



*Grady BOOCHE*

*James RUMBAUGH*

*Ivar JACOBSON*

# UML

## GUIA DO USUÁRIO

*Segunda Edição, totalmente revista e atualizada*

*O mais avançado tutorial sobre Unified Modeling Language (UML), elaborado pelos próprios criadores da linguagem*



*Grady BOOCH*  
*James RUMBAUGH*  
*Ivar JACOBSON*

# UML

## GUIA DO USUÁRIO

*Segunda Edição, totalmente revista e atualizada*

*O mais avançado tutorial sobre Unified Modeling Language (UML), elaborado pelos próprios criadores da linguagem*



Abrange  
**UML 2.0**



© 2012, Elsevier Editora Ltda.

Todos os direitos reservados e protegidos pela Lei nº 9.610, de 19/02/1998.  
Nenhuma parte deste livro, sem autorização prévia por escrito da editora, poderá ser  
reproduzida ou transmitida sejam quais forem os meios empregados: eletrônicos,  
mecânicos, fotográficos, gravação ou quaisquer outros.

*Copidesque*: Cristina de Amorim Machado

*Revisão Gráfica*: Marco Antonio Corrêa e Juliana Vieira Machado

*Editoração Eletrônica*: SBNigri Artes e Textos Ltda.

*Epub*: SBNigri Artes e Textos Ltda.

**Elsevier Editora Ltda.**

**Conhecimento sem Fronteiras**

Rua Sete de Setembro, 111 – 16º andar  
20050-006 – Centro – Rio de Janeiro – RJ – Brasil

Rua Quintana, 753 – 8º andar  
04569-011 – Brooklin – São Paulo – SP – Brasil

Serviço de Atendimento ao Cliente  
0800-0265340  
[atendimento1@elsevier.com](mailto:atendimento1@elsevier.com)

Consulte nosso catálogo completo, os últimos lançamentos e os serviços exclusivos no site  
[www.elsevier.com.br](http://www.elsevier.com.br)

ISBN: 978-85-352-1784-1

ISBN (versão eletrônica): 978-85-352-8565-9

**Nota:** Muito zelo e técnica foram empregados na edição desta obra. No entanto, podem ocorrer erros de digitação, impressão ou dúvida conceitual. Em qualquer das hipóteses, solicitamos a comunicação ao nosso Serviço de Atendimento ao Cliente, para que possamos esclarecer ou encaminhar a questão.

Nem a editora nem o autor assumem qualquer responsabilidade por eventuais danos ou perdas a pessoas ou bens, originados do uso desta publicação.

CIP-BRASIL. CATALOGAÇÃO-NA-FONTE  
SINDICATO NACIONAL DOS EDITORES DE LIVROS, RJ

B715u B715uBooch, Grady

UML: guia do usuário / Grady Booch, James Rumbaugh, Ivar Jacobson; tradução de Fábio Freitas da Silva e Cristina de Amorim Machado. – Rio de Janeiro: Elsevier, 2012. – 12<sup>a</sup> reimpressão.

184 p.: il. ; 23 cm.

Tradução de: The Unified Modeling Language user guide, 2nd ed  
Inclui bibliografia

ISBN 978-85-352-1784-1

1. Software – Desenvolvimento. 2. UML (Computação). I.  
Rumbaugh,James. II. Jacobson, Ivar. III. Título.

05-  
3807.

CDD: 005.1  
CDU: 004.4

À minha querida esposa, Jan, e à minha afilhada,  
Elyse, ambas me completam.  
Grady Booch

# Prefácio

A UML, Linguagem Unificada de Modelagem, é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas complexos de software. A UML proporciona uma forma-padrão para a preparação de planos de arquitetura de projetos de sistemas, incluindo aspectos conceituais, tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, esquemas de bancos de dados e componentes de software reutilizáveis.

Este livro ensina a usar a UML de maneira efetiva e aborda a UML versão 2.0.

## OBJETIVOS

Neste livro, você alcançará os seguintes objetivos:

- » Aprender o que é UML, o que não é e por que a UML é relevante para o processo de desenvolvimento de sistemas de software complexos.
- » Dominar o vocabulário, as regras e as expressões da UML, além de aprender, de uma forma geral, como “falar” a linguagem de maneira efetiva.

- » Compreender como aplicar a UML para a solução de vários problemas básicos de modelagem.

O guia do usuário fornece uma referência ao uso de características específicas da UML. Porém, não se pretende que este guia seja um manual de referência abrangente sobre a UML; esse é o objetivo de um outro livro, *The Unified Modeling Language Reference Manual, Second Edition* (Rumbaugh, Jacobson, Booch, Addison-Wesley, 2005).

O guia do usuário descreve um processo de desenvolvimento em que a UML poderá ser utilizada. No entanto, este guia não deverá fornecer uma referência completa para esse processo; isso é tarefa de outro livro: *The Unified Software Development Process* (Jacobson, Booch, Rumbaugh, Addison-Wesley, 1999).

Por fim, este livro oferece dicas e sugestões relacionadas à utilização da UML para solucionar vários problemas básicos de modelagem, mas não ensina a modelar. Da mesma maneira que um guia de usuário referente a uma linguagem de programação ensina a usar a linguagem, mas não ensina a programar.

## PÚBLICO-ALVO

A UML interessa a qualquer pessoa envolvida na produção, na implantação e na manutenção de software. O guia do usuário está direcionado principalmente para os membros de equipes de desenvolvimento responsáveis pela criação de modelos UML. Porém, também será adequado a quem lê esses modelos, trabalhando em conjunto para compreender, construir, testar e gerar um sistema complexo de software. Apesar de essa descrição corresponder praticamente a quase todas as funções encontradas em empresas de desenvolvimento de software, o guia do usuário será relevante principalmente para analistas e usuários finais (que determinam a estrutura e o funcionamento necessários de um sistema), para arquitetos (que projetam sistemas capazes de atender a esses requisitos), para desenvolvedores (que

transformam essas arquiteturas em código executável), para o pessoal de controle de qualidade (que verifica e valida a estrutura e o funcionamento do sistema), para o pessoal de documentação (que cria e faz o catálogo de componentes) e para gerentes de projetos e de programas (que geralmente lidam com o caos, exercem uma liderança e orientação, além de coordenarem os recursos necessários para o fornecimento de sistemas bem-sucedidos).

O guia do usuário pressupõe um conhecimento básico de conceitos orientados a objetos. Alguma experiência em linguagens ou métodos de programação orientada a objetos será útil, mas não necessária.

## **COMO USAR ESTE LIVRO**

Para um desenvolvedor em primeiro contato com a UML, será mais adequada a leitura linear do guia do usuário. Uma atenção especial deverá ser dedicada ao [Capítulo 2](#), que apresenta um modelo conceitual da UML. Todos os capítulos estão estruturados de modo que o conteúdo de um capítulo é elaborado a partir do conteúdo do capítulo anterior, em uma progressão linear. No caso de desenvolvedores experientes que procuram respostas para os problemas mais comuns de modelagem que ocorrem quando se utiliza a UML, este livro poderá ser lido em qualquer ordem. Uma atenção especial deverá ser dedicada aos problemas básicos de modelagem apresentados em cada capítulo.

## **ORGANIZAÇÃO E CARACTERÍSTICAS ESPECIAIS**

O guia do usuário está organizado em sete partes:

- › [Parte 1: Primeiros passos](#)
- › [Parte 2: Modelagem estrutural básica](#)
- › [Parte 3: Modelagem estrutural avançada](#)

- › Parte 4: Modelagem comportamental básica
- › Parte 5: Modelagem comportamental avançada
- › Parte 6: Modelagem da arquitetura
- › Parte 7: Criação do pacote

O guia do usuário contém dois apêndices: um resumo das notações da UML e um resumo do Rational Unified Process. Também é fornecido um glossário de termos básicos, seguido de um índice. Cada capítulo trata do uso de uma característica específica da UML, e quase todos os capítulos estão organizados de acordo com as seguintes seções:

1. Primeiros passos
2. Termos e conceitos
3. Técnicas básicas de modelagem
4. Dicas e sugestões

A terceira seção apresenta e, em seguida, soluciona um conjunto de problemas básicos de modelagem. Visando facilitar a navegação pelo guia à procura desses casos de uso de UML, cada problema é identificado por um título específico, como no próximo exemplo.

## **PADRÕES DE ARQUITETURA DA MODELAGEM**

Todos os capítulos são iniciados com um resumo das características examinadas, como no próximo exemplo:

### **Neste capítulo**

- *Objetos, processos e threads ativos*
- *Modelagem de fluxos de controle múltiplos*
- *Modelagem de comunicação entre processos*
- *Elaboração de abstrações com threads seguros*

De forma semelhante, comentários e orientações gerais são apresentados como notas, como mostra o próximo exemplo.

**Nota:** As operações abstratas são mapeadas para o que a linguagem C++ chama de operações virtuais puras; operações-folha na UML são mapeadas para operações não virtuais no C++.

A UML é semanticamente rica. Como consequência, a apresentação de um recurso poderá envolver naturalmente outros recursos. Nesses casos, as referências cruzadas são fornecidas na margem esquerda, indicadas por seta preta, como nesta página.

► Os componentes são examinados no [Capítulo 25](#).

As figuras, destacadas em cinza, diferenciam o texto que explica um modelo do texto que faz parte do próprio modelo, que é sempre apresentado em preto. O código é assinalado com uma fonte especial, como neste exemplo.

Agradecimento. Os autores desejam agradecer a Bruce Douglass, Per Krol e Joaquin Miller por sua ajuda na revisão do manuscrito da segunda edição.

## BREVE HISTÓRIA DA UML

A primeira linguagem orientada a objetos é geralmente reconhecida como sendo a Simula-67, desenvolvida por Dahl e Nygaard, na Noruega, em 1967. Essa linguagem nunca teve muitos adeptos, mas seus conceitos foram a principal inspiração das linguagens que vieram depois. A linguagem Smalltalk tornou-se amplamente disponível no início dos anos 1980, seguida por outras linguagens orientadas a objetos, como Objective C, C++ e Eiffel, no fim dos anos 1980. As linguagens de modelagem orientadas a objetos surgiram nos anos 1980, à medida que o pessoal envolvido com metodologia, diante de um novo gênero de linguagens de programação orientadas a objetos e de aplicações cada vez mais complexas, começou a experimentar métodos alternativos de análise e projeto. A quantidade de métodos orientados a objetos aumentou de pouco mais de 10 para mais de 50 durante o período de

1989 a 1994. Muitos usuários desses métodos tiveram dificuldade para encontrar uma linguagem de modelagem capaz de atender inteiramente às suas necessidades, alimentando, assim, a chamada guerra de métodos. Alguns métodos ganharam destaque, entre eles o método de Booch, a Engenharia de Software Orientada a Objetos de Jacobson (OOSE – Object Oriented Software Engineering) e a Técnica de Modelagem de Objetos de Rumbaugh (OMT – Object Modeling Technique). Outros métodos importantes eram o Fusion e os de Shlaer-Mellor e Coad-Yourdon. Todos eram métodos completos, apesar de cada um conter pontos fortes e fracos. Em termos simples, o método Booch era expressivo principalmente durante as fases de projeto e construção de sistemas; o OOSE fornecia excelente suporte para os casos como uma maneira de controlar a captura de requisitos, a análise e o projeto em alto nível; o OMT era mais útil para análise e sistemas de informações com uso intensivo de dados.

Um conjunto de ideias começou a tomar forma por volta da metade da década de 1990, quando Grady Booch (Rational Software Corporation), Ivar Jacobson (Objectory) e James Rumbaugh (General Electrics) começaram a adotar ideias provenientes de cada um desses métodos, que vinham sendo reconhecidos mundialmente como os principais métodos orientados a objetos. Como principais autores dos métodos Booch, OOSE e OMT, nos sentimos motivados a criar uma linguagem unificada de modelagem por três motivos. Primeiro, nossos métodos já estavam evoluindo um em direção ao outro de maneira independente. Fazia sentido continuar essa evolução em conjunto e não separadamente, eliminando a possibilidade de diferenças casuais e desnecessárias que apenas confundiriam os usuários. Segundo, a unificação dos métodos traria alguma estabilidade ao mercado orientado a objetos, permitindo que os projetos tivessem como base uma linguagem madura de modelagem e que os produtores de ferramentas fornecessem recursos mais úteis. Terceiro, esperávamos que nossa colaboração proporcionasse aprimoramentos para os três métodos anteriores, auxiliando-nos a captar as lições aprendidas e a lidar com problemas que nenhum de nossos métodos conseguira manipular de maneira adequada até então.

Ao iniciar a unificação, estabelecemos três objetivos para o nosso trabalho:

1. Fazer a modelagem de sistemas, do conceito ao artefato executável, com a utilização de técnicas orientadas a objetos.
2. Incluir questões de escala, inerentes a sistemas complexos e de missão crítica.
3. Criar uma linguagem de modelagem a ser utilizada por seres humanos e por máquinas.

Projetar uma linguagem a ser utilizada em análise e projeto orientados a objetos não é muito diferente de criar uma linguagem de programação. Primeiro, é necessário delimitar o problema: a linguagem deverá abranger a especificação de requisitos? A linguagem deverá ser suficiente para permitir programação visual? Segundo, é preciso alcançar um equilíbrio entre expressividade e simplicidade. Sendo muito simples, a linguagem limitaria a amplitude de problemas que poderiam ser solucionados; sendo muito complexa, a linguagem sobrecarregaria o desenvolvedor mortal. No caso da unificação de métodos existentes, também era necessário levar em consideração a base já instalada. Implementando muitas alterações, confundiríamos os usuários existentes; resistindo a tornar a linguagem mais avançada, perderíamos a oportunidade de conquistar um conjunto bem maior de usuários e de tornar a linguagem mais simples. A definição da UML busca obter os melhores resultados possíveis em cada uma dessas áreas.

Os esforços para a criação da UML se iniciaram oficialmente em outubro de 1994, quando Rumbaugh se juntou a Booch na Rational. O foco inicial de nosso projeto era a unificação dos métodos Booch e OMT. O esboço da versão 0.8 do Método Unificado (como então era chamado) foi lançado em outubro de 1995. Mais ou menos na mesma época, Jacobson se associou à Rational e o escopo do projeto da UML foi expandido com a finalidade de incorporar o OOSE. Nossos esforços resultaram no lançamento dos documentos da versão 0.9 da UML em junho de 1996. Ao longo de 1996, solicitamos e recebemos retornos da comunidade de engenharia de software em geral. Nesse período, também se tornou bem claro que muitas empresas

de software consideravam a UML um recurso estratégico para seus negócios. Estabelecemos um consórcio UML com várias empresas que desejavam dedicar recursos com o propósito de trabalhar a favor de uma definição mais forte e completa da UML. Entre os parceiros que contribuíram para a definição da UML 1.0, encontravam-se Digital Equipment Corporation, Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instruments e Unisys. Tal colaboração resultou na UML 1.0, uma linguagem de modelagem bem-definida, expressiva, poderosa e que poderia ser aplicada a uma grande variedade de tipos de problemas. Mary Loomis foi de grande ajuda ao convencer o OMG (Object Management Group) a solicitar uma proposta (RFP) de linguagem padrão de modelagem. A UML 1.0 foi oferecida para padronização ao OMG em janeiro de 1997, em resposta à sua RFP.

Entre janeiro e julho de 1997, o grupo original de parceiros se expandiu, passando a incluir praticamente todos os participantes e colaboradores da resposta inicial ao OMG, entre os quais se encontravam Andersen Consulting, Ericsson, ObjectTime Limited, Platinum Technology, PTech, Reich Technologies, Softeam, Sterling Software e Taskon. Um grupo de tarefas em semântica foi formado, liderado por Cris Kobryn da MCI Systemhouse e administrado por Ed Eykholt da Rational, com o propósito de formalizar a especificação da UML e de integrar a linguagem a outros esforços de padronização. Uma versão revisada da UML (versão 1.1) foi oferecida para padronização ao OMG em julho de 1997. Em setembro do mesmo ano, essa versão foi aceita pela ADTF (Analysis and Design Task Force) e pelo Architecture Board do OMG e, a seguir, submetida ao voto de todos os membros do OMG. A UML 1.1 foi adotada pelo OMG em 14 de novembro de 1997.

Por vários anos, a manutenção da UML foi assumida pela RTF (Revision Task Force) do OMG, que produziu as versões 1.3, 1.4 e 1.5. De 2000 a 2003, um novo e maior grupo de parceiros produziu e atualizou a especificação da UML, versão 2.0. Essa versão foi revista ao longo de um ano por uma FTF (Finalization Task Force) coordenada por Bran Selic, da

IBM, e a versão oficial da UML 2.0 foi adotada pelo OMG no início de 2005. A UML 2.0 é uma revisão importante da UML 1 e inclui muitos recursos adicionais. Além disso, muitas alterações foram feitas nos construtos anteriores baseadas na experiência com a versão anterior. Os documentos oficiais da especificação da UML encontram-se no site do OMG, em [www.omg.org](http://www.omg.org).

A UML é o trabalho de várias pessoas, e as ideias que ali se encontram vêm de muitos trabalhos anteriores. Seria um trabalho importante de pesquisa histórica reconstruir uma lista completa das fontes e ainda mais difícil identificar os muitos precursores que influenciaram a UML, de maneira mais ou menos relevante. Como em qualquer pesquisa científica e prática de engenharia, a UML é uma pequena colina sobre uma grande montanha de experiência anterior.

# SUMÁRIO

## **PARTE 1: PRIMEIROS PASSOS**

### **Capítulo 1 – Por que fazer a Modelagem?**

A importância da modelagem

Princípios da modelagem

A modelagem orientada a objetos

### **Capítulo 2 – Introdução à UML**

Uma visão geral da UML

A UML é uma linguagem

A UML é uma linguagem para visualização

A UML é uma linguagem para especificação

A UML é uma linguagem para documentação

Onde a UML pode ser utilizada?

Um modelo conceitual da UML

Blocos de construção da UML

Itens da UML

Regras da UML

Mecanismos básicos da UML

Arquitetura

O ciclo de vida do desenvolvimento do software

### **Capítulo 3 – Hello, World!**

[Principais abstrações](#)  
[Mecanismos](#)  
[Artefatos](#)

## **PARTE 2: MODELAGEM ESTRUTURAL BÁSICA**

### **Capítulo 4 – Classes**

[Primeiros passos](#)  
[Termos e conceitos](#)  
    [Atributos](#)  
    [Responsabilidades](#)  
    [Outras características](#)  
[Técnicas básicas de modelagem](#)  
    [A modelagem de itens que não são software](#)  
[Dicas e sugestões](#)

### **Capítulo 5 – Relacionamentos**

[Primeiros passos](#)  
[Termos e conceitos](#)  
    [Dependência](#)  
    [Generalização](#)  
    [Associação](#)  
    [Outras características](#)  
    [Estilos de desenho](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

### **Capítulo 6 – Mecanismos**

[Primeiros passos](#)  
[Termos e conceitos](#)  
    [Notas](#)  
    [Estereótipos](#)  
[Valores atribuídos](#)

[Restrições](#)

[Elementos-padrão](#)

[Perfis](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **[Capítulo 7 – Diagramas](#)**

[Primeiros passos](#)

[Termos e conceitos](#)

[Diagramas estruturais](#)

[Diagramas comportamentais](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **[Capítulo 8 – Diagramas de Classes](#)**

[Primeiros passos](#)

[Termos e conceitos](#)

[Propriedades básicas](#)

[Conteúdo](#)

[Usos básicos](#)

[Técnicas básicas de modelagem](#)

[Engenharia direta e reversa](#)

[Dicas e sugestões](#)

## **[PARTE 3: MODELAGEM ESTRUTURAL AVANÇADA](#)**

### **[Capítulo 9 – Classes Avançadas](#)**

[Primeiros passos](#)

[Termos e conceitos](#)

[Classificadores](#)

[Visibilidade](#)

[Escopo de instância e de estática](#)

[Elementos abstratos, raiz, folha e polimórficos](#)

[Multiplicidade](#)  
[Atributos](#)  
[Operações](#)  
[Classes template](#)  
[Elementos-padrão](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

## **Capítulo 10 – Relacionamentos Avançados**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Dependência](#)  
[Generalização](#)  
[Associação](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

## **Capítulo 11 – Interfaces, Tipos e Funções**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Nomes](#)  
[Operações](#)  
[Relacionamentos](#)  
[Compreendendo uma interface](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

## **Capítulo 12 – Pacotes**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Nomes](#)  
[Elementos pertinentes](#)  
[Visibilidade](#)

[Importação e exportação](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

## **Capítulo 13 – Instâncias**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Abstrações e instâncias](#)  
[Tipos](#)  
[Nomes](#)  
[Operações](#)  
[Estado](#)  
[Outras características](#)  
[Elementos-padrão](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

## **Capítulo 14 – Diagramas de Objetos**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Propriedades comuns](#)  
[Conteúdo](#)  
[Usos comuns](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

## **Capítulo 15 – Componentes**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Componentes e interfaces](#)  
[Substituição](#)  
[Organização de componentes](#)  
[Portas](#)

Estrutura interna  
TÉCNICAS BÁSICAS de MODELAGEM  
Dicas e sugestões

## **PARTE 4: MODELAGEM BÁSICA DE COMPORTAMENTO**

### **Capítulo 16 – Interações**

Primeiros passos  
Termos e conceitos  
Contexto  
Objetos e papéis  
Vínculos e conectores  
Mensagens  
Sequenciamento  
Criação, modificação e destruição  
Representação  
Técnicas básicas de modelagem  
Dicas e sugestões

### **Capítulo 17 – Casos de Uso**

Primeiros passos  
Termos e conceitos  
Assunto  
Nomes  
Casos de uso e atores  
Casos de uso e fluxo de eventos  
Casos de uso e cenários  
Casos de uso e colaborações  
Organização dos casos de uso  
Outras características  
Técnicas básicas de modelagem  
Dicas e sugestões

## **Capítulo 18 – Diagramas de Casos de Uso**

Primeiros passos

Termos e conceitos

Propriedades comuns

Conteúdo

Notação

Usos comuns

Técnicas básicas de modelagem

Dicas e sugestões

## **Capítulo 19 – Diagramas de Interação**

Primeiros passos

Termos e conceitos

Propriedades comuns

Conteúdo

Diagramas de sequências

Controle estruturado nos diagramas de sequências

Diagramas de atividades aninhadas

Diagramas de comunicação

Equivalência semântica

Usos comuns

Técnicas básicas de modelagem

Dicas e sugestões

## **Capítulo 20 – Diagramas de Atividades**

Primeiros passos

Termos e conceitos

Propriedades comuns

Conteúdo

Ações e nós de atividades

Fluxos de controle

Ramificação

[Bifurcação e união](#)  
[Raias de natação](#)  
[Fluxo de objetos](#)  
[Regiões de expansão](#)  
[Usos comuns](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

## **PARTE 5: MODELAGEM COMPORTAMENTAL AVANÇADA**

### **Capítulo 21 – Eventos e Sinais**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Tipos de eventos](#)  
[Sinais](#)  
[Eventos de chamada](#)  
[Eventos de tempo e alteração](#)  
[Eventos enviar e receber](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

### **Capítulo 22 – Máquinas de Estados**

[Primeiros passos](#)  
[Termos e conceitos](#)  
[Contexto](#)  
[Estados](#)  
[Transições](#)  
[Transições e estados avançados](#)  
[Subestados](#)  
[Técnicas básicas de modelagem](#)  
[Dicas e sugestões](#)

### **Capítulo 23 – Processos e Threads**

[Primeiros passos](#)

[Termos e conceitos](#)

[Fluxo de Controle](#)

[Classes e eventos](#)

[Comunicação](#)

[Sincronização](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **Capítulo 24 – Tempo e Espaço**

[Primeiros passos](#)

[Termos e conceitos](#)

[Tempo](#)

[Localização](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **Capítulo 25 – Diagramas de Estados**

[Primeiros passos](#)

[Termos e conceitos](#)

[Propriedades comuns](#)

[Conteúdo](#)

[Usos comuns](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **PARTE 6: MODELAGEM DA ARQUITETURA**

### **Capítulo 26 – Artefatos**

[Primeiros passos](#)

[Termos e conceitos](#)

[Nomes](#)

[Artefatos e classes](#)

[Tipos de artefatos](#)

[Elementos-padrão](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **[Capítulo 27 – Implantação](#)**

[Primeiros passos](#)

[Termos e conceitos](#)

[Nomes](#)

[Nós e artefatos](#)

[Organização dos nós](#)

[Conexões](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **[Capítulo 28 – Colaborações](#)**

[Primeiros passos](#)

[Termos e conceitos](#)

[Nomes](#)

[Estrutura](#)

[Comportamento](#)

[Organização das colaborações](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **[Capítulo 29 – Padrões e Frameworks](#)**

[Primeiros passos](#)

[Termos e conceitos](#)

[Padrões De Arquitetura](#)

[Mecanismos](#)

[Frameworks](#)

[Técnicas básicas de modelagem](#)

[Dicas e sugestões](#)

## **Capítulo 30 – Diagramas de Artefatos**

Primeiros passos

Termos e conceitos

Propriedades comuns

Conteúdo

Usos comuns

Técnicas básicas de modelagem

Dicas e sugestões

## **Capítulo 31 – Diagramas de Implantação**

Primeiros passos

Termos e conceitos

Propriedades comuns

Conteúdo

Usos Comuns

Técnicas básicas de modelagem

Dicas e sugestões

## **Capítulo 32 – Sistemas e Modelos**

Primeiros passos

Termos e conceitos

Sistemas e subsistemas

Modelos e visões

Rastreamento

Técnicas básicas de modelagem

Dicas e sugestões

## **PARTE 7: CONCLUSÃO**

## **Capítulo 33 – Aplicando a UML**

Fazendo a transição para a UML

Para onde ir agora

## APÊNDICE A: NOTAÇÃO DA UML

### Itens

- [Itens estruturais](#)
- [Itens comportamentais](#)
- [Itens de agrupamento](#)
- [Itens anotacionais](#)

### Relacionamentos

- [Dependência](#)
- [Associação](#)
- [Generalização](#)

### Extensão

### Diagramas

## APÊNDICE B: RATIONAL UNIFIED PROCESS

### Características do processo

### Fases e iterações

- [Fases](#)
- [Iterações](#)
- [Ciclos de desenvolvimento](#)

### Disciplinas

### Artefatos

- [Modelos](#)
- [Outros artefatos](#)

## GLOSSÁRIO

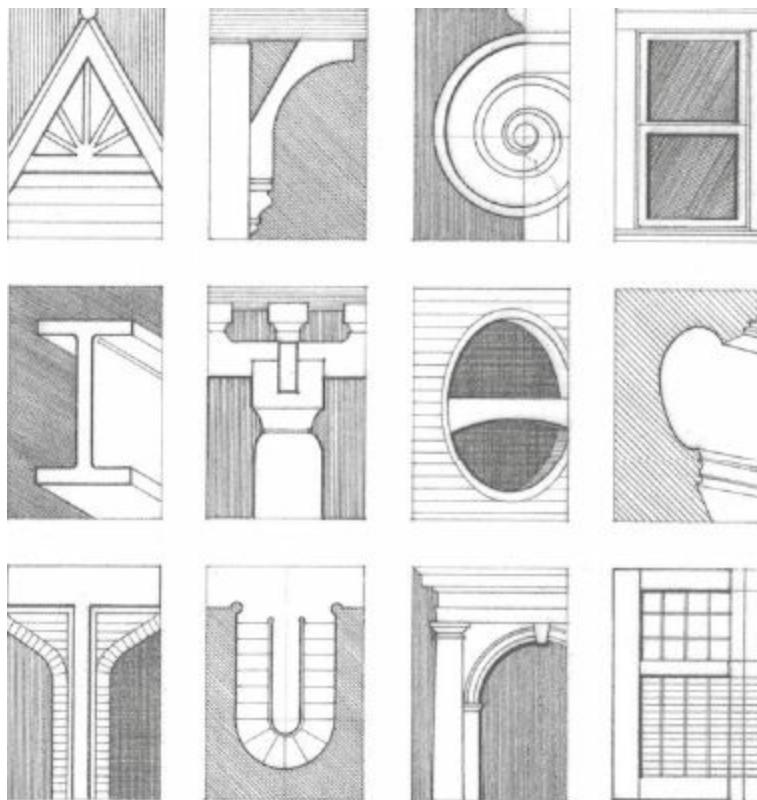
P A R T E

---

1

---

# PRIMEIROS PASSOS



CAPÍTULO

---

1

---

# Por que fazer a Modelagem?

## Neste capítulo

- › *A importância da modelagem*
- › *Os quatro princípios da modelagem*
- › *A esquematização essencial de sistemas de software*
- › *A modelagem orientada a objetos*

**U**ma empresa de software bem-sucedida é aquela que fornece software de qualidade e capaz de atender às necessidades dos respectivos usuários. Uma empresa que consiga desenvolver esse software de maneira previsível e em determinado período, com utilização eficiente e eficaz de recursos, será uma empresa com um negócio viável.

Existe uma implicação importante nesta mensagem: o principal produto de uma equipe de desenvolvimento não são documentos bonitos, reuniões sofisticadas, ótimos slogans ou linhas de código-fonte merecedoras do Prêmio Pulitzer. O principal produto é um bom software capaz de satisfazer às necessidades de seus usuários e respectivos negócios. Tudo o mais é secundário.

Infelizmente, muitas empresas de software confundem “secundário” com “irrelevante”. Para entregar um software que satisfaça ao propósito pretendido, será preciso reunir-se e interagir com os usuários de uma maneira disciplinada, com a finalidade de expor os requisitos reais do sistema. Para desenvolver software de qualidade duradoura, será necessário criar uma arquitetura de fundação sólida que aceite modificações. Para desenvolver software de forma rápida, eficiente e efetiva, com o mínimo de desperdício e de retrabalho de software, será preciso dispor das pessoas certas, das

ferramentas adequadas e do enfoque correto. Para fazer tudo isso de maneira previsível e consistente, com uma avaliação dos custos reais do sistema, você precisará de um processo seguro de desenvolvimento que possa ser adaptado às novas necessidades de seu negócio e de sua tecnologia.

A modelagem é uma parte central de todas as atividades que levam à implantação de um bom software. Construímos modelos para comunicar a estrutura e o comportamento desejados do sistema. Construímos modelos para visualizar e controlar a arquitetura do sistema. Construímos modelos para compreender melhor o sistema que estamos elaborando, muitas vezes expondo oportunidades de simplificação e reaproveitamento. Construímos modelos para gerenciar os riscos.

## **A IMPORTÂNCIA DA MODELAGEM**

Para construir uma casa para seu cachorro, você poderá começar juntando uma pilha de tábuas, alguns pregos e algumas ferramentas básicas, como martelo, serrote e metro. Em poucas horas, com pequeno planejamento prévio, provavelmente a casa de cachorro estará pronta, razoavelmente funcional, e por certo você será capaz de fazer tudo isso sem precisar da ajuda de ninguém. Desde que a nova casa seja suficientemente grande e não haja muitas goteiras, seu cão ficará feliz. Se não der certo, sempre será possível fazer tudo de novo ou arrumar um cachorro menos exigente.

Para construir uma casa para sua família, você poderá começar juntando uma pilha de tábuas, alguns pregos e algumas ferramentas básicas, mas precisará de um tempo muito maior e, com certeza, sua família será mais exigente que o cachorro. Nesse caso, a menos que já tenha construído várias casas, será melhor fazer um planejamento detalhado antes de prender o primeiro prego ou iniciar a fundação. Pelo menos, você fará alguns desenhos rápidos da aparência desejada para sua futura casa. Se quiser construir uma casa de qualidade, que atenda às necessidades de sua família e respeite os códigos de construção civil da região, também será preciso desenhar alguns esboços de projetos, com a finalidade de pensar sobre o uso pretendido para

cada cômodo e detalhes práticos de energia, circulação e encanamento. A partir desses planos, você poderá começar a fazer uma estimativa razoável da quantidade de tempo e de material necessários para essa tarefa. Embora seja humanamente possível construir uma casa sozinho, você logo descobrirá que será muito mais eficiente trabalhar com outras pessoas, talvez terceirizando vários serviços básicos ou comprando material pré-fabricado. Desde que você se mantenha fiel aos planos e permaneça dentro dos limites de tempo e custos, provavelmente sua família ficará satisfeita. Se não der certo, a solução não será trocar de família. Portanto, será melhor definir as expectativas desde o início e gerenciar qualquer modificação com muita cautela.

Para construir um prédio comercial com vários andares, não será uma boa ideia começar com uma pilha de tábuas, alguns pregos e algumas ferramentas básicas. Como provavelmente você estará usando o dinheiro de outras pessoas, como os acionistas da empresa, elas exigirão saber o tamanho, a forma e o estilo do futuro prédio. Muitas vezes, essas pessoas mudarão de ideia, mesmo depois de iniciada a construção. Valerá a pena fazer um planejamento rigoroso, pois os custos de qualquer erro serão altos. Você será apenas uma parte de um grupo bem maior, responsável pelo desenvolvimento e pela entrega do prédio. Assim, a equipe precisará de todos os modelos e esboços do projeto para que seus membros comuniquem-se uns com os outros. Desde que você consiga as pessoas certas e as ferramentas adequadas, além de gerenciar de maneira ativa o processo de transformação de um conceito de arquitetura em realidade, provavelmente acabará obtendo um prédio que satisfará seus futuros ocupantes. Caso pretenda continuar construindo prédios, você tentará encontrar um equilíbrio entre os desejos dos futuros ocupantes e a realidade das tecnologias de construção, além de manter um relacionamento profissional com os demais integrantes de sua equipe, nunca pondo-os em risco, nem exigindo tanto que eles acabem exaustos ou doentes.

Curiosamente, muitas empresas de desenvolvimento de software começam querendo construir prédios altos como se estivessem fazendo uma casinha de cachorro.

De vez em quando, a sorte ajuda. Com as pessoas certas no momento adequado e com todos os planetas em um alinhamento favorável, talvez, e apenas talvez, você consiga fazer com que sua equipe crie um produto de software capaz de encantar seus usuários. Tipicamente, entretanto, você não conseguirá todas as pessoas certas (que já estarão superocupadas), nunca será o momento adequado (ontem teria sido muito melhor) e dificilmente os planetas estarão alinhados (movendo-se de uma maneira que você não poderá modificar). Considerando a demanda cada vez maior de desenvolvimento de software nesta época de internet, as equipes de desenvolvimento costumam restringir-se à tarefa que realmente sabem fazer bem – digitar linhas de código. Esforços heroicos de programação já viraram lendas nessa indústria e, de uma forma geral, tem-se a impressão de que apenas trabalhar bastante seja a resposta adequada a qualquer crise que ocorra no processo de desenvolvimento. Porém, esse trabalho não produzirá necessariamente as linhas de código corretas, e projetos são de tal grandeza que o acréscimo de mais horas de trabalho não será suficiente para a sua conclusão.

Se você realmente quiser construir softwares equivalentes a uma casa ou a um prédio, o problema não se restringirá a uma questão de escrever uma grande quantidade de software – de fato, o segredo estará em criar o código correto e pensar em como será possível elaborar menos software. Isso faz com que o desenvolvimento de software de qualidade se torne uma questão de arquitetura, processo e ferramentas. Ainda assim, muitos projetos são iniciados parecendo uma casa de cachorro, mas crescem com a grandeza de um prédio simplesmente porque são vítimas de seu próprio sucesso. Chega um momento em que, caso não tenham sido consideradas questões referentes à arquitetura, a processos e a ferramentas, a casa de cachorro, agora ampliada para um grande prédio, sucumbirá ao seu próprio peso. Qualquer erro na casa de cachorro poderá deixar seu cão insatisfeito. A falha na construção de um grande prédio afetará materialmente seus ocupantes.

Projetos de software malsucedidos falham em relação a aspectos únicos e específicos de cada projeto, mas todos os projetos bem-sucedidos são semelhantes em diversos aspectos. Existem muitos elementos que contribuem

para uma empresa de software de sucesso; um desses componentes é a utilização da modelagem.

A modelagem é uma técnica de engenharia aprovada e bem-aceita. Construímos modelos de arquitetura de casas e de grandes prédios para auxiliar seus usuários a visualizar qual será o produto final. Podemos até elaborar modelos matemáticos com a finalidade de analisar quais serão os efeitos que ventos e tremores de terra poderão causar aos prédios que você construir.

A modelagem não faz parte apenas da indústria de construção. Seria inconcebível fornecer um novo avião ou automóvel sem primeiro construir os respectivos modelos – desde modelos de computadores, modelos físicos de túneis de vento até protótipos em larga escala. Novos dispositivos elétricos, desde microprocessadores a sistemas de telefonia, requerem algum grau de modelagem com o propósito de permitir uma melhor compreensão do sistema e a comunicação dessas ideias a outras pessoas. Na indústria cinematográfica, a elaboração de roteiros, que é uma forma de modelagem, tem um papel central em qualquer produção. Nas áreas de sociologia, economia e administração comercial, construímos modelos para validar nossas teorias, ou experimentar novas, com riscos e custos mínimos.

Afinal, o que é um modelo? Falando de uma maneira simples:

- *Um modelo é uma simplificação da realidade.*

Os modelos fornecem uma planta do projeto de um sistema. Os modelos poderão abranger planos detalhados, assim como planos mais gerais com uma visão panorâmica do sistema considerado. Um bom modelo inclui aqueles componentes que têm ampla repercussão e omite os componentes menores que não são relevantes em determinado nível de abstração. Todos os sistemas podem ser descritos sob diferentes aspectos, com a utilização de modelos distintos, e cada modelo será, portanto, uma abstração semanticamente específica do sistema. Os modelos podem ser estruturais, dando ênfase à

organização do sistema, ou podem ser comportamentais, dando ênfase à dinâmica do sistema.

Por que fazer a modelagem? Existe um motivo fundamental.

- ➡ *Construímos modelos para compreender melhor o sistema que estamos desenvolvendo.*

Com a modelagem, alcançamos quatro objetivos.

1. Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja.
2. Os modelos permitem especificar a estrutura ou o comportamento de um sistema.
3. Os modelos proporcionam um guia para a construção do sistema.
4. Os modelos documentam as decisões tomadas.

- ➡ *O Capítulo 2 apresenta como a UML direciona esses quatro aspectos.*

A modelagem não se restringe a grandes sistemas. Até os softwares equivalentes a casas de cachorro poderão receber os benefícios da modelagem. Porém, é absolutamente verdadeiro que, quanto maior e mais complexo for o sistema, maior será a importância da modelagem, por uma razão muito simples:

- ➡ *Construímos modelos de sistemas complexos porque não é possível compreendê-los em sua totalidade.*

Existem limites para a capacidade humana de compreender complexidades. Com a ajuda da modelagem, delimitamos o problema que estamos estudando, restringindo nosso foco a um único aspecto por vez. Em essência esse é o procedimento de “dividir-e-conquistar”, do qual Edsger Dijkstra falava há anos: ataque um problema difícil, dividindo-o em vários problemas menores que você pode solucionar. Além disso, com o auxílio da modelagem, somos capazes de ampliar o intelecto humano. Um modelo

escolhido de maneira adequada permitirá a quem usa a modelagem trabalhar em níveis mais altos de abstração.

Afirmar que a modelagem deveria ser feita não significa necessariamente que ela será utilizada. De fato, vários estudos sugerem que, em sua maioria, as empresas de software usam pouca, quando usam alguma modelagem formal. Compare o uso da modelagem com a complexidade de um projeto e você descobrirá que, quanto mais simples for o projeto, menos provável será a utilização da modelagem formal.

A palavra funcional aqui utilizada é “formal”. Na verdade, inclusive nos projetos mais simples, os desenvolvedores realizam alguma modelagem, apesar da maneira muito informal. O desenvolvedor poderá rabiscar uma ideia em um quadro-negro ou em uma folha de papel, com a finalidade de visualizar parte do sistema, ou a equipe poderá usar cartões CRC<sup>1</sup> para trabalhar em um cenário específico ou estruturar determinado mecanismo. Não há nada errado nesses modelos. Se funcionarem, sem dúvida deverão ser utilizados. Porém, esses modelos informais costumam ter um propósito específico e não oferecem uma linguagem básica que possa ser compartilhada com outras pessoas facilmente. Assim como existe uma linguagem básica para esboços de projetos na indústria de construção, na engenharia elétrica e na modelagem matemática, também uma empresa de desenvolvimento de software poderá usufruir os benefícios de utilizar uma linguagem básica para a modelagem de software.

Qualquer projeto será beneficiado pelo uso de algum tipo de modelagem. Mesmo nos casos de softwares descartáveis, em que algumas vezes é mais efetivo jogar fora programas inadequados em função da produtividade oferecida por linguagens visuais de programação, a modelagem pode ser útil para que a equipe de desenvolvimento visualize melhor o plano de seu sistema, permitindo que eles desenvolvam mais rapidamente ao ajudá-los a construir, diretamente, a coisa certa. Quanto mais complexo for o sistema, maior será a probabilidade de ocorrência de erros ou de construção de itens errados, caso não haja qualquer modelagem. Todos os sistemas úteis e

interessantes apresentam uma tendência natural para se transformar em algo mais complexo ao longo do tempo. Portanto, ainda que considere não ser preciso fazer a modelagem hoje, à medida que o seu sistema evoluir, você se arrependerá dessa decisão, quando for tarde demais.

## PRINCÍPIOS DA MODELAGEM

O uso da modelagem tem uma rica história em todas as disciplinas de engenharia. Essa experiência sugere quatro princípios básicos de modelagem. Em primeiro lugar:

- *A escolha dos modelos a serem criados tem profunda influência sobre a maneira como um determinado problema é atacado e como uma solução é definida.*

Em outras palavras, escolha bem os seus modelos. Os modelos corretos esclarecerão brilhantemente os problemas de desenvolvimento mais complicados, proporcionando conclusões que simplesmente não seriam possíveis de outra maneira; modelos inadequados causarão confusões, desviando a atenção para questões irrelevantes.

Deixando de lado o desenvolvimento de software por um instante, suponha que você esteja tentando solucionar um problema de física quântica. Certos problemas, como a interação de fótons no espaço-tempo, implicam uma matemática maravilhosamente complexa. Escolha um modelo que não seja o de cálculo e imediatamente essa complexidade inerente se tornará manipulável. Nesse campo, esse é precisamente o valor dos diagramas de Feynmann, capazes de oferecer representações gráficas de problemas bastante complexos. De modo semelhante, em um domínio inteiramente diferente, suponha que você esteja construindo um novo prédio e esteja interessado em saber como ele se comportará quando houver fortes ventanias. Construindo um modelo físico e submetendo-o a testes de túneis de vento, você aprenderá algumas coisas interessantes, apesar de os materiais em

escalas menores não se flexionarem exatamente como em escalas maiores. Assim, se elaborar um modelo matemático e depois submetê-lo a simulações, você aprenderá algumas coisas diferentes e provavelmente também será capaz de trabalhar com um número maior de novos cenários do que se estivesse utilizando modelos físicos. A partir de testes contínuos e rigorosos com seus modelos, você acabará obtendo um nível de confiança muito superior em relação ao fato de que o sistema, cuja modelagem foi realizada, de fato se comportará da maneira esperada no mundo real.

