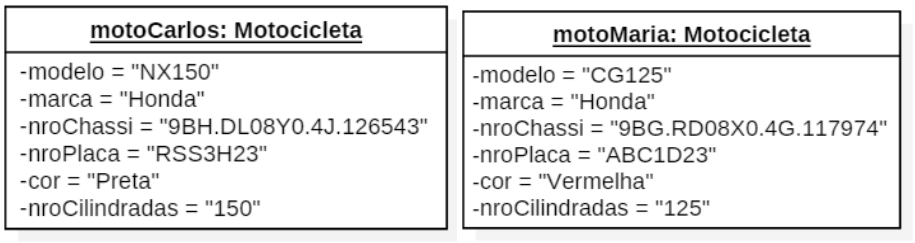
Essas são propriedades relevantes que descrevem as propriedades em comum dessas diferentes motocicletas. Todos esses diferentes objetos Motocicletas pertencem à mesma Classe Motocicleta, que poderia ser representada como é apresentada na figura a seguir por meio da notação de classe e da notação de objeto em UML:



Os objetos de uma classe possuem os mesmos atributos e comportamentos, a individualidade pode ser alcançada a partir das diferenças em seus valores de atributos e relacionamentos específicos com os outros objetos, compartilhando uma mesma finalidade semântica acima e além do requisito de terem atributos e comportamento comuns.

Os objetos da classe Motocicleta possuem características em comum, como **modelo, marca, nroChassi, nroPlaca, cor** e **nroCilindradas**. Você pode perceber

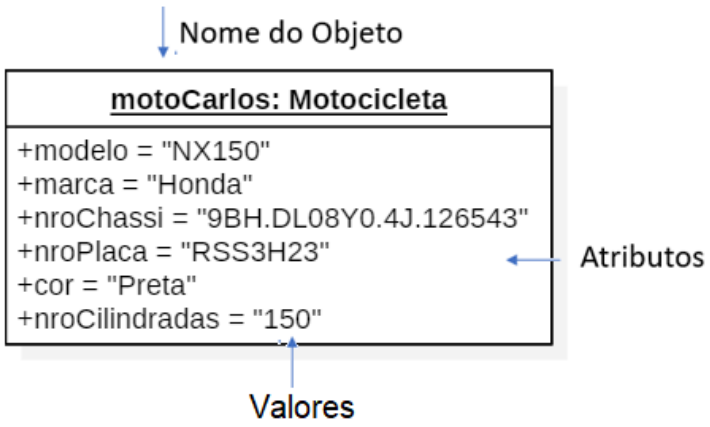
que os objeto da mesma classe também têm as mesmas operações que são: **licenciar** e **checaChassi**. Veja a figura a seguir. Você pode observar a representação de dois objetos diferentes que pertencem à mesma classe Motocicletas.



A moto de Carlos é uma motocicleta de modelo NX 150 da marca Honda; a motocicleta de Maria é uma CG 125 também da marca Honda, mas a motocicleta da Maria é da cor vermelha, e a motocicleta de Carlos é da cor preta.

Cada objeto "conhece" sua classe. A maioria das linguagens de programação pode determinar a classe de um objeto durante a execução. A classe de um objeto é uma propriedade implícita do objeto.

REPRESENTAÇÃO DE OBJETO



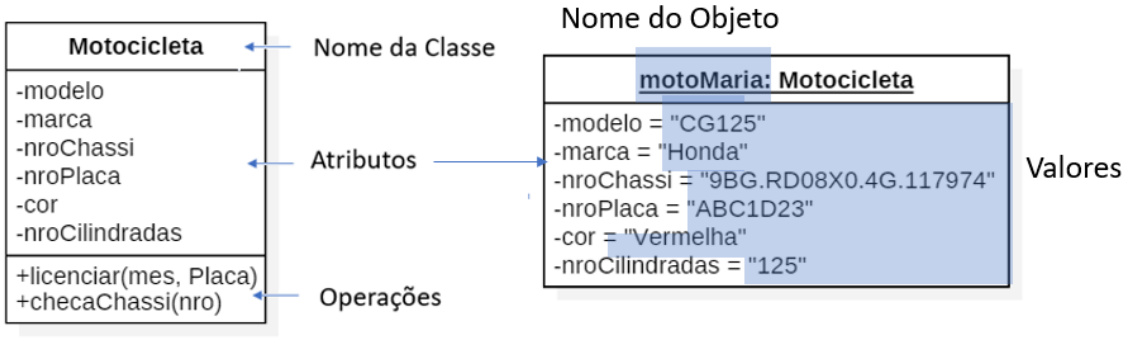
VALORES E ATRIBUTOS

Atributo é um dado (informação de estado) para o qual cada Objeto em uma Classe tem seu próprio valor.