Em relação aos softwares, a escolha de modelos poderá ser modificada, de maneira significativa, de acordo com sua visão de mundo. Construindo um sistema a partir da perspectiva de um desenvolvedor de bancos de dados, provavelmente você atribuirá o foco a modelos de relacionamentos entre entidades, cujo comportamento tende a privilegiar procedimentos armazenados e os eventos que os iniciam. Construindo um sistema a partir da perspectiva de um analista de análise estruturada, provavelmente usará modelos centrados em algoritmos, com o respectivo fluxo de dados de um processo para outro. Construindo um sistema a partir da perspectiva de um desenvolvedor orientado a objetos, provavelmente trabalhará com um sistema cuja arquitetura estará centrada em várias classes e os padrões de interação que determinarão como essas classes funcionarão em conjunto. Qualquer uma dessas soluções poderá ser correta para uma determinada aplicação e cultura de desenvolvimento, apesar de a experiência indicar que a perspectiva orientada a objetos é superior para a criação de arquiteturas flexíveis, inclusive no caso de sistemas que poderão conter grandes bancos de dados ou vários componentes computacionais. Apesar desse fato, o ponto mais importante é que cada visão de mundo conduz a um tipo diferente de sistema, com custos e benefícios diversos.

Em segundo lugar:

- *Cada modelo poderá ser expresso em diferentes níveis de precisão.*

Ao desenvolver um sistema complexo, às vezes poderá ser necessária uma visão panorâmica – por exemplo, para ajudar os investidores a visualizar a aparência e o funcionamento do futuro sistema. Em outras situações, poderá ser preciso retornar ao nível dos alicerces – por exemplo, quando existe uma rotina cuja execução é crítica ou um componente estrutural pouco comum.

O mesmo é verdadeiro em relação aos modelos de software. Às vezes, um modelo da interface para o usuário, de execução rápida e simples, será exatamente o que você precisará; em outros casos, será necessário retornar a níveis mais baixos, como ao especificar interfaces para várias plataformas ou ao enfrentar congestionamentos em uma rede. Em qualquer situação, os melhores tipos de modelos serão aqueles que permitem a escolha do grau de detalhamento, dependendo de quem esteja fazendo a visualização e por que deseja fazê-la. Um analista ou um usuário final dirigirá a atenção em direção a questões referentes ao que será visualizado; o desenvolvedor moverá o foco para a maneira como esses objetos funcionarão. Todos esses observadores desejarião visualizar o sistema em níveis diferentes de detalhamento em situações distintas.

Em terceiro lugar:

► *Os melhores modelos estão relacionados à realidade.*

O modelo físico de um prédio, que não responda da mesma forma que os materiais reais, terá um valor apenas limitado; o modelo matemático de um avião, em que são consideradas apenas condições de voo ideais e fabricação perfeita, poderá ocultar características potencialmente fatais do avião de verdade. Será melhor utilizar modelos que tenham uma clara conexão com a realidade e, nos casos em que essa conexão seja fraca, saber, de maneira exata, como esses modelos diferem do mundo real. Todos os modelos simplificam a realidade; o segredo será ter certeza de que sua simplificação não ocultará detalhes importantes.

No caso dos softwares, o tendão de Aquiles das técnicas de análise estruturada está no fato de não haver uma conexão básica entre o modelo de

análise e o modelo de projeto do sistema. Falhando a ponte sobre essa fenda, ao longo do tempo aparecerá uma divergência entre o sistema concebido e o sistema construído. Nos sistemas orientados a objetos, é possível estabelecer alguma conexão entre todos os pontos de vista quase independentes de um sistema em uma mesma totalidade semântica.

Em quarto lugar:

- *Nenhum modelo único é suficiente. Qualquer sistema não trivial será mais bem investigado por meio de um pequeno conjunto de modelos quase independentes com vários pontos de vista.*

Para a construção de um prédio, não existe um único conjunto de esboços de projetos capaz de revelar todos os detalhes da construção. Pelo menos, serão necessárias plantas baixas, aéreas, elétricas, de circulação e de água e esgoto.

A expressão funcional aqui utilizada é “quase independente”. Nesse contexto, a expressão significa modelos que possam ser criados e estudados separadamente, mas que continuam inter-relacionados. Assim como no caso de um prédio, você poderá estudar apenas as plantas relacionadas à energia elétrica, mas também poderá ver o respectivo mapa na planta baixa e talvez até sua interação com o direcionamento de canos na planta de água e esgoto.

O mesmo é verdadeiro em relação a sistemas de software orientados a objetos. Para compreender a arquitetura desses sistemas, você precisará recorrer a várias visões complementares e inter-relacionadas: a visão dos casos de uso (expondo os requisitos do sistema), a visão de projeto (captando o vocabulário do espaço do problema e do espaço da solução), a visão do processo (modelando a distribuição dos processos e threads do sistema), a visão da implementação do sistema (direcionando a realização física do sistema) e a visão da implantação (com o foco voltado para questões de engenharia de sistemas). Cada uma dessas visões poderá conter aspectos estruturais, como também aspectos comportamentais. Em conjunto, essas visões representam a base do projeto do software.

► O [Capítulo 2](#) apresenta os cinco pontos de vista de uma arquitetura.

Dependendo da natureza do sistema, alguns modelos poderão ser mais importantes do que outros. Por exemplo, no caso de sistemas que utilizam muitos dados, predominarão modelos voltados para a visão estática do projeto. Em sistemas centrados no uso de GUI, são importantes as visões estáticas e dinâmicas dos casos de uso. Em sistemas de execução crítica em tempo real, a visão dinâmica do processo tende a ser mais importante. Por fim, em sistemas distribuídos, como aqueles encontrados em aplicações que utilizam a Web, os modelos de implementação e de implantação são os mais importantes.

## A MODELAGEM ORIENTADA A OBJETOS

Os engenheiros civis constroem vários tipos de modelos. Com maior frequência, encontram-se modelos estruturais que ajudam as pessoas a visualizar e a especificar partes de sistemas e os relacionamentos existentes entre essas partes. Dependendo de ser mais importante o interesse comercial ou a questão de engenharia, os engenheiros também poderão elaborar modelos dinâmicos – por exemplo, com a finalidade de ajudá-los a estudar o comportamento de determinada estrutura em relação a tremores de terra. Cada tipo de modelo é organizado de modo diferente e cada um tem seu próprio foco. No caso de software, existem várias maneiras de se definir um modelo. As duas maneiras mais comuns são provenientes da perspectiva de um algoritmo ou da perspectiva orientada a objetos.

A visão tradicional no desenvolvimento de softwares adota a perspectiva de um algoritmo. Nessa visão, o principal bloco de construção do software é o procedimento ou a função. Essa perspectiva conduz os desenvolvedores a voltar seu foco de atenção para questões referentes ao controle e à decomposição de algoritmos maiores em outros menores. Não existe nenhuma grande desvantagem nessa solução, com exceção da tendência a permitir sistemas instáveis. À medida que os requisitos se modificam (e isso

certamente ocorrerá) e o sistema cresce (o que também acontecerá), será difícil fazer a manutenção de sistemas construídos a partir do foco em algoritmos.

A visão contemporânea no desenvolvimento de software adota uma perspectiva orientada a objetos. Nessa visão, o principal bloco de construção de todos os sistemas de software é o objeto ou a classe. Explicando de uma maneira simples, um objeto é alguma coisa geralmente estruturada a partir do vocabulário do espaço do problema ou do espaço da solução; uma classe é a descrição de um conjunto de objetos comuns. Todos os objetos têm uma identidade (você pode atribuir-lhes nomes ou diferenciá-los dos demais objetos de alguma maneira), um estado (costuma haver dados a eles associados) e um comportamento (você poderá fazer algo com o objeto ou ele poderá fazer algo com outros objetos).

Por exemplo, considere uma arquitetura simples com três componentes para um sistema de cobrança, incluindo a interface para o usuário, uma camada intermediária e um banco de dados. Na interface para o usuário, você encontrará objetos concretos, como botões, menus e caixas de diálogo. No banco de dados, haverá objetos concretos como tabelas, que representam entidades provenientes do domínio do problema, incluindo clientes, produtos e pedidos. Na camada intermediária, você encontrará objetos como transações e regras de negócios, além de visões de alto nível relacionadas às entidades do problema, como clientes, produtos e pedidos.

O método orientado a objetos para o desenvolvimento de software é, com certeza, uma parte do fluxo principal, simplesmente porque tem sido provado seu valor para a construção de sistemas em todos os tipos de domínios de problemas, abrangendo todos os graus de tamanho e de complexidade. Além disso, muitas linguagens, sistemas operacionais e ferramentas contemporâneos são, de alguma forma, orientados a objetos, fortalecendo a visão de mundo em termos de objetos. O desenvolvimento orientado a objetos fornece os fundamentos conceituais para a montagem de sistemas a partir de componentes com a utilização de tecnologias como J2EE ou .NET.

Várias consequências decorrem da escolha da visão de mundo orientada a

objetos: o que é a estrutura de uma boa arquitetura orientada a objetos? Quais artefatos o projeto deverá criar? Quem deverá criá-los? Como esses artefatos poderão ser medidos?

➡ *O Capítulo 2 examina essas questões.*

A visualização, a especificação, a construção e a documentação de sistemas orientados a objetos é exatamente o objetivo da UML.

---

<sup>1</sup> CRC significa Classe, Responsabilidade e Colaboração. Cada CRC Card funciona como um script que representa as funções que cada módulo do sistema desempenhará e como ele se relacionará com os outros módulos. Ver: <http://miud.in/83r>

CAPÍTULO

---

2

---

# Introdução à UML

## Neste capítulo

- » *Uma visão geral da UML*
- » *Os três passos para compreender a UML*
- » *A arquitetura de softwares*
- » *O processo de desenvolvimento de software*

**A**UML (Unified Modeling Language) é uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software.

A UML é adequada para a modelagem de sistemas, cuja abrangência poderá incluir sistemas de informação corporativos a serem distribuídos a aplicações baseadas na Web e até sistemas complexos embutidos de tempo real. É uma linguagem muito expressiva, abrangendo todas as visões necessárias ao desenvolvimento e implantação desses sistemas. Apesar de sua expressividade, não é difícil compreender e usar a UML. Aprender a aplicar a UML de maneira efetiva tem início com a formação de um modelo conceitual da linguagem, o que pressupõe o entendimento de três principais elementos: os blocos básicos de construção da UML, as regras que determinam como esses blocos de construção deverão ser combinados e alguns mecanismos básicos que se aplicam a toda a linguagem.

A UML é apenas uma linguagem e, portanto, é somente uma parte de um método para desenvolvimento de software. A UML é independente do processo, apesar de ser perfeitamente utilizada em processo orientado a casos de usos, centrado na arquitetura, iterativo e incremental.

# UMA VISÃO GERAL DA UML

A UML é uma linguagem destinada a:

- » Visualizar
- » Especificar
- » Construir
- » Documentar os artefatos de um sistema complexo de software

## A UML É UMA LINGUAGEM

As linguagens fornecem um vocabulário e as regras para a combinação de palavras desse vocabulário com a finalidade de comunicar algo. Uma linguagem de *modelagem* é a linguagem cujo vocabulário e regras têm seu foco voltado para a representação conceitual e física de um sistema. Portanto, uma linguagem de modelagem, como a UML, é uma linguagem-padrão para a elaboração da estrutura de projetos de software.

A modelagem permite a compreensão de um sistema. Nenhum modelo é inteiramente suficiente. Sempre serão necessários vários modelos, conectados entre si, para tornar possível entender qualquer aspecto, ainda que seja o sistema mais trivial. No caso de sistemas que fazem uso intenso de software, torna-se necessária uma linguagem capaz de abranger as diferentes visões relacionadas à arquitetura do sistema, como essa arquitetura evolui ao longo do ciclo de vida do desenvolvimento do software.

► Os princípios básicos da modelagem são apresentados no [Capítulo 1](#).

O vocabulário e as regras de uma linguagem como a UML indicam como criar e ler modelos bem formados, mas não apontam quais modelos deverão ser criados, nem quando você deverá criá-los. Essa tarefa cabe ao processo de desenvolvimento do software. Um processo bem definido servirá como guia para decidir quais artefatos serão produzidos, quais atividades e trabalhadores

serão escolhidos para criá-los e gerenciá-los e como esses artefatos serão empregados para medir e controlar o projeto como um todo.

## A UML É UMA LINGUAGEM PARA VISUALIZAÇÃO

Para muitos programadores, não existe diferença entre pensar em uma implementação e transformá-la em código. Pensar em algo é criar o código correspondente. De fato, alguns aspectos podem ser mais bem delineados diretamente no código. O texto é uma forma maravilhosamente mínima e direta para escrever expressões e algoritmos.

Nesses casos, o programador ainda estará realizando algum tipo de modelagem, apesar de inteiramente no plano mental. Ele até poderá rabiscar algumas ideias em um quadro ou em um guardanapo. Entretanto, esse procedimento apresenta alguns problemas. Primeiro, a comunicação desses modelos conceituais a outras pessoas estará sujeita a erros, a menos que todos os envolvidos usem a mesma linguagem. Tipicamente, empresas e projetos desenvolvem suas próprias linguagens e acaba se tornando difícil compreender o que está se passando, se você não pertencer ao grupo ou for um novo participante. Segundo, existem aspectos nos sistemas de software que não podem ser efetivamente compreendidos a menos que você construa modelos capazes de transcender a linguagem de programação textual. Por exemplo, o significado de uma hierarquia de classes pode ser inferido, mas não percebido diretamente, examinando-se o código à procura de todas as classes possíveis na hierarquia. De maneira semelhante, a distribuição física e a possível migração de objetos em sistemas baseados na Web podem ser inferidas, mas não percebidas diretamente apenas pelo estudo do código do sistema. Terceiro, se o desenvolvedor decidiu fazer uma remoção no código, mas nunca escreveu os modelos existentes em sua mente, essas informações estarão perdidas para sempre ou, na melhor das hipóteses, serão recriadas parcialmente somente a partir da própria implementação, uma vez que o desenvolvedor não está mais à disposição.

O uso da UML para elaborar modelos abrange uma terceira questão: modelos explícitos facilitam a comunicação.

Para determinados itens, poderá ser melhor uma modelagem textual; para outros itens, poderá ser mais conveniente uma modelagem gráfica. De fato, todos os sistemas interessantes contêm estruturas que transcendem o que uma linguagem de programação é capaz de representar. Sendo uma linguagem gráfica, a UML atende ao segundo problema descrito anteriormente. A UML é mais do que um mero punhado de símbolos gráficos. Por trás de cada símbolo empregado na notação da UML existe uma semântica bem definida. Dessa maneira, um desenvolvedor poderá usar a UML para escrever seu modelo, e qualquer outro desenvolvedor, ou até outra ferramenta, será capaz de interpretá-lo sem ambiguidades. Dessa maneira, a UML também atende à primeira questão mencionada anteriormente.

- *O manual de referência da UML apresenta a semântica completa da linguagem.*

## A UML É UMA LINGUAGEM PARA ESPECIFICAÇÃO

No presente contexto, *especificar* significa construir modelos precisos, sem ambiguidades e completos. Em particular, a UML atende a todas as decisões importantes em termos de análise, projeto e implementação, que devem ser tomadas para o desenvolvimento e a implantação de sistemas complexos de software.

A UML é uma linguagem para construção; não é uma linguagem visual de programação, mas seus modelos podem ser diretamente conectados a várias linguagens de programação. Isso significa que é possível mapear os modelos da UML em linguagens de programação tais como Java, C++, Visual Basic ou até tabelas de bancos de dados relacionais ou o armazenamento de dados persistentes de um banco de dados orientado a objetos. A UML é capaz de representar tudo que possa ser mais bem expresso em termos gráficos, enquanto as linguagens de programação representam o que é mais bem expresso em termos textuais.

Esse mapeamento permite a realização de uma engenharia direta: a geração de código a partir de um modelo em UML para uma linguagem de programação. O processo inverso também é possível: você poderá reconstruir um modelo a partir de sua implementação, revertendo-a à UML. A engenharia reversa não faz mágicas. A menos que você as codifique na implementação, as informações são perdidas ao serem passadas dos modelos para o código. A engenharia reversa, portanto, requer o suporte de ferramentas em conjunto com a intervenção humana. A combinação dos procedimentos de geração de código e de engenharia reversa permite uma engenharia de ciclo completo, o que significa a capacidade de trabalhar em modos de visualização gráfica ou textual, enquanto as ferramentas asseguram a consistência desses dois modos.

- ➡ *A modelagem da estrutura de sistemas é apresentada nas Partes 2 e 3.*

Além desse mapeamento direto, a UML é suficientemente expressiva e não tem ambiguidades para permitir a execução direta dos modelos, a simulação de sistemas e a instrumentação de sistemas em execução.

- ➡ *A modelagem do comportamento de sistemas é apresentada nas Partes 4 e 5.*

## **A UML É UMA LINGUAGEM PARA DOCUMENTAÇÃO**

Uma empresa de software saudável produz todos os tipos de artefatos, além do código executável bruto. Esses artefatos incluem (mas não estão limitados a) o seguinte:

- › Requisitos
- › Arquitetura
- › Projeto
- › Código-fonte

- › Planos do projeto
- › Testes
- › Protótipos
- › Versões

Dependendo da cultura de desenvolvimento, alguns desses artefatos são tratados de uma maneira mais ou menos formal do que outros. Esses artefatos não são apenas a parte do projeto a ser entregue, mas também são críticos para controlar, medir e comunicar determinado sistema durante seu desenvolvimento e após sua implantação. A UML abrange a documentação da arquitetura do sistema e de todos os seus detalhes. A UML também proporciona uma linguagem para a expressão de requisitos e para a realização de testes. Por fim, a UML oferece uma linguagem para a modelagem das atividades de planejamento do projeto e de gerenciamento de versões.

## **ONDE A UML PODE SER UTILIZADA?**

A UML se destina principalmente a sistemas complexos de software. Tem sido empregada de maneira efetiva em domínios como os seguintes:

- › Sistemas de informações corporativos
- › Serviços bancários e financeiros
- › Telecomunicações
- › Transportes
- › Defesa/espaço aéreo
- › Vendas de varejo
- › Eletrônica médica
- › Científico
- › Serviços distribuídos baseados na Web

A UML não está restrita à modelagem de software. Na verdade, a UML é suficientemente expressiva para modelar sistemas que não sejam de software, como o fluxo de trabalho no sistema legal, a estrutura e o comportamento de sistemas de saúde e o projeto de hardware.

## **UM MODELO CONCEITUAL DA UML**

Para compreender a UML, você precisará formar um modelo conceitual da linguagem e isso pressupõe aprender três elementos principais: os blocos de construção básicos da UML, as regras que determinam como esses blocos poderão ser combinados e alguns mecanismos comuns aplicados na UML. Após entender essas ideias, você será capaz de ler modelos da UML e criar outros modelos básicos. À medida que acumular experiência na aplicação da UML, você pode construir novos modelos a partir desse modelo conceitual, usando características mais avançadas da linguagem.

## **BLOCOS DE CONSTRUÇÃO DA UML**

O vocabulário da UML abrange três tipos de blocos de construção:

1. Itens
2. Relacionamentos
3. Diagramas

Os itens são as abstrações identificadas como cidadãos de primeira classe em um modelo; os relacionamentos reúnem esses itens; os diagramas agrupam coleções interessantes de itens.

## **ITENS DA UML**

Existem quatro tipos de itens na UML:

1. Itens estruturais
2. Itens comportamentais
3. Itens de agrupamentos

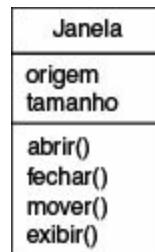
#### 4. Itens anotacionais

Esses itens constituem os blocos de construção básicos orientados a objetos da UML e você os utilizará para escrever modelos bem formados.

**Itens estruturais** Os *itens estruturais* são os substantivos utilizados em modelos da UML. São as partes mais estáticas do modelo, representando elementos conceituais ou físicos. Coletivamente, os itens estruturais são chamados *classificadores*.

As *classes* são descrições de conjuntos de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. As classes implementam uma ou mais interfaces. Graficamente, as classes são representadas por retângulos, geralmente incluindo seu nome, atributos e operações, conforme mostra a [Figura 2.1](#).

► As *classes* são examinadas nos Capítulos 4 e 9.



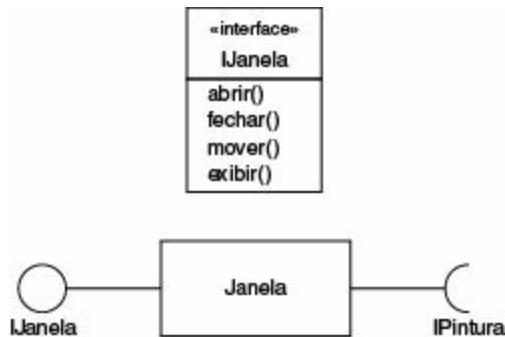
**Figura 2.1:**

Classes

Uma *interface* é uma coleção de operações que especificam serviços de uma classe ou componente. Portanto, uma interface descreve o comportamento externamente visível desse elemento. Uma interface poderá representar todo o comportamento de uma classe ou componente, como também apenas parte desse comportamento. A interface define um conjunto de especificações de operações (suas assinaturas), mas nunca um conjunto de implementações de operações. A declaração de uma interface é semelhante a uma classe com a palavra-chave «interface» acima do nome; os atributos não são relevantes, exceto, às vezes, para mostrar as constantes. Entretanto, uma interface raramente aparece sozinha. Uma interface fornecida por uma classe

ao mundo externo é exibida como um pequeno círculo anexado à caixa de classes por uma linha. Uma interface requerida por uma classe de alguma outra classe é exibida como um pequeno semicírculo anexado à caixa de classes por uma linha, conforme mostra a [Figura 2.2](#).

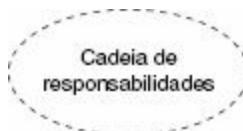
- As interfaces são examinadas no [Capítulo 11](#).



**Figura 2.2:**  
Interfaces

As colaborações definem interações e são sociedades de papéis e outros elementos que funcionam em conjunto para proporcionar um comportamento cooperativo superior à soma de todos os elementos. Portanto, as colaborações contêm dimensões estruturais, assim como comportamentais. Uma determinada classe poderá participar em várias colaborações. Assim, essas colaborações representam a implementação de padrões que formam um sistema. Graficamente, as colaborações são representadas como elipses com linhas tracejadas, geralmente incluindo somente seu nome, conforme mostra a [Figura 2.3](#).

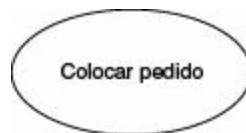
- As colaborações são examinadas no [Capítulo 28](#).



**Figura 2.3:**  
Colaborações

Um *caso de uso* é a descrição de sequências de ações realizadas pelo sistema que proporciona resultados observáveis de valor para um determinado ator. Um caso de uso é utilizado para estruturar o comportamento de itens em um modelo. Um caso de uso é realizado por uma colaboração. Graficamente, um caso de uso é representado por uma elipse com linhas contínuas, geralmente incluindo somente seu nome, conforme mostra a [Figura 2.4](#).

- ➡ Os casos de uso são examinados no [Capítulo 17](#).

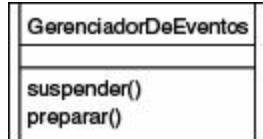


**Figura 2.4:**

Casos de uso

Os outros três itens restantes – classes ativas, componentes e nós – são semelhantes a classes, ou seja, descrevem conjuntos de entidades que compartilham os mesmos atributos, operações, relacionamentos e semânticas. Porém, esses três são suficientemente diferentes e necessários para a modelagem de certos aspectos de sistemas orientados a objetos e, portanto, merecem um tratamento especial. As *classes ativas* são classes cujos objetos têm um ou mais processos ou threads e, portanto, podem iniciar a atividade de controle. Uma classe ativa é semelhante a uma classe, exceto pelo fato de que seus objetos representam elementos cujo comportamento é concorrente com o de outros elementos. Graficamente, as classes ativas são representadas da mesma maneira como as classes, mas com linhas duplas à esquerda e à direita, incluindo seus nomes, atributos e operações, conforme mostra a [Figura 2.5](#).

- ➡ As classes ativas são examinadas no [Capítulo 23](#).



**Figura 2.5:**

Classes ativas

Os *componentes* são partes modulares de um sistema, que ocultam a sua implementação atrás de um conjunto de interfaces externas. Em um sistema, os componentes que compartilham as mesmas interfaces podem ser substituídos ao mesmo tempo em que preservam o mesmo comportamento lógico. A implementação de um componente pode ser expressa por meio da ligação de peças e conectores; as peças podem incluir componentes menores. Graficamente, os componentes são representados como uma classe com um ícone especial no canto superior direito, como mostra a [Figura 2.6](#).

- ➡ Os *componentes* e a estrutura interna são examinados no [Capítulo 15](#).



**Figura 2.6:**

Componentes

Os dois elementos restantes – componentes e nós – também são diferentes. Esses elementos representam itens físicos, enquanto os cinco itens anteriores representam itens conceituais ou lógicos. Um *artefato* é uma peça física substituível de um sistema que contém informações físicas (“bits”). Em um sistema, você encontrará diferentes tipos de artefatos de implantação, como arquivos de código-fonte, executáveis e scripts. Um artefato normalmente representa a embalagem física da fonte ou as informações de tempo de execução. Graficamente, um artefato é representado como um retângulo com a palavra-chave «artefato» acima do nome, como na [Figura 2.7](#).

► Os artefatos são examinados no [Capítulo 26](#).

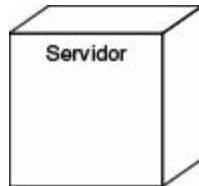


**Figura 2.7:**

Artefatos

Um *nó* é um elemento físico existente em tempo de execução que representa um recurso computacional, geralmente com pelo menos alguma memória e, frequentemente, capacidade de processamento. Um conjunto de componentes poderá estar contido em um nó e também poderá migrar de um nó para outro. Graficamente, um nó é representado como um cubo, normalmente incluindo somente seu nome, como mostra a [Figura 2.8](#).

► Os nós são apresentados no [Capítulo 27](#).



**Figura 2.8:**

Nós

Esses elementos – classes, interfaces, colaborações, casos de uso, classes ativas, componentes, artefatos e nós – são os itens estruturais básicos que você poderá incluir em um modelo da UML. Também existem variações desses elementos, como atores, sinais e utilitários (tipos de classes); processos e threads (tipos de classes ativas); e aplicações, documentos, arquivos, bibliotecas, páginas e tabelas (tipos de artefatos).

**Itens comportamentais** Os *itens comportamentais* são as partes dinâmicas dos modelos de UML. São os verbos de um modelo, representando comportamentos no tempo e no espaço. Ao todo, existem dois tipos principais de itens comportamentais.

- O [Capítulo 17](#) examina os casos de uso, que são utilizados para estruturar os itens comportamentais de um modelo; o [Capítulo 16](#) apresenta as interações.

Primeiro, uma *interação* é um comportamento que abrange um conjunto de mensagens trocadas entre um conjunto de objetos em determinado contexto para a realização de propósitos específicos. O comportamento de uma sociedade de objetos ou de uma operação individual poderá ser especificado por meio de uma interação. As interações envolvem outros elementos, inclusive mensagens, ações e ligações (as conexões entre os objetos). Graficamente, uma mensagem é representada como linha cheia com seta, quase sempre incluindo o nome de suas operações, conforme mostra a [Figura 2.9](#).

exibir ►

**Figura 2.9:**

Mensagens

Segundo, uma *máquina de estado* é um comportamento que especifica as sequências de estados pelas quais objetos ou interações passam durante sua existência em resposta a eventos, bem como suas respostas a esses eventos. O comportamento de uma classe individual ou de uma colaboração de classes pode ser especificado por meio de uma máquina de estados. Uma máquina de estado abrange outros elementos, incluindo estados, transições (o fluxo de um estado a outro), eventos (itens que disparam uma transição) e atividades (as respostas às transições). Graficamente, o estado é representado como retângulo com ângulos arredondados, geralmente incluindo seu nome e respectivos subestados, se houver, conforme mostra a [Figura 2.10](#).

- As máquinas de estados são examinadas no [Capítulo 22](#).

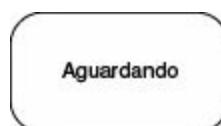


Figura 2.10:

Estados

Terceiro, uma atividade é um comportamento que especifica a sequência de etapas que um processo computacional realiza. Em uma interação, o foco está no conjunto de objetos que interage. Em uma máquina de estado, o foco é no ciclo de vida de um objeto por vez. Em uma atividade, o foco está nos fluxos entre as etapas, independente de qual objeto realiza cada etapa. Uma etapa de uma atividade é chamada de *ação*. Graficamente, uma ação é representada como retângulo com ângulos arredondados com um nome que indica o seu propósito. Estados e ações distinguem-se por seus contextos diferentes.

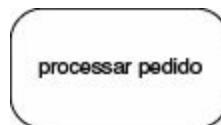


Figura 2.11:

Ações

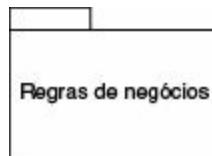
Esses três elementos – interações, máquinas de estados e atividades – são os itens comportamentais básicos que você pode incluir em um modelo da UML. Semanticamente, esses elementos costumam estar conectados a vários elementos estruturais, classes principais, colaborações e objetos.

**Itens de agrupamento** Os *itens de agrupamento* são as partes organizacionais dos modelos de UML. São os blocos em que os modelos podem ser decompostos. Ao todo, existe apenas um tipo principal de itens de agrupamento, chamado pacotes.

Um *pacote* é um mecanismo de propósito geral para a organização do próprio projeto, ao contrário das classes, que organizam os construtos de implementação. Os itens estruturais, os itens comportamentais e até outros itens de grupos podem ser colocados em pacotes. Ao contrário dos componentes (que existem em tempo de execução), um pacote é puramente conceitual (o que significa que existe apenas em tempo de desenvolvimento). Graficamente, um pacote é representado como diretórios com guias,

geralmente incluindo somente seus nomes e, às vezes, seu conteúdo, conforme mostra a [Figura 2.12](#).

- Os pacotes são examinados no [Capítulo 12](#).



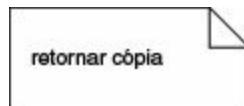
**Figura 2.12:**

Pacotes

Os pacotes são itens de agrupamento básico, com os quais você pode organizar modelos de UML. Também existem variações, como frameworks, modelos e subsistemas (tipos de pacotes).

**Itens anotacionais** Os *itens anotacionais* são as partes explicativas dos modelos de UML. São comentários, incluídos para descrever, esclarecer e fazer alguma observação sobre qualquer elemento do modelo. Existe um único tipo de item anotacional, chamado nota. Uma *nota* é apenas um símbolo para representar restrições e comentários anexados a um elemento ou a uma coleção de elementos. Graficamente, uma nota é representada por um retângulo em um dos cantos como uma dobra de página, acompanhado por texto ou comentário gráfico, conforme mostra a [Figura 2.13](#).

- As notas são examinadas no [Capítulo 6](#).



**Figura 2.13:**

Notas

Esse elemento é o único item anotacional básico que você poderá incluir em um modelo de UML. Geralmente você usará as notas para aprimorar seus diagramas com restrições ou comentários que possam ser mais bem expressos

por um texto formal ou informal. Também existem variações desse elemento, como os requisitos (que especificam determinado comportamento desejado sob uma perspectiva externa ao sistema).

**Relacionamentos na UML** Existem quatro tipos de relacionamentos na UML:

1. Dependência
2. Associação
3. Generalização
4. Realização

Esses relacionamentos são os blocos relacionais básicos de construção da UML. Você os utilizará para escrever modelos bem formados. Primeiro, uma *dependência* é um relacionamento semântico entre dois itens, nos quais a alteração de um (o item independente) pode afetar a semântica do outro (o item dependente). Graficamente, uma dependência é representada por linhas tracejadas, possivelmente com setas e ocasionalmente incluindo um rótulo, conforme mostra a [Figura 2.14](#).

► As dependências são examinadas nos Capítulos 5 e 10.



**Figura 2.14:**  
Dependências

Segundo, uma *associação* é um relacionamento estrutural entre classes que descreve um conjunto de ligações, em que as ligações são conexões entre objetos que são instâncias das classes. A agregação é um tipo especial de associação, representando um relacionamento estrutural entre o todo e suas partes. Graficamente, uma associação é representada por linhas sólidas, possivelmente direcionadas, ocasionalmente incluindo rótulos e, frequentemente, contendo outros adornos, como nomes de papéis e multiplicidades, conforme mostra a [Figura 2.15](#).

► As associações são apresentadas nos Capítulos 5 e 10.



**Figura 2.15:**

---

Associações

Terceiro, uma *generalização* é um relacionamento de especialização/generalização, no qual os objetos dos elementos especializados (os filhos) são substituíveis por objetos do elemento generalizado (os pais). Dessa maneira, os filhos compartilham a estrutura e o comportamento dos pais. Graficamente, um relacionamento de generalização é representado como linha sólida com uma seta em branco apontando o pai, conforme mostra a [Figura 2.16](#).

- As *generalizações* são examinadas nos [Capítulos 5 e 10](#).



**Figura 2.16:**

---

Generalizações

Quarto, uma *realização* é um relacionamento semântico entre classificadores, em que um classificador especifica um contrato que outro classificador garante executar. Os relacionamentos de realizações serão encontrados em dois locais: entre interfaces e as classes ou componentes que as realizam; e entre casos de uso e as colaborações que os realizam. Graficamente, um relacionamento de realização é representado por uma linha tracejada com seta branca entre uma generalização e um relacionamento de dependência, conforme mostra a [Figura 2.17](#).

- As *realizações* são examinadas no [Capítulo 10](#).



**Figura 2.17:**

---

Realizações

Esses quatro elementos são os itens relacionais básicos que você pode incluir em um modelo de UML. Também existem variações desses quatro

elementos, como refinamentos, rastros, inclusões e extensões.

**Diagramas na UML** Um *diagrama* é a apresentação gráfica de um conjunto de elementos, geralmente representadas como gráficos de vértices (itens) e arcos (relacionamentos). São desenhados para permitir a visualização de um sistema sob diferentes perspectivas; nesse sentido, um diagrama constitui uma projeção de um determinado sistema. Em todos os sistemas, com exceção dos mais triviais, um diagrama representa uma visão parcial dos elementos que compõem o sistema. O mesmo elemento pode aparecer em todos os diagramas, em apenas alguns (o caso mais comum) ou em nenhum diagrama (um caso muito raro). Na teoria, um diagrama pode conter qualquer combinação de itens e de relacionamentos. Na prática, porém, aparecerá um pequeno número de combinações comuns, que são consistentes com as cinco visões mais úteis da arquitetura de um sistema complexo de software. Por isso, a UML inclui 13 desses diagramas:

► As cinco visões de uma arquitetura são examinadas na próxima seção.

1. Diagrama de classes
2. Diagrama de objetos
3. Diagrama de componentes
4. Diagrama de estruturas compostas
5. Diagrama de casos de uso
6. Diagrama de sequências
7. Diagrama de comunicações
8. Diagrama de estados
9. Diagrama de atividades
10. Diagrama de implantação
11. Diagrama de pacote
12. Diagrama de temporização
13. Diagrama de visão geral da interação

Um *diagrama de classes* exibe um conjunto de classes, interfaces e colaborações, bem como seus relacionamentos. Esses diagramas são encontrados com maior frequência em sistemas de modelagem orientados a objeto e abrangem uma visão estática da estrutura do sistema. Os diagramas de classes que incluem classes ativas direcionam a perspectiva do processo estático do sistema. Os diagramas de componentes são variantes dos diagramas de classes.

- *Os diagramas de classes são examinados no Capítulo 8.*

Um *diagrama de objetos* exibe um conjunto de objetos e seus relacionamentos. Representa retratos estáticos de instâncias de itens encontrados em diagramas de classes. São diagramas que abrangem a visão estática da estrutura ou do processo de um sistema, como ocorre nos diagramas de classes, mas sob perspectiva de casos reais ou de protótipos.

- *Os diagramas de objetos são examinados no Capítulo 14.*

Um *diagrama de componentes* exibe uma classe encapsulada e suas interfaces, portas e estrutura interna que consiste de componentes aninhados e conectores. Os diagramas de componentes abrangem a visão de implementação do projeto estático de um sistema. Eles são importantes para criar grandes sistemas a partir de partes menores. (A UML distingue um *diagrama de estrutura composta*, aplicável a qualquer classe, de um diagrama de componentes, mas juntamos as duas análises porque a distinção entre um componente e uma classe estruturada é desnecessariamente sutil.)

- *Os diagramas de componentes e a estrutura interna são examinados no Capítulo 15.*

Um *diagrama de casos de uso* exibe um conjunto de casos de uso e atores (um tipo especial de classe) e seus relacionamentos. Diagramas de caso de uso abrangem a visão estática de casos de uso do sistema. Esses diagramas

são importantes principalmente para a organização e a modelagem de comportamentos do sistema.

- Os diagramas de casos de uso são examinados no Capítulo 18.

Tanto os diagramas de sequências como os de comunicações são tipos de diagramas de interações. Um *diagrama de visão geral de interação* exibe uma interação, consistindo de um conjunto de objetos ou papéis, incluindo as mensagens que podem ser trocadas entre eles. Diagramas de interações abrangem a visão dinâmica de um sistema. Um *diagrama de sequências* é um diagrama de interação cuja ênfase está na ordenação temporal das mensagens; o *diagrama de comunicações* é um diagrama de interação cuja ênfase está na organização estrutural dos objetos ou papéis que enviam e recebem mensagens. Os diagramas de sequências e de comunicações representam conceitos básicos similares, mas cada um enfatiza uma visão diferente dos conceitos. Os diagramas de sequências enfatizam a ordem temporal, e os de comunicações, a estrutura de dados por meio da qual a mensagem passa. Um *diagrama de temporização* (não abordado neste livro) mostra os tempos reais nos quais as mensagens são trocadas.

- Os diagramas de visão geral de interações são examinados no Capítulo 19.

Os *diagramas de estados* exibem uma máquina de estados, formada por estados, transições, eventos e atividades. Os diagramas de estados abrangem a visão dinâmica de um objeto. São importantes principalmente para a modelagem de comportamentos de uma interface, classe ou colaboração e para dar ênfase a comportamentos de um objeto ordenados por eventos, o que é de grande ajuda para a modelagem de sistemas reativos.

- Os diagramas de estados são examinados no Capítulo 25.

Um *diagrama de atividades* exibe a estrutura de um processo ou outra computação, como o fluxo de controle e os dados de cada etapa de uma

computação. Abrange a visão dinâmica do sistema e é importante principalmente para a modelagem da função de um sistema e dá ênfase ao fluxo de controle entre objetos.

- Os *diagramas de atividades* são examinados no [Capítulo 20](#).

Um *diagrama de implantação* mostra a configuração dos nós de processamento em tempo de execução e os componentes neles existentes. Abrange a visão estática de implantação de uma arquitetura. Tipicamente, um nó inclui um ou mais artefatos.

- Os *diagramas de implantação* são examinados no [Capítulo 31](#).

Um *diagrama de artefato* mostra os constituintes físicos de um sistema no computador. Os artefatos incluem arquivos, bancos de dados e coleções de bits físicas similares. Os artefatos são usados frequentemente junto com diagramas de implantação e mostram as classes e componentes que implementam. (A UML trata os diagramas de artefatos como um tipo de diagrama de implantação, mas nós os trataremos separadamente.)

- Os *diagramas de artefatos* são examinados no [Capítulo 30](#).

Um *diagrama de pacote* mostra a decomposição do próprio modelo em unidades organizacionais e suas dependências.

- Os *diagramas de pacote* são examinados no [Capítulo 12](#).

Um *diagrama de temporização* é um diagrama de interação que mostra os tempos reais em diferentes objetos ou papéis, em vez das sequências de mensagens relativas. Um *diagrama de visão geral de interação* é um híbrido de um diagrama de atividades e um diagrama de sequências. Esses diagramas têm usos especializados, mas esse tema não é abordado neste livro. Consulte o *UML Reference Manual* para obter mais detalhes.

Essa não é uma lista completa de diagramas. Algumas ferramentas podem usar a UML para fornecer outros tipos de diagramas, apesar de que certamente esses tipos serão os mais encontrados na prática.

## REGRAS DA UML

Os blocos de construção da UML não podem ser simplesmente combinados de uma forma aleatória. Como ocorre em qualquer linguagem, a UML tem um determinado número de regras que especificam o que deverá ser um modelo bem formado. Os *modelos bem formados* são aqueles autoconsistentes semanticamente e em harmonia com todos os modelos a eles relacionados.

A UML dispõe de regras semânticas para:

- » Nomes Quais nomes podem ser atribuídos a itens, relacionamentos e diagramas.
- » Escopo O contexto que determina um significado específico para um nome.
- » Visibilidade Como esses nomes podem ser vistos e utilizados pelos outros.
- » Integridade Como os itens se relacionam entre si de forma adequada e consistente.
- » Execução O que significa executar ou simular um modelo dinâmico.

Os modelos construídos durante o desenvolvimento de sistemas complexos de software tendem a evoluir e podem ser visualizados pelos participantes de diferentes maneiras e em momentos distintos. Por isso, é comum a equipe de desenvolvimento não construir apenas modelos bem formados, mas também modelos:

- › Parciais      Certos elementos ficam ocultos para simplificar a visão do modelo.
- › Incompletos    Certos elementos podem ser omitidos.
- ›                  A integridade do modelo não é assegurada.
- Inconsistentes

É inevitável a utilização desses modelos que não são bem formados como um detalhamento de um sistema em formação durante o ciclo de vida de desenvolvimento de um software. As regras da UML incentivam – mas não obrigam – a considerar as questões mais importantes de análise, projeto e implementação que levam esses modelos a se tornar bem formados ao longo do tempo.

## MECANISMOS BÁSICOS DA UML

Qualquer construção se torna mais simples e harmoniosa devido à adequação a um padrão de características básicas. Uma casa poderá ser construída em um estilo francês ou vitoriano apenas pela utilização de certos padrões arquitetônicos que definem esses estilos. O mesmo se aplica à UML, que se torna mais simples pela presença de quatro mecanismos básicos, aplicados de maneira consistente na linguagem.

1. Especificações
2. Adornos
3. Divisões comuns
4. Mecanismos de extensão

**Especificações** A UML é mais do que apenas uma linguagem gráfica. Por trás de cada parte de suas notações gráficas, existe uma especificação capaz de fornecer uma declaração textual da sintaxe e da semântica do respectivo bloco de construção. Por exemplo, relacionada a um ícone de classe, existe uma especificação que fornece o conjunto completo de atributos, operações (incluindo suas assinaturas completas) e comportamentos da classe; visualmente, o ícone de uma classe poderia apenas exibir uma pequena parte

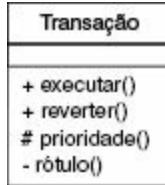
dessa especificação. Além disso, poderia haver outra visão referente a essa classe, capaz de apresentar um conjunto completamente diferente do conjunto de partes, mas inteiramente consistente com a especificação subjacente da classe. Você usa a notação gráfica da UML para visualizar um sistema e utiliza a especificação da UML para determinar os detalhes do sistema. Levando em consideração esses dois aspectos, será possível construir modelos de maneira incremental, desenhando diagramas e depois acrescentando uma semântica às especificações do modelo ou diretamente pela criação de uma especificação, talvez com a aplicação de engenharia reversa a um sistema existente, seguida pela criação dos diagramas que constituem projeções nessas especificações.

As especificações da UML fornecem um repertório semântico, contendo todas as partes de todos os modelos de determinado sistema, cada parte relacionada às demais de uma forma consistente. Assim, os diagramas da UML são apenas projeções visuais a partir desse repertório, cada diagrama revelando um aspecto interessante específico do sistema.

**Adornos** Em sua maioria, os elementos da UML têm uma notação gráfica única e direta, que proporciona uma representação visual dos aspectos mais importantes do elemento. Por exemplo, a notação para uma classe é intencionalmente projetada para ser desenhada com facilidade, pois as classes são os elementos mais comumente encontrados em sistemas de modelagem orientados a objetos. A notação de classe também expõe os aspectos mais importantes da classe, ou seja, seu nome, atributos e operações.

► *Notas e outros adornos são examinados no Capítulo 6.*

A especificação da classe pode incluir outros detalhes, como se a classe fosse abstrata ou como é a visibilidade de seus atributos e operações. Muitos desses detalhes podem ser representados como adornos gráficos ou textuais para a notação retangular básica da classe. Por exemplo, a [Figura 2.18](#) mostra uma classe ornamentada com a finalidade de indicar que é uma classe abstrata com duas operações públicas, uma protegida e uma privada.



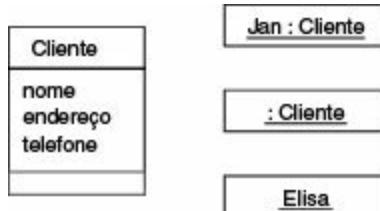
**Figura 2.18:**

Adornos

Todos os elementos da notação da UML são iniciados com um símbolo básico, ao qual pode ser acrescentada uma variedade de adornos específicos desse símbolo.

**Divisões comuns** Na modelagem de sistemas orientados a objetos, o mundo costuma ser dividido pelo menos de duas maneiras. Primeiro, existe a divisão de classes e objetos. Uma classe é uma abstração; um objeto é uma manifestação concreta dessa abstração. Na UML, você pode modelar as classes, assim como os objetos, conforme mostra a [Figura 2.19](#). Graficamente, para diferenciar os objetos, a UML utiliza os mesmos símbolos das respectivas classes e depois simplesmente sublinha os nomes dos objetos.

► Os objetos são examinados no [Capítulo 13](#).



**Figura 2.19:**

Classes e objetos

Nessa figura, existe uma única classe, chamada Cliente, juntamente com três objetos: Jan (marcado explicitamente como um objeto Cliente), Cliente (um objeto Cliente anônimo) e Elisa (em cuja especificação está marcado como um tipo de objeto Cliente, apesar de isso não ser apresentado aqui explicitamente). Quase todos os blocos de construção disponíveis na UML apresentam o

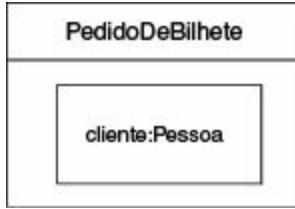
mesmo tipo de dicotomia classe/objeto. Por exemplo, você pode utilizar casos de uso e instâncias de caso de uso, componentes e instâncias de componente, nós e instâncias de nó e assim por diante. Segundo, existe uma separação de interface e implementação. Uma interface declara um contrato, e a implementação representa uma realização completa desse contrato, responsável pela manutenção fiel da semântica completa da interface. Na UML, você é capaz de modelar as interfaces e suas implementações, conforme mostra a [Figura 2.20](#).

- As interfaces são examinadas no [Capítulo 11](#).



**Figura 2.20:**  
Interfaces e suas implementações

Nessa figura, existe um componente chamado `AssistenteDeOrthografia.dll`, que implementa duas interfaces, `IDesconhecido` e `IOrtografia`. Ele também requer uma interface, `IDicionário`, que deve ser fornecida por outro componente. Quase todos os blocos de construção disponíveis na UML apresentam esse mesmo tipo de dicotomia interface/implementação. Por exemplo, pode haver casos de uso e as colaborações que as realizam, assim como operações e métodos que as implementam. Terceiro, há a separação de tipo e papel. O tipo declara a classe de uma entidade, como um objeto, um atributo ou um parâmetro. Um papel descreve o significado de uma entidade em seu contexto, como uma classe, um componente ou uma colaboração. Qualquer entidade que faça parte da estrutura de outra entidade, como um atributo, tem estas duas características: deriva uma parte do seu significado do tipo inerente e a outra parte do seu papel no contexto ([Figura 2.21](#)).



**Figura 2.21:**

Parte com papel e tipo

**Mecanismos de extensibilidade** A UML fornece uma linguagem-padrão para a elaboração de estrutura de projetos de software, mas não é possível que uma única linguagem fechada seja suficiente para expressar todas as nuances possíveis de todos os modelos em qualquer domínio o tempo todo. Por isso, a UML é aberta, permitindo que você amplie a linguagem de uma maneira controlada. Os mecanismos de extensibilidade da UML incluem as seguintes características:

- » *Os mecanismos de extensibilidade são apresentados no Capítulo 6.*
- » Estereótipos
- » Valores atribuídos
- » Restrições

Um *estereótipo* amplia o vocabulário da UML, permitindo a criação de novos tipos de blocos de construção que são derivados dos já existentes, mas específicos a determinados problemas. Por exemplo, ao trabalhar com uma linguagem de programação, como Java ou C++, com frequência você desejará modelar as exceções. Nessas linguagens, as exceções são apenas classes, apesar de serem tratadas de forma muito especial. Tipicamente, é suficiente permitir que sejam iniciadas e identificadas e nada mais. Em seus modelos, você poderá transformar as exceções em cidadãos de primeira classe – isso significa que serão tratadas como blocos de construção básicos – marcando-as com um estereótipo adequado, como no caso da classe Overflow mostrada na [Figura 2.22](#).

Um *valor atribuído* estende as propriedades dos blocos de construção da UML, permitindo a criação de novas informações na especificação de um elemento. Por exemplo, ao trabalhar em um produto que tenha muitas versões com o passar do tempo, certamente você desejará rastrear as versões e os autores de determinadas abstrações críticas. A versão e o autor não são conceitos primitivos na UML, mas podem ser acrescentados a qualquer bloco de construção, como uma classe, a partir da introdução de novos valores marcados em um certo bloco de construção. Na [Figura 2.22](#), por exemplo, a classe FilaDeEventos é estendida por uma marcação explícita de sua versão e autor.

Uma *restrição* amplia as semânticas dos blocos de construção da UML, permitindo acrescentar novas regras ou modificar as já existentes. Por exemplo, você poderia restringir a classe FilaDeEventos com a finalidade de que todos os acréscimos sejam feitos ordenadamente. Conforme mostra a [Figura 2.22](#), é possível adicionar uma restrição e marcar explicitamente esses acréscimos para a operação adicionar.



**Figura 2.22:**  
Mecanismos de extensibilidade

Em conjunto, esses três mecanismos de extensibilidade permitem definir a forma e ampliar a UML de acordo com as necessidades de seus projetos. Esses mecanismos ainda permitem que a UML se adapte a novas tecnologias de software, como o possível surgimento de linguagens mais poderosas de programação distribuídas. Você pode adicionar novos blocos de construção, modificar a especificação dos já existentes e até alterar sua semântica. Naturalmente, é importante fazer tudo isso de maneira controlada, para que,

ao utilizar essas extensões, você permaneça fiel ao propósito da UML – a comunicação de informações.

## ARQUITETURA

Visualizar, especificar, construir e documentar sistemas complexos de software são tarefas que requerem a visualização desses sistemas sob várias perspectivas. Diferentes participantes – usuários finais, analistas, desenvolvedores, integradores de sistemas, o pessoal que testa os sistemas, escritores técnicos e gerentes de projetos – trazem contribuições próprias ao projeto e observam o sistema de maneira distinta em momentos diferentes ao longo do desenvolvimento do projeto. A arquitetura do sistema talvez seja o artefato mais importante a ser utilizado com o objetivo de gerenciar esses diferentes pontos de vista e, assim, tornar possível um controle do desenvolvimento iterativo e incremental de um sistema durante seu ciclo de vida.

- » *A necessidade de visualizar sistemas complexos a partir de diferentes perspectivas é examinada no [Capítulo 1](#).*

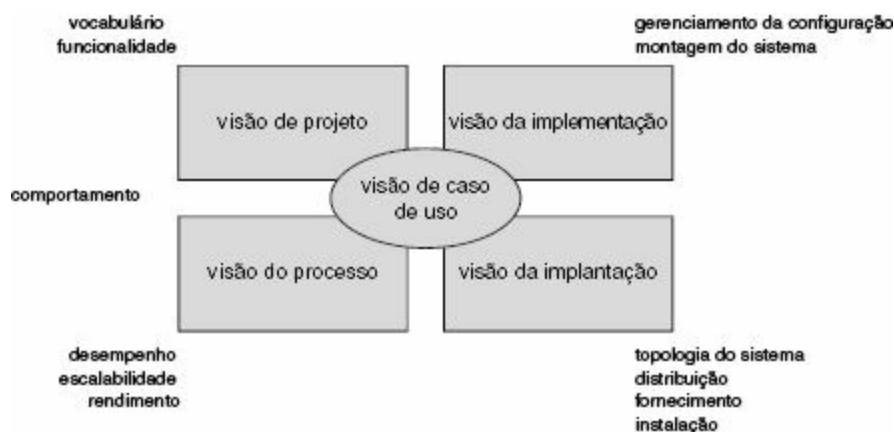
A arquitetura é o conjunto de decisões significativas acerca dos seguintes itens:

- » A organização do sistema de software.
- » A seleção dos elementos estruturais e suas interfaces, que compõem o sistema.
- » Seu comportamento, conforme especificado nas colaborações entre esses elementos.
- » A composição desses elementos estruturais e comportamentais em subsistemas progressivamente maiores.
- » O estilo de arquitetura que orienta a organização: os elementos estáticos e dinâmicos e as respectivas interfaces, colaborações e composição.

A arquitetura de software não está apenas relacionada à estrutura e ao comportamento, mas também ao uso, à funcionalidade, ao desempenho, à flexibilidade, à reutilização, à abrangência, a adequações e a restrições de caráter econômico e tecnológico, além de questões estéticas.

Conforme mostra a [Figura 2.23](#), a arquitetura de um sistema complexo de software pode ser descrita mais adequadamente por cinco visões interligadas. Cada visão constitui uma projeção na organização e estrutura do sistema, cujo foco está voltado para determinado aspecto desse sistema.

- A modelagem da arquitetura de sistemas é examinada no [Capítulo 32](#).



**Figura 2.23:**

A modelagem da arquitetura de um sistema

A *visão do caso de uso* abrange os casos de uso que descrevem o comportamento do sistema conforme é visto pelos seus usuários finais, analistas e pessoal de teste. Essa visão não especifica realmente a organização do sistema de um software. Porém, ela existe para especificar as forças que determinam a forma da arquitetura do sistema. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de caso de uso, enquanto os aspectos dinâmicos são capturados em diagramas de interação, diagramas de estados e diagramas de atividades.

A *visão de projeto* de um sistema abrange as classes, interfaces e colaborações que formam o vocabulário do problema e de sua solução. Essa perspectiva proporciona principalmente um suporte para os requisitos funcionais do sistema, ou seja, os serviços que o sistema deverá fornecer a seus usuários finais. Com a UML, os aspectos estáticos dessa visão são captados em diagramas de classes e de objetos; os aspectos dinâmicos são captados em diagramas de interações, diagramas de estados e diagramas de atividades. O diagrama da estrutura interna de uma classe é particularmente útil.

A *visão de interação* de um sistema mostra o fluxo de controle entre as várias partes, incluindo mecanismos de concorrência e de sincronização. Essa visão cuida principalmente de questões referentes ao desempenho, à escalabilidade e ao rendimento do sistema. Com a UML, os aspectos estáticos e dinâmicos dessa visão são captados nos mesmos tipos de diagramas da visão de projeto, mas com o foco voltado para as classes ativas que controlam o sistema e as mensagens que passam por elas.

A *visão de implementação* de um sistema abrange os componentes e os artefatos utilizados para a montagem e o fornecimento do sistema físico. Essa visão envolve principalmente o gerenciamento da configuração das versões do sistema, compostas por componentes e arquivos de alguma maneira independentes, que podem ser reunidos de diferentes formas para a produção de um sistema executável. Diz respeito também ao mapeamento de classes lógicas e componentes para artefatos físicos. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de componentes; os aspectos dinâmicos são capturados em diagramas de interações, de estados e de atividades.

A *visão de implantação* de um sistema abrange os nós que formam a topologia de hardware em que o sistema é executado. Essa visão direciona principalmente a distribuição, o fornecimento e a instalação das partes que constituem o sistema físico. Com a UML, os aspectos estáticos dessa visão são capturados em diagramas de implantação; os aspectos dinâmicos são

capturados em diagramas de interações, diagramas de estados e diagramas de atividades.

Cada uma dessas cinco visões pode ser considerada isoladamente, permitindo que diferentes participantes dirijam seu foco para os aspectos da arquitetura do sistema que mais lhes interessam. Essas cinco visões também interagem entre si – os nós na visão de implantação contêm componentes desta visão, que, por sua vez, representa a realização física de classes, interfaces, colaborações e classes ativas provenientes das visões de projeto e de processo. A UML permite expressar cada uma dessas cinco visões.

## O CICLO DE VIDA DO DESENVOLVIMENTO DO SOFTWARE

A UML é amplamente independente do processo. Isso significa que não se limita ao ciclo de vida de desenvolvimento de determinado software. Porém, para obter o máximo proveito da UML, será preciso levar em consideração um processo com as seguintes características:

- » *O resumo sobre o Rational Unified Process é apresentado no Apêndice B; uma descrição do tratamento completo desse processo é apresentada em The Unified Software Development Process e em The Rational Unified Process.*

- » Orientado a caso de uso
- » Centrado na arquitetura
- » Iterativo e incremental

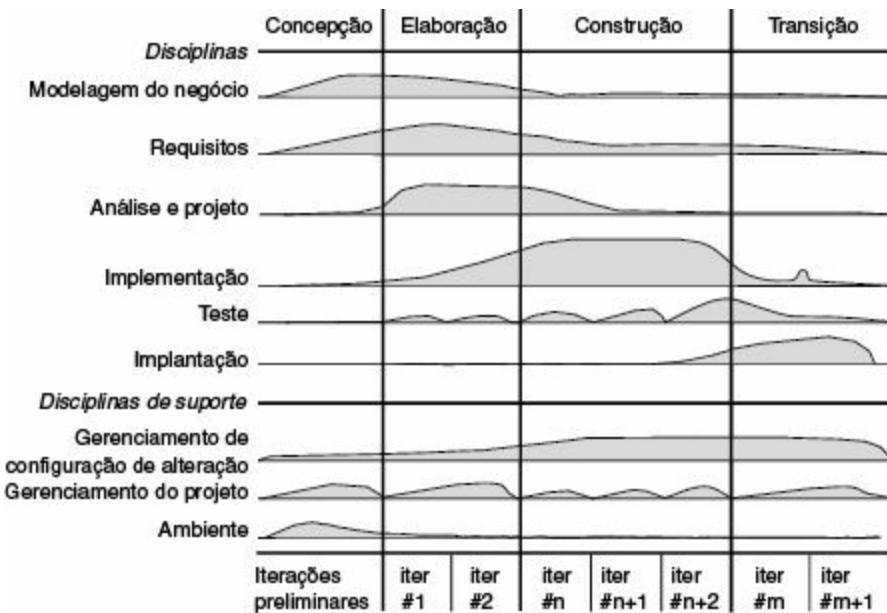
*Orientado a caso de uso* significa que esses casos são utilizados como o principal artefato para o estabelecimento do comportamento desejado do sistema, para a verificação e a validação da arquitetura do sistema, para a realização de testes e para a comunicação entre os participantes do projeto.

*Centrado na arquitetura* significa que a arquitetura do sistema é utilizada como principal artefato para a conceituação, a construção, o gerenciamento e a evolução do sistema em desenvolvimento.

Um *processo iterativo* é aquele que envolve o gerenciamento de um fluxo de versões executáveis. Um *processo incremental* é aquele que envolve a integração contínua da arquitetura do sistema para a produção dessas versões, de maneira que cada nova versão incorpore os aprimoramentos incrementais em relação às demais. Em conjunto, um processo iterativo e incremental é *orientado a riscos*, ou seja, cada nova versão tem como foco atacar e reduzir os riscos mais significativos para o sucesso do projeto.

Esse processo orientado a caso de uso, centrado na arquitetura e processo iterativo/incremental, pode ser desmembrado em fases. Uma *fase* é o intervalo de tempo decorrido entre dois importantes pontos marcos do processo, quando um conjunto bem definido de objetivos é alcançado, os artefatos são concluídos e decisões são tomadas para passar à fase seguinte. Conforme mostra a [Figura 2.24](#), existem quatro fases no ciclo de desenvolvimento de um software: concepção, elaboração, construção e transição. No diagrama, os fluxos de trabalho são representados nessas fases, indicando as variações de graus do foco ao longo do tempo.

A *concepção* é a primeira fase do processo, em que a ideia inicial para o desenvolvimento é levada até o ponto de ser – pelo menos internamente – suficientemente bem fundamentada para assegurar a passagem à fase de elaboração.



**Figura 2.24:**

Ciclo de vida de desenvolvimento de um software

A *elaboração* é a segunda fase do processo, quando a visão do produto e sua arquitetura são definidas. Nessa fase, os requisitos do sistema são articulados e são definidas as prioridades e baseline. Os requisitos do sistema podem abranger desde declarações de caráter geral até critérios precisos de avaliação, em que cada requisito especifica determinado comportamento funcional ou não funcional e proporciona uma base para a realização de testes.

A *construção* é a terceira fase do processo, em que o software chega a uma arquitetura executável básica e destinada à transferência para a comunidade de usuários. Também aqui os requisitos do sistema e principalmente seus critérios de avaliação são constantemente reexaminados em relação às necessidades comerciais do projeto e os recursos são alocados de modo adequado a atacar ativamente os riscos ao projeto.

A *transição* é a quarta fase do processo, em que o software chega às mãos da comunidade de usuários. Raramente o processo de desenvolvimento do software termina aqui, pois, até durante essa fase, o sistema é aprimorado continuamente, bugs são eliminados e são acrescentadas novas características.

O único elemento que diferencia esse processo e que está presente em todas as quatro fases é uma iteração. Uma *iteração* é um conjunto distinto de atividades, com um plano básico e critérios de avaliação que resultam em um sistema que pode ser executado, testado e avaliado. O sistema não precisa ser divulgado externamente. Como a iteração gera um produto executável, o progresso pode ser avaliado e os riscos podem ser reavaliados após cada iteração. Isso significa que o ciclo de vida do desenvolvimento de um software pode ser caracterizado como um fluxo contínuo de evolução de versões executáveis da arquitetura do sistema com correções após cada iteração, a fim de diminuir o risco potencial. É a ênfase na arquitetura como um artefato importante que orienta a UML para o foco na modelagem das diferentes visões da arquitetura de um sistema.

CAPÍTULO

---

3

---

# Hello, World!

## Neste capítulo

- } *Classes e artefatos*
- } *Modelos estáticos e modelos dinâmicos*
- } *Coneções entre modelos*
- } *Estendendo a UML*

**B**rian Kernighan e Dennis Ritchie, os autores da linguagem de programação C, afirmam que “a única maneira de aprender uma nova linguagem de programação é utilizá-la para escrever programas”. O mesmo se aplica à UML. A única forma de aprender a UML é utilizá-la para escrever modelos.

O primeiro programa que muitos desenvolvedores escrevem ao conhecer uma nova linguagem é um programa simples, envolvendo apenas exibir na tela a sequência de caracteres “Hello, World!”. É um ponto de partida razoável, pois dominar essa aplicação trivial proporciona uma gratificação imediata. Além disso, manipula-se toda a infraestrutura necessária para fazer algo ser executado.

É assim que começaremos a usar a UML. Modelar “Hello, World!” será o uso mais simples da UML que você poderá encontrar. Entretanto, essa aplicação é apenas aparentemente fácil, pois, subjacentes à aplicação, existem alguns mecanismos interessantes que a fazem funcionar. Esses mecanismos podem ser modelados facilmente com a UML, proporcionando uma visão enriquecida dessa aplicação simples.

# PRINCIPAIS ABSTRAÇÕES

Em Java, para o applet exibir “Hello, World!” em um navegador Web é bastante simples:

```
import java.awt.Graphics;
class HelloWorld extends java.applet.Applet {
    public void paint (Graphics g) {
        g.drawString("Hello, World!", 10, 10);
    }
}
```

A primeira linha do código:

```
import java.awt.Graphics;
```

faz com que a classe Graphics fique diretamente disponível ao código indicado a seguir. O prefixo `java.awt` especifica o pacote Java em que se encontra a classe `Graphics`.

A segunda linha do código:

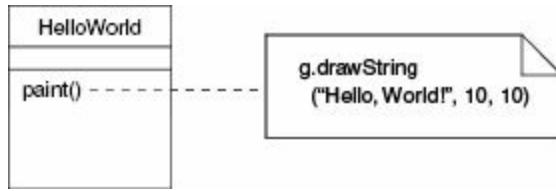
```
class HelloWorld extends java.applet.Applet {
```

introduz uma nova classe chamada `HelloWorld` e especifica que se trata de um tipo de classe semelhante a `Applet`, encontrada no pacote `java.applet`. As três linhas de código seguintes:

```
public void paint (Graphics g) {
    g.drawString ("Hello, World!", 10, 10);
}
```

declaram uma operação chamada `paint`, cuja implementação inicia outra operação, chamada `drawString`, responsável pela exibição da sequência de caracteres “Hello, World!” nas coordenadas fornecidas. No modo orientado a objetos usual, `drawString` é uma operação de um parâmetro chamado `g`, cujo tipo é a classe `Graphics`. É simples a modelagem dessa aplicação na UML. Conforme mostra a [Figura 3.1](#), a classe `HelloWorld` pode ser representada graficamente como um ícone retangular. A operação `paint` é mostrada nessa figura com todos os seus parâmetros formais omitidos e sua implementação especificada na nota anexa.

► As classes são examinadas nos Capítulos 4 e 9.

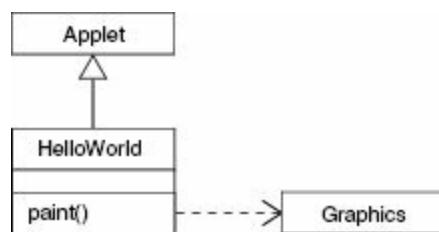


**Figura 3.1:**

As abstrações-chave para HelloWorld

**Nota:** A UML não é uma linguagem visual de programação, apesar de, conforme mostra a figura, permitir – mas sem exigir – uma estreita relação com várias linguagens de programação, como a linguagem Java. A UML se destina a permitir que modelos sejam transformados em código, como também aplicar uma engenharia reversa para converter código em modelos. Alguns itens podem ser escritos mais facilmente na sintaxe de uma linguagem de programação textual (por exemplo, as expressões matemáticas), enquanto outros itens são visualizados graficamente com maior clareza na UML (por exemplo, as hierarquias de classes).

Esse diagrama de classes capta o básico da aplicação “Hello, World!”, mas não inclui vários itens. Conforme especifica o código apresentado anteriormente, duas outras classes – Applet e Graphics – estão envolvidas nessa aplicação e cada uma delas é utilizada de uma maneira diferente. A classe Applet é empregada como mãe de HelloWorld e a classe Graphics é usada na assinatura e implementação de uma de suas operações, paint. Você pode representar essas classes e seus diferentes relacionamentos com a classe HelloWorld, usando um diagrama de classes, conforme mostra a [Figura 3.2](#).



**Figura 3.2:**

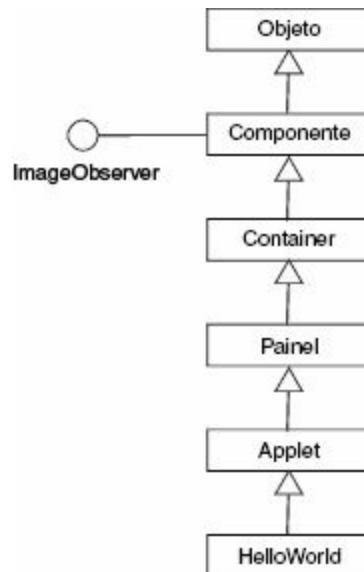
Vizinhos imediatos de HelloWorld

As classes Applet e Graphics são representadas como ícones retangulares. As operações dessas classes não são mostradas e, portanto, seus ícones estão ocultos. A linha com a seta que vai de HelloWorld a Applet representa a generalização, significando, nesse caso, que a classe HelloWorld é filha de Applet. A linha pontilhada que vai de HelloWorld a Graphics representa um relacionamento de dependência, significando que HelloWorld usa a classe Graphics.

► *Os relacionamentos são examinados nos Capítulos 5 e 10.*

Esse não é o final da estrutura a partir da qual HelloWorld é construída. Pesquisando as bibliotecas de Java à procura de Applet e Graphics, você descobrirá que essas duas classes são parte de uma hierarquia maior. Considerando somente as classes que Applet estende e implementa, é possível gerar outro diagrama de classes, conforme mostra a Figura 3.3.

**Nota:** *Essa figura é um bom exemplo de um diagrama gerado pela engenharia reversa de um sistema existente. A engenharia reversa é a criação de um modelo a partir de determinado código.*



**Figura 3.3:**

A hierarquia de heranças da classe HelloWorld

Essa figura deixa claro que HelloWorld é apenas uma ramificação em uma hierarquia de classes muito maior. HelloWorld é filha de Applet; Applet é filha de Panel; Panel é filha de Container; Container é filha de Component; e Component é filha de Object, que é a classe-mãe de todas as classes em Java. Portanto, esse modelo corresponde à biblioteca de Java – cada classe-filha estende a classe-mãe imediata.

O relacionamento entre ImageObserver e Component é um pouco diferente e o diagrama de classes reflete essa diferença. Na biblioteca Java, ImageObserver é uma interface; entre outras coisas, isso significa que não possui uma implementação e requer que outras classes a implementem. Conforme mostra a figura, uma interface pode ser representada como um círculo na UML. O fato de Component implementar ImageObserver está representado pela linha sólida que vai da implementação (Component) até sua interface (ImageObserver).

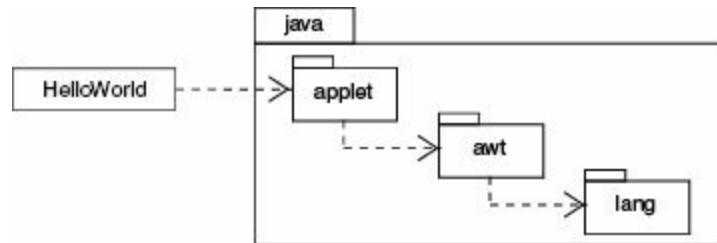
► As interfaces são examinadas no [Capítulo 11](#).

Conforme mostram essas figuras, HelloWorld colabora diretamente apenas com duas classes (Applet e Graphics), que, por sua vez, são apenas uma pequena parte da biblioteca maior de classes predefinidas de Java. Para gerenciar essa extensa coleção, a linguagem Java organiza suas interfaces e classes em vários pacotes diferentes. O pacote-raiz do ambiente Java é chamado, como era de esperar, Java. Aninhados nesse pacote existem diversos outros pacotes, contendo outros pacotes, interfaces e classes. Object se encontra no pacote lang e, portanto, seu caminho completo é java.lang.Object. De modo semelhante, Panel, Container e Component se encontram em awt; a classe Applet se encontra no pacote applet. A interface ImageObserver está no pacote image, que, por sua vez, pertence ao pacote awt; portanto, seu caminho completo é a extensa sequência java.awt.image.ImageObserver.

No diagrama de classes da [Figura 3.4](#), os pacotes são representados na UML por diretórios com guias. Os pacotes podem estar aninhados, com linhas tracejadas representando as dependências entre esses pacotes. Por

exemplo, HelloWorld depende do pacote java.applet e java.applet depende do pacote java.awt.

- Os pacotes são examinados no [Capítulo 12](#).



**Figura 3.4:**

Os pacotes de HelloWorld

## MECANISMOS

A tarefa mais difícil para dominar uma biblioteca tão rica como a da linguagem Java é aprender como suas partes trabalham em conjunto. Por exemplo, como invocar a operação `paint` de `HelloWorld`? Quais operações você deve utilizar para modificar o comportamento desse applet, como exibir a sequência de caracteres em outra cor? Para responder a essa e a outras perguntas, você precisará dispor de um modelo conceitual da maneira como essas classes trabalham em conjunto dinamicamente.

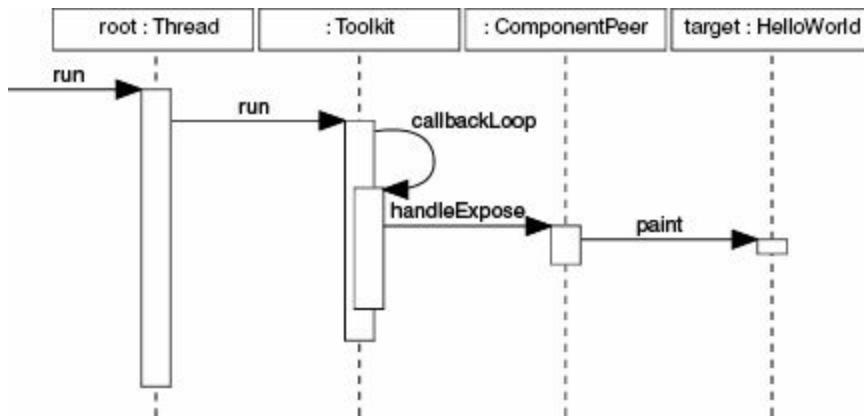
- Padrões e estruturas são examinados no [Capítulo 29](#).

Um estudo da biblioteca de Java revelará que a operação `paint` de `HelloWorld` está na relação de herança de `Component`. Isso ainda mantém a pergunta de como essa operação é invocada. A resposta é que `paint` é chamada como uma parte da execução do thread que contém o applet, conforme mostra a [Figura 3.5](#).

- Os processos e os threads são examinados no [Capítulo 23](#).

A [Figura 3.5](#) mostra a colaboração de vários objetos, incluindo uma instância da classe `HelloWorld`. Os demais objetos fazem parte do ambiente de Java e assim, na maioria dos casos, são encontrados no segundo plano dos applets que você criar. Mostra também uma colaboração entre objetos que podem ser aplicados muitas vezes. Cada coluna mostra um papel na colaboração, ou seja, uma parte que pode ser representada por um objeto diferente em cada execução. Na UML, os papéis são representados da mesma forma que as classes, mas com seus nomes e tipos. Os dois papéis intermediários nesse diagrama são anônimos, porque seus tipos são suficientes para identificá-los na colaboração (mas os dois-pontos e a ausência de sublinhado os identificam como papéis). O Thread inicial chama-se `root`, e o papel `HelloWorld` tem um nome (`target`) conhecido pelo papel `ComponentPeer`.

► As instâncias são examinadas no [Capítulo 13](#).



**Figura 3.5:**

O mecanismo da operação `paint`

Você pode modelar a ordem de eventos utilizando um diagrama de sequência, conforme mostra a [Figura 3.5](#). A sequência inicia-se com a execução do objeto `Thread`, que, por sua vez, chama a operação `run` de `Toolkit`. O objeto `Toolkit` então chama uma de suas próprias operações (`callbackloop`), que, a seguir, chama a operação `handleExpose` de `ComponentPeer`. O objeto `ComponentPeer` então chama a operação `paint` de seu alvo. O objeto `ComponentPeer` assume que

seu atributo é um Component (denominado HelloWorld) e assim a operação paint de HelloWorld é realizada polimorficamente.

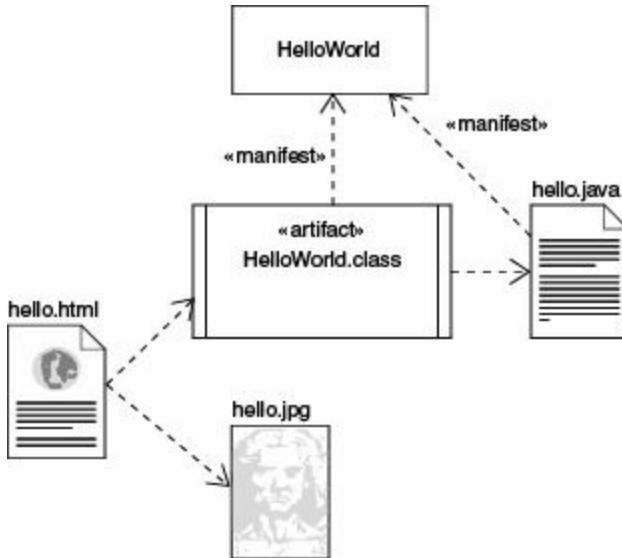
- Os diagramas de sequência são examinados no [Capítulo 19](#).

## ARTEFATOS

Por ser implementado como um applet, “HelloWorld!” nunca aparecerá sozinho, mas tipicamente é parte de alguma página Web. O applet é executado quando a página que o contém estiver aberta, iniciado por algum mecanismo do navegador que executa o objeto Thread do applet. Entretanto, a classe HelloWorld não é diretamente uma parte da página Web. Mais propriamente, é uma forma binária dessa classe, criada por um compilador Java capaz de transformar o código-fonte que representa a classe em um artefato que possa ser executado. Isso sugere uma perspectiva muito diferente do sistema. Enquanto todos os diagramas apresentados anteriormente representavam uma visão lógica do applet, agora estamos considerando uma visão dos artefatos físicos do applet.

Você pode fazer a modelagem dessa visão física usando um diagrama de artefatos, conforme mostra a [Figura 3.6](#).

- Os artefatos são examinados no [Capítulo 26](#).



**Figura 3.6:**  
Artefatos de `HelloWorld`

A classe lógica `HelloWorld` é exibida em cima como um retângulo de classe. Cada um dos ícones mostrados nessa figura representa um artefato da UML na visão da implementação do sistema. Um artefato é uma representação física, como um arquivo. O artefato chamado `hello.java` representa o código-fonte para a classe lógica `HelloWorld`; portanto, trata-se de um arquivo que pode ser manipulado pelas ferramentas de gerenciamento de configuração e ambientes de desenvolvimento. Esse código-fonte pode ser transformado no applet binário `hello.class` por um compilador Java, permitindo ser executado por uma máquina virtual Java em um determinado computador. Tanto o código-fonte quanto o applet binário manifestam – implementam fisicamente – a classe lógica. Esse procedimento é indicado pelas setas pontilhadas com a palavra-chave “manifest”.

O ícone de um artefato é um retângulo com a palavra-chave “artefact”. O applet binário `HelloWorld.class` é uma variação desse símbolo básico, cujas linhas mais grossas indicam que se trata de um artefato executável (assim como ocorre com uma classe ativa). O ícone para o artefato `hello.java` foi substituído por um ícone definido pelo usuário, representando um arquivo de texto. O ícone para a página Web `hello.html` foi definido de maneira semelhante, como uma extensão da notação da UML. Conforme indica a figura, essa página

Web contém um outro artefato, hello.jpg, representado por um ícone de artefato definido pelo usuário, nesse caso fornecendo uma miniatura da imagem gráfica. Como esses três últimos artefatos incluem símbolos gráficos definidos pelo usuário, seus nomes foram colocados fora dos ícones. As dependências entre os artefatos são indicadas pelas setas pontilhadas.

- Os mecanismos de extensibilidade da UML são examinados no Capítulo 6.

**Nota:** Os relacionamentos entre a classe (HelloWorld), seu código-fonte (hello.java) e seu código-objeto (HelloWorld.class) raramente são modelados explicitamente, apesar de que, de vez quando, é útil proceder dessa maneira para visualizar a configuração física do sistema. Por outro lado, é comum visualizar a organização de um sistema baseado na Web como esse, utilizando-se diagramas de artefatos para a modelagem de suas páginas e de outros artefatos executáveis.

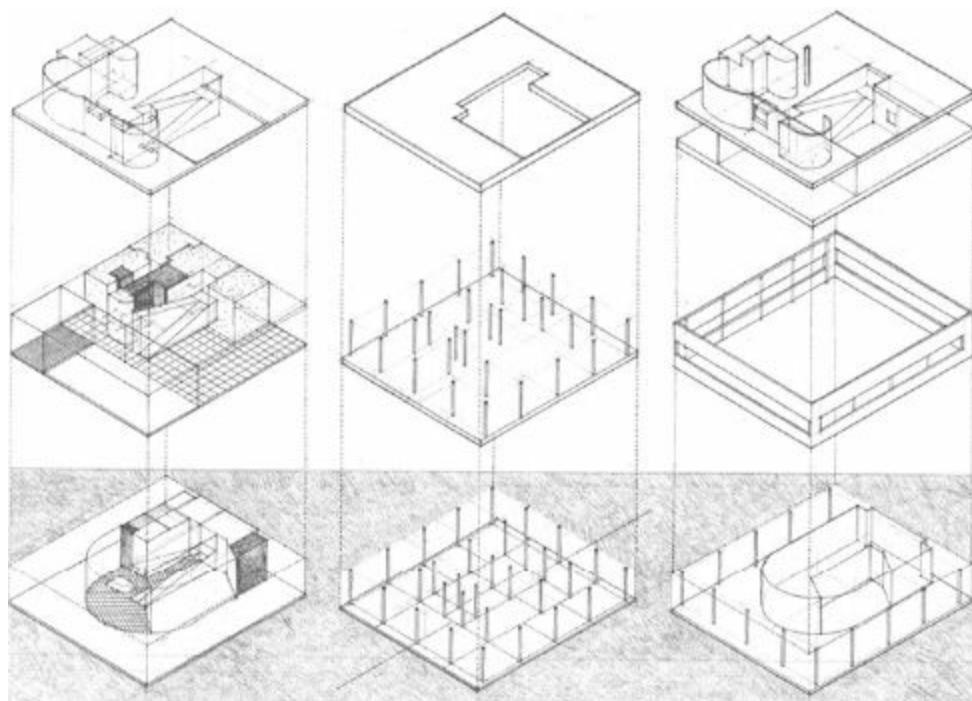
P A R T E

---

2

---

# **MODELAGEM ESTRUTURAL BÁSICA**



CAPÍTULO

---

4

---

# Classes

## Neste capítulo

- » *Classes, atributos, operações e responsabilidades*
- » *A modelagem do vocabulário do sistema*
- » *A modelagem da distribuição de responsabilidades em um sistema*
- » *A modelagem de itens que não são software*
- » *A modelagem de tipos primitivos*
- » *Criando abstrações de qualidade*

**A**s classes são os blocos de construção mais importantes de qualquer sistema orientado a objetos. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. Uma classe implementa uma ou mais interfaces. Você utiliza classes para capturar o vocabulário do sistema que está em desenvolvimento. Essas classes podem incluir abstrações que são parte do domínio do problema, assim como as classes que fazem uma implementação. Você pode usar classes para representar itens de software, de hardware e até itens que sejam puramente conceituais.

► Os recursos avançados das classes são examinados no [Capítulo 9](#).

Classes bem estruturadas anulam fronteiras e formam parte de uma distribuição equilibrada de responsabilidades em um sistema.

## PRIMEIROS PASSOS

A modelagem de um sistema envolve a identificação de itens considerados importantes de acordo com uma determinada visão. Esses itens formam o vocabulário do sistema a ser modelado. Por exemplo, se você está construindo uma casa, paredes, portas, janelas, cômodos e luzes são alguns dos itens que serão importantes para você como proprietário da casa. Cada um desses itens pode ser diferenciado dos demais.

Cada um desses itens também tem um conjunto de propriedades. As paredes têm altura e largura e são sólidas. As portas também têm altura e largura e são sólidas, mas têm o comportamento adicional que permite ser abertas em uma direção. As janelas são semelhantes às portas por ter aberturas que passam através das paredes, mas janelas e portas têm propriedades ligeiramente diferentes. Geralmente (mas nem sempre) as janelas são projetadas de forma a permitir que você olhe através delas em vez de utilizá-las como uma passagem.

Paredes, portas e janelas raramente existem isoladamente; portanto, também é preciso levar em consideração como as instâncias específicas desses itens são distribuídas no conjunto. Os itens que você identificar e os relacionamentos que decidir estabelecer entre eles serão afetados pela maneira como você pretende usar os vários cômodos da casa, como espera passar de um cômodo a outro e o estilo geral e o funcionamento que você deseja criar com essa organização.

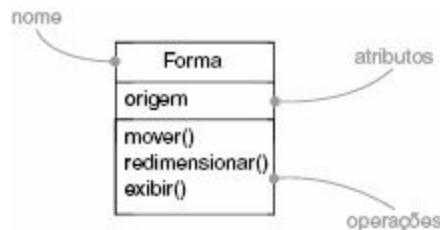
Os usuários estarão interessados em diferentes itens. Por exemplo, os encanadores que ajudam a construir a casa estarão interessados em itens como ralos, sifões e aberturas para ventilação. Você, como proprietário da casa, não cuidará necessariamente desses itens, exceto quando interagirem com o que coincide com o seu ponto de vista, como o local em que deveria ficar um ralo ou onde deveria ser posicionada uma abertura para ventilação em relação ao teto.