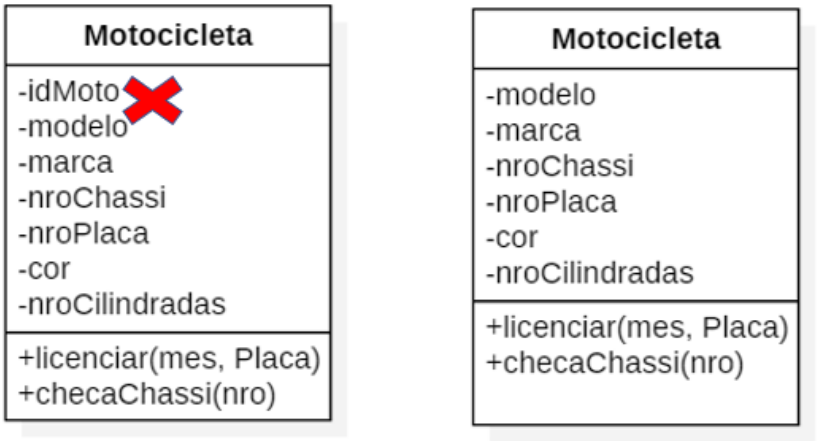
Um ***valor*** é um elemento dos dados. Você pode achar valores por meio do exame da documentação do problema. Um ***atributo*** é uma propriedade nomeada

de uma classe, que descreve um valor retido por cada objeto da classe. Pode-se encontrar atributos procurando adjetivos ou abstraindo valores típicos.

Analogia: objeto está para classe assim como valor está para atributo.



Alguns meios de implementação exigem que um objeto tenha um identificador exclusivo. Esses identificadores são implícitos em um modelo de classes – você não precisa e não deve listá-los explicitamente.



Identificadores de objeto (idMoto). Não liste identificadores, pois eles são implícitos nos modelos.

Os identificadores são um artefato computacional e não possuem significado intrínseco; os identificadores internos são uma conveniência de implementação, ao contrário do número do contribuinte, número da placa e número do telefone, pois possuem significado no mundo real.

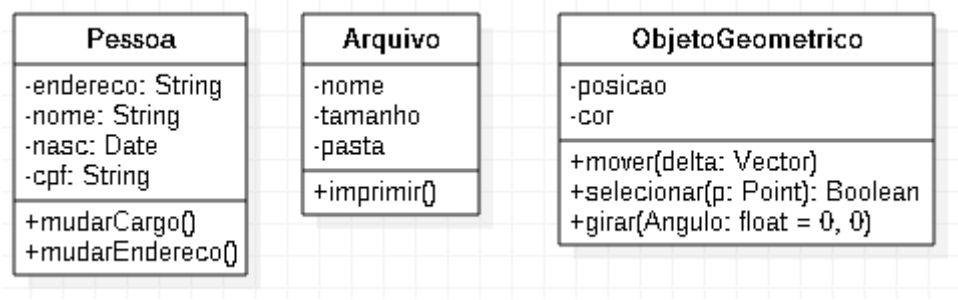
OPERAÇÕES E MÉTODOS

Uma ***operação*** é uma função ou um procedimento que pode ser aplicado a ou por objetos em uma classe. *Contratar, demitir e pagarDividendos* são operações

da classe *Empresa,* por exemplo. Cada operação possui um objeto destino como um argumento implícito. O comportamento da operação depende da classe de seu destino. Um objeto "conhece" sua classe e, portanto, a implementação correta da operação.

A mesma operação pode ser aplicada a muitas classes diferentes. Tal operação é ***polimórfica***, ou seja, a mesma operação assume diferentes formas em diferentes classes. Um ***método*** é a implementação de uma operação na classe. Uma operação pode ter argumentos além de seu objeto destino, esses argumentos podem ser valores ou outros objetos. A escolha de um método depende inteiramente da classe do objeto destino, e não de quaisquer argumentos objeto que uma operação possa ter.

É importante você observar estas três classes na figura a seguir: a classe *Pessoa* possui os atributos *nome* e *dataNascimento* e, como operações, *mudarCargo* e *mudarEndereço*; a classe *Arquivo* possui os atributos *nome, tamanho* e *pasta* (localização em um estrutura de armazenamento) e, como operação, *imprimir*; o *ObjetoGeometrico*, por sua vez, tem como atributos *posição* e *cor* e, como operações, *mover*, *selecionar* e *girar*. Tanto os atributos quanto as operações são características de cada uma dessas classes.