Em UML, todos esses itens são modelados como classes. Uma classe é uma abstração de itens que fazem parte de seu vocabulário. A classe não é um objeto individual, mas representa um conjunto inteiro de objetos. Portanto, você pode pensar conceitualmente em “parede” como uma classe de

objetos com determinadas propriedades comuns, como altura, largura, espessura, capacidade de suportar ou não pesos e assim por diante. Você pode também considerar instâncias individuais de paredes, como “a parede do lado sudoeste do meu escritório”.

► Os objetos são examinados no [Capítulo 13](#).

No caso de software, muitas linguagens de programação dispõem de suporte direto para o conceito de classe. Isso é excelente, pois significa que as abstrações que você criar podem ser mapeadas diretamente para a linguagem de programação, ainda que sejam abstrações de itens que não sejam software, como “cliente”, “transação” ou “conversação”. A UML permite a representação gráfica de classes, conforme mostra a [Figura 4.1](#). Essa notação permite visualizar uma abstração independente de qualquer linguagem de programação específica e de uma maneira que torna possível dar ênfase às partes mais importantes de uma abstração: seu nome, atributos e operações.



**Figura 4.1:**  
Classes

## TERMOS E CONCEITOS

Uma *classe* é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. Uma classe é representada graficamente como um retângulo.

**Nomes** Cada classe deve ter um nome que a diferencie de outras classes. O nome é uma sequência de caracteres. Sozinho, é conhecido como *nome simples*; o *nome qualificado* é o nome da classe, tendo como prefixo o nome

do pacote a que essa classe pertence. Uma classe pode ser representada mostrando apenas seu nome, conforme mostra a [Figura 4.2](#).

- *O nome de uma classe deve ser único no pacote que a contém, conforme é apresentado no [Capítulo 12](#).*



**Figura 4.2:**

Nomes simples e nomes qualificados

**Nota:** O nome de uma classe pode ser um texto composto por qualquer número de caracteres e determinados sinais de pontuação (exceto alguns sinais, como os dois-pontos, utilizados para separar o nome da classe e o nome do pacote que a contém) e pode se estender por várias linhas. Na prática, os nomes das classes são substantivos ou expressões breves, definidos a partir do vocabulário do sistema cuja modelagem está sendo feita. Tipicamente, aparece como maiúsculo o primeiro caractere de cada palavra existente no nome da classe, como em Cliente ou SensorDeTemperatura.

## ATRIBUTOS

Um *atributo* é uma propriedade nomeada de uma classe que descreve um intervalo de valores que as instâncias da propriedade podem apresentar. Uma classe pode ter qualquer número de atributos ou mesmo nenhum atributo. Um atributo representa alguma propriedade do item que está sendo modelado, compartilhado por todos os objetos dessa classe. Por exemplo, toda parede tem altura, largura e espessura; você poderia fazer a modelagem de seus clientes de maneira que cada cliente tenha um nome, endereço, número de telefone e data de nascimento. Um atributo é, portanto, uma abstração do tipo de dados ou estados que os objetos da classe podem abranger. Em

determinado momento, um objeto de uma classe terá valores específicos para cada um dos atributos de sua classe. Graficamente, os atributos são listados em um compartimento imediatamente abaixo do nome da classe. Os atributos podem ser representados, exibindo apenas seus nomes, conforme mostra a [Figura 4.3](#).

- *Os atributos estão relacionados à semântica de agregação, conforme é examinado no [Capítulo 10](#).*

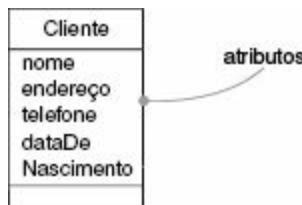


Figura 4.3:  
Atributos

**Nota:** O nome de um atributo pode ser um texto, como os nomes das classes. Na prática, o nome de um atributo é um substantivo ou expressão que, breve, representa alguma propriedade da classe correspondente. Tipicamente, aparece como maiúsculo o primeiro caractere de cada palavra existente no nome do atributo, exceto a primeira letra, como em `nome` ou `suporteDeCarga`. Você pode indicar um atributo, indicando sua classe e possivelmente um valor inicial padrão, conforme mostra a [Figura 4.4](#).

- *Você pode especificar outras características de um atributo, como marcá-lo somente para leitura ou compartilhá-lo com todos os objetos da classe, conforme é examinado no [Capítulo 9](#).*

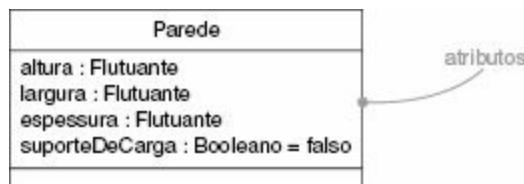


Figura 4.4:  
Atributos e sua classe

**Operações** Uma operação é a implementação de um serviço que pode ser solicitado por algum objeto da classe para modificar o comportamento. Em outras palavras, uma operação é uma abstração de algo que pode ser feito com um objeto e que é compartilhado por todos os objetos dessa classe. Uma classe pode ter qualquer número de operações ou até não ter nenhuma operação. Por exemplo, em uma biblioteca de janelas, como aquela encontrada no pacote `awt` da linguagem Java, todos os objetos da classe `Retângulo` podem ser movidos, redimensionados ou ter suas propriedades examinadas. Muitas vezes (mas nem sempre), a chamada a uma operação em determinado objeto altera os dados ou o estado do objeto. Graficamente, as operações aparecem listadas em um compartimento imediatamente abaixo dos atributos da classe. As operações podem ser representadas, exibindo somente seus nomes, conforme mostra a [Figura 4.5](#).

- Você pode especificar a implementação de uma operação usando uma nota, conforme descreve o [Capítulo 6](#), ou usando um diagrama de atividades, conforme é apresentado no [Capítulo 20](#).



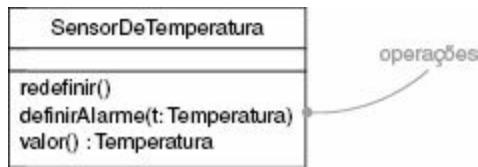
**Figura 4.5:**

Operações

**Nota:** O nome de uma operação pode ser um texto, como os nomes das classes. Na prática, o nome de uma operação é um verbo ou uma locução verbal breve, representando algum comportamento da classe correspondente. Tipicamente, aparece como maiúsculo o primeiro caractere de cada palavra existente no nome da operação, exceto a primeira letra, como em `mover` ou `está Vazio`.

Você poderá especificar uma operação, indicando sua assinatura, que contém o nome, o tipo e o valor-padrão de todos os parâmetros e (no caso das funções) o tipo a ser retornado, conforme mostra a [Figura 4.6](#).

- Você pode especificar outras características de uma operação, marcando-a como polimórfica ou constante, ou especificando sua visibilidade, conforme é apresentado no [Capítulo 9](#).



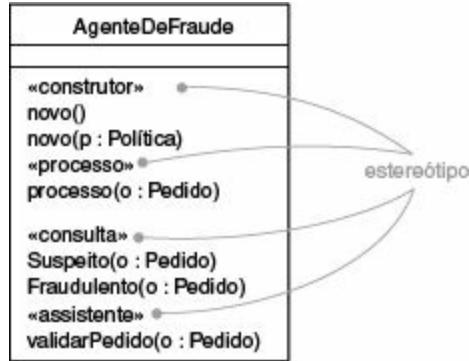
**Figura 4.6:**

Operações e suas assinaturas

**Organização de atributos e operações** Ao representar uma classe, não é preciso exibir todos os atributos e operações ao mesmo tempo. Na verdade, na maioria dos casos, isso não é possível (haverá um número muito grande de itens a serem incluídos em uma mesma figura), nem é adequado (somente um subconjunto desses atributos e operações será relevante para uma determinada visão). Por esses motivos, você poderá suprimir elementos de uma classe, significando que podem ser escolhidos somente alguns ou nenhum dos atributos e operações da classe a serem mostrados. Para especificar explicitamente a existência de mais atributos ou propriedades do que as apresentadas, termine cada lista com reticências (...). Você também pode suprimir o comportamento inteiramente. Nesse caso, não será possível informar se há atributos ou operações ou quantos são.

Para uma melhor organização de listas extensas de atributos e operações, cada grupo pode receber um prefixo com uma categoria descritiva, utilizando estereótipos, conforme mostra a [Figura 4.7](#).

- Os estereótipos são examinados no [Capítulo 6](#).



**Figura 4.7:**

Estereótipos para as características da classe

## RESPONSABILIDADES

Uma *responsabilidade* é um contrato ou obrigações de uma determinada classe. Ao criar uma classe, você estará criando uma declaração de que todos os objetos dessa classe têm o mesmo tipo de estado e o mesmo tipo de comportamento. Em um nível mais abstrato, esses atributos e operações correspondentes são apenas as características com as quais as responsabilidades das classes são executadas. Uma classe Parede é responsável pelo conhecimento acerca de altura, largura e espessura; uma classe AgenteDeFraude, como costuma haver em aplicações de cartão de crédito, é responsável pelo processamento de pedidos e pela identificação de que são legítimos, suspeitos ou fraudulentos; uma classe SensorDeTemperatura é responsável por medir a temperatura e disparar um alarme, caso a temperatura alcance determinado ponto.

- As *responsabilidades* são exemplos de estereótipos definidos, conforme é examinado no [Capítulo 6](#).

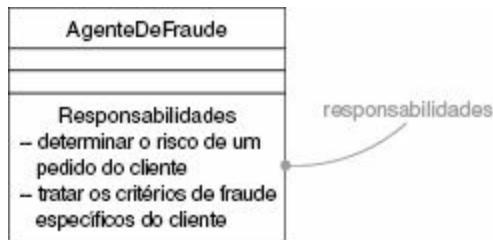
Ao fazer a modelagem de classes, um bom ponto de partida é especificar as responsabilidades dos itens existentes em seu vocabulário. Técnicas como cartões CRC e análises baseadas em casos de uso são de grande ajuda aqui. Uma classe pode ter qualquer número de responsabilidades, apesar de que, na prática, toda classe bem estruturada tem pelo menos uma responsabilidade e, quase sempre, um punhado de responsabilidades. À medida que aprimorar

seus modelos, você traduzirá essas responsabilidades em um conjunto de atributos e operações, capazes de atender da melhor maneira às responsabilidades da classe.

- A modelagem da semântica de uma classe é examinada no [Capítulo 9](#).

As responsabilidades podem ser representadas graficamente em um compartimento separado, no final do ícone da classe, conforme mostra a [Figura 4.8](#).

- Você também pode representar as responsabilidades de uma classe em uma nota, conforme é apresentado no [Capítulo 6](#).



**Figura 4.8:**  
Responsabilidades

**Nota:** As responsabilidades são apenas texto em formato livre. Na prática, uma única responsabilidade pode ser escrita como uma expressão, uma oração ou um breve parágrafo.

## OUTRAS CARACTERÍSTICAS

Atributos, operações e responsabilidades são as características mais comuns de que você precisará ao criar abstrações. De fato, na maioria dos modelos que você construir, a forma básica dessas três características será suficiente para estabelecer a semântica mais importante de suas classes. De vez em quando, porém, será necessário visualizar ou especificar outras características, como a visibilidade de atributos e operações individuais; características específicas da linguagem em relação a uma operação, como a

possibilidade de ser polimórfica ou constante; ou mesmo as exceções que os objetos da classe poderiam produzir ou manipular. Essas e muitas outras características podem ser expressas em UML, mas são tratadas como conceitos avançados.

- *Conceitos de classes avançadas são apresentados no [Capítulo 9](#).*

Ao construir modelos, você logo descobrirá que praticamente todas as abstrações criadas são algum tipo de classe. Às vezes, você desejará separar a implementação de uma classe de sua especificação e isso pode ser expresso na UML pela utilização de interfaces.

- *As interfaces são examinadas no [Capítulo 11](#).*

Ao começar a projetar a implementação de uma classe, você precisará modelar sua estrutura interna como um conjunto de peças conectadas. Você pode expandir uma classe de nível superior por meio de várias camadas de estrutura interna para obter o projeto final.

- *A estrutura interna é examinada no [Capítulo 15](#).*

Ao começar a construir modelos mais complexos, você também descobrirá que serão encontrados os mesmos tipos de classes repetidamente, como as classes que representam processos e threads concorrentes ou classes que representam itens físicos, como applets, Java Beans, arquivos, páginas Web e hardware. Uma vez que esses tipos de classes são tão comuns e representam abstrações de arquitetura importantes, a UML oferece classes ativas (representando processos e threads) e classificadores, como artefatos (representando componentes físicos de software) e nós (representando dispositivos de hardware).

- *As classes ativas, os componentes e os nós são examinados nos Capítulos [23](#), [25](#) e [27](#), e os artefatos, no Capítulo 26.*

Por fim, as classes raramente aparecem sozinhas. Em vez disso, ao construir seus modelos, tipicamente você dirigirá o foco para grupos de classes que interagem entre si. Na UML, essas sociedades de classes formam colaborações e costumam ser visualizadas em diagramas de classes.

- » *Os diagramas de classes são examinados no [Capítulo 8](#).*

## TÉCNICAS BÁSICAS DE MODELAGEM

### A MODELAGEM DO VOCABULÁRIO DO SISTEMA

Geralmente você usará as classes para fazer a modelagem de abstrações definidas a partir do problema que você está tentando solucionar ou a partir da tecnologia empregada para implementar uma solução para esse problema. Cada uma dessas abstrações é uma parte do vocabulário do sistema, que, em conjunto, representa as coisas importantes para os usuários e implementadores.

Para os usuários, não é difícil identificar a maioria das abstrações, pois elas costumam ser definidas a partir das coisas que os usuários já utilizam para descrever seu sistema. Técnicas como os cartões CRC e a análise baseada em casos de uso são formas excelentes para auxiliar os usuários a descobrir essas abstrações. Para os implementadores, tipicamente as abstrações são os itens disponíveis na tecnologia que constituem partes da solução.

- » *Os casos de uso são examinados no [Capítulo 17](#).*

Para fazer a modelagem do vocabulário de um sistema:

- » Identifique os itens que os usuários ou os implementadores usam para descrever o problema ou a solução. Use os cartões CRC e a análise baseada em casos de uso para ajudar a descoberta dessas abstrações.
- » Para cada uma das abstrações, identifique um conjunto de responsabilidades. Verifique se todas as classes estão bem definidas e se

existe um bom equilíbrio de responsabilidades entre as classes.

- » Forneça os atributos e as operações necessárias para a execução dessas responsabilidades para cada uma das classes.

A [Figura 4.9](#) mostra um conjunto de classes definidas a partir de um sistema de vendas de varejo, incluindo Cliente, Pedido e Produto. Essa figura inclui algumas outras abstrações relacionadas, estabelecidas a partir do vocabulário do problema, como Entrega (utilizada para rastrear os pedidos), Fatura (utilizada para solicitar pedidos) e Armazém (o local em que os produtos se encontram antes da entrega). Também existe uma abstração relacionada à solução, Transação, que se aplica a pedidos e fretes.

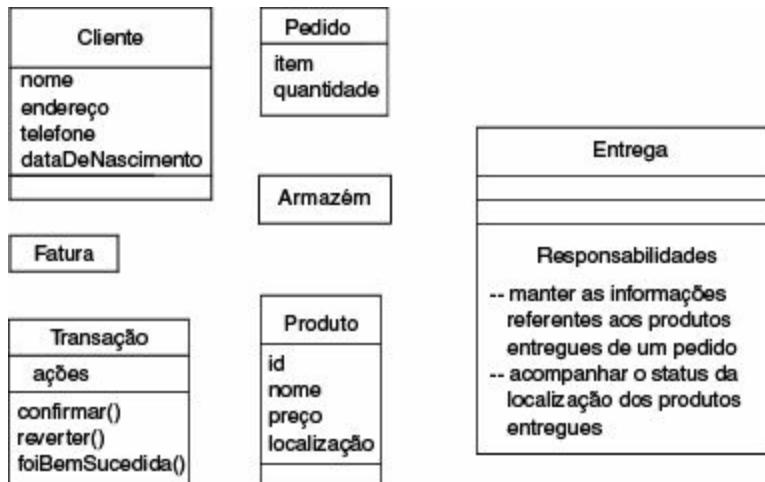


Figura 4.9:

A modelagem do vocabulário de um sistema

À medida que aumenta o tamanho de seus modelos, muitas das classes que você encontrará tenderão a ser reunidas em grupos conceitual e semanticamente relacionadas. Na UML, você pode usar os pacotes para fazer a modelagem desses agrupamentos de classes.

- » Os pacotes são examinados no [Capítulo 12](#).

Em sua maioria, seus modelos raramente serão completamente estáticos. Em vez disso, muitas abstrações existentes no vocabulário de seu sistema

deverão interagir entre si de maneira dinâmica. Na UML, existem várias formas para a modelagem desse comportamento dinâmico.

- » *A modelagem do comportamento é examinada nas Partes 4 e 5.*

## **A MODELAGEM DA DISTRIBUIÇÃO DE RESPONSABILIDADES EM UM SISTEMA**

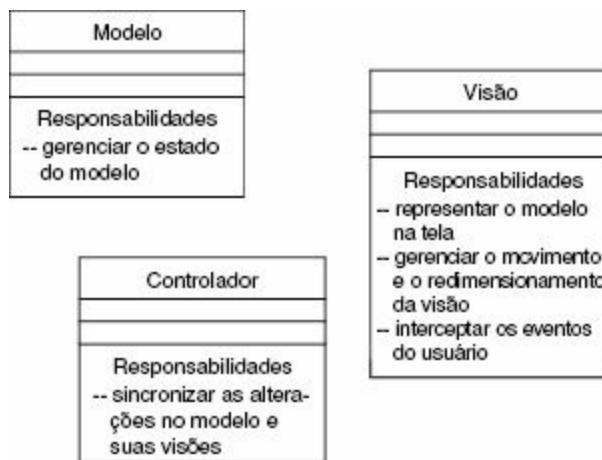
Após começar a modelar mais do que um punhado de classes, você desejará ter certeza de que suas abstrações proporcionam um conjunto equilibrado de responsabilidades. Isso significa que você não deseja classes muito grandes ou pequenas. Cada classe deverá fazer algo de modo eficiente. Se você criar abstrações de classes muito grandes, logo descobrirá que será difícil modificar ou reutilizar seus modelos. Se as suas abstrações de classes forem muito pequenas, acabará havendo um número muito maior de abstrações do que você será capaz de gerenciar ou compreender de modo razoável. Você pode utilizar a UML para auxiliá-lo a visualizar e a especificar esse equilíbrio de responsabilidades. Para fazer a modelagem da distribuição de responsabilidades em um sistema:

- » Identifique um conjunto de classes que trabalham em conjunto com a finalidade de executar algum comportamento.
- » Identifique um conjunto de responsabilidades para cada uma dessas classes.
- » Considere esse conjunto de classes como um todo, divida em abstrações menores as classes que tenham muitas responsabilidades, reúna classes pequenas, com responsabilidades triviais, em classes maiores e redistribua as responsabilidades de forma que todas as abstrações permaneçam razoavelmente independentes.
- » Leve em consideração os modos como essas classes colaboram umas com as outras e redistribua suas responsabilidades para que, em uma colaboração, nenhuma classe tenha muito ou pouco a fazer.

► As colaborações são examinadas no [Capítulo 28](#).

Por exemplo, a [Figura 4.10](#) mostra um conjunto de classes estabelecido pelo Smalltalk, mostrando a distribuição de responsabilidades entre as classes Modelo, Visão e Controlador. Observe como todas essas classes trabalham em conjunto, de modo que nenhuma classe tenha muito ou pouco a fazer.

► Esse conjunto de classes forma um padrão, conforme é apresentado no [Capítulo 29](#).



**Figura 4.10:**

A modelagem da distribuição de responsabilidades em um sistema

## A MODELAGEM DE ITENS QUE NÃO SÃO SOFTWARE

De vez em quando, as coisas a serem modeladas podem não apresentar qualquer elemento análogo de software. Por exemplo, as pessoas que enviam pedidos e os robôs que empacotam automaticamente os pedidos para entregar a partir de um armazém poderiam fazer parte do fluxo de trabalho a ser modelado em um sistema de vendas de varejo. Sua aplicação poderia não conter qualquer parte de software que os represente (ao contrário dos clientes do exemplo anterior, pois o sistema provavelmente deverá manter informações sobre esses clientes). Para fazer a modelagem de itens que não são software:

» Faça a modelagem do item que você está abstraindo como uma classe.

» Para diferenciar esses itens a partir dos blocos de construção definidos na UML, crie um novo bloco de construção usando estereótipos para especificar a nova semântica e fornecer uma indicação visual distintiva.

► *Os estereótipos são examinados no [Capítulo 6](#).*

» Se o item a ser modelado for algum tipo de hardware que contenha seu próprio software, considere essa modelagem também como um tipo de nó, para ser capaz de expandir sua estrutura posteriormente.

► *Os nós são examinados no [Capítulo 27](#).*

**Nota:** A UML se destina principalmente à modelagem de sistemas complexos de software, apesar de que, em conjunção com linguagens textuais para modelagem de hardware, tal como VHDL, a UML pode ser suficientemente expressiva para a modelagem de sistemas de hardware. O OMG também produziu uma extensão UML chamada SysML para modelagem de sistemas.

Como exemplifica a [Figura 4.11](#), é perfeitamente normal fazer a abstração de humanos (como AgenteDeContasAReceber) e de hardware (como Robô) como classes, pois cada um representa um conjunto de objetos com estrutura e comportamento comuns.

► *Itens que são externos ao sistema costumam ser modelados como atores, conforme é apresentado no [Capítulo 17](#).*



**Figura 4.11:**

A modelagem de itens que não são software

## A MODELAGEM DE TIPOS PRIMITIVOS

No outro extremo, os itens a serem modelados podem ser definidos diretamente a partir da linguagem de programação que está sendo empregada

para implementar uma solução. Tipicamente, essas abstrações envolvem tipos primitivos, como inteiros, caracteres, sequências de caracteres e até tipos enumerados que você mesmo poderia criar.

- ➡ Os tipos são examinados no [Capítulo 11](#).

Para fazer a modelagem de tipos primitivos:

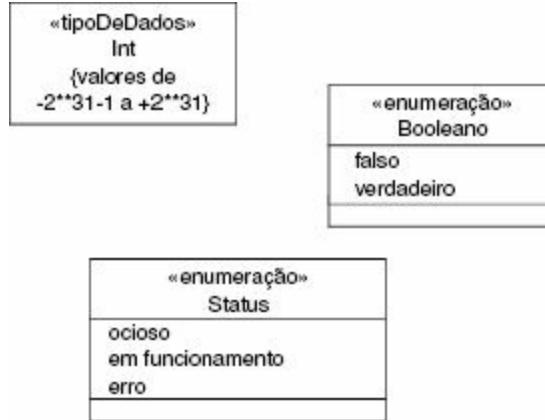
- » Faça a modelagem do item que você está abstraindo como um tipo ou uma enumeração, representada por uma notação de classe com o estereótipo apropriado.
- » Caso seja necessário especificar o intervalo de valores associados a esse tipo, utilize restrições.

- ➡ As restrições são descritas no [Capítulo 6](#).

Conforme mostra a [Figura 4.12](#), esses itens podem ser modelados na UML como tipos ou enumerações, representadas da mesma forma que as classes, mas explicitamente marcados via estereótipos. Tipos primitivos, como inteiros (representados pela classe `Int`), são modelados como tipos, e você pode usar uma restrição para indicar explicitamente o intervalo de valores que esses itens podem receber; a semântica de tipos primitivos deve ser definida externamente à UML. Tipos de enumeração, como `Boolean` e `Status`, podem ser modelados como enumerações, com seus valores individuais listados no compartimento do atributo (observe que eles não são atributos). Tipos de enumeração também podem definir operações.

- ➡ Os tipos são apresentados no [Capítulo 11](#).

**Nota:** Algumas linguagens, como C e C++, permitem definir um valor inteiro equivalente para uma enumeração. Você pode fazer a modelagem dessa característica na UML anexando uma nota a uma enumeração como diretriz de implementação. Valores inteiros não são necessários para a modelagem lógica.



**Figura 4.12:**

A modelagem de tipos primitivos

## DICAS E SUGESTÕES

Ao fazer a modelagem de classes na UML, lembre-se de que todas as classes devem ser mapeadas em alguma abstração tangível ou conceitual no domínio do usuário final ou do implementador. Uma classe bem estruturada:

- › Proporciona uma abstração clara de algo definido a partir do vocabulário do domínio do problema ou do domínio da solução.
- › Incorpora um conjunto pequeno e bem definido de responsabilidades e é capaz de executá-las todas de modo eficiente.
- › Fornece uma nítida separação da especificação da abstração e sua implementação.
- › É compreensível e simples, além de extensível e adaptável.

Ao definir uma classe na UML:

- › Mostre somente aquelas propriedades da classe que são importantes para a compreensão da abstração em seu contexto.
- › Organize extensas listas de atributos e operações, agrupando-os de acordo com suas categorias.
- › Exiba, em um mesmo diagrama de classes, classes que sejam relacionadas.

CAPÍTULO

---

5

---

# Relacionamentos

## Neste capítulo

- » *Relacionamentos de dependência, generalização e associação*
- » *A modelagem de dependências simples*
- » *A modelagem de herança simples*
- » *A modelagem de relacionamentos estruturais*
- » *A criação de redes de relacionamentos*

Ao construir abstrações, você descobrirá que há um número muito pequeno de classes que trabalham sozinhas. Em vez disso, a maioria das classes colabora com outras de várias maneiras. Portanto, ao fazer a modelagem de um sistema, será necessário não identificar somente os itens que formam o vocabulário do sistema, mas também modelar como esses itens relacionam-se entre si.

Na modelagem orientada a objetos, existem três tipos de relacionamentos especialmente importantes: *dependências*, que representam relacionamentos de utilização entre as classes (incluindo relacionamentos de refinamento, rastreamento e vínculos); *generalizações*, que relacionam classes generalizadas a suas especializações; e *associações*, que representam relacionamentos estruturais entre objetos. Cada um desses relacionamentos fornece uma forma diferente de combinações de abstrações.

- As características avançadas dos relacionamentos são apresentadas no [Capítulo 10](#).

A construção de redes de relacionamentos não é diferente da criação de uma distribuição equilibrada de responsabilidades entre as classes. Exagerando na engenharia, você acabará obtendo uma massa confusa de relacionamentos que tornarão seu modelo incompreensível; minimizando a engenharia, perderá boa parte da riqueza de seu sistema embutida na forma de colaboração dos itens.

## PRIMEIROS PASSOS

Quando você está construindo uma casa, itens como paredes, portas, janelas, prateleiras e luzes farão parte de seu vocabulário. Entretanto, nenhum desses itens funciona sozinho. As paredes estão conectadas a outras paredes. Portas e janelas são colocadas nas paredes, formando passagens para as pessoas e para a luz natural. Prateleiras e luzes estão anexadas fisicamente a paredes e tetos. Paredes, portas, janelas, prateleiras e luzes são agrupadas para a formação de itens de um nível superior, como salas.

Entre os itens não serão encontrados apenas relacionamentos estruturais, mas também relacionamentos de outros tipos. Por exemplo, certamente sua casa tem janelas, mas provavelmente existem muitos tipos de janelas. Poderia haver janelas maiores que não são abertas, como também pequenas janelas na cozinha que podem ser abertas. Algumas janelas poderiam ser abertas no sentido vertical e outras, no sentido horizontal. Algumas têm um único painel de vidro e outras têm vários. Quaisquer que sejam as diferenças, existe algo próprio de janelas em cada uma delas: todas consistem em aberturas nas paredes, destinadas à passagem de luz, de ar e, de vez em quando, de pessoas.

Na UML, os modos pelos quais os itens podem estar conectados a outros, isto é, logicamente ou fisicamente, são modelados como relacionamentos. Na modelagem orientada a objetos, existem três tipos de relacionamentos que são os mais importantes: dependências, generalizações e associações.

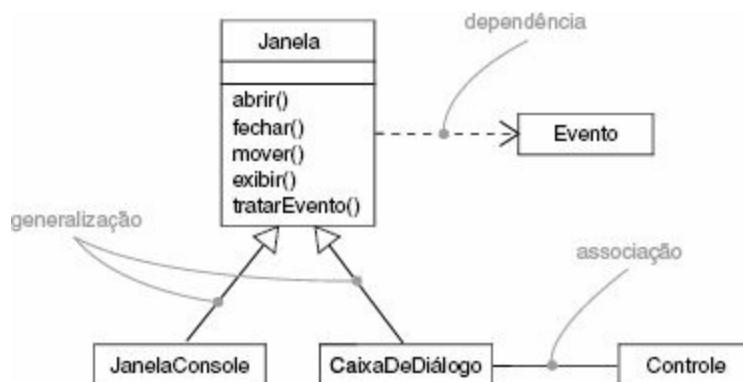
1. As *dependências* são relacionamentos de utilização. Por exemplo, os canos dependem do aquecedor para fornecerem água quente.

2. As *associações* são relacionamentos estruturais entre instâncias. Por exemplo, as salas são formadas por paredes e outros itens; as próprias paredes podem ter portas e janelas embutidas; os canos podem ser passados por dentro das paredes.
3. As *generalizações* conectam classes generalizadas a outras mais especializadas, o que é conhecido como relacionamentos subclasse/superclasse ou filha/mãe. Por exemplo, janelas panorâmicas são grandes e com painéis fixos de vidro; janelas de quartos costumam ter mais de um painel de vidro e abrir de um lado a outro.

Esses três tipos de relacionamentos abrangem os modos mais importantes de colaboração entre os itens. Não causa nenhuma surpresa o fato de esses tipos de relacionamentos também mapearem adequadamente as formas fornecidas pela maioria das linguagens de programação em relação à conexão de objetos.

- Outros tipos de relacionamentos como *realizações* e *refinamentos* são examinados no [Capítulo 10](#).

A UML oferece uma representação gráfica para cada um desses tipos de relacionamentos, conforme mostra a [Figura 5.1](#). Essa notação permite a visualização de relacionamentos, independentemente de qualquer linguagem específica de programação, e de uma maneira que dá ênfase às partes mais importantes do relacionamento: seu nome, os itens por ele conectados e suas propriedades.



## TERMOS E CONCEITOS

Um *relacionamento* é uma conexão entre itens. Em uma modelagem orientada a objetos, os três relacionamentos mais importantes são as dependências, as generalizações e as associações. Um relacionamento é representado graficamente como um caminho, com tipos diferentes de linhas para diferenciar os tipos de relacionamentos.

### DEPENDÊNCIA

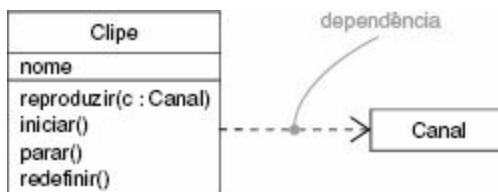
Uma *dependência* é um relacionamento de utilização, determinando que um item (por exemplo, a classe Janela) usa as informações e os serviços de outro item (por exemplo, a classe Evento), mas não necessariamente o inverso. A dependência é representada graficamente como linhas tracejadas apontando o item do qual o outro depende. Use as dependências sempre que quiser indicar algum item dependendo de outro.

Com muita frequência, você usará as dependências no contexto das classes para mostrar que uma classe usa outra como argumento na assinatura de uma operação; ver a [Figura 5.2](#). Esse é essencialmente o caso de um relacionamento de utilização – se a classe utilizada for modificada, a operação da outra classe pode ser afetada, pois a classe utilizada pode agora apresentar uma interface ou comportamento diferente. Na UML, também podem ser criadas dependências entre muitos outros itens, principalmente anotações e pacotes.

- As notas são examinadas no [Capítulo 6](#); os pacotes são apresentados no [Capítulo 12](#).

**Nota:** Uma dependência pode ter um nome, apesar de raramente os nomes serem necessários, a menos que você tenha um modelo com muitas dependências e precise fazer referência a essas dependências ou diferenciá-las. Será mais comum usar estereótipos para distinguir vários tipos de dependências.

- Diferentes tipos de dependências são examinados no [Capítulo 10](#); estereótipos são examinados no [Capítulo 6](#).



**Figura 5.2:**  
Dependências

---

## GENERALIZAÇÃO

Uma *generalização* é um relacionamento entre itens gerais (chamados superclasses ou classes-mãe) e tipos mais específicos desses itens (chamados subclasses ou classes-filha). Muitas vezes, as generalizações são chamadas relacionamentos “é um tipo de”: um item (como a classe JanelaOval) é um tipo de item mais geral (por exemplo, a classe Janela). A generalização significa que os objetos da classe-filha podem ser utilizados em qualquer local em que a classe-mãe ocorra, mas não vice-versa. Em outras palavras, a generalização significa que a filha é substituível por uma declaração da mãe. A filha herda as propriedades da mãe, principalmente seus atributos e operações. Frequentemente – mas não sempre – as filhas têm atributos e operações além daqueles encontrados nas respectivas mães. A operação de uma filha, que tenha a mesma assinatura de uma operação da mãe, prevalecerá em relação à operação da mãe; isso é conhecido como polimorfismo. Para serem iguais, duas operações devem ter a mesma assinatura (mesmos nome e parâmetros). Uma generalização é representada graficamente como linhas sólidas com uma grande seta triangular apontando a mãe, conforme mostra a [Figura 5.3](#). Use as generalizações quando desejar mostrar os relacionamentos mãe/filha. Uma classe pode ter zero, uma ou mais classes-mãe. Uma classe que não tenha uma classe-mãe (superior), mas tenha uma ou mais classes-filha, é chamada classe-raiz ou classe de base.

Uma classe que não tem classes-filha é chamada classe-folha. Uma classe que tem exatamente uma única classe-mãe é identificada como usando uma única herança; uma classe com mais de uma classe-mãe é identificada como usando heranças múltiplas.

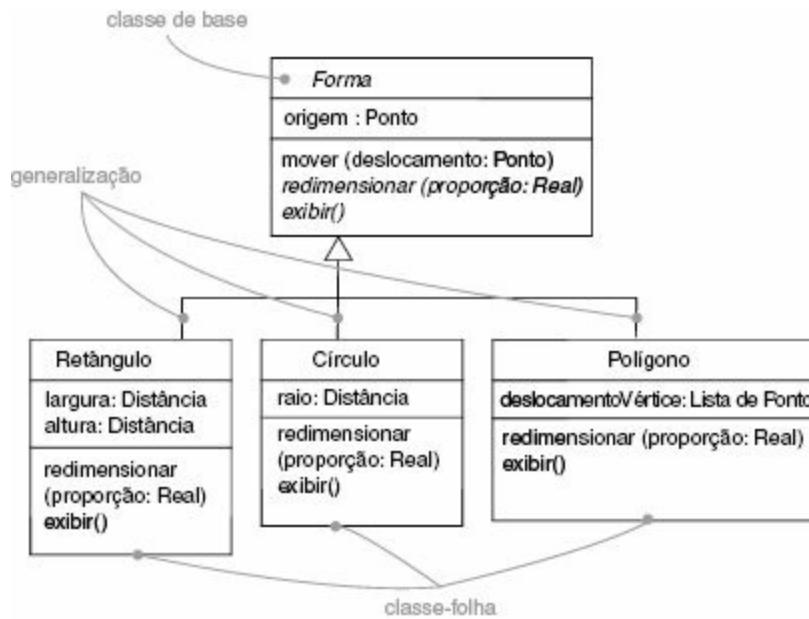
Com muita frequência, você usará as generalizações existentes entre classes e interfaces para exibir relacionamentos de herança. Na UML, você também pode criar generalizações entre outros tipos de classificadores – como nós.

- Os pacotes são examinados no [Capítulo 12](#).

## ASSOCIAÇÃO

Uma *associação* é um relacionamento estrutural que especifica objetos de um item conectados a objetos de outro item. A partir de uma associação conectando duas classes, você é capaz de navegar do objeto de uma classe até o objeto de outra classe e vice-versa. É inteiramente válido ter as duas extremidades do círculo de uma associação retornando à mesma classe. Isso significa que, a partir de um objeto da classe, você poderá criar vínculos com outros objetos da mesma classe. Uma associação que estabelece uma conexão exata a duas classes é chamada uma associação binária. Apesar de não ser comum, você poderá ter associações conectando mais de duas classes, chamadas *associações enésimas*. Uma associação é representada graficamente como uma linha sólida conectando a mesma classe ou classes diferentes. Use as associações sempre que desejar exibir relacionamentos estruturais.

- As associações e as dependências (mas não os relacionamentos de generalização) poderão ser recíprocos, conforme é apresentado no [Capítulo 10](#).



**Figura 5.3:**  
Generalização

**Nota:** Uma generalização com um nome indica uma decomposição das subclasses de uma superclasse em um determinado aspecto, ao que chamamos conjunto de generalizações. Vários conjuntos de generalizações são ortogonais; a superclasse pretende ser especializada usando a herança múltipla para selecionar uma subclass de cada conjunto de generalizações. Esse é um tópico avançado que não será abordado neste livro.

Além dessa forma básica, existem quatro tipos de aprimoramentos a serem aplicados às associações.

**Nome** Uma associação pode ter um nome, que pode ser utilizado para descrever a natureza do relacionamento. Portanto, não havendo ambiguidade acerca de seu significado, você pode atribuir uma direção para o nome, fornecendo um triângulo de orientação que aponta a direção como o nome deverá ser lido, conforme mostra a [Figura 5.4](#).

- *Não confunda a direção do nome com a navegação da associação, conforme é apresentado no [Capítulo 10](#).*



**Figura 5.4:**  
Nomes de associações

**Nota:** Apesar da possibilidade de a associação ter um nome, tipicamente não será necessário incluí-lo, se você atribuir explicitamente nomes de extremidades à associação. Se houver mais de uma associação conectando as mesmas classes, será necessário usar nomes de associação ou nomes de extremidades da associação para distingui-las. Se uma associação tiver mais de uma extremidade na mesma classe, será necessário usar os nomes de extremidades da associação para distinguir as extremidades. Se houver apenas uma associação entre um par de classes, alguns modeladores omitirão os nomes, mas é melhor fornecê-los para que o propósito da associação fique claro.

**Papel** Quando uma classe participa de uma associação, ela tem um papel específico a executar nesse relacionamento; o papel é apenas a face que a classe próxima a uma das extremidades apresenta à classe encontrada na outra extremidade da associação. É possível nomear explicitamente o papel desempenhado por uma classe na associação. O papel desempenhado por uma extremidade da associação chama-se *nome da extremidade*. Na [Figura 5.5](#), Pessoa, desempenhando o papel de funcionário, está associado a empresa, que desempenha o papel de empregador.

- Os papéis estão relacionados à semântica das interfaces, conforme é apresentado no [Capítulo 11](#).



**Figura 5.5:**  
Nomes de extremidades da associação (nomes de papéis)

**Nota:** A mesma classe pode executar papéis iguais ou diferentes em outras associações.

**Nota:** Um atributo pode ser considerado como uma associação de mão única de uma classe. O nome do atributo corresponde ao nome da extremidade na extremidade da associação da classe.

**Multiplicidade** Uma associação representa um relacionamento estrutural existente entre objetos. Em muitas situações de modelagem, é importante determinar a quantidade de objetos que podem ser conectados pela instância de uma associação. Essa “quantidade” é chamada de multiplicidade do papel de uma associação. Ela representa um intervalo de inteiros que especifica o tamanho possível do conjunto de objetos relacionados. É escrita como uma expressão com valores mínimo e máximo, que podem ser iguais; dois pontos são usados para separar os valores mínimo e máximo. Ao determinar a multiplicidade em uma das extremidades de uma associação, você está especificando que, para cada objeto da classe encontrada na extremidade oposta, deve haver a mesma quantidade de objetos na próxima extremidade. O número de objetos deve estar no intervalo dado. Pode ser apresentada uma multiplicidade de exatamente um (1), zero ou um (0..1), muitos (0..\*) ou um ou mais (1..\*). É possível fornecer um intervalo inteiro (como 2..5). Você pode até determinar o número exato (por exemplo, 3, que equivale a 3..3).

Por exemplo, na Figura 5.6, cada objeto empresa tem como funcionário um ou mais objetos pessoa (multiplicidade 1..\*); cada objeto pessoa tem um empregador zero ou mais objetos empresa (multiplicidade \*, que equivale a 0..\*).

- A instância de uma associação é chamada de vínculo, conforme explica o Capítulo 16.



**Figura 5.6:**  
Multiplicidade

**Agregação** Uma pura associação entre duas classes representa um relacionamento estrutural entre pares, significando que essas duas classes estão conceitualmente em um mesmo nível, sem que uma seja mais importante do que a outra. Em alguns casos, você desejará fazer a modelagem de um relacionamento “todo/parte”, no qual uma classe representa um item maior (o “todo”), formado por itens menores (as “partes”). Esse tipo de relacionamento é chamado de agregação e representa um relacionamento do tipo “tem-um”, o que significa que um objeto do todo contém os objetos das partes. A agregação, na verdade, é apenas um tipo especial de associação, especificada utilizando uma associação simples com um diamante aberto na extremidade do todo, conforme mostra a [Figura 5.7](#).

- A agregação admite muitas variações importantes, conforme explica o [Capítulo 10](#).



**Figura 5.7:**  
Agregação

**Nota:** O significado dessa forma simples de agregação é inteiramente conceitual. O diamante aberto diferencia o “todo” da “parte” e nada mais. Isso significa que essa agregação simples não altera o significado da navegação pela associação entre o todo e suas partes, nem está vinculada ao período de vida do todo e de suas partes. Consulte a seção sobre composição no [Capítulo 10](#) para obter uma forma restrita de agregação.

## OUTRAS CARACTERÍSTICAS

Dependências, generalizações e associações, simples e sem adornos, com nomes, multiplicidades e papéis são os recursos utilizados com maior frequência para a criação de abstrações. De fato, na maioria dos modelos que você criar, a forma básica desses três relacionamentos será suficiente para definir a semântica mais importante de seus relacionamentos. De vez em quando, porém, será necessário visualizar ou especificar outras características, como agregação composta, navegação, discriminações, classes de associação e tipos especiais de dependências e de generalizações. Essas e muitas outras características podem ser expressas na UML, mas são tratadas como conceitos avançados.

- Os conceitos avançados de relacionamentos são examinados no [Capítulo 10](#).

Dependências, generalizações e associações são itens estáticos, definidos no nível das classes. Na UML, esses relacionamentos costumam ser visualizados em diagramas de classes.

- Os diagramas de classes são apresentados no [Capítulo 8](#).

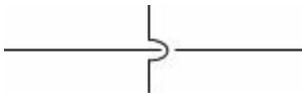
Quando a modelagem é iniciada no nível dos objetos e, especialmente, quando você começa a trabalhar com colaborações dinâmicas desses objetos, serão encontrados vínculos, que são instâncias de associações representando conexões entre objetos por intermédio dos quais as mensagens são enviadas.

- Os vínculos são examinados no [Capítulo 16](#).

## ESTILOS DE DESENHO

Os relacionamentos são apresentados em diagramas por linhas de um ícone a outro. As linhas têm vários adornos, como setas ou diamantes, para distinguir diferentes tipos de relacionamentos. Normalmente, os modeladores escolhem um dos dois estilos para desenhar linhas:

- » Linhas oblíquas em qualquer ângulo. Use um segmento de linha, a menos que vários segmentos sejam necessários para evitar outros ícones.
- » Linhas retilíneas desenhadas paralelamente nos lados da página. A menos que uma linha conecte dois ícones alinhados, a linha deve ser desenhada como uma série de segmentos de linha conectados pelos ângulos direitos. Esse é o estilo mais usado neste livro. Com cuidado, a maioria dos cruzamentos de linhas pode ser evitada. Se um cruzamento de linha for necessário e houver ambiguidade sobre como os caminhos são conectados, um pequeno arco pode ser usado para indicar um cruzamento de linha.



**Figura 5.8:**

Símbolo de cruzamento de linha

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE DEPENDÊNCIAS SIMPLES

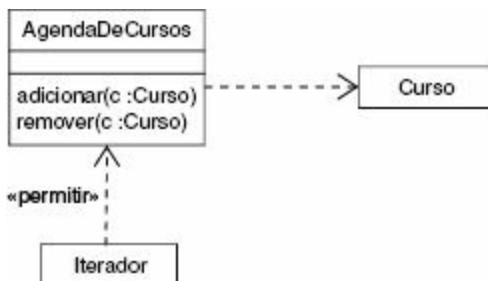
O tipo mais comum de relacionamento de dependência é a conexão entre uma classe que somente utiliza outra classe como parâmetro para determinada operação.

Para fazer a modelagem usando este relacionamento:

- » Crie uma dependência apontando, a partir da classe que executa a operação, a classe utilizada como parâmetro nessa operação.

Por exemplo, a [Figura 5.9](#) mostra um conjunto de classes definidas em um sistema responsável pela distribuição de alunos e professores em cursos de uma universidade. A figura mostra a dependência de `AgendaDeCursos` em relação a `Curso`, pois `Curso` é empregado nas operações adicionar e remover de `AgendaDeCursos`.

Fornecendo a assinatura completa da operação como ocorre nessa figura, normalmente você não necessita mostrar a dependência, pois o uso da classe já está explícito na assinatura. Entretanto, em outras situações, você desejará mostrar essa dependência, especialmente se você ocultar assinaturas de operações ou o modelo apresentar outros relacionamentos para as classes utilizadas.



**Figura 5.9:**  
Relacionamentos de dependência

Essa figura mostra uma outra dependência, que não envolve classes em operações, mas a modelagem de algo comum na linguagem C++. A dependência de Iterador indica que Iterador usa AgendaDeCursos, que, por sua vez, nada sabe sobre Iterador. A dependência é marcada com um estereótipo “permitir”, que é similar a uma declaração friend no C++.

► Outros estereótipos de relacionamentos são examinados no [Capítulo 10](#).

**Modelagem de herança simples** Ao fazer a modelagem do vocabulário de seu sistema, você sempre encontrará classes semelhantes estrutural ou comportamentalmente. Cada uma dessas classes poderá ser modelada como abstrações distintas e não relacionadas. Uma solução melhor seria extrair quaisquer características estruturais e comportamentais que sejam comuns e colocá-las em classes mais gerais a partir das quais serão herdadas por classes mais especializadas. Para fazer a modelagem de relacionamentos de herança:

- › Considerando um conjunto de classes, procure responsabilidades, atributos e operações que sejam comuns a duas ou mais classes.
- › Eleve essas responsabilidades, atributos e operações comuns para uma classe mais geral. Se necessário, crie uma nova classe à qual esses elementos possam ser atribuídos (mas tenha cuidado para não incluir uma quantidade excessiva de níveis).
- › Especifique que as classes mais específicas herdarão da classe mais geral, determinando um relacionamento de generalização, definido a partir de cada classe especializada para a classe-mãe mais geral.

Por exemplo, a [Figura 5.10](#) mostra um conjunto de classes estabelecido em uma aplicação comercial. Existe um relacionamento de generalização a partir de quatro classes – EmEspécie, Ações, Obrigações e Bens – para a classe mais geral Título. Título é a classe-mãe e EmEspécie, Ações, Obrigações e Bens são classes-filha. Cada uma dessas classes-filha especializada é um tipo de Título. Observe que Título inclui duas operações: valorPresente e histórico. Como Título é a classe-mãe, EmEspécie, Ações, Obrigações e Bens herdam essas duas operações e também quaisquer outros atributos e operações de Título que não são mostrados na figura.

Observe que os nomes Título e valorPresente são escritos de maneira um pouco diferente dos demais. Existe uma razão para isso. Ao construir hierarquias como aquela apresentada na figura anterior, você muitas vezes encontrará classes que não são classes-folha, que estão incompletas ou simplesmente você não deseja que contenham objetos. Essas classes são chamadas *abstratas*. Você pode especificar uma classe como abstrata na UML, escrevendo seu nome em itálico, como ocorre com a classe Título. Essa convenção se aplica a operações como valorPresente e significa que determinada operação fornece uma assinatura, mas por outro lado está incompleta e, portanto, deve ser implementada por algum método em um nível mais baixo de abstração. De fato, conforme mostra a figura, todas as quatro classes-filha

immediatas de Título são concretas (o que significa que não são abstratas) e também fornecem uma implementação concreta da operação valorPresente.

- Operações e classes abstratas são examinadas no [Capítulo 9](#).

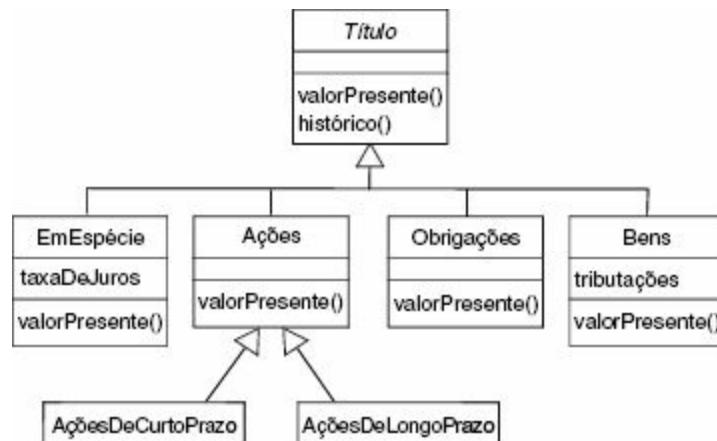


Figura 5.10:

Relacionamentos de herança

Suas hierarquias de generalização/especialização não precisam ficar limitadas a apenas dois níveis. De fato, conforme mostra a figura, é comum haver mais de dois níveis de herança. AçõesDeCurtoPrazo e AçõesDeLongoPrazo são filhas de Ações, que, por sua vez, é filha de Título. Assim, Título é uma classe-base, pois não tem uma classe-mãe. AçõesDeCurtoPrazo e AçõesDeLongoPrazo são classes-folha, porque não têm classes-filha. Ações tem uma classe-mãe assim como classes-filha e, portanto, não é uma classe-raiz, nem uma classe-folha.

Apesar de não ser mostrado aqui, você também pode criar classes com mais de uma classe-mãe. Isso é chamado herança múltipla e significa que determinada classe tem todos os atributos, operações e associações de todas as suas classes-mãe.

- A herança múltipla é examinada no [Capítulo 10](#).

Certamente, poderá não haver ciclos em uma rede de heranças; uma determinada classe não poderá ser sua própria classe-mãe.

## MODELAGEM DE RELACIONAMENTOS ESTRUTURAIS

Ao fazer a modelagem com dependências ou relacionamentos de generalizações, você está modelando classes que representam níveis diferentes de importância ou níveis diferentes de abstração. Considerando uma dependência entre duas classes, uma classe depende da outra, mas a outra classe não tem nenhum conhecimento sobre aquela que é dependente. Considerando-se um relacionamento de generalização entre duas classes, a classe-filha herda de sua classe-mãe, mas a classe-mãe não terá qualquer conhecimento específico sobre as classes-filha. Resumindo, a dependência e os relacionamentos de generalização são unilaterais. Ao fazer a modelagem com relacionamentos de associação, você está modelando classes que são pares umas das outras. Considerando uma associação entre duas classes, uma depende da outra de alguma forma e você pode navegar nas duas direções. Sempre que a dependência é um relacionamento de utilização e a generalização é um relacionamento "é um tipo de", uma associação especifica um caminho estrutural por meio do qual os objetos das classes interagem.

- » *Como padrão, as associações são bidirecionais; você pode limitar as direções, conforme é apresentado no [Capítulo 10](#).*

Para fazer a modelagem de relacionamentos estruturais:

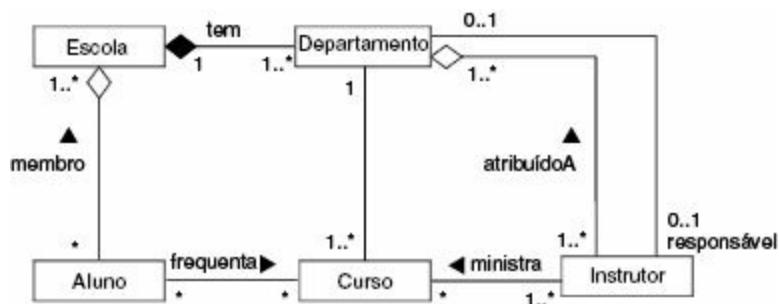
- » Para cada par de classes, se for necessário navegar de um objeto para outro, especifique uma associação entre elas. Isso constitui uma visão de associações orientadas-a-dados.
- » Para cada par de classes, se os objetos de uma classe precisarem interagir com os objetos de outras classes como parâmetros de uma operação, especifique uma associação entre as duas classes. Isso é mais uma visão de associações orientadas-a-comportamento.
- » Para cada uma dessas associações, especifique uma multiplicidade (principalmente quando a multiplicidade não for\*, que é o padrão), além de nomes de papéis (principalmente se ajudar a explicar o modelo).

- Se uma das classes de uma associação for, estrutural ou organizacionalmente, um todo quando comparada às classes encontradas na outra extremidade, que, por sua vez, parecem partes, marque esse caso como uma agregação, incluindo um adorno na associação que se encontra na extremidade junto ao todo com um diamante.

Como você poderá saber quando os objetos de uma determinada classe devem interagir com os objetos de outra classe? A resposta é que os cartões CRC e a análise de caso de uso serão de grande ajuda, obrigando você a considerar cenários estruturais e comportamentais. Sempre que descobrir que duas ou mais classes interagem, especifique uma associação.

► Os casos de uso são examinados no [Capítulo 17](#).

A [Figura 5.11](#) mostra um conjunto de classes estabelecidas em um sistema de informações destinado a uma escola. Iniciando na parte inferior esquerda do diagrama, encontram-se as classes Aluno, Curso e Instrutor. Existe uma associação entre Aluno e Curso, especificando que os alunos frequentam cursos. Além disso, cada aluno poderá frequentar vários cursos e cada curso poderá ter vários alunos. De maneira semelhante, existe uma associação entre Curso e Instrutor, especificando que os instrutores ministram os cursos. Para cada curso, há pelo menos um instrutor e todo instrutor poderá ministrar zero ou mais cursos. Cada curso pertence a exatamente um departamento.



**Figura 5.11:**

Relacionamentos estruturais

Os relacionamentos entre Escola e as classes Aluno e Departamento são um pouco diferentes. Nesse caso, encontram-se relacionamentos de agregação. Uma escola tem zero ou mais alunos, cada aluno poderá estar registrado em uma ou mais escolas, uma escola tem um ou mais departamentos, cada departamento pertence exatamente a uma única escola. Você poderia não incluir os adornos de agregação e usar associações puras, mas, ao especificar que Escola é um todo e que Aluno e Departamento são algumas de suas partes, fica claro qual deles é organizacionalmente superior ao outro. Assim, as escolas são definidas, de alguma forma, pelos seus alunos e departamentos. De modo semelhante, os alunos e os departamentos não existem fora da escola a que pertencem. Mais propriamente, parte de sua identidade é obtida a partir de sua escola.

- *O relacionamento de agregação entre Escola e Departamento é uma agregação composta, conforme é apresentado no [Capítulo 10](#).*

Observe que existem duas associações entre Departamento e Instrutor. Uma dessas associações especifica que todo instrutor é atribuído a um ou mais departamentos e cada departamento tem um ou mais instrutores. Esse caso é modelado como uma agregação, pois, em termos organizacionais, os departamentos se encontram em um nível mais alto da estrutura da escola do que os instrutores. A outra associação especifica que, para cada departamento, existe somente um instrutor como responsável pelo departamento. Da forma como esse modelo está especificado, um instrutor pode ser responsável por apenas um departamento e alguns instrutores não são responsáveis por nenhum departamento.

**Nota:** Podem ocorrer exceções nesse modelo, por não refletir a sua realidade. Sua escola poderá não ter departamentos. Poderá haver responsáveis que não sejam instrutores ou até poderá haver alunos que sejam instrutores. Isso não significa que o modelo aqui apresentado esteja errado, mas é apenas diferente. A modelagem não pode ser realizada isoladamente e todos os modelos como esse sempre dependem de como você pretende utilizá-los.

## DICAS E SUGESTÕES

Ao fazer a modelagem de relacionamentos em UML:

- › Use dependências somente quando o relacionamento que estiver sendo modelado não for estrutural.
- › Use a generalização somente quando houver um relacionamento “é um tipo de”; as heranças múltiplas quase sempre podem ser substituídas por agregação.
- › Tenha cuidado com a inclusão de relacionamentos de generalização cílicos.
- › Mantenha os relacionamentos de generalização geralmente equilibrados; as redes de herança não deverão ser muito profundas (questione a existência de cinco ou mais níveis), nem muito extensas (ao contrário, busque a possibilidade de classes abstratas intermediárias).
- › Use as associações principalmente no caso de relacionamentos estruturais entre objetos. Não as use para mostrar relacionamentos transitórios, como parâmetros ou variáveis locais de procedimentos.

Ao definir relacionamentos na UML:

- › Use linhas retilíneas ou oblíquas de maneira consistente. As linhas retilíneas fornecem uma orientação visual que dá ênfase às conexões existentes entre itens relacionados, todas apontando na direção de um item comum. Em diagramas complexos, as linhas oblíquas costumam ser mais eficientes em termos de espaço. A utilização dos dois tipos de linhas em um mesmo diagrama é útil para dar destaque a grupos diferentes de relacionamentos.
- › Evite linhas que se cruzam.
- › Mostre apenas aqueles relacionamentos necessários à compreensão de determinado agrupamento de itens. Relacionamentos supérfluos (principalmente as associações redundantes) deverão ser omitidos.

CAPÍTULO

---

6

---

# Mecanismos

## Neste capítulo

- » *Notas*
- » *Estereótipos, valores atribuídos e restrições*
- » *A modelagem de comentários*
- » *A modelagem de novos blocos de construção*
- » *A modelagem de novas propriedades*
- » *A modelagem de novas semânticas*
- » *Estendendo a UML*

**A**UML se torna mais simples devido à presença de quatro mecanismos básicos que são aplicados de maneira consistente na linguagem: especificações, adornos, divisões comuns e mecanismos de extensibilidade. Este capítulo explica o uso de dois desses mecanismos básicos: adornos e mecanismos de extensibilidade.

► *Esse mecanismo básico é examinado no [Capítulo 2](#).*

As notas são o tipo mais importante de adorno que pode aparecer sozinho. Uma nota é um símbolo gráfico para a representação de restrições ou de comentários anexados a um elemento ou a uma coleção de elementos. Use as notas para anexar informações a um modelo, como requisitos, observações, revisões e explicações. Os mecanismos de extensibilidade da UML permitem estender a linguagem de uma maneira controlada. Esses mecanismos incluem estereótipos, valores atribuídos e restrições. Os estereótipos ampliam o vocabulário da UML, permitindo a criação de novos tipos de blocos de

construção, derivados de outros já existentes, mas específicos a determinado problema. Um valor atribuído estende as propriedades de blocos de construção da UML, permitindo a criação de novas informações nas especificações desses elementos. Uma restrição estende a semântica dos blocos de construção da UML, permitindo adicionar novas regras ou modificar as existentes. Use esses mecanismos para ajustar a UML às necessidades específicas do seu domínio e cultura de desenvolvimento.

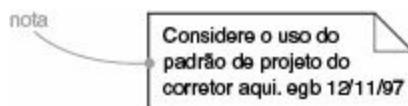
## PRIMEIROS PASSOS

Algumas vezes, será necessário usar a criatividade para incluir informações adicionais. Por exemplo, no local da obra, um arquiteto poderá rabiscar algumas notas nas plantas baixas do projeto de um prédio, com a finalidade de comunicar um pequeno detalhe aos operários da construção. Em um estúdio de gravações, um compositor poderá inventar uma nova notação musical para representar algum efeito pouco usual que deseja obter do músico que toca os teclados. Nos dois casos, já existem linguagens bem definidas – a linguagem dos planos estruturais de um projeto e a linguagem da notação musical –, mas, às vezes, é necessário adaptar ou estender essas linguagens, de uma maneira controlada, para comunicar suas intenções.

A modelagem está inteiramente relacionada com a comunicação. A UML já oferece todas as ferramentas necessárias para você visualizar, especificar, construir e documentar os artefatos de uma grande variedade de sistemas complexos de software. Porém, em determinadas circunstâncias, você poderá querer adaptar ou estender a UML. Isso ocorre o tempo todo com as linguagens humanas (por isso, novos dicionários são publicados todos os anos), pois nenhuma linguagem estática pode ser suficiente para abranger tudo o que se deseja comunicar o tempo todo. Ao usar uma linguagem de modelagem como a UML, lembre-se de que você o está fazendo para se comunicar e isso significa que desejará manter a linguagem básica, a menos que exista alguma importante razão em contrário. Ao inserir informações adicionais, você deverá fazê-lo de uma maneira controlada. Caso contrário, as

pessoas não conseguirão compreender o que você fez. As notas são o mecanismo fornecido pela UML com a finalidade de permitir captar restrições e comentários arbitrários que auxiliem a esclarecer os modelos que você esteja criando.

As notas podem representar artefatos que desempenham papéis importantes no ciclo de vida de desenvolvimento do software, como requisitos, ou simplesmente representam observações de forma livre, revisões ou explicações. A UML oferece uma representação gráfica para os comentários e as restrições, chamada nota, conforme mostra a [Figura 6.1](#). Essa notação permite que você visualize um comentário diretamente. Em conjunto com as ferramentas adequadas, as notas também oferecem um marcador de lugar para a vinculação ou a incorporação de outros documentos.



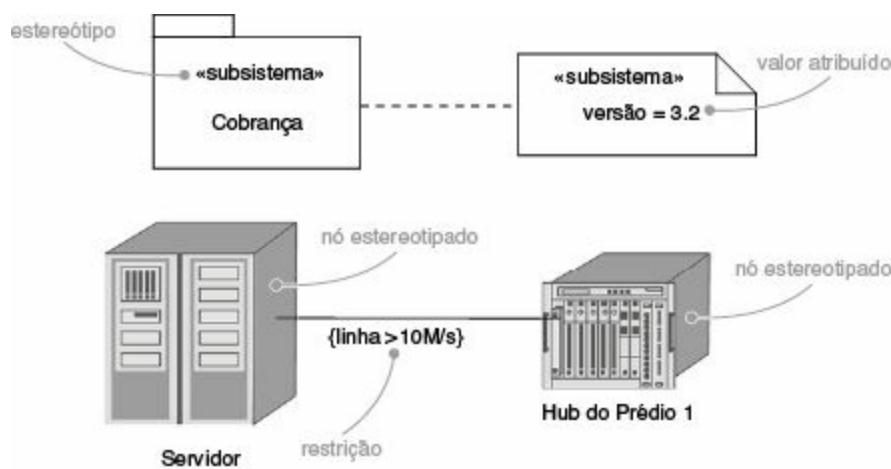
**Figura 6.1:**

Notas

Estereótipos, valores atribuídos e restrições são os mecanismos fornecidos pela UML para permitir adicionar novos blocos de construção, criar novas propriedades e especificar uma nova semântica. Por exemplo, ao fazer a modelagem de uma rede, você poderá querer símbolos para roteadores e hubs; experimente utilizar nós estereotipados para fazer com que esses itens apareçam como blocos de construção primitivos. De maneira semelhante, se você fizer parte da equipe de implantação do projeto, responsável por montar, testar e depois implantar as versões, talvez queira acompanhar o número de versões e resultados de testes para cada um dos principais subsistemas. Você pode usar valores atribuídos e adicionar essas informações aos seus modelos. Por fim, caso esteja fazendo a modelagem de sistemas em tempo real, talvez queira incluir adornos em seus modelos com as informações sobre custos e

datas-limite; você pode usar as restrições para captar os requisitos referentes a tempo.

A UML oferece uma representação textual para estereótipos, valores atribuídos e restrições, conforme mostra a [Figura 6.2](#). Os estereótipos também permitem incluir novos símbolos gráficos, de modo que você seja capaz de fornecer indicações visuais em seus modelos, falando a linguagem própria do seu domínio e cultura de desenvolvimento.



**Figura 6.2:**

Estereótipos, valores atribuídos e restrições

## TERMOS E CONCEITOS

Uma *nota* é um símbolo gráfico para a representação de restrições ou de comentários anexados a um elemento ou a uma coleção de elementos. Uma nota é representada graficamente como um retângulo com uma dobra em um dos cantos, incluindo um comentário gráfico ou visual.

Um *estereótipo* é uma extensão do vocabulário da UML, permitindo a criação de novos tipos de blocos de construção semelhantes aos existentes, mas específicos a determinado problema. Um estereótipo é representado graficamente como um nome entre ângulos (« »)<sup>2</sup>, colocado acima do nome de outro elemento. Como uma opção, o elemento estereotipado pode ser representado com a utilização de um novo ícone associado a esse estereótipo.

Um *valor atribuído* é uma propriedade de um estereótipo, permitindo a criação de novas informações na especificação desse estereótipo. Um valor atribuído é representado graficamente como uma sequência de caracteres na forma nome = valor em uma nota anexada ao objeto.

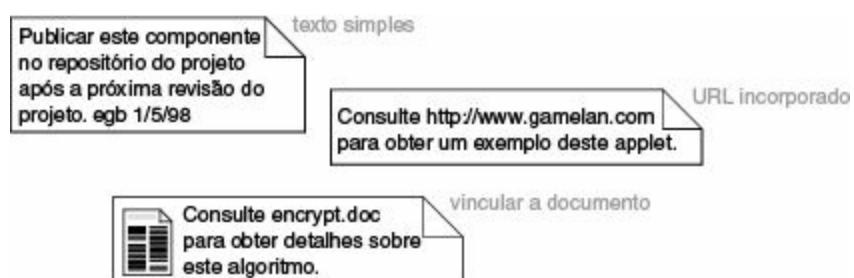
Uma *restrição* é uma especificação textual da semântica de um elemento da UML, permitindo adicionar novas regras ou modificar as existentes. Uma restrição é representada graficamente como uma sequência de caracteres entre chaves, colocada próxima ao elemento associado ou conectada a esse elemento ou elementos por meio de relacionamentos de dependência. Como uma alternativa, você pode representar uma restrição como uma nota.

## NOTAS

Uma nota que representa um comentário não tem qualquer impacto semântico, ou seja, seu conteúdo não altera o significado do modelo ao qual está anexada. Por isso as notas são empregadas para especificar itens como requisitos, observações, revisões e explanações, além da representação de restrições.

Uma nota pode conter qualquer combinação de texto e imagem gráfica. Caso a sua implementação permita, você poderá incluir um endereço URL em uma nota ou até vincular ou incorporar outro documento. Desse modo, é possível usar a UML para organizar todos os artefatos que você poderá gerar ou utilizar durante o desenvolvimento, conforme mostra a [Figura 6.3](#).

- As notas podem ser anexadas a mais de um elemento a partir da utilização de dependências, conforme explica o Capítulo 5.



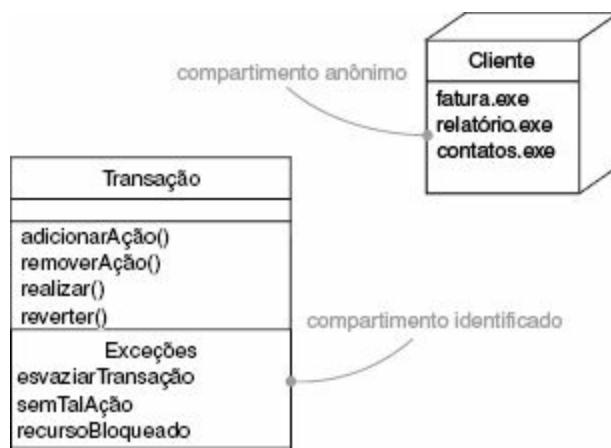
**Figura 6.3:**

Notas

**Outros adornos** Os adornos são itens gráficos ou visuais, adicionados à notação básica de um elemento e empregados para a visualização de detalhes a partir da especificação do elemento. Por exemplo, a notação básica para uma associação é uma linha, mas podem ser incluídos adornos referentes a detalhes como o papel e a multiplicidade de cada extremidade. Durante a utilização da UML, a regra geral a ser respeitada é a seguinte: inicie com a notação básica para cada elemento e depois adicione outros adornos somente se forem necessários para fornecer informações específicas que sejam importantes para o modelo.

- A notação básica para uma associação, além de seus adornos, é apresentada nos Capítulos [5](#) e [10](#).

Muitos adornos são representados colocando um texto próximo ao elemento de interesse ou adicionando um símbolo gráfico à notação básica. Porém, algumas vezes será conveniente incluir adornos a um elemento com mais detalhes do que costuma ser possível acomodar em textos simples ou em imagens gráficas. No caso de itens como classes, componentes e nós, você pode adicionar um compartimento extra, abaixo dos compartimentos usuais, para fornecer essas informações, conforme mostra a [Figura 6.4](#).



**Figura 6.4:**

Compartimentos extras

**Nota:** A menos que seja óbvio devido ao respectivo conteúdo, é de boa prática dar nomes explicitamente a qualquer compartimento extra, evitando qualquer confusão em relação ao seu significado. Também é de boa prática usar compartimentos extras com cautela, pois, caso sejam utilizados em excesso, acabam congestionando os diagramas.

## ESTEREÓTIPOS

A UML fornece uma linguagem para itens estruturais, itens comportamentais, agrupamentos de itens e itens notacionais. Esses quatro tipos de itens abrangem a maioria dos sistemas cuja modelagem você precisará fazer. Entretanto, às vezes você desejará incluir itens novos, capazes de falar o vocabulário próprio do seu domínio e com a aparência de blocos de construção primitivos.

- *Esse quatro elementos básicos da UML são apresentados no [Capítulo 2](#).*

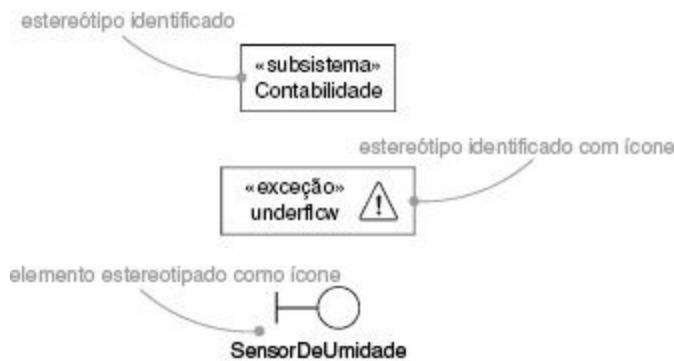
Um estereótipo não é o mesmo que uma classe-mãe em um relacionamento de generalização mãe-filha. Em vez disso, considere o estereótipo como um metatipo, pois cada um cria o equivalente a uma nova classe no metamodelo da UML. Por exemplo, ao fazer a modelagem de um processo de negócio, você incluirá itens como funcionários, documentos e estratégias. De forma similar, caso você siga um processo de desenvolvimento como o Rational Unified Process (processo unificado de engenharia de software criado pela Rational Software Corporation, hoje uma empresa da IBM), você desejará modelar usando limite (ou fronteira), controle e entidades de classes. Nesse caso, revela-se o verdadeiro valor dos estereótipos. Ao aplicar um estereótipo a um elemento, como um nó ou uma classe, você está estendendo a UML com a criação de um novo bloco de construção semelhante a algum já existente, mas com suas próprias propriedades especiais (cada estereótipo pode fornecer seu próprio conjunto de valores atribuídos), sua semântica (cada estereótipo pode determinar suas

próprias restrições) e notação (cada estereótipo poderá fornecer seu próprio ícone).

■ *O Rational Unified Process é resumido no Apêndice B.*

Em sua forma mais simples, um estereótipo é representado como um nome entre ângulos (por exemplo, <<nome>>), colocado acima do nome de outro elemento. Como uma indicação visual, você pode definir um ícone para o estereótipo e apresentá-lo à direita do nome (caso esteja utilizando a notação básica para o elemento) ou usar esse ícone como símbolo básico para o item estereotipado. Todos esses três procedimentos são ilustrados na [Figura 6.5](#).

**Nota:** Ao definir ícones para os estereótipos, experimente usar cores como um destaque para compor uma indicação visual sutil (mas use as cores de maneira criteriosa). A UML permite a utilização de qualquer forma para esses ícones e, caso sua implementação permita, os ícones poderão aparecer como ferramentas primitivas, de modo que os usuários responsáveis pela criação de diagramas da UML terão uma paleta de itens que consideram básicos e capazes de expressar o vocabulário de seu domínio.



**Figura 6.5:**  
Estereótipos

## VALORES ATRIBUÍDOS

Tudo na UML tem seu próprio conjunto de propriedades: as classes têm nomes, atributos e operações; as associações têm nomes e duas ou mais

extremidades (cada uma com suas próprias propriedades); e assim por diante. Utilizando os estereótipos, você pode adicionar itens novos à UML; usando valores atribuídos, pode acrescentar novas propriedades.

Você pode definir etiquetas para os elementos existentes da UML ou pode definir etiquetas que se aplicam a estereótipos individuais de maneira que tudo em relação a esse estereótipo tenha esse valor atribuído. Um valor atribuído não é o mesmo que os atributos de uma classe. Mais propriamente, considere os valores atribuídos como metadados, pois seus valores se aplicam aos próprios elementos e não às respectivas instâncias. Por exemplo, conforme mostra a [Figura 6.6](#), você poderá especificar a capacidade requerida de uma classe de servidor ou requerer que somente um tipo de servidor seja usado em um determinado sistema.

► Os atributos são examinados nos [Capítulos 4 e 9](#).



**Figura 6.6:**

Definições de estereótipos e etiquetas

Os valores atribuídos são colocados em uma nota anexada ao elemento afetado, como mostra a [Figura 6.7](#). Cada valor atribuído é representado como uma sequência de caracteres que inclui um nome (a etiqueta), um separador (o símbolo =) e um valor (atribuído). Você pode especificar apenas o valor, caso seu significado não seja ambíguo, como ocorre quando o valor é o nome de uma enumeração.

**Nota:** Uma das utilizações mais comuns dos valores atribuídos é a especificação de propriedades relevantes para geração de código ou de gerenciamento de configurações. Por exemplo, você pode usar os valores atribuídos para especificar a linguagem de programação para a qual é mapeada uma determinada classe. De modo semelhante, pode-se utilizar os valores atribuídos para especificar o autor e a versão de um componente.

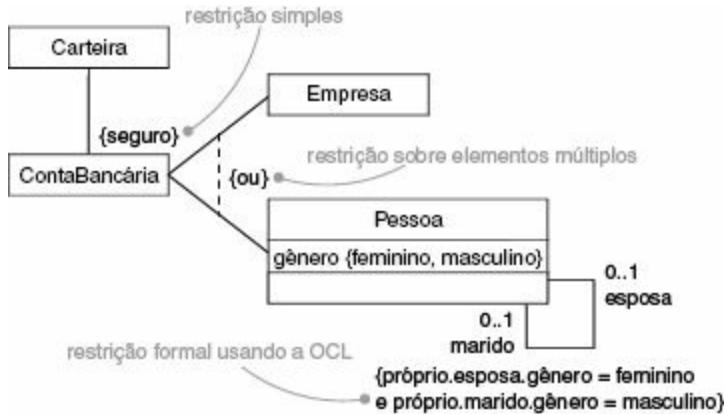


**Figura 6.7:**  
Valores atribuídos

## RESTRIÇÕES

Tudo na UML tem sua própria semântica. A generalização implica o princípio de substituição de Liskov<sup>3</sup>, e a existência de associações múltiplas conectadas a uma mesma classe denota relacionamentos distintos. Usando restrições, você pode adicionar uma nova semântica ou modificar as regras existentes. Uma restrição especifica condições que devem ser mantidas como verdadeiras para que o modelo seja bem formado. Por exemplo, conforme mostra a Figura 6.8, você poderá especificar que, em uma determinada associação, a comunicação é segura; uma configuração que viole essa restrição é inconsistente com o modelo. De maneira semelhante, você poderá especificar que, em um conjunto de associações conectadas a uma certa classe, uma determinada instância poderá ter vínculos somente com uma associação no conjunto.

- Restrições de tempo e espaço, muito utilizadas para a modelagem de sistemas em tempo real, são examinadas no [Capítulo 24](#).



**Figura 6.8:**

Restrição

**Nota:** As restrições podem ser escritas como texto em forma livre. Se preferir especificar sua semântica de maneira mais precisa, você pode usar a OCL (Object Constraint Language) da UML, descrita em detalhes no Unified Modeling Language Reference Manual.

Uma restrição é representada como uma sequência de caracteres entre chaves, colocada próxima ao elemento associado. Essa notação também é utilizada como um adorno para a notação básica de um elemento, com a finalidade de permitir a visualização de partes da especificação de um elemento que não tenha qualquer indicação visual. Por exemplo, algumas propriedades das associações (ordem e mutabilidade) são representadas utilizando uma notação de restrição.

- As restrições podem ser anexadas a mais de um elemento pela utilização de dependências, conforme é explicado no [Capítulo 5](#).

## ELEMENTOS-PADRÃO

A UML define um conjunto de estereótipos-padrão para classificadores, componentes, relacionamentos e outros elementos da modelagem. Existe um estereótipo-padrão, que interessa principalmente aos construtores de ferramentas, capaz de permitir a modelagem dos próprios estereótipos.

- Os classificadores são discutidos no [Capítulo 9](#).

### » estereótipo

Especifica que o classificador é um estereótipo que pode ser aplicado a outros elementos

Você usará esse estereótipo quando quiser modelar explicitamente os estereótipos que definiu em seu projeto.

## PERFIS

Frequentemente, vale a pena definir uma versão de UML adequada a uma determinada finalidade ou área de domínio. Por exemplo, se você deseja usar um modelo de UML para geração de código em uma determinada linguagem, é útil definir estereótipos que possam ser aplicados a elementos para fornecer dicas ao gerador de código (como um Ada pragma). Entretanto, os estereótipos definidos seriam diferentes para Java e C++. Como outro exemplo, você poderia usar a UML para modelar bancos de dados. Alguns recursos da UML, como modelagem dinâmica, são menos importantes, mas você deseja adicionar conceitos, como chaves candidatas e índices. É possível ajustar a UML usando perfis.

Um *perfil* é uma UML com um conjunto de estereótipos predefinidos, valores atribuídos, restrições e classes de base. Ele também seleciona um subconjunto dos tipos de elementos da UML para uso, de maneira que um modelador não fique confuso pelos tipos de elementos que não são necessários para a área de aplicação particular. Um perfil define uma versão especializada da UML para uma determinada área. Como o perfil é criado sobre elementos comuns da UML, ele não representa uma nova linguagem e pode ser suportado por ferramentas comuns da UML.

A maioria dos modeladores não construirá seus próprios perfis. A maioria dos perfis será construída por criadores de ferramentas, criadores de frameworks e projetistas similares de recursos gerais. Entretanto, muitos modeladores usarão perfis. São como as bibliotecas de sub-rotina tradicionais – alguns especialistas as criam, mas muitos programadores as usam. Esperamos que os perfis sejam criados para linguagens de programação e

bancos de dados, para diferentes plataformas de implementação, diferentes ferramentas de modelagem e várias áreas de aplicação de domínio comercial.

## **TÉCNICAS BÁSICAS DE MODELAGEM**

### **MODELAGEM DE COMENTÁRIOS**

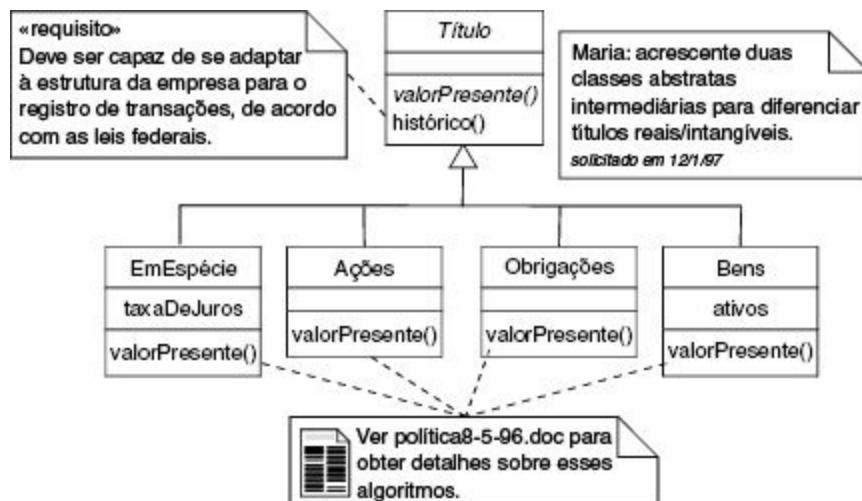
O propósito mais comum das notas será escrever, em forma livre, observações, revisões ou explanações. Incluindo esses comentários diretamente em seus modelos, estes podem se transformar em um repositório comum para os mais diversos artefatos a serem criados durante o desenvolvimento. Você até pode usar notas para visualizar requisitos e mostrar como estão relacionados explicitamente às partes do modelo. Para fazer a modelagem de comentários:

- › Inclua seu comentário como texto em uma nota e coloque-a próxima ao elemento a que ela se refere. Você pode mostrar um relacionamento mais explícito, conectando a nota aos seus elementos por meio de relacionamentos de dependência.
- › Lembre-se de que é possível ocultar ou tornar visíveis os elementos de seu modelo, conforme você desejar. Isso significa que não é necessário manter visíveis os componentes em todos os locais em que aparecem os elementos a que eles estão anexados. Em vez disso, exiba comentários em seus diagramas somente na medida em que sejam necessários para a comunicação de informações em um determinado contexto.
- › Se o comentário for muito extenso ou envolver algo mais rebuscado do que texto simples, considere a possibilidade de colocá-lo em um documento externo e depois vincular ou incorporar esse documento em uma nota anexada ao modelo.
- › À medida que o modelo evoluir, mantenha aqueles comentários que registram decisões significativas e que não possam ser inferidos a partir do próprio modelo e – a menos que tenham algum interesse histórico –

exclua os demais comentários.

Por exemplo, a [Figura 6.9](#) mostra o modelo de um trabalho em andamento de uma hierarquia de classes, exibindo alguns requisitos que dão forma ao modelo, assim como algumas notas provenientes de uma revisão de projeto.

- A *generalização simples* é apresentada no [Capítulo 5](#); formas avançadas de generalização são examinadas no [Capítulo 10](#).



**Figura 6.9:**

Modelagem de comentários

Nesse exemplo, a maior parte dos comentários é texto simples (como a nota para Maria), mas um deles (a nota encontrada na parte inferior do diagrama) oferece um hyperlink para outro documento.

## MODELAGEM DE NOVAS PROPRIEDADES

As propriedades básicas dos blocos de construção da UML – atributos e operações para classes, o conteúdo de pacotes e assim por diante – são suficientemente genéricas para abranger a maioria dos itens cuja modelagem você desejará fazer. Entretanto, se você quiser estender as propriedades desses blocos de construção básicos (ou de novos blocos criados com o uso de estereótipos), é necessário utilizar valores atribuídos. Para fazer a modelagem de novas propriedades:

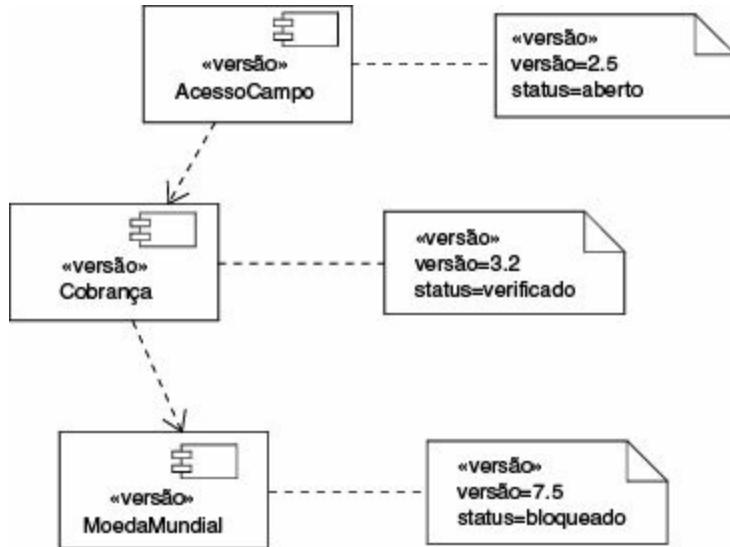
- › Em primeiro lugar, verifique se já não existe uma forma para expressar o que você deseja, utilizando apenas a notação básica UML.
- › Caso esteja convencido de que não existe outra maneira de expressar essa semântica, adicione a nova propriedade a um elemento individual ou a um estereótipo. As regras de generalização serão aplicadas – os valores atribuídos definidos para um tipo de elemento são aplicados aos respectivos elementos-filho.

Por exemplo, suponha que você queira manter a coesão entre os modelos criados para o sistema de gerenciamento da configuração de seu projeto. Entre outras coisas, isso significa manter um acompanhamento do número da versão, do status atual de verificação de entrada/saída e talvez até das datas de criação e de modificação de cada subsistema. Por serem informações específicas do processo, não são partes básicas da UML, apesar de ser possível adicioná-las como valores atribuídos. Além disso, essas informações nem são atributos de classes. O número da versão de um subsistema é parte de seus metadados e não parte do modelo.