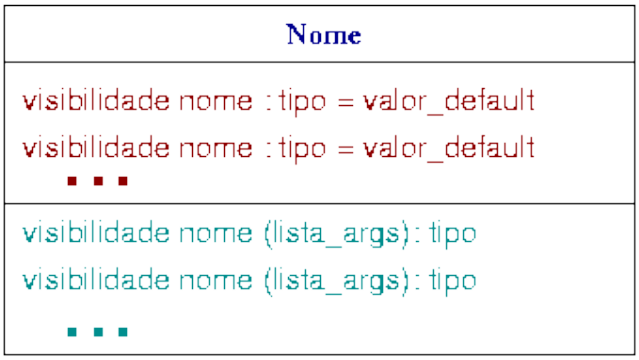
A ***Característica*** é uma palavra genérica para um atributo ou uma operação.

A notação UML lista operações no terceiro compartimento da caixa de classe. Nossa convenção é listar o nome da operação em fonte regular, alinhado à esquerda na caixa e com a primeira letra maiúscula. Os detalhes opcionais como uma lista de argumentos e tipo de resultado podem seguir cada nome de operação.

Parênteses delimitam uma lista de argumentos, separados por vírgulas; um sinal de dois-pontos precede o tipo do resultado. Uma lista de argumentos vazia entre parênteses mostra explicitamente que não existem argumentos; caso contrário, você não poderá concluir nada.

Resumo da Notação de Classes

Uma caixa representa uma classe e pode ter até três compartimento, de cima para baixo: nome de classe, lista de atributos e lista de operações; detalhes opcionais como tipo e valor-padrão podem seguir cada nome de atributo e, na lista de operações, a lista de argumentos e os tipos de resultado podem vir acompanhando cada nome de operação.



VISIBILIDADE

A visibilidade refere-se à capacidade de um método referenciar uma característica de outra classe e tem os seguintes valores possíveis: público, protegido, privado e pacote.

É importante ressaltar que o significado exato depende da linguagem de programação.

Por exemplo:

Visibilidade	UML	Descrição
Público	+	O método ou atributo pode ser acessado externamente por outro código.
Privada	-	O método ou atributo pode ser acessado somente dentro da própria classe.
Protegido	#	O método ou atributo pode ser acessado pela própria classe ou por classes-filhas (aquelas que herdam da classe base).
Pacote	~	O método ou atributo pode ser acessado pela própria classe ou por classes que participam do mesmo pacote.

Devem-se considerar várias questões quando se escolhe a visibilidade:

- Compreensão: Você precisa compreender todas as características públicas para entender as capacidades de uma classe, ao contrário, você pode ignorar características privadas, protegidas e de pacote, uma vez que elas são simplesmente uma conveniência de implementação.

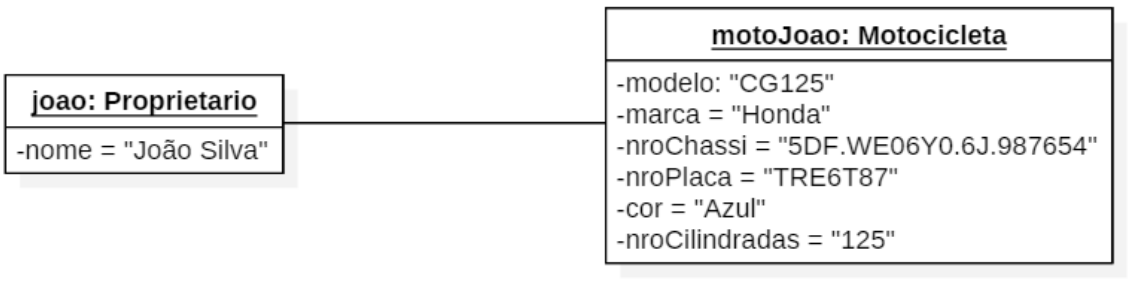
- Capacidade de extensão: muitas classes podem depender de métodos públicos, de modo que pode ser altamente prejudicial mudar sua assinatura (número de argumentos, tipos de argumentos, tipo de valor de retorno). Com menos classes dependentes de métodos privados, protegidos e de pacotes, há mais espaço para alterações.

- Contexto: Métodos privados, protegidos e de pacote podem se basear em precondições ou informações de estado criadas por outros métodos na classe. Aplicado fora do contexto, um método privado pode calcular resultados incorretos ou causar falha do objeto.

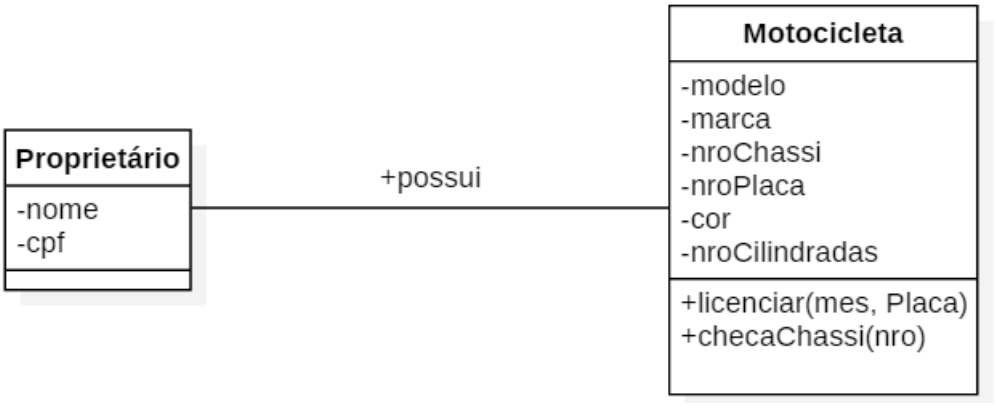
Ligação e Associação

São o meio de estabelecer relacionamento entre objetos e classes.

Uma ***ligação*** é uma conexão física ou conceitual entre objetos. Você pode observar um exemplo de ligação por meio do diagrama de objetos apresentado a seguir: João Silva *possui uma motocicleta CG125*.



Uma associação é uma descrição de um grupo de ligações com estrutura e semântica comuns. Agora, neste segundo diagrama de classes, você pode observar a representação de um Proprietário que possui Motocicleta.



As ligações de uma associação conectam objetos das mesmas classes. Uma associação descreve um conjunto de ligações em potencial, da mesma maneira que uma classe descreve um conjunto de objetos em potencial.

As ligações e associações normalmente aparecem como verbos em relatos de problemas, como você pode observar na figura: **Proprietário possui Motocicleta**.

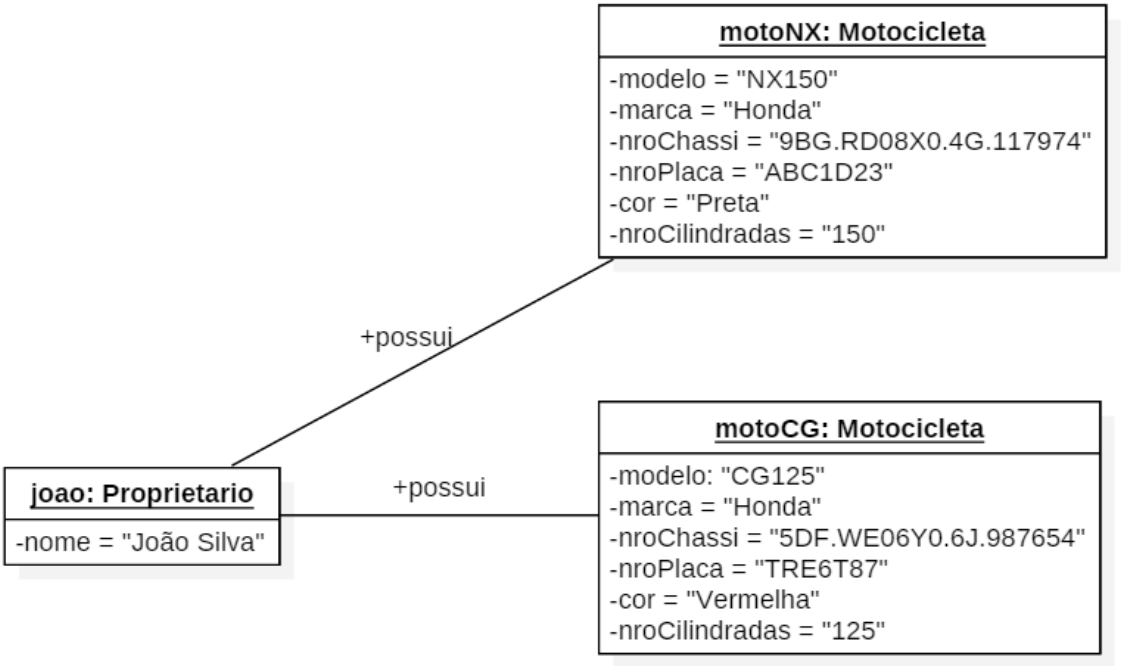
Entretanto, alguém pode ter mais de uma motocicleta, não? E uma motocicleta necessariamente deve pertencer alguém? Como podemos representar isso?