► *Os subsistemas são examinados no [Capítulo 32](#).*

A [Figura 6.10](#) apresenta três subsistemas, cada um dos quais foi estendido para incluir o respectivo status e número da versão.

**Nota:** Os valores de etiquetas como *versão* e *status* são itens que podem ser definidos pelas ferramentas. Em vez de estabelecer esses valores em seu modelo manualmente, você pode usar um ambiente de desenvolvimento que integre suas ferramentas de gerenciamento de configuração e suas ferramentas de modelagem com a finalidade de manter esses valores para você.



**Figura 6.10:**

A modelagem de novas propriedades

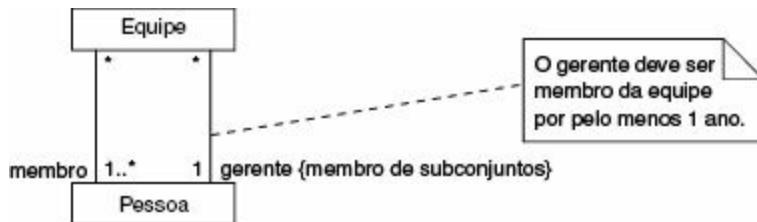
## MODELAGEM DE UMA NOVA SEMÂNTICA

Ao usar a UML para criar um modelo, você trabalha de acordo com as regras em que a UML se baseia. Isso é bom, pois significa que você é capaz de comunicar tudo que desejar sem ambiguidade a qualquer pessoa que saiba como ler a UML.

Porém, se for necessário expressar uma nova semântica não disponível na UML ou modificar as regras da UML, você precisa escrever uma restrição. Para modelar a nova semântica:

- › Em primeiro lugar, verifique se já não existe uma forma para expressar o que você deseja, utilizando apenas a notação UML básica.
- › Caso esteja convencido de que não existe outra maneira de expressar essa semântica, escreva a nova semântica como texto em uma restrição e coloque-a ao lado do elemento ao qual ela se refere. Para mostrar um relacionamento mais explícito, experimente usar um relacionamento de dependência para conectar uma restrição aos respectivos elementos.
- › Se for necessário especificar sua semântica de modo mais preciso e formal, escreva a nova semântica usando a OCL.

Por exemplo, a [Figura 6.11](#) exibe a modelagem de uma pequena parte de um sistema corporativo de recursos humanos.



**Figura 6.11:**

A modelagem de uma nova semântica

Esse diagrama mostra que cada Pessoa pode ser um membro de zero ou mais Equipes e cada Equipe deve ter pelo menos uma Pessoa como membro. O diagrama também indica que cada Equipe deve ter exatamente uma Pessoa como gerente e cada Pessoa pode ser o gerente de zero ou mais Equipes. Toda essa semântica pode ser expressa utilizando apenas a UML simples. Entretanto, a afirmação de que um gerente também precisa ser um membro do departamento envolve várias associações e não pode ser expressa pela UML simples. Para estabelecer esse aspecto invariante, é necessário escrever uma restrição que apresente o gerente como um subconjunto dos membros da Equipe, conectando as duas associações e a restrição. Há também a restrição que determina que o gerente deve ser membro há pelo menos um ano.

## DICAS E SUGESTÕES

Ao usar notas para adornar um modelo:

- » Use notas somente para aqueles requisitos, observações, revisões e explanações que não possam ser expressos de modo simples ou significativo usando as características existentes na UML.
- » Use notas como um tipo de lembrete eletrônico, com a finalidade de acompanhar o andamento de seu trabalho.

Ao estabelecer suas notas:

› Não congestione seus modelos com grandes blocos de comentários. Em vez disso, se realmente for necessário algum comentário extenso, use as notas como um marcador de posição para vincular ou incorporar um documento contendo o comentário completo.

Ao estender modelos com estereótipos, valores atribuídos ou restrições:

› Utilize uma padronização baseada em um pequeno conjunto de estereótipos, valores atribuídos e restrições a serem usados em seu projeto e evite que desenvolvedores individuais criem muitas extensões novas.

› Escolha nomes curtos e significativos para seus estereótipos e valores atribuídos.

› Sempre que a precisão não for necessária, use texto em forma livre para especificar restrições. Quando precisar de maior rigor, use a OCL para escrever as expressões das restrições.

Ao estabelecer estereótipos, valores atribuídos ou restrições:

› Use os estereótipos gráficos criteriosamente. Você pode modificar inteiramente a notação básica da UML com os estereótipos, mas ao fazê-lo será impossível que outras pessoas compreendam os seus modelos.

› Considere o uso de cores ou sombras simples para os estereótipos gráficos, assim como para os ícones mais complicados. As notações simples costumam ser melhores, e até as indicações visuais mais sutis podem ser mais eficientes para comunicar determinado significado.

---

<sup>2</sup> Esta notação de ângulos duplos (guillemets) é usada no idioma francês para citações, substituindo as aspas.

<sup>3</sup> Na programação orientada a objeto, o princípio da substituição de Liskov é uma definição particular para o conceito de subtipo. Foi introduzido em 1993 por Barbara Liskov e Jeannette Wing através de um artigo de nome “Family Values: A Behavioral Notion of Subtyping”. Fonte: Wikipedia.

CAPÍTULO

---

7

---

# Diagrams

## Neste capítulo

- » *Diagrams, visões e modelos*
- » *A modelagem das diferentes visões de um sistema*
- » *A modelagem dos diferentes níveis de abstração*
- » *A modelagem de visões complexas*
- » *A organização de diagramas e outros artefatos*

**A**o fazer uma modelagem, você cria uma simplificação da realidade para entender melhor o sistema em desenvolvimento. Usando a UML, você constrói seus modelos a partir de blocos de construção básicos, como classes, interfaces, colaborações, componentes, nós, dependências, generalizações e associações.

► *O conceito de modelagem é apresentado no Capítulo 1.*

Os diagramas são meios utilizados para a visualização desses blocos de construção. Um diagrama é uma apresentação gráfica de um conjunto de elementos, geralmente representados como um gráfico conectado de vértices (itens) e arcos (relacionamentos). Use os diagramas para visualizar o seu sistema sob diferentes perspectivas. Uma vez que nenhum sistema complexo pode ser compreendido em sua totalidade a partir de uma única perspectiva, a UML define um número de diagramas que permite dirigir o foco para aspectos diferentes de seu sistema de maneira independente.

Bons diagramas facilitam a compreensão do sistema que você está desenvolvendo. Escolhendo o conjunto correto de diagramas para a

modelagem de seu sistema, você precisa fazer as perguntas adequadas sobre o seu sistema e auxiliar a identificação das implicações de suas decisões.

## PRIMEIROS PASSOS

Ao trabalhar com um arquiteto para projetar uma casa, você inicia com três itens: uma lista do que deseja (como “eu quero uma casa com três quartos” e “não quero gastar mais do que x”), alguns esboços simples ou desenhos de outras casas representando algumas de suas principais características (como o desenho de uma entrada com uma escada circular) e algumas ideias gerais sobre o estilo (como “gostaria de algo francês com toques californianos”). A tarefa do arquiteto é coletar esses requisitos incompletos, sujeitos a alterações e possivelmente contraditórios, e transformá-los em um projeto.

Para cumprir essa tarefa, provavelmente o arquiteto começará com um esquema de projeto de planta baixa. Esse artefato fornece um meio para que você e ele visualizem a casa pronta, especifiquem detalhes e documentem suas decisões. Em cada revisão, você desejará fazer algumas modificações, como mudar paredes, reorganizar os cômodos, colocar janelas e portas. Desde o início, esses esquemas de projeto são alterados com frequência. À medida que o projeto for amadurecendo e você ficar satisfeito por ter um projeto capaz de atender da melhor maneira a todas as restrições de forma, função, duração e custos, esses esquemas de projeto se estabilizarão até o ponto de poderem ser utilizados para a construção da casa. Ainda quando a casa estiver sendo construída, provavelmente você modificará alguns desses diagramas e também criará outros.

Ao longo do tempo, você desejará visualizar perspectivas da casa diferentes daquela proporcionada pela planta baixa. Por exemplo, você desejará ver uma planta de elevação, mostrando os diferentes lados da casa. À medida que você começa a especificar detalhes para que a tarefa possa ser significativamente explicitada, seu arquiteto precisará criar plantas das instalações elétricas, plantas de aquecimento e ventilação e plantas de água e

esgoto. Se o seu projeto exigir características pouco usuais (como um longo vão acima do porão) ou se determinada característica for muito importante para você (como uma lareira junto à qual será colocado um telão), você e seu arquiteto criará esboços esclarecendo esses detalhes.

A prática de criar diagramas para a visualização de sistemas sob diferentes perspectivas não está limitada à indústria da construção. Isso sempre será encontrado na área de engenharia, envolvendo a criação de sistemas complexos, desde a engenharia civil até a engenharia aeronáutica, construção naval, indústria e software.

No contexto do software, existem cinco visões complementares que são as mais importantes para a visualização, a especificação, a construção e a documentação da arquitetura de um software: a visão do caso de uso, a visão do projeto, a perspectiva do processo, a visão da implementação e a visão da implantação. Cada uma dessas visões envolve uma modelagem estrutural (modelagem de itens estáticos), assim como uma modelagem comportamental (modelagem de itens dinâmicos). Em conjunto, essas visões diferentes captam as decisões mais importantes sobre o sistema. Individualmente, cada uma dessas visões permite voltar sua atenção para uma perspectiva do sistema e analisar suas decisões com clareza.

- As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#).

Ao visualizar um sistema de software sob qualquer perspectiva usando a UML, você utiliza diagramas para organizar os elementos relevantes. A UML define nove tipos de diagramas, que podem ser combinados para determinar cada visão. Por exemplo, a visão de implementação referente aos aspectos estáticos de um sistema pode ser visualizada com a utilização de diagramas de classes; os aspectos dinâmicos da mesma visão de implementação poderão ser visualizados com a utilização de diagramas de interação.

- A modelagem da arquitetura de um sistema é examinada no [Capítulo 32](#).

Certamente, você não está limitado aos tipos de diagramas predefinidos. Na UML, os tipos de diagramas são definidos porque representam o pacote mais comum dos elementos visualizados. Para atender às necessidades de seu projeto ou organização, você pode criar seus próprios tipos de diagramas com a finalidade de visualizar elementos da UML de diferentes maneiras.

Você usará os diagramas da UML de duas maneiras básicas: para especificar modelos a partir dos quais será construído um sistema executável (engenharia direta); e para reconstruir modelos a partir de partes de um sistema executável (engenharia reversa). Em qualquer uma dessas duas maneiras, assim como o arquiteto de um prédio, você tenderá a criar seus diagramas incrementalmente (ampliando-os uma parte de cada vez) e iterativamente (repetindo o processo de projetar uma pequena parte e construí-la).

► *O processo iterativo e incremental é resumido no Apêndice B.*

## TERMOS E CONCEITOS

Um *sistema* é uma coleção de subsistemas organizados para a realização de um objetivo e descritos por um conjunto de modelos, possivelmente sob diferentes pontos de vista. Um *subsistema* é um agrupamento de elementos, alguns dos quais constituem uma especificação do comportamento proporcionado pelos outros elementos contidos. Um *modelo* é uma abstração semanticamente fechada de um sistema, o que significa que representa uma simplificação autoconsistente e completa da realidade, criada com a finalidade de permitir uma melhor compreensão a respeito do sistema. No contexto da arquitetura, uma *visão* é uma projeção da organização e estrutura do modelo de um sistema, cujo foco está voltado para um único aspecto desse sistema. Um *diagrama* é a apresentação gráfica de um conjunto de elementos, geralmente representada como um gráfico conectado de vértices (itens) e arcos (relacionamentos).

► *Sistemas, modelos e visões são examinados no Capítulo 32.*

Dito de outra forma, um sistema representa um item que está sendo desenvolvido, visualizado sob diferentes perspectivas por modelos distintos, sendo essas visões apresentadas na forma de diagramas.

Um diagrama é apenas uma projeção gráfica dos elementos que formam o sistema. Por exemplo, poderá haver várias centenas de classes no projeto de um sistema de recursos humanos de uma empresa. Nunca será possível visualizar a estrutura ou o comportamento desse sistema por meio de um único grande diagrama contendo todas essas classes e todos os seus relacionamentos. Em vez disso, você desejará criar vários diagramas, cada um com o foco em uma única perspectiva. Por exemplo, poderá haver um diagrama de classes, incluindo classes como Pessoa, Departamento e Escritório, elaborado para construir o esquema de um banco de dados. Você poderá encontrar algumas dessas classes, juntamente com outras classes, em outro diagrama que apresenta uma API utilizada pelas aplicações-cliente. Provavelmente você verá algumas dessas classes mencionadas em um diagrama de interação, especificando a semântica de uma transação que atribui uma Pessoa a um novo Departamento.

Conforme mostra esse exemplo, um mesmo item em um sistema (como a classe Pessoa) poderá aparecer várias vezes no mesmo diagrama ou até em diagramas diferentes. Em cada caso, será o mesmo item. Cada diagrama oferece uma visão acerca dos elementos que formam o sistema.

Na modelagem de sistemas reais, independentemente do domínio do problema, você criará os mesmos tipos de diagramas, pois eles representam visões de modelos comuns. Tipicamente, você visualizará as partes estáticas de um sistema, utilizando um dos seguintes diagramas:

1. Diagrama de classes
2. Diagrama de componentes
3. Diagrama de estrutura composta
4. Diagrama de objetos
5. Diagrama de implantação

## 6. Diagrama de artefatos

Com frequência, você usará cinco diagramas adicionais para visualizar as partes dinâmicas de um sistema.

1. Diagrama de caso de uso
2. Diagrama de sequências
3. Diagrama de comunicação
4. Diagrama de estados
5. Diagrama de atividades

Todos os diagramas que você criar provavelmente corresponderão a um desses nove tipos ou ocasionalmente a um outro tipo, definido para seu projeto ou organização. Todos os diagramas devem ter nomes únicos em seus contextos, permitindo que você se refira a um diagrama específico e diferencie um do outro. Em todos os casos, com exceção dos sistemas mais triviais, você desejará organizar seus diagramas em pacotes.

► *Os pacotes são apresentados no [Capítulo 12](#).*

Você pode projetar qualquer combinação de elementos da UML em um mesmo diagrama. Por exemplo, poderá mostrar classes e objetos no mesmo diagrama ou poderá até mostrar classes e componentes no mesmo diagrama (válido, mas menos comum). Apesar de nada impedir que você inclua tipos distintos de elementos de modelagem no mesmo diagrama, é mais comum haver os mesmos tipos de itens em um diagrama. De fato, os diagramas definidos na UML recebem nomes de acordo com os elementos neles incluídos, e com o de maior frequência. Por exemplo, se quiser visualizar um conjunto de classes e seus relacionamentos, você usará um diagrama de classes. De forma semelhante, para visualizar um conjunto de componentes, usará um diagrama de componentes.

## DIAGRAMAS ESTRUTURAIS

Os diagramas estruturais da UML existem para visualizar, especificar, construir e documentar os aspectos estáticos de um sistema. Considere os

aspectos estáticos do sistema como uma representação de seu esqueleto e estrutura relativamente estáveis. Assim como os aspectos estáticos de uma casa incluem a existência e a colocação de itens como paredes, portas, janelas, canos, fios e respiradouros, também os aspectos estáticos de um sistema de software abrangem a existência e a colocação de itens como classes, interfaces, colaborações, componentes e nós.

Os diagramas estruturais da UML são organizados em função dos principais grupos de itens encontrados na modelagem de um sistema.

- |                                   |                                    |
|-----------------------------------|------------------------------------|
| 1. Diagrama de classes            | Classes, interfaces e colaborações |
| 2. Diagrama de componentes        | Componentes                        |
| 3. Diagrama de estrutura composta | Estrutura interna                  |
| 4. Diagrama de objetos            | Objetos                            |
| 5. Diagrama de artefatos          | Artefatos                          |
| 6. Diagrama de implantação        | Nós                                |

**Diagrama de classes** Um *diagrama de classes* mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos. Os diagramas de classes são os diagramas mais encontrados em sistemas de modelagem orientados a objetos. Use esses diagramas para ilustrar a visão estática do projeto de um sistema. Os diagramas de classes que incluem classes ativas são empregados para direcionar a visão estática do processo de um sistema.

► *Os diagramas de classes são examinados no Capítulo 8.*

**Diagrama de componentes** Um *diagrama de componentes* mostra as partes internas, os conectores e as portas que implementam um componente. Quando o componente é instanciado, as cópias de suas partes internas também são instanciadas.

**Diagrama de estrutura composta** Um *diagrama de estrutura composta* mostra a estrutura interna de uma classe ou colaboração. A diferença entre

componentes e estrutura composta é pequena, e este livro os trata igualmente como diagramas de componentes.

- Os diagramas de estrutura composta e de componentes são examinados no [Capítulo 15](#).

**Diagrama de objetos** Um *diagrama de objetos* mostra um conjunto de objetos e seus relacionamentos. Use esses diagramas para ilustrar as estruturas de dados, registros estáticos de instâncias dos itens encontrados nos diagramas de classes. Os diagramas de objetos direcionam a visão estática do projeto de um sistema ou a visão estática do processo de um sistema, tais quais os diagramas de classes, mas considerando casos reais ou prototípicos.

- Os diagramas de objetos são examinados no [Capítulo 14](#).

**Diagrama de artefatos** Um *diagrama de artefatos* mostra um conjunto de artefatos e seus relacionamentos com outros artefatos e com as classes que implementam. Use esses diagramas para mostrar as unidades de implementação física do sistema. (A UML considera os artefatos como parte dos diagramas de implantação, mas nós os separamos para facilitar a discussão.)

- Os diagramas de artefatos são examinados no [Capítulo 30](#).

**Diagrama de implantação** Um *diagrama de implantação* mostra um conjunto de nós e seus relacionamentos. Use esses diagramas para ilustrar a visão estática da implantação de uma arquitetura. Os diagramas de implantação estão relacionados aos diagramas de componentes, pois tipicamente um nó contém um ou mais componentes.

- Os diagramas de implantação são examinados no [Capítulo 31](#).

**Nota:** Existem algumas variações básicas desses diagramas, cujos nomes refletem seu conteúdo principal. Por exemplo, você poderá criar o diagrama de um subsistema para ilustrar a decomposição estrutural de um sistema em subsistemas.

*Um diagrama de subsistemas é apenas um diagrama de classes que contém principalmente os subsistemas.*

## DIAGRAMAS COMPORTAMENTAIS

Os cinco diagramas comportamentais da UML são utilizados para visualizar, especificar, construir e documentar os aspectos dinâmicos de um sistema. Considere os aspectos dinâmicos de um sistema como uma representação de suas partes que sofrem alterações. Assim como os aspectos dinâmicos de uma casa abrangem a circulação de ar e a passagem de pessoas pelos cômodos da casa, também os aspectos dinâmicos de um sistema de software envolvem itens como o fluxo de mensagens ao longo do tempo e a movimentação física de componentes em uma rede. Os diagramas comportamentais da UML são basicamente organizados a partir das principais maneiras disponíveis para fazer a modelagem da dinâmica de um sistema.

1. Diagrama de Organiza os comportamentos do sistema.  
caso de uso
2. Diagrama de Enfatiza a ordem temporal das mensagens.  
sequência
3. Diagrama de Enfatiza a organização estrutural de objetos que  
comunicação enviam e recebem mensagens.
4. Diagrama de Enfatiza o estado de mudança de um sistema  
estados orientado por eventos.
5. Diagrama de Enfatiza o fluxo de controle de uma atividade para  
atividades outra.

**Diagrama de caso de uso** Um *diagrama de caso de uso* mostra um conjunto de casos de uso e atores (um tipo especial de classe) e seus relacionamentos. Aplique esses diagramas para ilustrar a visão estática do caso de uso de um sistema. Os diagramas de caso de uso são importantes

principalmente para a organização e modelagem dos comportamentos de um sistema.

- Os diagramas de caso de uso são examinados no [Capítulo 18](#).

**Diagrama de interação** é o nome coletivo atribuído a diagramas de sequências e de comunicação. Todos os diagramas de sequência e de comunicação são diagramas de interação, e qualquer diagrama de interação é um diagrama de sequência ou de comunicação. Esses diagramas compartilham o mesmo modelo subjacente, embora, na prática, eles enfatizem itens diferentes. (Os diagramas de temporização constituem outro tipo de diagrama de interação que não é tratado neste livro.)

**Diagrama de sequência** Um *diagrama de sequência* é um diagrama de interação que dá ênfase à ordenação temporal de mensagens. Um diagrama de sequência mostra um conjunto de papéis e as mensagens enviadas e recebidas pelas instâncias que representam os papéis. Use os diagramas de sequência para ilustrar a visão dinâmica de um sistema.

- Os diagramas de sequência são examinados no [Capítulo 19](#).

**Diagrama de comunicação** Um *diagrama de comunicação* é um diagrama de interação que dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens. Um diagrama de comunicação mostra um conjunto de papéis, as conexões existentes entre esses papéis e as mensagens enviadas e recebidas pelas instâncias que representam os papéis. Use os diagramas de comunicação para ilustrar a visão dinâmica de um sistema.

- Os diagramas de comunicação são examinados no [Capítulo 19](#).

**Diagrama de estados** Um *diagrama de estados* mostra uma máquina de estados, que consiste de estados, transições, eventos e atividades. Use os diagramas de estados para ilustrar a visão dinâmica de um sistema. Esses diagramas são importantes principalmente para fazer a modelagem do comportamento de uma interface, classe ou colaboração. Os diagramas de

estados dão ênfase ao comportamento de um objeto, solicitado por eventos, que é de grande ajuda para a modelagem de sistemas reativos.

- Os diagramas de estados são examinados no [Capítulo 25](#).

**Diagrama de atividades** Um *diagrama de atividades* mostra o fluxo de uma atividade para outra em um sistema. Uma atividade mostra um conjunto de atividades, o fluxo sequencial ou ramificado de uma atividade para outra e os objetos que realizam ou sofrem ações. Use os diagramas de atividades para ilustrar a visão dinâmica de um sistema. Esses diagramas são importantes principalmente para fazer a modelagem da função de um sistema. Os diagramas de atividades dão ênfase ao fluxo de controle na execução de um comportamento.

- Os diagramas de atividades, um caso especial dos diagramas de estados, são examinados no [Capítulo 20](#).

**Nota:** Existem óbvias limitações práticas para ilustrar algo que é inherentemente dinâmico (o comportamento de um sistema) pela utilização de diagramas (artefatos inherentemente estáticos, principalmente quando desenhados em uma folha de papel, um quadro ou no verso de um envelope). Representados no monitor de um computador, é possível fazer a animação de diagramas comportamentais com a finalidade de simular um sistema executável ou refletir o comportamento real de um sistema que está sendo executado. A UML permite a criação de diagramas dinâmicos e a utilização de cores ou de outras indicações visuais para a “animação” do diagrama. Algumas ferramentas já demonstraram esse uso avançado da UML.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE DIFERENTES VISÕES DE UM SISTEMA

Ao fazer a modelagem de um sistema sob diferentes visões, na verdade você está construindo seu sistema em múltiplas dimensões simultaneamente. Escolhendo o conjunto correto de visões, você configura um processo que o obriga a formular boas perguntas sobre o sistema e a expor os riscos a serem

atacados. Se a escolha das visões não for adequada ou o foco ficar limitado a uma única visão em detrimento de todas as outras, você corre o risco de ocultar questões e protelar problemas que eventualmente destruirão qualquer chance de sucesso.

Para fazer a modelagem de um sistema sob diferentes visões:

- › Decida quais visões são necessárias para expressar da melhor maneira a arquitetura de seu sistema e para expor os riscos técnicos do projeto. As cinco visões de uma arquitetura, descritas anteriormente, são um bom ponto de partida.
  - › Para cada uma dessas visões, decida quais artefatos você precisa criar com a finalidade de captar detalhes essenciais dessa visão. Na maioria dos casos, esses artefatos serão formados por vários diagramas da UML.
  - › Como parte do planejamento do processo, decida quais desses diagramas deverão ser colocados sob algum tipo de controle formal ou semiformal. Esses são os diagramas para os quais você programará revisões e que deverão ser preservados como documentação do projeto.
  - › Guarde os diagramas que não tenham sido aproveitados. Esses diagramas transitórios ainda são úteis para explorar as implicações de suas decisões e para a realização de experiências com as alterações.

Por exemplo, se você está fazendo a modelagem de uma aplicação monolítica simples, executada em um único equipamento, talvez sejam necessários somente os seguintes diagramas:

- » Visão de caso de uso Diagramas de caso de uso
  - » Visão de projeto Diagramas de classes (para a modelagem estrutural)
  - » Visão de interação Diagramas de interação (para a modelagem comportamental)

## › Visão de implementação

### › Visão de implantação Nenhum diagrama é necessário

No caso de um sistema reativo ou de um sistema cujo foco é o fluxo do processo, provavelmente você desejará incluir diagramas de estados e diagramas de atividades, respectivamente, para fazer a modelagem do comportamento do seu sistema.

De maneira semelhante, no caso de um sistema cliente/servidor, provavelmente você desejará incluir diagramas de componentes e diagramas de implantação para fazer a modelagem de detalhes físicos do seu sistema.

Por fim, se você estiver fazendo a modelagem de um sistema distribuído e complexo, você precisará empregar todos os tipos de diagramas da UML para expressar a arquitetura do seu sistema e os riscos técnicos do projeto, conforme é apresentado adiante.

### › Visão de caso de uso Diagramas de caso de uso Diagramas de sequência

### › Visão de projeto Diagramas de classes (para a modelagem estrutural)

Diagramas de interação (para a modelagem comportamental)

Diagramas de estados (para a modelagem comportamental)

### › Visão de interação Diagramas de classes (para a modelagem estrutural)

Diagramas de interação (para modelagem comportamental)

## › Visão de Diagramas de componentes implementação

» Visão de Diagramas de implantação  
implantação

## **MODELAGEM DE DIFERENTES NÍVEIS DE ABSTRAÇÃO**

Não apenas você precisa visualizar o sistema sob vários ângulos, mas também as pessoas envolvidas no desenvolvimento necessitarão da mesma visão do sistema, mas em diferentes níveis de abstração. Por exemplo, considerando um determinado conjunto de classes que captam o vocabulário do espaço do problema, um programador poderá querer uma visão detalhada no nível de atributos, operações e relacionamentos de cada classe. Por outro lado, um analista, que está trabalhando em alguns cenários de casos de uso com um usuário final, provavelmente desejará apenas uma visão mais geral das mesmas classes. Nesse contexto, o programador está trabalhando em um nível inferior de abstração, enquanto o analista e o usuário final estão trabalhando em um nível superior de abstração, mas todos estão trabalhando a partir do mesmo modelo. De fato, como os diagramas são apenas apresentações gráficas dos elementos que formam o modelo, você pode criar vários diagramas relacionados ao mesmo modelo ou a modelos diferentes, ocultando ou exibindo, em cada um deles, diferentes conjuntos desses elementos em níveis distintos de detalhe.

Basicamente, existem duas maneiras de fazer a modelagem de um sistema em níveis diferentes de abstração: pela apresentação de diagramas com níveis diferentes de detalhe para o mesmo modelo; ou pela criação de modelos em níveis diferentes de abstração com diagramas que rastreiam um modelo no outro.

Para fazer a modelagem de um sistema em níveis diferentes de abstração apresentando diagramas com níveis distintos de detalhe:

- » Leve em consideração as necessidades de seus leitores e inicie com um determinado modelo.
- » Se o seu leitor está usando o modelo para construir uma implementação, ele precisará de diagramas em um nível inferior de abstração, ou seja, os

diagramas deverão exibir muitos detalhes. Se estiver usando o modelo para apresentar um modelo conceitual a um usuário final, o leitor precisará de diagramas em um nível superior de abstração, ou seja, os diagramas deverão ocultar vários detalhes.

- » Dependendo de como você se definir nessa variação de níveis mais baixos ou mais altos de abstração, crie um diagrama no nível de abstração correto, ocultando ou exibindo as quatro seguintes categorias de itens em seu modelo:
  1. *Blocos de construção e relacionamentos*: oculte aqueles que não são relevantes para o propósito do diagrama ou para as necessidades do leitor.
  2. *Adornos*: exiba somente os adornos dos blocos de construção e relacionamentos que são essenciais à compreensão do seu propósito.
  3. *Fluxo*: no contexto dos diagramas comportamentais, procure expandir somente aquelas mensagens ou transições que são essenciais à compreensão do seu propósito.
  4. *Estereótipos*: no contexto dos estereótipos utilizados para a classificação de listas de itens, como atributos e operações, exiba somente os itens estereotipados que são essenciais à compreensão do seu propósito.

- ➡ As mensagens são examinadas no [Capítulo 16](#); as transições são apresentadas no [Capítulo 22](#); os estereótipos são explicados no [Capítulo 6](#).

A principal vantagem dessa solução é que você sempre está fazendo a modelagem a partir de um repositório semântico comum. A principal desvantagem é que as modificações de diagramas em um nível de abstração podem tornar obsoletos os diagramas em outros níveis de abstração.

Para fazer a modelagem de um sistema em níveis diferentes de abstração com a criação de modelos em níveis distintos de abstração:

- » Considere as necessidades de seus leitores e decida qual nível de abstração deverá ser adequado a cada um, formando modelos separados para cada nível.
- » Em geral, preencha os modelos em alto nível de abstração com abstrações simples e os modelos em baixo nível de abstração com abstrações detalhadas. Estabeleça dependências de rastreamento entre os elementos relacionados dos diferentes modelos.

► *As dependências de rastreamento são examinadas no [Capítulo 32](#).*

- » Na prática, se você seguir as cinco visões de uma arquitetura, haverá quatro situações básicas a serem encontradas quando fizer a modelagem de um sistema em níveis diferentes de abstração:

1. *Casos de uso e suas realizações*: em um modelo de caso de uso, os casos de uso conduzirão as colaborações em um modelo de projeto.
2. *Colaborações e suas realizações*: as colaborações conduzirão a uma sociedade de classes que funcionam em conjunto para a execução da colaboração.
3. *Componentes e seus projetos*: os componentes de um modelo de implementação conduzirão aos elementos de um modelo de projeto.
4. *Nós e seus componentes*: os nós de um modelo de implantação conduzirão aos componentes em um modelo de implementação.

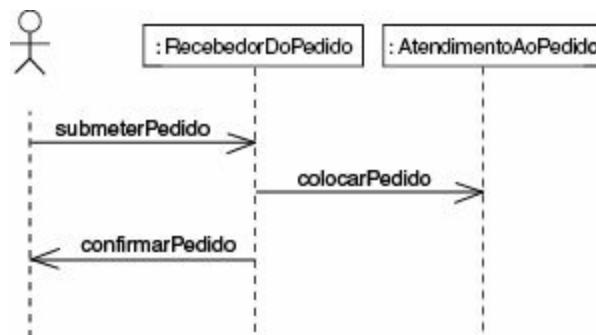
► *Os casos de uso são examinados no [Capítulo 17](#); as colaborações são apresentadas no [Capítulo 28](#); os componentes são explicados no [Capítulo 15](#); e os nós são examinados no [Capítulo 27](#).*

A principal vantagem dessa solução é que os diagramas em níveis diferentes de abstração permanecem mais remotamente acoplados. Isso significa que as modificações de um modelo terão menor efeito direto em outros modelos. A principal desvantagem é a necessidade de gastar recursos com a finalidade de manter sincronizados esses modelos e seus diagramas.

Isso se aplica principalmente quando seus modelos são construídos em paralelo nas diferentes fases do ciclo de vida do desenvolvimento do software, tal qual decidir manter um modelo de análise separado de um modelo de projeto.

Por exemplo, suponha que você esteja fazendo a modelagem de um sistema para um comércio na Web – um dos principais casos de uso desse sistema deverá ser a inclusão de pedidos. Sendo um analista ou um usuário final, provavelmente você criaria alguns diagramas de interação em um alto nível de abstração para exibir a ação de incluir um pedido, conforme mostra a [Figura 7.1](#).

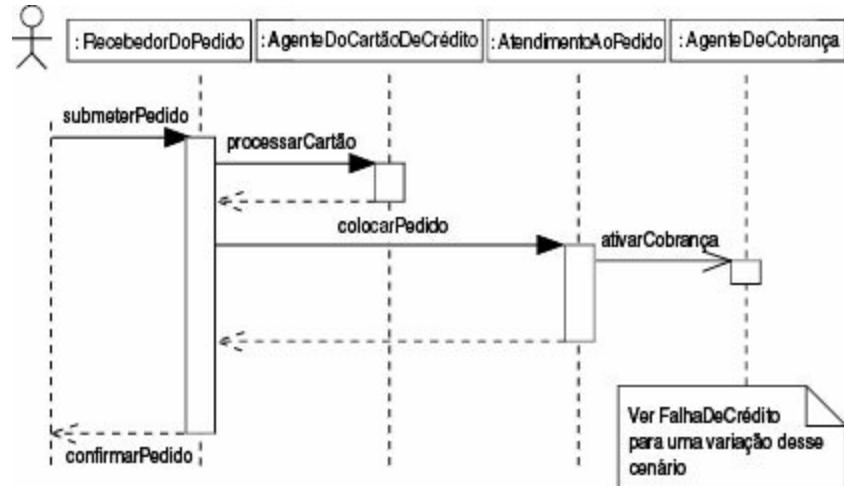
- Os diagramas de interação são examinados no [Capítulo 19](#).



**Figura 7.1:**

Diagrama de interação em alto nível de abstração

Por outro lado, o programador responsável pela implementação desse cenário precisaria trabalhar a partir desse diagrama, expandindo certas mensagens e adicionando outros papéis nessa interação, conforme mostra a [Figura 7.2](#).



**Figura 7.2:**

Interação em baixo nível de abstração

Esses dois diagramas funcionam a partir do mesmo modelo, mas em níveis diferentes de detalhe. O segundo diagrama tem mensagens e papéis adicionais. É razoável dispor de muitos diagramas como esses, principalmente se as suas ferramentas facilitam a navegação de um diagrama para outro.

## MODELAGEM DE VISÕES COMPLEXAS

Independentemente da forma como você desmembra os seus modelos, de vez em quando será necessário criar diagramas extensos e complexos. Por exemplo, para analisar todo o esquema inteiro de um banco de dados abrangendo 100 ou mais abstrações, realmente é valioso estudar um diagrama mostrando todas as classes e suas associações. A partir disso, você será capaz de verificar padrões comuns de colaboração. Exibindo esse modelo em um nível superior de abstração, omitindo alguns detalhes, você perderia as informações necessárias para chegar a conclusões adequadas. Para fazer a modelagem de visões complexas:

- › Em primeiro lugar, convença a si próprio de que não existe uma maneira significativa de apresentar essas informações em um nível superior de abstração, talvez omitindo algumas partes do diagrama e mantendo detalhes de outras partes.

- › Se você ocultou tantos detalhes quanto possível e seu diagrama ainda está complexo, considere a possibilidade de agrupar alguns desses elementos em pacotes ou colaborações em seu diagrama.
  - ➡ *Os pacotes são examinados no [Capítulo 12](#); as colaborações são apresentadas no [Capítulo 28](#).*
- › Se o seu diagrama ainda está complexo, use notas e cores como indicações visuais com a finalidade de chamar a atenção do leitor para os pontos que você deseja esclarecer.
- › Se o seu diagrama ainda está complexo, imprima-o todo e pendure-o na parede. Você perderá a interatividade proporcionada pela versão on-line do diagrama, mas poderá visualizar todo o diagrama e estudá-lo à procura de padrões básicos.

## DICAS E SUGESTÕES

Quando criar um diagrama:

- › Lembre-se de que o propósito de um diagrama da UML não é exibir figuras atraentes, mas visualizar, especificar, construir e documentar. Os diagramas são um meio de chegar ao fim da implantação de um sistema executável.
- › Nem todos os diagramas precisam ser preservados. Considere a possibilidade de elaborar diagramas durante o trabalho, fazendo consultas aos elementos encontrados em seu sistema e use esses diagramas para analisar o sistema enquanto está sendo construído. Muitos desses tipos de diagramas podem ser rejeitados após atenderem aos respectivos propósitos (mas a semântica a partir da qual foram criados permanecerá como uma parte do modelo).
- › Evite diagramas irrelevantes ou redundantes. Esses diagramas apenas congestionam os seus modelos.

- › Em cada diagrama revele somente os detalhes suficientes para esclarecer os aspectos desejados. Informações irrelevantes podem desviar a atenção do leitor em relação ao ponto-chave que você deseja esclarecer.
- › Por outro lado, não crie diagramas mínimos, a menos que seja realmente necessário apresentar algo em um nível muito alto de abstração. Uma supersimplificação pode ocultar detalhes importantes para a compreensão do modelo.
- › Mantenha um equilíbrio entre os diagramas estruturais e os comportamentais em seu sistema. Pouquíssimos sistemas são totalmente estáticos ou totalmente dinâmicos.
- › Não crie diagramas muito extensos (é difícil navegar em diagramas que ocupam várias páginas impressas) ou muito breves (considere a possibilidade de reunir vários diagramas simples em um só).
- › Atribua a cada diagrama um nome significativo, capaz de expressar a sua intenção com clareza.
- › Mantenha organizados os seus diagramas. Agrupe-os em pacotes de acordo com a visão.
- › Não se preocupe exageradamente com o formato do diagrama. Deixe que as ferramentas o ajudem.

Um diagrama bem estruturado:

- › Enfatiza a comunicação de determinado aspecto de uma visão do sistema.
- › Contém somente aqueles elementos que são essenciais à compreensão desse aspecto.
- › Oferece detalhes consistentes com o respectivo nível de abstração (exiba apenas os adornos essenciais à compreensão).
- › Não seja tão minimalista que acabe confundindo o leitor sobre os aspectos importantes da semântica.

Ao elaborar um diagrama:

- › Atribua um nome que comunique o seu propósito.
- › Distribua os elementos de modo a minimizar linhas que se cruzem.
- › Organize espacialmente os elementos, de modo que os itens semanticamente relacionados fiquem próximos.
- › Use notas e cores como indicações visuais com a finalidade de chamar a atenção para as características importantes do diagrama. Entretanto, use as cores com cuidado, pois muita gente não as vê; as cores devem ser usadas como um realce, e não para conter informações essenciais.

CAPÍTULO

---

8

---

# Diagramas de Classes

## Neste capítulo

- › A modelagem de colaborações simples
- › A modelagem do esquema lógico de um banco de dados
- › Engenharia direta e reversa

**O**s diagramas de classes são os diagramas encontrados com maior frequência na modelagem de sistemas orientados a objetos. Um diagrama de classes mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos.

Use os diagramas de classes para fazer a modelagem da visão estática do projeto de um sistema. Na maioria dos casos, isso envolve a modelagem do vocabulário do sistema, a modelagem de colaborações ou a modelagem de esquemas.

Os diagramas de classes também são a base para um par de diagramas relacionados: os diagramas de componentes e os diagramas de implantação. Os diagramas de classes são importantes não só para a visualização, a especificação e a documentação de modelos estruturais, mas também para a construção de sistemas executáveis por intermédio de engenharia direta e reversa.

## PRIMEIROS PASSOS

Ao construir uma casa, você começa com um vocabulário que inclui blocos de construção básicos, como paredes, piso, janelas, portas, tetos e vigas. Esses itens são amplamente estruturais (as paredes têm altura, largura e

espessura), mas também são de alguma forma comportamentais (tipos diferentes de paredes podem suportar diferentes cargas, as portas se abrem e fecham, existem restrições em relação ao vão de um piso sem suportes). De fato, você não pode considerar essas características estruturais e comportamentais de maneira independente. Em vez disso, ao construir sua casa, é necessário considerar como esses itens interagem. Assim, o processo de determinar a arquitetura da casa envolve a reunião desses itens de uma maneira única e agradável para satisfazer a todos os requisitos funcionais e não funcionais. As plantas do projeto que você criar para visualizar sua casa e para especificar detalhes destinados aos construtores contratados são, na verdade, apresentações gráficas desses itens e de seus relacionamentos.

A construção de um software tem muitas dessas características, com exceção de que, considerada a fluidez do software, você tem a capacidade de definir seus próprios blocos de construção básicos desde o ponto zero. Com a UML, você utiliza os diagramas de classes para visualizar os aspectos estáticos desses blocos de construção e seus relacionamentos e para especificar detalhes da construção, conforme mostra a [Figura 8.1](#).

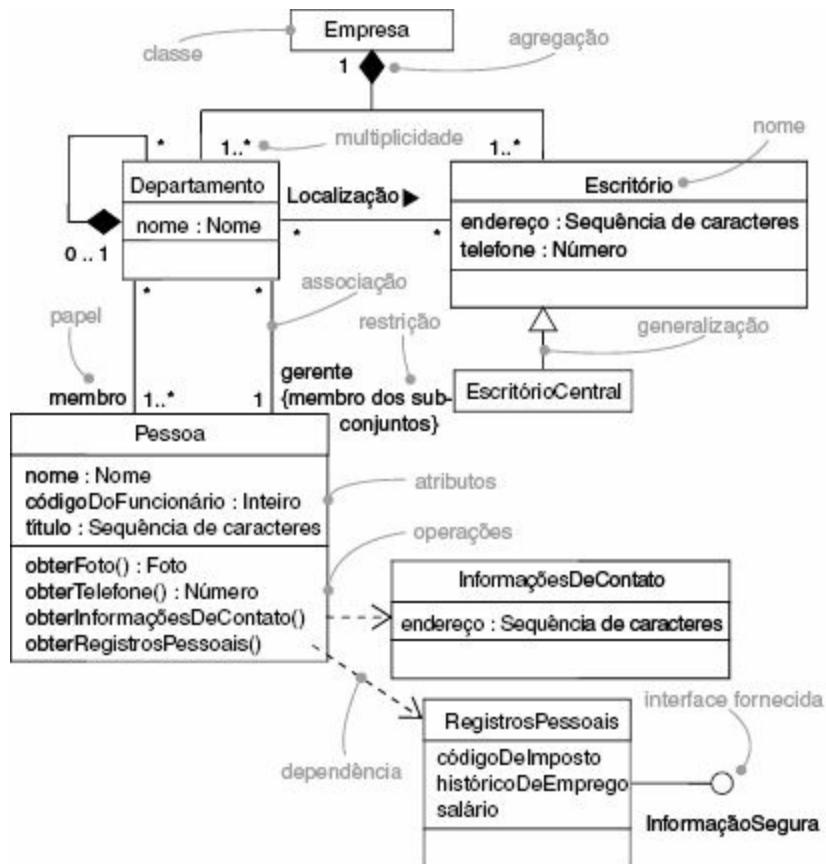


Figura 8.1:

Um diagrama de classes

## TERMOS E CONCEITOS

Um *diagrama de classes* é um diagrama que mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos. Graficamente, um diagrama de classes é uma coleção de vértices e arcos.