Multiplicidade

A *Multiplicidade* especifica o número de instâncias de uma classe que pode se relacionar a uma única instância de uma classe associada. A multiplicidade restringe o número de objetos relacionados. A literatura normalmente descreve a multiplicidade como sendo “um” ou “muitos”, mas, geralmente, ela é um subconjunto de inteiros não negativos finitos.

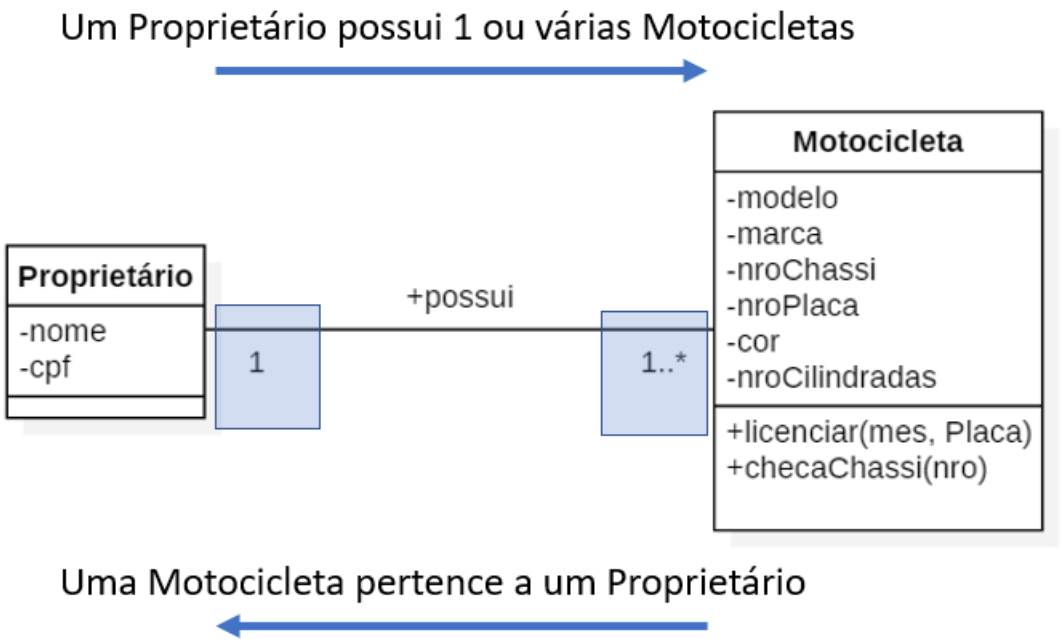
Assim, podemos ter o seguinte caso: João Silva possui uma CG125 vermelha e uma NX 150 preta, e esses duas motos pertencem a um único proprietário (uma moto não pode pertencer a duas entidades). Como podemos representar isso?

A figura a seguir representa esse cenário por meio de um diagrama de objetos: temos o objeto **joao** ligado aos objetos **motoNX** e **motoCG,** representando que o objeto joao tem duas instâncias de objetos do tipo Motocicleta.



Um proprietário poderia ter mais de duas motocicletas? Sim, ele poderia ter várias, não poderia?

Para demonstrar isso, podemos usar o símbolo "*", que representa várias ocorrências de um objeto em outro. Poderíamos, então, elaborar um diagrama de classes:



Ainda estudaremos mais sobre multiplicidade!

É importante você sempre lembrar que:

O **Diagrama de classes** oferece uma notação gráfica para modelar classes e seus relacionamentos, descrevendo, assim, possíveis objetos. Os diagramas de classes são úteis para a modelagem abstrata e para o projeto de programas reais. Eles são concisos, fáceis de entender e funcionam bem na prática; usamos diagramas de classes para representar a estrutura das aplicações.

Ocasionalmente, também usaremos **diagramas de objetos**. Um diagrama de objetos mostra objetos individuais e seus relacionamentos, são úteis para documentar casos de teste e discutir exemplos, um diagrama de classes corresponde a um conjunto infinito de diagramas de objetos.

REFERÊNCIAS

BEZERRA, E. *Princípios de análise e projeto de sistemas com UML*. 3. ed. Rio de Janeiro: Elsevier-Campus, 2015.

FOWLER, M. *UML essencial:* um breve guia para linguagem padrão. 3. ed. Porto Alegre: Bookman, 2005.

PAGE-JONES, M. *Fundamentos do desenho orientado a objeto com UML*. São Paulo: Pearson, 2001.