## PROPRIEDADES BÁSICAS

Um diagrama de classes é apenas um tipo especial de diagrama e compartilha as mesmas propriedades de todos os outros diagramas – um nome e um conteúdo gráfico que são uma projeção em um modelo. O que diferencia os diagramas de classes dos outros tipos de diagramas é o seu conteúdo particular.

- As propriedades gerais dos diagramas são examinados no [Capítulo 7](#).

## CONTEÚDO

Os diagramas de classes costumam conter os seguintes itens:

- » Classes
- » Interfaces
- » Relacionamentos de dependência, generalização e associação

- As classes são examinadas nos Capítulos [4](#) e [9](#); as interfaces, no [Capítulo 11](#); os relacionamentos, nos Capítulos [5](#) e [10](#); os pacotes, no [Capítulo 12](#); os subsistemas no [Capítulo 32](#); as instâncias, no [Capítulo 13](#).

Assim como os demais diagramas, os diagramas de classes podem conter notas e restrições.

Os diagramas de classes também podem conter pacotes ou subsistemas, utilizados para agrupar elementos do seu modelo em um conjunto maior. Às vezes, você ainda desejará incluir instâncias em seus diagramas de classes, principalmente quando quiser visualizar o tipo (possivelmente dinâmico) de uma instância.

**Nota:** Os diagramas de componentes e os diagramas de implantação são semelhantes aos diagramas de classes, exceto pelo fato de que, em vez de conterem classes, eles contêm componentes e nós, respectivamente.

## USOS BÁSICOS

Os diagramas de classes são utilizados para fazer a modelagem da visão estática de um sistema. Essa visão oferece principalmente suporte para os requisitos funcionais de um sistema – os serviços que o sistema deverá fornecer aos usuários finais.

- As visões de projeto são examinadas no [Capítulo 2](#).

Ao fazer a modelagem da visão estática de um sistema, tipicamente você usará diagramas de classes em uma dentre as três formas.

1. Para fazer a modelagem do vocabulário de um sistema.

- A modelagem do vocabulário de um sistema é apresentada no [Capítulo 4](#).

A modelagem do vocabulário de um sistema envolve uma decisão a respeito de quais abstrações fazem parte do sistema considerado e quais estão fora dos limites do sistema. Use os diagramas de classes para especificar essas abstrações e suas responsabilidades.

2. Para fazer a modelagem de colaborações simples.

- As colaborações são apresentadas no [Capítulo 28](#).

Uma colaboração é uma sociedade de classes, interfaces e outros elementos que funcionam em conjunto para proporcionar algum comportamento cooperativo, maior do que a soma de todos os elementos. Por exemplo, quando você está fazendo a modelagem da semântica de uma transação em um sistema distribuído, não basta observar uma única classe para compreender o que está acontecendo. Em vez disso, essa semântica é realizada por um conjunto de classes que trabalham em conjunto. Use os diagramas de classes para visualizar e especificar esse conjunto de classes e seus relacionamentos.

3. Para fazer a modelagem do esquema lógico de um banco de dados.

- A persistência é examinada no [Capítulo 24](#); a modelagem de bancos de dados físicos é apresentada no [Capítulo 30](#).

Imagine um esquema como se fosse a planta para o projeto conceitual de uma base de dados. Em muitos domínios, você desejará armazenar informações persistentes em um banco de dados relacional ou em um banco

de dados orientado a objetos. Você pode fazer a modelagem de esquemas para esses bancos de dados, utilizando diagramas de classes.

## **TÉCNICAS BÁSICAS DE MODELAGEM**

### **MODELAGEM DE COLABORAÇÕES SIMPLES**

Nenhuma classe é utilizada sozinha. Em vez disso, cada classe funciona em colaboração com outras, para a realização de alguma semântica maior do que cada uma delas individualmente. Portanto, além de captar o vocabulário do seu sistema, você também precisará prestar atenção para visualizar, especificar, construir e documentar as várias formas em que esses itens trabalham em conjunto em seu vocabulário. Use os diagramas de classes para representar tais colaborações.

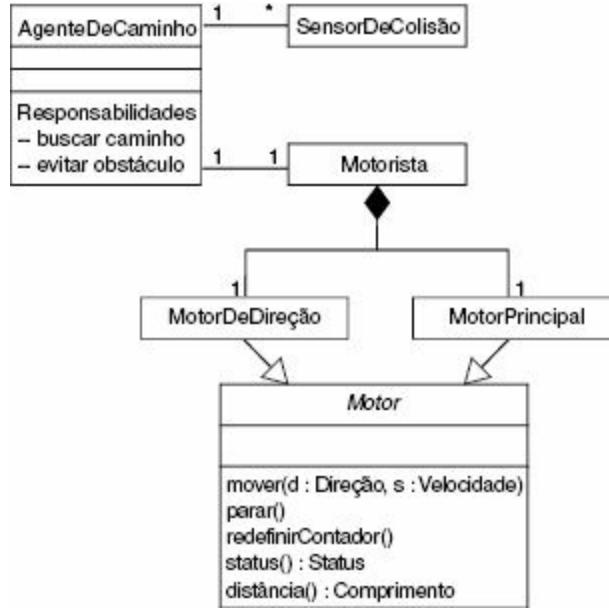
Para fazer a modelagem de uma colaboração:

- › Identifique o mecanismo cuja modelagem você deseja fazer. Um mecanismo representa alguma função ou comportamento da parte do sistema que você está modelando e que resulta da interação de uma sociedade de classes, interfaces e outros itens.
- › Para cada mecanismo, identifique as classes, interfaces e outras colaborações que participam dessa colaboração. Identifique também os relacionamentos existentes entre esses itens.
- › Use cenários para percorrer esses itens. Ao longo do caminho, você descobrirá partes do seu modelo que estão faltando e partes semanticamente erradas.
- › Certifique-se de preencher esses elementos com o respectivo conteúdo. No caso das classes, comece obtendo um bom equilíbrio de responsabilidades. Depois, com o passar do tempo, converta-as em operações e atributos concretos.

► *Mecanismos como esse costumam estar acoplados a casos de uso, conforme é apresentado no Capítulo 17; os cenários são threads em um caso de uso, conforme é examinado no Capítulo 16.*

Por exemplo, a [Figura 8.2](#) mostra um conjunto de classes estabelecido a partir da implementação de um robô autônomo. A figura tem como foco as classes envolvidas no mecanismo para mover o robô por um determinado caminho. Existe uma classe abstrata (Motor) com duas classes-filha concretas MotorDeDireção e MotorPrincipal. Essas duas classes herdam as cinco operações da classe-mãe, Motor. As duas classes, por sua vez, são mostradas como partes de outra classe, Motorista. A classe AgenteDoCaminho tem uma associação de um-para-um com Motorista e uma associação de um-para-muitos com SensorDeColisão. Nenhum atributo ou operação é mostrado para AgenteDoCaminho, apesar de suas responsabilidades serem definidas.

Existem muitas outras classes envolvidas nesse sistema, mas esse diagrama tem como foco somente aquelas abstrações que estão diretamente envolvidas na movimentação do robô. Você encontrará algumas dessas mesmas classes em outros diagramas. Por exemplo, apesar de não ser mostrado aqui, a classe AgenteDoCaminho colabora com pelo menos outras duas classes (Ambiente e AgenteDoDestino) em um mecanismo de nível mais alto para o gerenciamento dos objetivos conflitantes que o robô poderá ter em determinado momento. De modo semelhante, também não mostrado aqui, as classes SensorDeColisão e Motorista (e suas partes) que colaboram com uma outra classe (AgenteDeFalhas) em um mecanismo responsável por uma verificação contínua do hardware do robô à procura de erros. Considerando cada uma dessas colaborações como o foco em diferentes diagramas, você fornece uma visão compreensível do sistema sob vários ângulos.



**Figura 8.2:**

A modelagem de colaborações simples

## MODELAGEM DO ESQUEMA LÓGICO DE UM BANCO DE DADOS LÓGICO

Em muitos sistemas cuja modelagem você fará, haverá objetos persistentes. Isso significa que poderão ser armazenados em um banco de dados para serem recuperados posteriormente. Com muita frequência, você usará um banco de dados relacional, um banco de dados orientado a objetos ou um banco de dados híbrido relacional/orientado a objetos para armazenamentos de itens persistentes. A UML é adequada para a modelagem de esquemas lógicos de bancos de dados, além dos próprios bancos de dados físicos.

- *A modelagem da distribuição e da migração de objetos persistentes é examinada no [Capítulo 24](#); a modelagem de bancos de dados físicos é apresentada no [Capítulo 30](#).*

Os diagramas de classes da UML são um superconjunto dos diagramas de entidade-relacionamento (E-R), uma ferramenta básica de modelagem para o projeto lógico de bancos de dados. Enquanto os diagramas E-R clássicos têm

seu foco apenas nos dados, os diagramas de classes vão um pouco além, permitindo ainda a modelagem de comportamentos. No banco de dados físico, essas operações lógicas costumam servir como procedimentos armazenados ou para iniciar operações.

Para fazer a modelagem de um esquema:

- › Identifique as classes existentes em seu modelo, cujo estado deve transcender o tempo de vida de suas aplicações.
- › Crie um diagrama de classes contendo essas classes. Você pode definir seu próprio conjunto de estereótipos e valores atribuídos para direcionar detalhes específicos do banco de dados.
- › Amplie os detalhes estruturais dessas classes. Em geral, isso significa especificar os detalhes de seus atributos e ter como foco as associações e suas cardinalidades que estruturam essas classes.
- › Procure padrões comuns que complicam o projeto do banco de dados físico, como associações cíclicas, associações de um-para-um e associações diárias. Onde for necessário, crie abstrações intermediárias para simplificar sua estrutura lógica.
- › Considere também o comportamento dessas classes, expandindo as operações que são importantes para o acesso e a integridade dos dados. Em geral, para proporcionar uma melhor separação de questões, as regras de negócios referentes à manipulação de conjuntos desses objetos deverão ser encapsuladas em uma camada superior a essas classes persistentes.
- › Onde for possível, use ferramentas para ajudá-lo a transformar seu projeto lógico em um projeto físico.

**Nota:** *O projeto lógico de um banco de dados ultrapassa o escopo deste livro. Aqui o foco se resume simplesmente a apresentar como você pode fazer a modelagem de esquemas utilizando a UML. Na prática, você acabará usando estereótipos adequados ao tipo de banco de dados (relacional ou orientado a objetos) que está utilizando.*

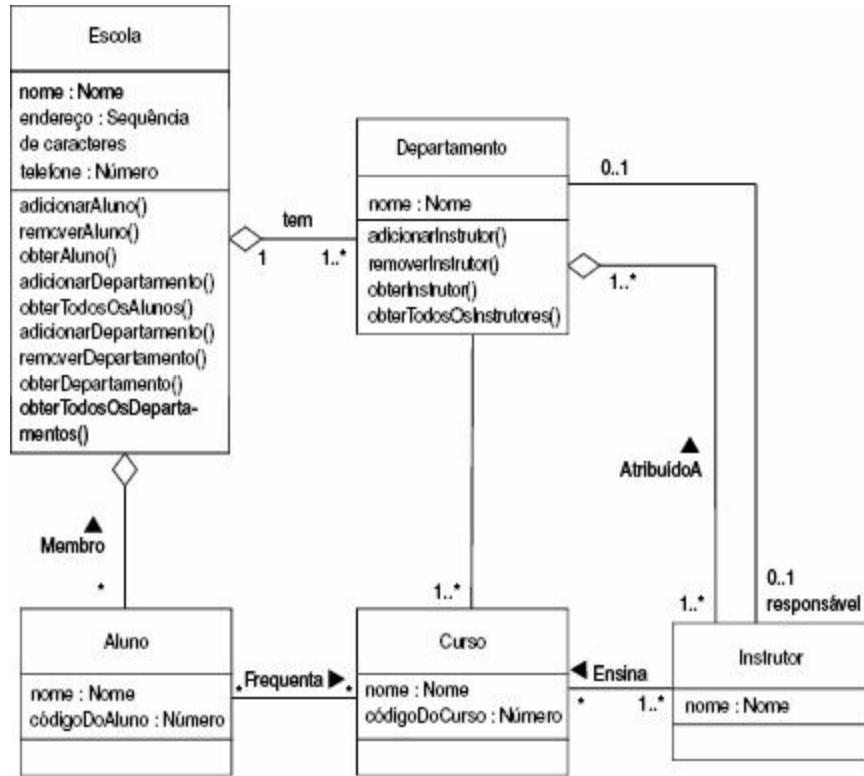
► Os estereótipos são examinados no [Capítulo 6](#).

A [Figura 8.3](#) mostra um conjunto de classes definido a partir de um sistema de informações para uma escola. Essa figura é expandida a partir de um diagrama de classes anterior e você observará os detalhes dessas classes revelados em um nível suficiente para a construção de um banco de dados físico. Iniciando na parte inferior esquerda desse diagrama, você encontrará as classes chamadas Aluno, Curso e Instrutor. Existe uma associação entre Aluno e Curso, especificando que os alunos frequentam cursos. Além disso, qualquer aluno poderá frequentar qualquer número de cursos e todos os cursos poderão ter qualquer número de alunos.

Esse diagrama expõe os atributos de todas essas seis classes. Observe que todos os atributos são tipos primitivos. Ao fazer a modelagem de um esquema, geralmente você desejará fazer a modelagem do relacionamento com quaisquer tipos não primitivos, usando uma associação explícita no lugar de um atributo.

► A modelagem de tipos primitivos é examinada no [Capítulo 4](#); a agregação é apresentada nos [Capítulos 5 e 10](#).

Duas dessas classes (Escola e Departamento) expõem várias operações para a manipulação de suas partes. Essas operações são incluídas porque são importantes para manter a integridade dos dados (o acréscimo ou a exclusão de um Departamento, por exemplo, poderá ter alguns efeitos em cadeia). Existem muitas outras operações a serem consideradas para essas e outras classes, tal como fazer consultas aos pré-requisitos de um curso antes de atribuir-lhe um aluno. Trata-se mais propriamente de regras de negócio do que operações para a integridade do banco de dados e, portanto, é melhor colocá-las em um nível mais alto de abstração do que esse esquema.



**Figura 8.3:**  
A modelagem de um esquema

## ENGENHARIA DIRETA E REVERSA

A modelagem é importante, mas é necessário lembrar que o produto principal de uma equipe de desenvolvimento é o software e não os diagramas. É claro que o motivo de criação dos modelos é o de entregar, de modo previsível, no momento certo, o software correto que satisfará aos objetivos dos usuários e do negócio. Por essa razão, é importante que os modelos criados e as implementações fornecidas sejam mapeados uns em relação aos outros e que isso seja feito de modo a minimizar ou até eliminar os custos de manter seus modelos e suas implementações sincronizados uns com os outros.

- A importância da modelagem é examinada no Capítulo 1.

Em alguns usos da UML, os modelos criados nunca serão mapeados no código. Por exemplo, se você estiver fazendo a modelagem de um processo de negócio, utilizando diagramas de atividades, muitas das atividades modeladas envolverão pessoas, e não computadores. Em outros casos, você desejará fazer a modelagem de sistemas cujas partes são, em seu nível de abstração, apenas um componente de hardware (apesar de que, em outro nível de abstração, seja melhor supor que esse hardware contenha software e um computador embutidos).

- Os *diagramas de atividades* são examinados no [Capítulo 20](#).

Na maioria dos casos, entretanto, os modelos que você criar serão mapeados no código. A UML não especifica um determinado mapeamento para alguma linguagem de programação orientada a objetos, mas a UML foi desenvolvida, considerando esses mapeamentos. Isso se aplica principalmente aos diagramas de classes, cujo conteúdo tem um claro mapeamento para todas as linguagens orientadas a objeto de capacidade industrial, como Java, C++, Smalltalk, Eiffel, Ada, ObjectPascal e Forte. A UML também foi projetada com a finalidade de ser mapeada em várias linguagens comerciais baseadas em objetos, como o Visual Basic.

**Nota:** O mapeamento da UML em linguagens específicas de implementação para engenharia direta ou reversa ultrapassa o escopo deste livro. Na prática, você acabará usando estereótipos e valores atribuídos adequados à linguagem de programação que estiver sendo utilizada.

- Os *estereótipos* e os *valores atribuídos* são apresentados no [Capítulo 6](#).

A *engenharia direta* é o processo de transformar um modelo em código pelo mapeamento de uma linguagem de implementação. A engenharia direta resulta em uma perda de informações, pois os modelos escritos na UML são semanticamente mais ricos do que qualquer linguagem atual de programação orientada a objetos. De fato, essa é uma das principais razões por que você

precisará de modelos além do código. Características estruturais, como as colaborações, e características comportamentais, como as interações, podem ser visualizadas na UML com clareza, mas não tão claramente no código bruto.

Para criar um diagrama de classes na engenharia direta:

- » Identifique as regras para o mapeamento em sua linguagem de implementação ou linguagens à sua escolha. Isso é algo que você desejará fazer para o seu projeto ou organização como um todo.
- » Dependendo da semântica das linguagens que você escolher, poderá ser necessário restringir o uso de certas características da UML. Por exemplo, a UML permite fazer a modelagem de heranças múltiplas, mas a Smalltalk somente permite a herança simples. Você poderá proibir os desenvolvedores de fazer a modelagem com herança múltipla (o que fará com que os modelos dependam da linguagem) ou desenvolver idiomas capazes de transformar essas características mais ricas para a linguagem de implementação (o que tornará mais complexo o mapeamento).
- » Use valores atribuídos para especificar sua linguagem-destino. Isso poderá ser feito no nível das classes individuais, caso seja necessário um controle preciso. Você também poderá fazer isso em um nível mais alto, como o nível de colaborações ou o de pacotes.
- » Use as ferramentas para gerar código.

A [Figura 8.4](#) ilustra um diagrama de classes simples, especificando uma instanciação da cadeia de padrões de responsabilidades. Essa instanciação particular envolve três classes: Client, EventHandler e GUIEventHandler. Client e EventHandler são mostradas como classes abstratas, enquanto GUIEventHandler é concreta. EventHandler tem a operação usual esperada em relação a esse padrão (`handleRequest`), embora dois atributos privados tenham sido adicionados para essa instanciação.

► Os padrões são examinados no [Capítulo 29](#).

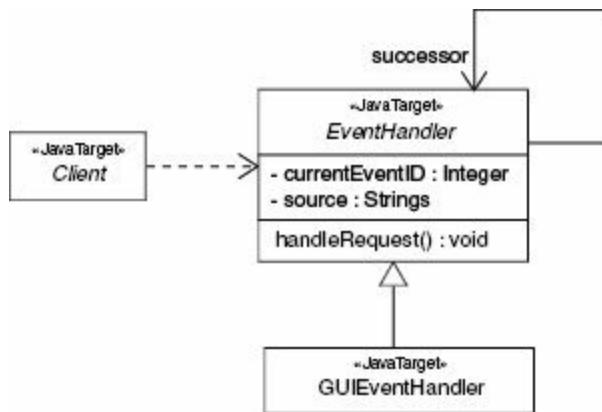


Figura 8.4:

Engenharia direta

Todas essas classes especificam um mapeamento para a linguagem Java, conforme é observado nos valores atribuídos. A engenharia direta aplicada às classes desse diagrama para a linguagem Java será direta, com a utilização de uma ferramenta. Na engenharia direta, a classe EventHandler geraria o seguinte código:

```
public abstract class EventHandler {  
    EventHandler successor;  
    private Integer currentEventID;  
    private String source;  
    EventHandler( ) {}  
    public void handleRequest( ) {}  
}
```

A *engenharia reversa* é o processo de transformação de código em um modelo por meio do mapeamento a partir de uma linguagem de implementação específica. A engenharia reversa resulta de uma grande quantidade de informações, algumas das quais estão em um nível mais baixo de detalhe do que será necessário para a construção de modelos úteis. Ao mesmo tempo, a engenharia reversa é incompleta. Existe uma perda de informações quando a engenharia direta converte os modelos em código e, consequentemente, não será possível recriar inteiramente um modelo a partir do código, a menos que suas ferramentas codifiquem informações nos

comentários de origem, que vão além da semântica da linguagem de implementação.

Para criar um diagrama de classes através da engenharia reversa:

- › Identifique as regras para o mapeamento a partir de sua linguagem de implementação ou de linguagens à sua escolha. Isso é algo que você desejará fazer para o seu projeto ou organização como um todo.
- › Usando uma ferramenta, aponte o código cuja engenharia reversa você deseja fazer. Use sua ferramenta para gerar um novo modelo ou para modificar um já existente, gerado pela engenharia direta. Não faz sentido esperar que se consiga fazer engenharia reversa de um modelo único e conciso a partir da totalidade do código. Você deve selecionar parcelas do código e construir o modelo desde o princípio.
- › Usando ferramenta, crie um diagrama de classes, realizando consultas ao modelo. Por exemplo, comece com uma ou mais classes e depois amplie o diagrama, seguindo relacionamentos específicos ou outras classes vizinhas. Exiba ou oculte detalhes do conteúdo desse diagrama de classes, conforme seja necessário para comunicar o seu propósito.
- › Adicione manualmente informações do projeto ao modelo para expressar a finalidade do projeto que está ausente ou oculta no código.

## DICAS E SUGESTÕES

Ao criar diagramas de classes na UML, lembre-se de que todo diagrama de classes é apenas uma apresentação gráfica da visão estática do projeto de um sistema. Nenhum diagrama de classes precisa captar tudo sobre a visão de projeto do sistema. Coletivamente, todos os diagramas de classes representam a visão estática completa do projeto do sistema; individualmente, cada diagrama representa somente um aspecto.

Um diagrama de classes bem estruturado:

- › Enfatiza a comunicação de um único aspecto da visão estática do projeto do sistema.
- › Contém somente elementos essenciais à compreensão desse aspecto.
- › Fornece detalhes consistentes com o respectivo nível de abstração, exibindo somente os adornos essenciais à compreensão.
- › Não é tão minimalista que prejudique a informação do leitor sobre a semântica importante.

Ao criar um diagrama de classes:

- › Atribua-lhe um nome que comunique seu propósito.
- › Distribua seus elementos de modo a minimizar o cruzamento de linhas.
- › Organize espacialmente seus elementos de modo que os itens semanticamente relacionados fiquem fisicamente próximos.
- › Use notas e cores como indicações visuais com a finalidade de chamar a atenção para características importantes do diagrama.
- › Procure não exibir uma quantidade excessiva de tipos de relacionamentos. Em geral, um único tipo de relacionamento tenderá a predominar em cada diagrama de classes.

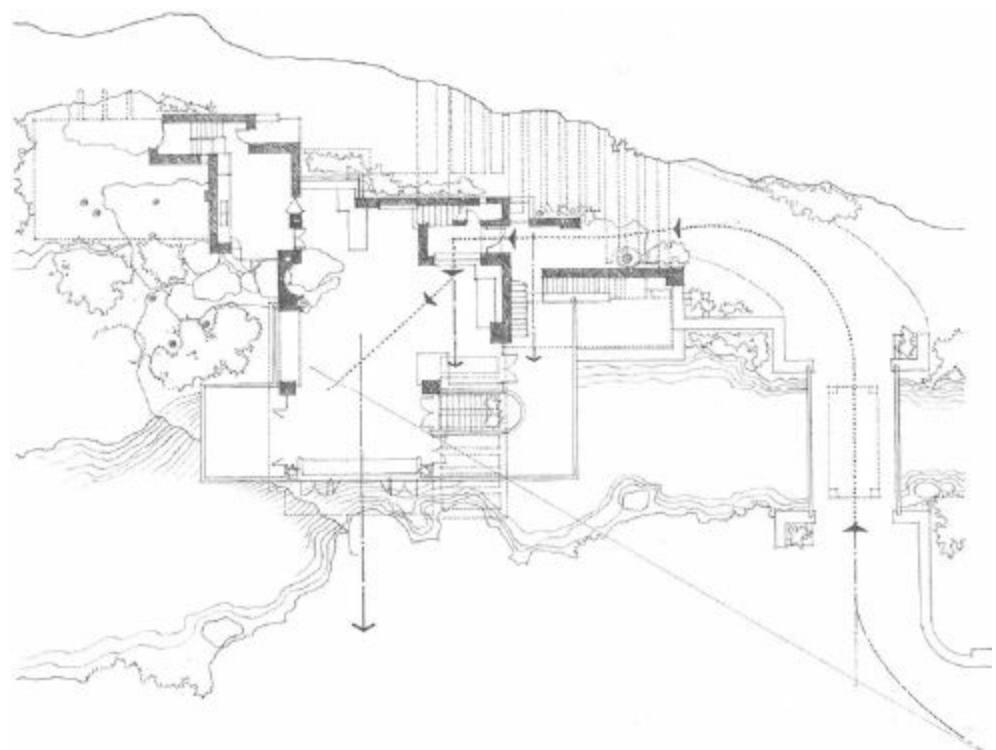
P A R T E

---

3

---

# **MODELAGEM ESTRUTURAL AVANÇADA**



CAPÍTULO

---

9

---

# Classes Avançadas

## Neste capítulo

- » *Classificadores, propriedades especiais de atributos e operações e diferentes tipos de classes*
- » *A modelagem da semântica de uma classe*
- » *A escolha do tipo correto de classificador*

**A**s classes realmente são os blocos de construção mais importantes de qualquer sistema orientado a objetos. Porém, as classes são apenas um dos tipos de um bloco de construção ainda mais geral existente na UML: os classificadores. Um classificador é um mecanismo que descreve características estruturais e comportamentais. Os classificadores incluem classes, interfaces, tipos de dados, sinais, componentes, nós, casos de uso e subsistemas.

Além das propriedades mais simples de atributos e operações, descritas na parte anterior, os classificadores (e principalmente as classes) têm características avançadas: é possível modelar a multiplicidade, a visibilidade, as assinaturas, o polimorfismo e outras características. Na UML, você pode fazer a modelagem da semântica de uma classe e estabelecer seu significado em qualquer grau de formalidade desejado.

► As propriedades básicas das classes são examinadas no [Capítulo 4](#).

Na UML, existem vários tipos de classificadores e de classes; é importante que você escolha aqueles que sejam mais adequados à modelagem de sua abstração do mundo real.

## PRIMEIROS PASSOS

Quando você constrói uma casa, em algum ponto do projeto você tomará uma decisão de arquitetura sobre os materiais da construção. Inicialmente, é suficiente definir materiais como madeira, pedra ou aço. Esse é um nível de detalhamento suficiente para iniciar a construção. O material escolhido será afetado pelos requisitos do seu projeto – aço e concreto seriam uma boa escolha, se você está construindo em uma área sujeita a furacões, por exemplo. À medida que você avança, o material escolhido afetará as decisões sobre o seu projeto a serem tomadas a seguir – a escolha de madeira ou de aço afetará a massa a ser suportada, por exemplo.

► A arquitetura é examinada no [Capítulo 2](#).

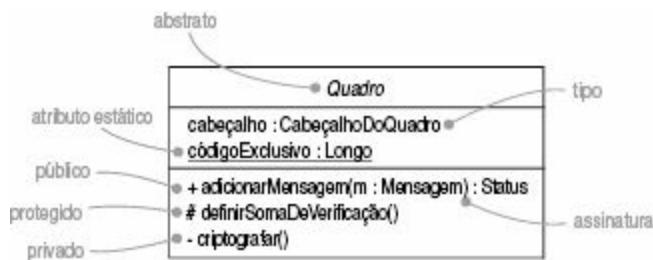
À medida que o projeto prosseguir, será preciso aprimorar essas decisões básicas sobre o projeto e adicionar detalhes suficientes para que um engenheiro de estruturas possa validar a segurança do projeto e para que um construtor continue com a construção. Por exemplo, poderá ser necessário especificar não só madeira, mas madeira de determinado tipo, tratada adequadamente para resistir a insetos.

O mesmo ocorre quando você constrói um software. No início do projeto, é suficiente afirmar que será incluída uma classe `Cliente` que executará determinadas responsabilidades. À medida que você aprimora sua arquitetura e avança a construção, será preciso decidir qual será a estrutura da classe (seus atributos) e qual será seu comportamento (suas operações), de maneira suficiente e necessária para a execução dessas responsabilidades. Por fim, enquanto o sistema executável evolui, você precisará fazer a modelagem de detalhes, como a visibilidade de operações e atributos individuais, a semântica concorrente da classe como um todo e suas operações individuais, além das interfaces realizadas pela classe.

► As responsabilidades são examinadas no [Capítulo 6](#).

A UML fornece uma representação para várias propriedades avançadas, conforme mostra a [Figura 9.1](#). Essa notação permite que você visualize, especifique, construa e documente uma classe em qualquer nível de detalhe desejado, até o suficiente para dar suporte à engenharia direta e reversa de modelos e código.

- ➡ As engenharias direta e reversa são examinadas nos Capítulos [8, 14, 18, 19, 20, 25, 30 e 31](#).



**Figura 9.1:**  
Classes avançadas

## TERMOS E CONCEITOS

Um *classificador* é um mecanismo que descreve características estruturais e comportamentais. Os classificadores incluem classes, interfaces, tipos de dados, sinais, componentes, nós, casos de uso e subsistemas.

## CLASSIFICADORES

Ao fazer uma modelagem, você revela abstrações que representam itens do mundo real e itens que existem apenas na sua solução. Por exemplo, se você está construindo um sistema de pedidos baseado na Web, o vocabulário do seu projeto provavelmente incluirá uma classe Cliente (representando as pessoas que solicitam produtos) e uma classe Transação (um artefato de implementação, representando uma ação atômica). No sistema implantado, poderá haver um componente Preço, com instâncias em todos os nós cliente.

Cada uma dessas abstrações terá instâncias; separar a essência e a instância dos itens em seu mundo é uma parte importante da modelagem.

- A modelagem do vocabulário de um sistema é examinada no [Capítulo 4](#); a dicotomia classe/objeto é apresentada no [Capítulo 2](#).

Alguns itens em UML não têm instâncias – por exemplo, os pacotes e os relacionamentos de generalização. Em geral, esses elementos da modelagem que podem ter instâncias são chamados classificadores (as associações e as mensagens também podem ter instâncias, mas suas instâncias não são exatamente o mesmo que as instâncias de uma classe). Ainda mais importante, um classificador apresenta características estruturais (na forma de atributos), além de características comportamentais (na forma de operações). Todas as instâncias de um determinado classificador compartilham as mesmas características, mas cada instância tem seu próprio valor para cada atributo.

- As instâncias são examinadas no [Capítulo 13](#); os pacotes, no [Capítulo 12](#); a generalização, nos Capítulos [5](#) e [10](#); as associações, nos Capítulos [5](#) e [10](#); as mensagens, no [Capítulo 16](#); as interfaces, no [Capítulo 11](#); os tipos de dados, nos Capítulos [4](#) e [11](#); os sinais, no [Capítulo 21](#); os componentes, no [Capítulo 15](#); os nós, no [Capítulo 27](#); os casos de uso, no [Capítulo 17](#); os subsistemas, no [Capítulo 32](#).

O tipo mais importante de estática na UML é a classe. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. Entretanto, as classes não são o único tipo de classificador. A UML oferece vários outros tipos de classificadores para ajudá-lo a fazer sua modelagem.

» Interface

Uma coleção de operações utilizadas para a especificação do serviço de uma classe ou de

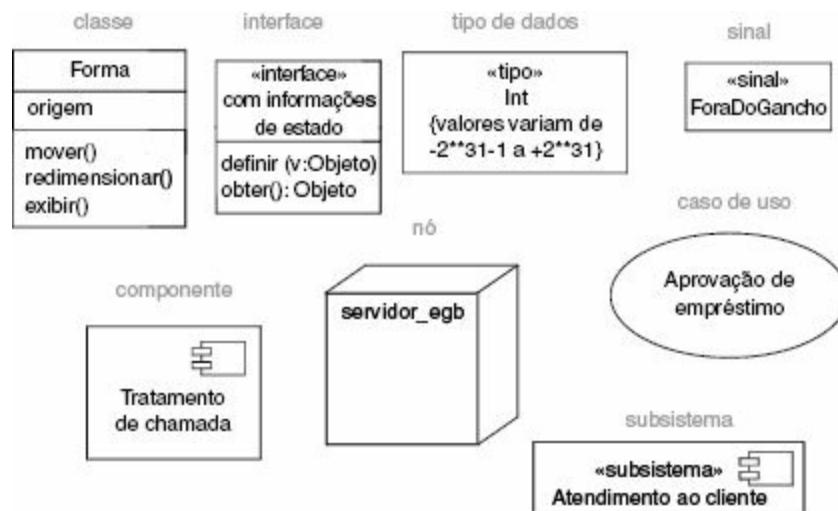
um componente.

» Tipo de dados	Um tipo cujos valores não têm identidade, incluindo tipos primitivos predefinidos (como números ou sequências de caracteres), além de tipos enumerados (como booleano).
» Associação	Uma descrição de um conjunto de vínculos, cada um relacionando dois ou mais objetos.
» Sinal	A especificação de um estímulo assíncrono, comunicado entre as instâncias.
» Componente	Uma parte modular de um sistema, que oculta a sua implementação atrás de um conjunto de interfaces externas.
» Nó	Um elemento físico que existe em tempo de execução e que representa um recurso computacional, geralmente com pelo menos alguma memória e frequentemente com capacidade de processamento.
» Caso de uso	Uma descrição de um conjunto de uma sequência de ações, incluindo variantes, que um sistema realiza, proporcionando um resultado observável do valor para um determinado ator.
» Subsistema	Um componente que representa uma parte importante de um sistema.

Na maioria dos casos, todos os tipos de classificadores têm características estruturais e comportamentais. Além disso, ao fazer a modelagem com qualquer um desses classificadores, você pode empregar todas as características avançadas descritas neste capítulo com a finalidade de oferecer o nível de detalhe necessário para captar o significado da abstração.

Graficamente, a UML diferencia os classificadores, conforme mostra a Figura 9.2.

**Nota:** Uma solução minimalista utilizaria um mesmo ícone para todos os classificadores. Isso não teria sentido, pois, por exemplo, as classes e os componentes são abstrações muito diferentes (uma é lógica e a outra é física) e, portanto, é muito importante ter uma indicação visual distintiva. De maneira semelhante, uma solução máxima teria utilizado ícones diferentes para cada tipo de classificador. Isso também não faria sentido, pois, por exemplo, as classes e os tipos de dados não apresentam essa diferença. O projeto em UML permite um equilíbrio – esses classificadores materialmente diferentes dos demais têm um ícone próprio e aqueles que não são materialmente diferentes usam palavras-chave especiais (como tipo, sinal ou subsistema).



**Figura 9.2:**  
Classificadores

## VISIBILIDADE

Um dos mais importantes detalhes que você pode especificar para os atributos e operações de um classificador é a sua visibilidade. A visibilidade de uma característica especifica se ela pode ser utilizada por outros classificadores. Na UML, você pode determinar qualquer um dos quatro níveis de visibilidade.

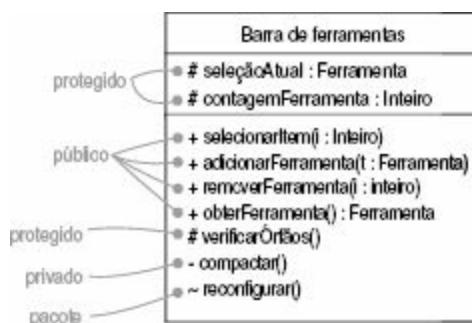
1. público Qualquer classificador externo com visibilidade para que determinado classificador seja capaz de usar a característica;

especificado por ser antecedido pelo símbolo +.

2. protegido Qualquer descendente do classificador é capaz de usar a característica; especificado por ser antecedido pelo símbolo #.
3. privado Somente o próprio classificador é capaz de usar a característica; especificado por ser antecedido pelo símbolo -.
4. pacote Somente classificadores declarados no mesmo pacote podem usar a característica; especificado por ser antecedido pelo símbolo ~.

■ *Um classificador é capaz de visualizar outro classificador, caso este se encontre em seu escopo e se houver um relacionamento explícito ou implícito para o alvo; os relacionamentos são examinados nos Capítulos 5 e 10; os descendentes provêm de relacionamentos de generalização, conforme apresenta o Capítulo 5; a permissão faz com que um classificador compartilhe suas partes privadas, conforme é apresentado no Capítulo 10.*

A Figura 9.3 mostra uma combinação de valores públicos, protegidos e privados para a classe Barra de ferramentas. Ao especificar a visibilidade das características de um classificador, geralmente você deseja ocultar todos os seus detalhes de implementação e exibir somente aquelas características necessárias para a execução das responsabilidades da abstração. Essa é a base para ocultar informações, o que é essencial para a construção de sistemas sólidos e relevantes. Se não incluir explicitamente um adorno para uma característica, usando um símbolo de visibilidade, geralmente você pode considerá-la como pública.



**Figura 9.3:**

Visibilidade

**Nota:** A propriedade de visibilidade da UML atende à semântica comum à maioria das linguagens de programação, incluindo C++, Java, Ada e Eiffel. Entretanto, observe que as linguagens distinguem-se ligeiramente em sua semântica de visibilidade.

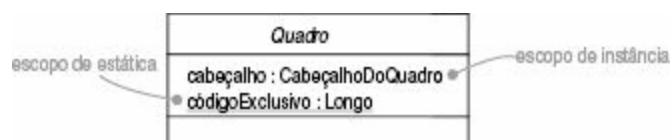
## ESCOPO DE INSTÂNCIA E DE ESTÁTICA

Outro importante detalhe que você pode especificar para os atributos e operações de um classificador é o respectivo escopo. O escopo de uma característica especifica se cada instância do classificador tem seu próprio valor distinto da característica ou se haverá um único valor da característica para todas as instâncias do classificador. Na UML, você pode especificar dois tipos de escopo do proprietário.

► As instâncias são apresentadas no [Capítulo 13](#).

1. instância Cada instância do classificador mantém seu próprio valor para a característica. Esse é o tipo padrão e não requer notação adicional.
2. estática Existe apenas um valor da característica para todas as instâncias do classificador. Também é chamado de *escopo de classe*. Sua notação é o sublinhado.

Conforme mostra a [Figura 9.4](#) (uma simplificação da [Figura 9.1](#)), uma característica pertencente ao escopo do classificador é representada pelo nome sublinhado da característica. A ausência de adornos significa que a característica tem seu escopo determinado pela instância.



**Figura 9.4:**

Escopo do proprietário

Em geral, a maioria das características dos classificadores em sua modelagem terá seu escopo definido pela instância. O uso mais comum das características com escopo do classificador é destinado a atributos privados que devem ser compartilhados em um conjunto de instâncias, como no caso de gerar códigos únicos para novas instâncias de uma classe.

**Nota:** *O escopo de estática é mapeado para o que as linguagens C++ e Java chamam de operações e atributos estáticos.*

O escopo de estática funciona de maneira um pouco diferente para operações. Uma operação de instância tem um parâmetro implícito correspondente ao objeto que está sendo manipulado. Uma operação estática não tem esse parâmetro e comporta-se como um procedimento global tradicional que não tem objeto-alvo. As operações estáticas são usadas para operações que criam instâncias ou operações que manipulam atributos estáticos.

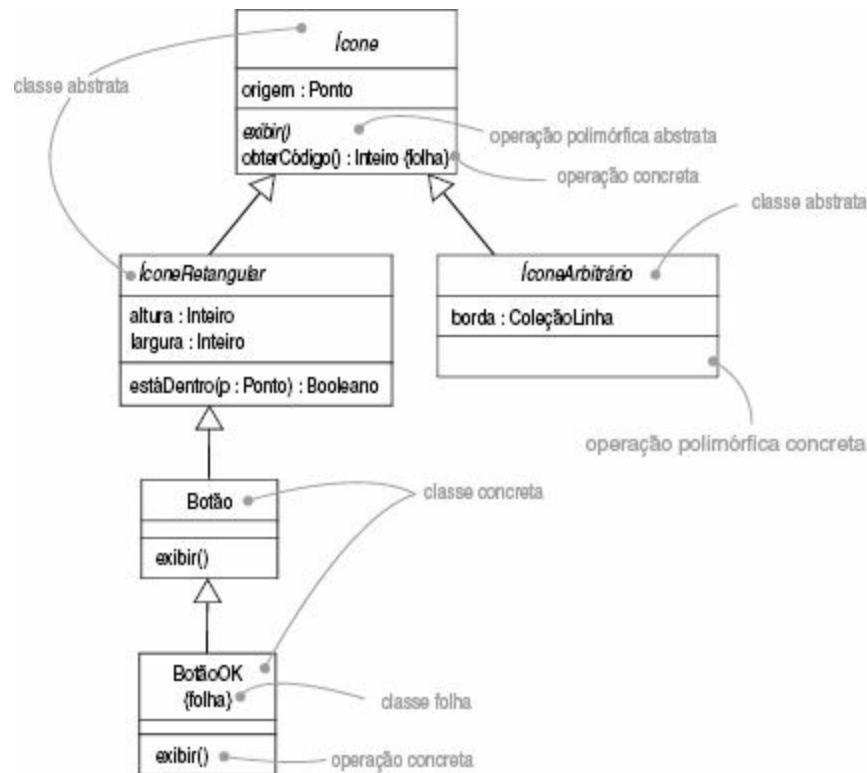
## ELEMENTOS ABSTRATOS, RAIZ, FOLHA E POLIMÓRFICOS

Você usa os relacionamentos de generalização para fazer a modelagem de uma estrutura de classes, com abstrações mais gerais no topo da hierarquia e outras mais específicas na parte inferior. Nessas hierarquias, é comum especificar que certas classes são abstratas – significando que poderão não apresentar instâncias diretas. Na UML, você especifica que uma classe é abstrata escrevendo seu nome em itálico. Por exemplo, conforme mostra a [Figura 9.5](#), *Ícone*, *ÍconeRetangular* e *ÍconeArbitrário* são todas classes abstratas. Por contraste, uma classe concreta (como *Botão* e *BotãoOK*) é aquela que pode ter instâncias diretas.

- A generalização é examinada nos Capítulos [5](#) e [10](#); as instâncias são apresentadas no [Capítulo 13](#).

Sempre que utilizar uma classe, provavelmente você desejará que as características sejam herdadas de outras classes mais gerais e que outras

classes mais específicas possam herdar características dessa classe. Essa é a semântica normal obtida das classes na UML. Entretanto, também é possível especificar que uma classe não terá classes-filha. Esse tipo de elemento é chamado classe-folha e é especificado na UML pela escrita da propriedade `leaf` abaixo do nome da classe. Por exemplo, na figura, `BotãoOK` é uma classe-folha e não terá classes-filha.



**Figura 9.5:**

Operações e classes abstratas e concretas

As operações têm propriedades semelhantes. Tipicamente, uma operação é polimórfica, significando que, em uma hierarquia de classes, você pode especificar operações com a mesma assinatura em pontos diferentes da hierarquia. As operações presentes nas classes-filha anulam o comportamento das operações existentes nas classes-mãe. Quando uma mensagem é despachada em tempo de execução, a operação a ser invocada na hierarquia é escolhida polimorficamente, ou seja, uma correspondência é determinada em tempo de execução de acordo com o tipo de objeto. Por exemplo, `exibir` e

estáDentro são duas operações polimórficas. Além disso, a operação `Ícone` : : `exibir( )` é abstrata, significando que é incompleta e necessita de uma filha para fornecer uma implementação da operação. Na UML, você especifica uma operação abstrata, escrevendo o respectivo nome em itálico, do mesmo modo como é feito no caso das classes. Por contraste, `Ícone` : : `obterCódigo( )` é uma operação-folha, conforme é designado pela propriedade `leaf`. Isso significa que a operação não é polimórfica e não poderá ser anulada. (É similar a uma operação final no Java.)

► As mensagens são examinadas no [Capítulo 16](#).

**Nota:** As operações abstratas são mapeadas para o que a linguagem C++ chama de operações virtuais puras; as operações-folha na UML são mapeadas para operações não virtuais no C++.

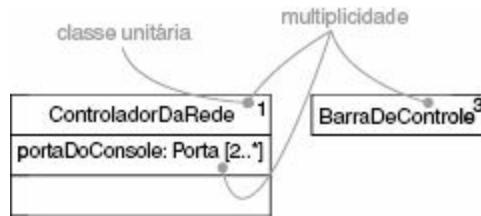
## MULTIPLICIDADE

Sempre que você utiliza uma classe, é razoável assumir que poderá haver qualquer quantidade de instâncias dessa classe (a menos, é claro, que seja uma classe abstrata e, portanto, não tenha qualquer instância direta, apesar de poder haver qualquer quantidade de instâncias de suas classes-filha). Em alguns casos, porém, você desejará restringir a quantidade de instâncias que uma classe poderá ter. Com muita frequência, você desejará especificar zero instância (nesse caso, trata-se de uma classe utilitária que exibe somente as operações e os atributos do escopo da classe), uma instância (classe unitária), um número específico de instâncias ou muitas instâncias (o caso padrão).

► As instâncias são examinadas no [Capítulo 13](#).

O número de instâncias que uma classe pode ter é chamada sua multiplicidade. A multiplicidade é a especificação do intervalo permitido de cardinalidade que uma entidade poderá assumir. Na UML, você pode especificar a multiplicidade de uma classe, escrevendo uma expressão de multiplicidade no canto superior direito do ícone da classe. Por exemplo,

na [Figura 9.6](#), ControladorDaRede é uma classe unitária. De modo semelhante, existem exatamente três instâncias da classe BarraDeControle no sistema.



**Figura 9.6:**  
Multiplicidade

- A multiplicidade se aplica também às associações, conforme é explicado nos [Capítulos 5 e 10](#).

A multiplicidade também se aplica aos atributos. Você pode especificar a multiplicidade de um atributo, escrevendo uma expressão adequada entre colchetes logo após o nome do atributo. Por exemplo, na figura, existem duas ou mais instâncias de `portaDoConsole` na instância de `Controladordarede`.

- Os atributos estão relacionados à semântica da associação, conforme é apresentado no [Capítulo 10](#).

**Nota:** A multiplicidade de uma classe aplica-se a um determinado contexto. Há um contexto implícito para o sistema inteiro no nível superior. O sistema inteiro pode ser considerado como um classificador estruturado.

## ATRIBUTOS

No nível mais abstrato, ao fazer a modelagem das características estruturais de uma classe (ou seja, seus atributos), você simplesmente escreve o nome de cada atributo. Essa informação costuma ser suficiente para que o leitor médio consiga compreender o propósito do modelo. Entretanto, conforme as seções anteriores descrevem, você também pode especificar a visibilidade, o escopo e a multiplicidade de cada atributo. Além disso, você

também poderá especificar o tipo, o valor inicial e a mutabilidade de cada atributo.

- Você também poderá usar estereótipos para designar conjuntos de atributos relacionados, como os atributos de manutenção, conforme é apresentado no [Capítulo 6](#).

Em sua forma plena, a sintaxe de um atributo na UML é a seguinte:

[visibilidade] nome  
[‘:’ tipo] [‘[‘ multiplicidade] ‘]’]  
[‘=’ valor-inicial]  
[string-propriedade { ‘,’ string-propriedade}]

Por exemplo, os seguintes exemplos são declarações de atributos válidas:

}` origin	Nome apenas
`+ origin	Visibilidade e nome
` origin : Point	Nome e tipo
` name : String [0..1]	Nome, tipo e multiplicidade
` origin : Point = (0,0)	Nome, tipo e valor inicial
` id : Integer {readonly}	Nome, tipo e propriedade

A menos que seja especificado o contrário, os atributos sempre podem ser alterados. Você pode usar a propriedade `readonly` para indicar que o valor do atributo não pode ser alterado após a inicialização do objeto. Você desejará usar a propriedade `readonly` principalmente ao fazer a modelagem de atributos constantes ou a serem gravados uma única vez.

**Nota:** A propriedade `readonly` é mapeada para `const` no C++.

## OPERAÇÕES

No nível mais abstrato, ao fazer a modelagem de características comportamentais de uma classe (ou seja, suas operações e sinais), você

simplesmente escreverá o nome de cada operação. Essa informação costuma ser suficiente para que o leitor médio consiga compreender o propósito do seu modelo. Porém, conforme descrevem as seções anteriores, você também pode especificar a visibilidade e o escopo de cada operação. Adicionalmente, você pode especificar os parâmetros, o tipo de retorno, a semântica de concorrência e outras propriedades de cada operação. Coletivamente, o nome de uma operação acompanhado de seus parâmetros (incluindo seu tipo de retorno, se houver) é chamado de assinatura da operação.

- Os sinais são examinados no [Capítulo 21](#).

**Nota:** A UML faz uma diferença entre operação e método. Uma operação especifica um serviço que pode ser solicitado por qualquer objeto da classe para afetar o comportamento; um método é a implementação de uma operação. Toda operação não abstrata de uma classe deve ter um método, que fornece um algoritmo executável como um corpo (geralmente designado em alguma linguagem de programação ou texto estruturado). Em uma estrutura de herança, poderá haver muitos métodos para a mesma operação e o polimorfismo seleciona qual método existente na hierarquia é empregado em tempo de execução.

Em sua forma plena, a sintaxe de uma operação na UML é a seguinte:

[visibilidade] nome [‘(‘ lista-de-parâmetros ‘)’] [‘:’ tipo-de-retorno] [string-propriedade] {‘,’ string-propriedade}]

- Você também pode usar estereótipos para designar conjuntos de operações relacionadas, como funções auxiliares, conforme é apresentado no [Capítulo 6](#).

Por exemplo, os seguintes exemplos são declarações de operações válidas:

}` display	Apenas nome
`+ display	Visibilidade e nome
}` set(n : Name, s : String)	Nome e parâmetros
}` getID( ) : Integer	Nome e tipo de retorno

`} restart( ) {guarded}`

Nome e propriedade

Na assinatura de uma operação, poderão ser fornecidos zero ou mais parâmetros, cada um seguindo esta sintaxe:

[direção] nome : tipo [= valor padrão]

A direção poderá ser um dos seguintes valores:

- `} in` Um parâmetro de entrada; poderá não ser modificado.
- `} out` Um parâmetro de saída; poderá ser modificado para comunicar informações a quem fez a chamada.
- `} inout` Um parâmetro de entrada; poderá ser modificado para comunicar informações a quem fez a chamada.

**Nota:** Um parâmetro `out` ou `inout` é equivalente a um parâmetro de retorno e um parâmetro `in`. `Out` e `inout` fornecem compatibilidade com as linguagens de programação mais antigas. Use parâmetros de retorno explícitos em seu lugar.

Além das propriedades `leaf` e `abstract` descritas anteriormente, existem propriedades definidas que você poderá utilizar com as operações.

1. `query` A execução da operação mantém inalterado o estado do sistema. Em outras palavras, a operação é uma função pura, sem efeitos secundários.
2. `sequential` Os autores das chamadas devem coordenar externamente o objeto, de maneira que exista um único fluxo no objeto por vez. Na presença de vários fluxos de controle, a semântica e a integridade do objeto não podem ser garantidas.
3. `guarded` A semântica e a integridade do objeto são garantidas na presença de vários fluxos de controle, devido à definição de uma sequência de todas as chamadas para todas as operações do objeto, que têm a

propriedade `guarded`. Como resultado, exatamente uma única operação por vez pode ser invocada para o objeto, reduzindo isso a uma semântica sequencial.

#### 4. concurrent

A semântica e a integridade do objeto são garantidas na presença de vários fluxos de controle, devido ao tratamento da operação como atômica. Várias chamadas a partir de fluxos concorrentes de controle poderão ocorrer simultaneamente para um objeto em qualquer operação concorrente e todas poderão prosseguir concorrentemente com a semântica correta; as operações concorrentes devem ser designadas para que possam ser executadas corretamente no caso de uma operação sequencial ou com a propriedade `guarded` no mesmo objeto.

#### 5. static

A operação não tem um parâmetro implícito para o objeto-alvo e se comporta como um procedimento tradicional global.

As propriedades de concorrência (`sequential`, `guarded` e `concurrent`) atendem à semântica de concorrência de uma operação, as propriedades que são relevantes somente na presença de objetos ativos, processos ou threads.

► *Objetos ativos, processos e threads são examinados no [Capítulo 23](#).*

## CLASSES TEMPLATE

Um template é um modelo de elemento que pode ser parametrizado. Em linguagens como C++ e Ada, é possível escrever classes template, cada uma definindo uma família de classes (também é possível escrever funções template, cada uma definindo uma família de funções). Um template possui espaços vazios (slots) que podem ser usados para classes, objetos e valores e esses slots servem como parâmetros do template. Um template não pode ser

utilizado diretamente; primeiro é preciso instanciá-lo. A instanciação envolve a vinculação desses parâmetros formais do template a valores reais. No caso de uma classe template, o resultado é uma classe concreta que pode ser empregada da mesma forma que outras classes comuns.

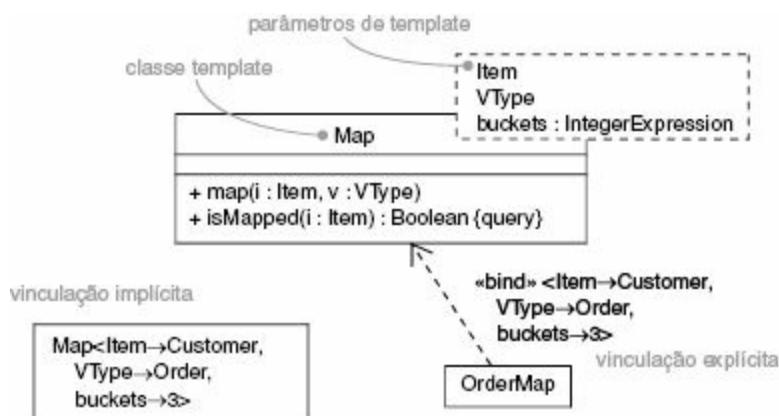
O uso mais comum das classes template é a especificação de recipientes que podem ser instanciados para elementos específicos, tornando-os um tipo seguro. Por exemplo, o seguinte fragmento de código em C++ declara uma classe Map parametrizada.

```
template<class Item, Class Value, int Buckets>class Map
{ public:
    virtual Boolean bind(const Item&, const Value&);
    virtual Boolean isBound(const Item&) const;
    ...
};
```

Você poderia instanciar esse template para mapear objetos Customer para objetos Order.

```
m : Map<Customer, Order, 3>;
```

Você também pode fazer a modelagem de classes template na UML. Conforme mostra a [Figura 9.7](#), uma classe template pode ser representada como qualquer outra classe, mas com uma caixa tracejada adicional no canto superior direito do ícone da classe, listando os parâmetros para o template.



**Figura 9.7:**  
Classes template

- As propriedades básicas das classes são apresentadas no Capítulo 4.

Conforme também mostra a figura, a modelagem da instanciação de uma classe template pode ser feita de duas maneiras. Primeiro, você pode fazê-lo implicitamente, declarando uma classe cujo nome proporciona a vinculação. Segundo, pode fazê-lo explicitamente, utilizando uma dependência estereotipada como `bind`, especificando que a origem instancia o template de destino usando os parâmetros reais.

- As dependências são examinadas nos Capítulos 5 e 10; os estereótipos são apresentados no [Capítulo 6](#).

## ELEMENTOS-PADRÃO

Todos os mecanismos de extensibilidade da UML se aplicam às classes. Com muita frequência, você usará valores atribuídos para ampliar as propriedades das classes (como a especificação da versão de uma classe) e os estereótipos para determinar novos tipos de componentes (como componentes que sejam específicos do modelo).

- Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).

A UML define quatro estereótipos-padrão que se aplicam às classes.

1. Metaclass Especifica um classificador cujos objetos são todas as classes.
2. powertype Especifica um classificador cujos objetos são classes-filha de uma determinada mãe.
3. stereotype Especifica que o classificador é um estereótipo que pode ser aplicado a outros elementos.
4. utility Especifica uma classe cujos atributos e operações pertencem ao escopo de estática.

**Nota:** Vários outros estereótipos-padrão e palavras-chave que se aplicam às classes são examinados em outras seções.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DA SEMÂNTICA DE UMA CLASSE

O propósito mais comum para o uso das classes é a modelagem de abstrações definidas a partir do problema que se está tentando solucionar ou da tecnologia empregada para implementar uma solução para esse problema. Uma vez identificadas essas abstrações, o passo seguinte será especificar sua semântica.

► Os usos básicos das classes são examinados no [Capítulo 4](#).

Na UML, você dispõe de uma ampla variedade de possibilidades de modelagem, desde a mais informal (responsabilidades) até a mais formal (OCL ou Linguagem de Restrição de Objeto). Considerando essas opções, você deve decidir o nível de detalhe que é adequado para comunicar o propósito do seu modelo. Se o propósito do modelo for a comunicação com usuários finais e especialistas no domínio, você tenderá a optar por uma menor formalidade. Se o propósito do modelo for o suporte para um trabalho completo de engenharia, alternando entre modelos e código, você tenderá a optar por uma maior formalidade. Se o propósito do modelo for testar rigorosa e matematicamente os seus modelos e provar sua exatidão, a tendência será para o máximo de formalidade.

► A modelagem é apresentada no [Capítulo 1](#); você também poderá fazer a modelagem da semântica de uma operação, usando um diagrama de atividades, conforme é explicado no [Capítulo 20](#).

**Nota:** Menos formalidade não significa menor precisão. Significa ser menos completo e menos detalhado. Em termos pragmáticos, você procurará um equilíbrio entre o informal e o muito formal. Isso significa oferecer detalhes suficientes para

*dar suporte à criação de artefatos executáveis, mas ainda assim ocultando detalhes para não confundir o leitor de seus modelos.*

Para fazer a modelagem de uma classe, escolha uma das seguintes possibilidades, organizadas do informal para o formal.

› Especifique as responsabilidades da classe. Uma responsabilidade é um contrato ou uma obrigação de um tipo ou classe e é representada em uma nota anexada à classe ou em um compartimento extra no ícone da classe.

► *As responsabilidades são examinadas no Capítulo 4.*

› Especifique a semântica da classe como um todo, usando um texto estruturado, representada em uma nota (estereotipada como semantics) anexada à classe.

› Especifique o corpo de cada método, usando um texto estruturado ou uma linguagem de programação, representado em uma nota anexada à operação por um relacionamento de dependência.

► *A especificação de um método é examinada no Capítulo 3.*

› Especifique as pré e pós-condições de cada operação, acompanhadas das invariantes da classe como um todo, usando um texto estruturado. Esses elementos são representados em notas (estereotipadas como postcondition e invariant) anexadas à operação ou classe por um relacionamento de dependência.

› Especifique uma máquina de estados para a classe. Uma máquina de estados é um comportamento que especifica as sequências de estados percorridos por um objeto durante seu tempo de vida, como uma resposta a eventos, juntamente com suas respostas a esses eventos. Especifique uma estrutura interna da classe.

- » Especifique uma colaboração que represente a classe. Uma colaboração é uma sociedade de papéis e de outros elementos que trabalham em conjunto com a finalidade de proporcionar algum comportamento cooperativo maior do que a soma de todos os elementos. As colaborações têm uma parte estrutural, além de uma parte dinâmica, permitindo que você as utilize para especificar todas as dimensões da semântica de uma classe.
- » Especifique as pré e pós-condições de cada operação, acompanhadas das invariantes da classe como um todo, usando uma linguagem formal como a OCL.
  - ➡ A especificação da semântica de uma operação é examinada no [Capítulo 20](#); as máquinas de estados, no [Capítulo 22](#); as colaborações, no [Capítulo 28](#); a estrutura interna, no [Capítulo 15](#). A OCL é discutida no [Manual de Referência da UML](#).

Em termos pragmáticos, você acabará usando alguma combinação dessas soluções para as diferentes abstrações encontradas em seu sistema.

**Nota:** Ao especificar a semântica de uma classe, lembre-se de que seu propósito é especificar o que a classe faz ou como faz. Especificar a semântica do que a classe faz representa sua perspectiva externa e pública; especificar a semântica de como a classe faz algo representa sua perspectiva interna e privada. Se preferir utilizar uma combinação dessas duas perspectivas, dê ênfase à perspectiva externa para os clientes da classe e à perspectiva interna para aqueles que implementam a classe.

## DICAS E SUGESTÕES

Ao fazer a modelagem de classificadores na UML, lembre-se de que existe uma ampla variedade de blocos de construção à sua disposição, desde interfaces para classes, até componentes e assim por diante. Você deve escolher aquele que for mais adequado à sua abstração. Um classificador bem estruturado:

- » Contém aspectos estruturais e comportamentais.

- › Tem alta coesão e baixo acoplamento.
- › Exibe somente as características necessárias para os clientes usarem a classe, ocultando todas as outras características.
- › Não é ambíguo quanto ao seu propósito e semântica.
- › Não é tão especificado, que exclua todos os graus de liberdade para seus implementadores.
- › Não é pouco especificado, tornando ambíguo o significado do classificador.

Ao criar um classificador na UML:

- › Exiba somente aquelas propriedades do classificador que são importantes para a compreensão da abstração em seu contexto.
- › Escolha uma versão estereotipada capaz de oferecer a melhor indicação visual para o propósito do classificador.

CAPÍTULO

---

**10**

---

# Relacionamentos Avançados

## Neste capítulo

- » *Relacionamentos avançados de dependência, generalização, associação, realização e refinamento*
- » *A modelagem de redes de relacionamentos*
- » *A criação de redes de relacionamentos*

Ao fazer a modelagem dos itens que formam o vocabulário do seu sistema, você também deve modelar a maneira como esses itens estão relacionados uns com os outros. Porém, os relacionamentos podem ser complexos. A visualização, especificação, construção e documentação de redes de relacionamentos requerem algumas características avançadas.

As dependências, generalizações e associações são os três blocos de construção relacionais mais importantes da UML. Esses relacionamentos têm mais propriedades do que aqueles descritos na seção anterior. Você também pode fazer a modelagem de heranças múltiplas, navegação, refinamentos e outras características. Com a utilização de um quarto tipo de relacionamento – a realização – você é capaz de fazer a modelagem da conexão existente entre uma interface e uma classe ou componente, ou entre um caso de uso e uma colaboração. Na UML, você pode fazer a modelagem da semântica dos relacionamentos em qualquer grau de formalidade.

- ➡ As propriedades básicas dos relacionamentos são examinadas no [Capítulo 5](#); as interfaces, no [Capítulo 11](#); os componentes, no [Capítulo 15](#); os casos de uso, no [Capítulo 17](#); e as colaborações

*no Capítulo 28.*

O gerenciamento de redes complexas de relacionamentos requer que você utilize os relacionamentos corretos no nível de detalhe, de maneira que não aplique uma super ou subengenharia ao seu sistema.

## PRIMEIROS PASSOS

Se você estiver construindo uma casa, uma tarefa crítica é decidir onde colocar cada cômodo em relação aos demais. Em um nível de abstração, você poderá decidir colocar o quarto principal no nível principal, longe da frente da casa. A seguir, você poderá imaginar cenários básicos para auxiliá-lo a analisar o uso dessa distribuição de cômodos. Por exemplo, considere o caso de carregar suas compras de mantimentos ao chegar à garagem. Não teria sentido caminhar da garagem e atravessar o quarto para chegar à cozinha; portanto, você rejeitaria essa distribuição.

► *Os casos de uso e os cenários são examinados no Capítulo 17.*

Você poderá formar uma ideia bem mais completa da planta baixa de sua casa considerando esses casos de uso e relacionamentos básicos. Entretanto, isso não será suficiente. Seu projeto poderá conter falhas, se você não levar em consideração relacionamentos mais complexos.

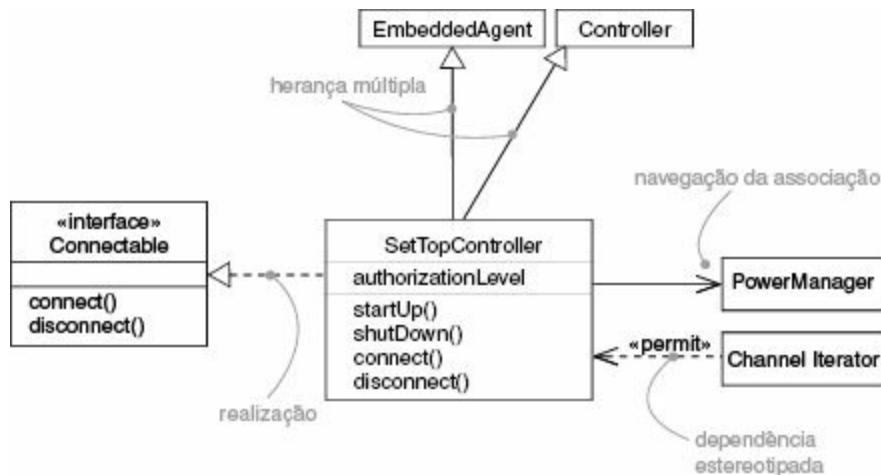
Por exemplo, você poderá gostar da distribuição dos cômodos em cada andar, mas os cômodos em andares diferentes poderão interagir de maneiras inesperadas. Suponha que você coloque o quarto de sua filha adolescente imediatamente acima do seu. Agora, suponha que sua filha adolescente resolva aprender a tocar bateria. É claro que você também rejeitaria essa planta.

De modo semelhante, será necessário levar em consideração como certos mecanismos básicos da casa deverão interagir na planta baixa. Por exemplo, os custos de sua construção aumentarão, se você não distribuir os cômodos de maneira a ter paredes comuns pelas quais passarão canos e tubos.

Ocorre o mesmo quando você constrói um software. Dependências, generalizações e associações são os relacionamentos mais comuns, encontrados na modelagem de sistemas complexos de software. Porém, são necessárias algumas características avançadas desses relacionamentos, com a finalidade de captar os detalhes de muitos sistemas – detalhes importantes a serem considerados, a fim de evitar falhas reais em seu projeto.

- A engenharia direta e a engenharia reversa são examinadas nos Capítulos 8, 14, 18, 19, 20, 24, 30 e 31.

A UML fornece uma representação para várias propriedades avançadas, conforme mostra a Figura 10.1. Essa notação permite que você visualize, especifique, construa e documente redes de relacionamentos em qualquer nível de detalhe desejado, suficiente inclusive para proporcionar suporte à engenharia direta e reversa de modelos e código.



**Figura 10.1:**  
Relacionamentos avançados

## TERMOS E CONCEITOS

Um *relacionamento* é uma conexão entre itens. Na modelagem orientada a objetos, os quatro relacionamentos mais importantes são as dependências, as generalizações, as associações e as realizações. Um relacionamento é

representado graficamente como um caminho, com tipos diferentes de linhas utilizadas para diferenciar os relacionamentos.

## DEPENDÊNCIA

Uma dependência é um relacionamento de utilização, especificando que uma alteração na especificação de um item (por exemplo, a classe SetTopController) poderá afetar outro item que a utilize (por exemplo, a classe ChannelIterator), mas não necessariamente o inverso. Uma dependência é representada graficamente por uma linha tracejada, apontando o item de que o outro depende. Aplique as dependências quando desejar mostrar um item usando outro.

- As propriedades básicas das dependências são examinadas no [Capítulo 5](#).

Um relacionamento de dependência simples e sem adornos é suficiente para a maioria dos relacionamentos de utilização que você encontrará. Entretanto, se quiser especificar algum aspecto do significado, a UML define um número de estereótipos que poderão ser aplicados aos relacionamentos de dependência. Há vários estereótipos, que podem ser organizados em vários grupos.

- Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).

Primeiro, existem estereótipos que se aplicam aos relacionamentos de dependência entre classes e objetos em um diagrama de classes.

- Os diagramas de classes são examinados no [Capítulo 8](#).

- |         |  |
|---------|--|
| 1. bind | Especifica que a origem instancia o template de destino, usando os parâmetros atuais fornecidos. Use a dependência bind quando |
|---------|--|

quiser fazer a modelagem de detalhes das classes template. Por exemplo, o relacionamento entre uma classe repositório de template e uma instanciação dessa classe seria modelado como uma dependência bind. A dependência bind inclui a lista de argumentos atuais que são mapeados para os argumentos formais do template.

► Os templates e as dependências bind são examinados no [Capítulo 9](#).

- |           |   |
|-----------|---|
| 2. derive | Especifica que a origem poderá ser computada a partir do destino. |
|-----------|---|

Você usará derive quando quiser fazer a modelagem do relacionamento entre dois atributos ou duas associações, em que um é concreto e o outro é conceitual. Por exemplo, uma classe Pessoa poderá ter o atributo DataDeNascimento (que é concreto) e também o atributo Idade (que pode ser derivado de DataDeNascimento e, portanto, não é manifestado separadamente na classe). Para exibir o relacionamento entre Idade e DataDeNascimento, você utilizaria uma dependência derive, mostrando Idade derivado de DataDeNascimento.

► Os atributos são examinados nos Capítulos 4 e 9; as associações, no [Capítulo 5](#) e adiante neste capítulo.

- |           |  |
|-----------|--|
| 3. permit | Especifica que a origem recebe visibilidade especial no destino. Use permit quando quiser permitir que uma classe acesse as características privadas de outra classe, como aquelas encontradas em classes friend da linguagem C++. |
|-----------|--|

► As dependências permit são examinadas no [Capítulo 5](#).

- |                |   |
|----------------|---|
| 4. instanceOf  | Especifica que o objeto de origem é uma instância do classificador de destino. Normalmente, apresenta uma notação de texto na forma origem : destino. |
| 5. instantiate | Especifica que a origem cria instâncias do destino.   |

Esses dois últimos estereótipos permitem fazer a modelagem de relacionamentos classe/objeto explicitamente. Use instanceOf quando quiser fazer a modelagem do relacionamento entre uma classe e um objeto no mesmo diagrama ou entre uma classe e sua metaclassse. Use instantiate quando quiser especificar qual elemento cria objetos a partir de outros.

► A dicotomia classe/objeto é examinada no [Capítulo 2](#).

- |              |  |
|--------------|--|
| 6. powertype | Especifica que o alvo é um powertype do destino. Um powertype é um classificador cujos objetos são filhos de um determinado pai. Use powertype quando quiser fazer a modelagem de classes que abrangem outras classes, como aquelas que você encontrará ao fazer a modelagem de bancos de dados. |
|--------------|--|

► A modelagem de bancos de dados lógicos é examinada no Capítulo 8; a modelagem de bancos de dados físicos é apresentada no [Capítulo 30](#).

- |           |   |
|-----------|---|
| 7. refine | Especifica que a origem é um grau mais apurado de abstração do que o destino. Use refine quando quiser fazer a modelagem de |
|-----------|---|

classes que sejam essencialmente as mesmas, mas em níveis diferentes de abstração. Por exemplo, durante a análise, você poderá encontrar uma classe Cliente que, ao longo do projeto, é refinada como uma classe Cliente mais detalhada, completa com sua respectiva implementação.

8. use Especifica que a semântica do elemento de origem depende da semântica da parte pública do destino. Você aplicará use quando quiser marcar explicitamente uma dependência como um relacionamento de uso, em contraste com sombras de dependências fornecidas por outros estereótipos. Continuando, existem dois estereótipos que são aplicados aos pacotes de relacionamentos de dependência.

► Os pacotes são examinados no [Capítulo 12](#).

1. import Especifica que o conteúdo público do pacote de destino fornece o espaço do nome público da origem, como se tivesse sido declarado na origem.
2. access Especifica que o conteúdo público do pacote de destino fornece o espaço do nome privado da origem. Os nomes não qualificados podem ser usados na origem, mas não podem ser exportados novamente.

Use import e access quando quiser usar elementos declarados em outros pacotes. A importação de elementos evita a necessidade de usar um nome

totalmente qualificado para fazer referência a um elemento de outro pacote em uma expressão de texto.

Dois estereótipos são aplicados aos relacionamentos de dependência existentes entre casos de uso:

► *Os casos de uso são examinados no [Capítulo 17](#).*

1. extend                          Especifica que o caso de uso de destino estende o comportamento da origem.

2. include                          Especifica que o caso de uso de origem incorpora explicitamente o comportamento de outro caso de uso em um local especificado pela origem. Use extend e include (e generalização simples) quando quiser decompor casos de uso em partes reutilizáveis. Você encontrará um estereótipo no contexto de interações entre objetos.

► *As interações são examinadas no [Capítulo 16](#).*

► *As máquinas de estados são examinadas no [Capítulo 22](#).*

} send                              Especifica que a classe de origem envia o evento de destino. Use send quando quiser fazer a modelagem de uma operação (como aquela encontrada na ação associada à transição de estados) disparando um determinado evento para um objeto de destino (que, por sua vez, poderá ter uma máquina de estados associada). De fato, a dependência send permite reunir máquinas de estados independentes. Por fim, o estereótipo encontrado no contexto da

organização de elementos do sistema em subsistemas e modelos é o seguinte:

» trace

Especifica que o destino é um antecessor histórico da origem, de um estágio anterior de desenvolvimento.

► *Os modelos e os sistemas são examinados no [Capítulo 32](#).*

Use trace quando quiser fazer a modelagem de relacionamentos entre elementos de modelos diferentes. Por exemplo, no contexto da arquitetura de um sistema, um caso de uso em um modelo de casos de uso (representando um requisito funcional) poderá ser rastreado em um pacote do modelo de projeto correspondente (representando os artefatos que realizam esse caso de uso).

► *As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#).*

**Nota:** Todos os relacionamentos, incluindo a generalização, a associação e a realização, são tipos de dependências. Generalização, associação e realização têm uma semântica suficiente importante, assegurando que sejam tratadas como tipos distintos de relacionamentos na UML. Os estereótipos listados acima representam sombras de dependências, cada um dos quais tem sua própria semântica, mas sem estar semanticamente distantes das dependências comuns, assegurando um tratamento como tipos distintos de relacionamentos. Essa é uma chamada de julgamento por parte da UML, mas a experiência mostra que essa solução permite um equilíbrio entre dar um destaque aos tipos importantes de relacionamentos a serem encontrados sem sobrecarregar o modelador com muitas opções em excesso. Você não estará errado, se, em primeiro lugar, fizer a modelagem de generalizações, associações e realizações e depois considerar todos os outros relacionamentos como tipos de dependências.

## GENERALIZAÇÃO

Uma generalização é um relacionamento entre um item geral (chamado de superclasse ou classe-mãe) e um tipo mais específico desse item (chamado de subclasse ou classe-filha). Por exemplo, você poderá encontrar a classe

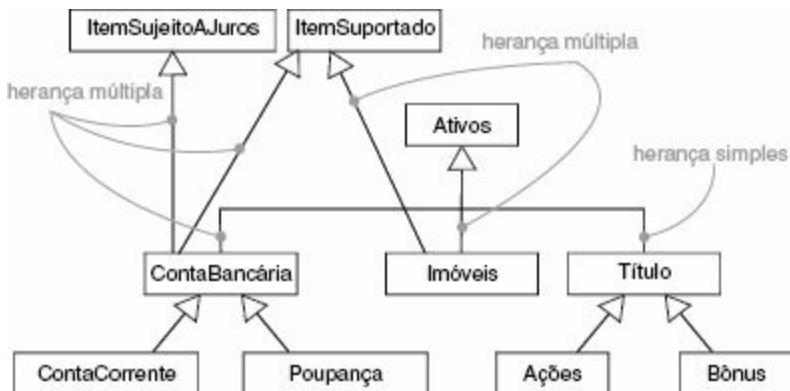
geral Janela e seu tipo mais específico JanelaDupla. Com um relacionamento de generalização da classe-filha para a classe-mãe, a filha (JanelaDupla) herdará toda a estrutura e o comportamento da mãe (Janela). A filha até poderá adicionar uma nova estrutura ou comportamento ou poderá modificar o comportamento da mãe. Em um relacionamento de generalização, as instâncias da filha poderão ser utilizadas em qualquer lugar em que se aplicam instâncias da mãe – significando que a filha pode ser substituída pela mãe.

- As propriedades básicas das generalizações são examinadas no [Capítulo 5](#).

Na maioria dos casos, você considerará suficiente a herança simples. Quando uma classe tem exatamente uma classe-mãe, considera-se que essa classe utiliza uma herança simples. Em alguns casos, porém, a herança múltipla será mais adequada e você também poderá fazer a modelagem desses relacionamentos na UML. Por exemplo, a [Figura 10.2](#) mostra um conjunto de classes estabelecidas em uma aplicação de serviços financeiros. A classe Ativos tem três classes-filha: ContaBancária, Imóveis e Título.

Duas dessas classes-filha (ContaBancária e Título) têm suas próprias filhas. Por exemplo, Ações e Bônus são filhas de Título. Duas dessas classes-filha (ContaBancária e Imóveis) têm herança de várias classes-mãe. Imóveis, por exemplo, é um tipo de Ativos, como também é um tipo de ItemSuportado, e ContaBancária é um tipo de Ativos, como também é um tipo de ItemSujeitoAJuros e um ItemSuportado.

Algumas superclasses só são usadas para adicionar comportamento (normalmente) e estrutura (ocasionalmente) a classes que herdam sua estrutura principal de superclasses comuns. Essas classes adicionais são chamadas mistas, pois não aparecem sozinhas e são destinadas a serem combinadas com outras superclasses em um relacionamento de herança múltipla. Por exemplo, ItemSuportado e ItemSujeitoAJuros são mistos na [Figura 10.2](#).



**Figura 10.2:**

Herança múltipla

**Nota:** Use a herança múltipla com cautela. Poderá haver problemas se uma classe-filha tiver várias classes-mãe cujas estruturas ou comportamentos se sobreponham. De fato, na maioria dos casos, a herança múltipla poderá ser substituída pela delegação, na qual a classe-filha herdará apenas a partir de uma classe-mãe e, então, usará a agregação para obter a estrutura e o comportamento de mais classes-mãe subordinadas. Por exemplo, em vez de especializar Veículo em VeículoTerrestre, VeículoAquático e VeículoAéreo, em uma dimensão, e em Gasolina, Vento e Músculo, em outra dimensão, deixe Veículo conter meioDePropulsão como uma parte. A principal desvantagem dessa solução é a perda da semântica de substituição com essas classes-mãe subordinadas.

Um relacionamento de generalização simples e sem adornos é suficiente na maioria dos casos de relacionamentos de herança que você encontrará. Entretanto, se você quiser especificar mais claramente o significado, a UML define quatro restrições que poderão ser aplicadas aos relacionamentos de generalização:

- Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).

1. complete

Especifica que todas as filhas na generalização foram especificadas no modelo (apesar de que algumas podem ter sido omitidas no diagrama) e que nenhuma filha adicional é permitida.

2. incomplete
- Especifica que nem todas as filhas na generalização foram especificadas no modelo (ainda que algumas tenham sido omitidas) e que filhas adicionais são permitidas.

A menos que seja determinado o contrário, você pode assumir que todos os diagramas exibem apenas uma visão parcial da estrutura de heranças e, portanto, algumas são omitidas. Porém, essa omissão é diferente de o modelo estar completo. Especificamente, use a restrição complete quando quiser mostrar de modo explícito já ter determinado por completo uma hierarquia no modelo (apesar de nenhum diagrama exibir essa hierarquia); use incomplete para mostrar de modo explícito que ainda não determinou toda a especificação da hierarquia no modelo (ainda que algum diagrama possa exibir tudo o que existe no modelo).

► As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

3. disjoint
- Especifica que os objetos da mãe não poderão ter mais de uma filha como um tipo. Por exemplo, a classe Pessoa pode ser especializada em classes disjuntas Homem e Mulher.
4. overlapping
- Especifica que os objetos da mãe poderão ter mais de uma filha como um tipo. Por exemplo, a classe Veículo pode ser especializada nas subclasses sobrepostas VeículoTerrestre e VeículoAquático (um veículo anfíbio).

Essas duas restrições aplicam-se somente no contexto de heranças múltiplas. Você poderá usar disjoint para mostrar que as classes em um conjunto são mutuamente incompatíveis; uma subclasse não pode herdar de

mais de uma classe. Você usará Overlapping para indicar que uma classe pode multiplicar a herança de mais de uma classe no conjunto.

- Os tipos e as interfaces são examinados no [Capítulo 11](#); as interações são apresentadas no [Capítulo 16](#).

**Nota:** Na maioria dos casos, um objeto tem um único tipo em tempo de execução; esse é um caso de classificação estática. Se o objeto puder mudar de tipo em tempo de execução, esse é o caso de uma classificação dinâmica. A modelagem da classificação dinâmica é complexa, mas a UML permite utilizar uma combinação de herança múltipla (para mostrar os tipos potenciais de um objeto) e tipos e interações (para mostrar o tipo de um objeto que se modifica em tempo de execução).

## ASSOCIAÇÃO

Uma associação é um relacionamento estrutural, especificando que os objetos de um item estão conectados a objetos de um outro item. Por exemplo, uma classe Biblioteca poderá ter uma associação de um-para-muitos com uma classe Livro, indicando que cada uma das instâncias de Livro pertence a uma instância de Biblioteca. Além disso, considerado um Livro, você pode encontrar a Biblioteca a que ele pertence e, considerada uma Biblioteca, você pode navegar por todos os seus Livros. Uma associação é representada graficamente por linhas sólidas, conectando a mesma classe ou classes diferentes. Use as associações quando quiser exibir relacionamentos estruturais.

- As propriedades básicas das associações são examinadas no [Capítulo 5](#).

Existem quatro tipos básicos de adornos que são aplicados às associações: um nome, o papel em cada extremidade da associação, a multiplicidade em cada extremidade da associação e a agregação. Para usos avançados, existem algumas outras propriedades que podem ser utilizadas para a modelagem de detalhes sutis, como navegação, qualificação e vários tipos de agregação.

**Navegação** Considerando uma associação simples e sem adornos entre duas classes, como Livro e Biblioteca, é possível navegar de objetos de um tipo até objetos de outro tipo. A menos que seja especificado o contrário, a navegação por uma associação é bidirecional. Entretanto, em algumas situações, você desejará limitar a navegação a uma única direção. Por exemplo, conforme mostra a [Figura 10.3](#), ao fazer a modelagem dos serviços de um sistema operacional, você encontrará uma associação entre objetos Usuário e Senha. Considerando um Usuário, você desejará ser capaz de encontrar os objetos Senha correspondentes; mas, considerando uma Senha, não desejará identificar o Usuário correspondente. Para representar explicitamente a direção da navegação, você poderá incluir adornos na associação com setas apontando a direção a ser seguida.



**Figura 10.3:**  
Navegação

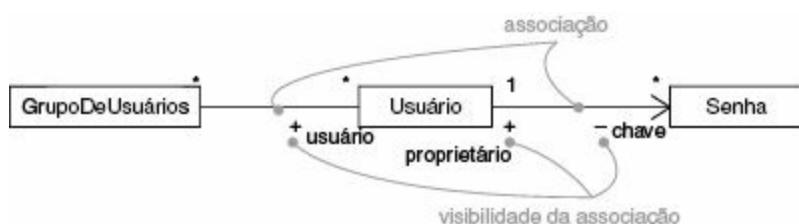
**Nota:** A especificação da direção a ser seguida não significa necessariamente que não será possível navegar dos objetos encontrados em uma extremidade da associação até os objetos existentes na outra extremidade. Em vez disso, a navegação é uma declaração da eficiência da direção a ser seguida. Por exemplo, na figura anterior, ainda será possível encontrar os objetos Usuário associados a Senha por meio de outras associações que envolvem outras classes ainda não apresentadas. Especificar que uma associação é navegável é uma declaração de que, considerando um objeto em uma extremidade, você poderá chegar direta e facilmente a objetos na outra extremidade, em geral porque o objeto de origem armazena algumas referências aos objetos de destino.

**Visibilidade** Considerando uma associação entre duas classes, os objetos de uma classe podem visualizar e navegar até objetos da outra classe, a menos que haja uma restrição em contrário, definida por uma declaração explícita de navegação. Entretanto, em determinadas situações você desejará

limitar a visibilidade em uma associação, relativa a objetos externos à associação. Por exemplo, conforme mostra a Figura 10.4, existe uma associação entre GrupoDeUsuários e Usuário e uma outra entre Usuário e Senha. Considerando um objeto Usuário, é possível identificar seus objetos Senha correspondentes. Porém, uma Senha é privada a um Usuário e, portanto, não está acessível externamente (a menos, é claro, que o Usuário exponha de modo explícito o acesso à Senha, talvez por meio de alguma operação pública). Consequentemente, conforme mostra a figura, dado um objeto GrupoDeUsuários, você pode navegar até seus objetos Usuário (e vice-versa), mas não pode visualizar os objetos Senha do objeto Usuário. Na UML, podem ser especificados três níveis de visibilidade para uma associação, da mesma forma que para as características de uma classe, anexando um símbolo de visibilidade ao nome de um papel. A menos que seja observado o contrário, a visibilidade de um papel é pública. A visibilidade privada indica que os objetos existentes nessa extremidade não estão acessíveis a qualquer objeto externo à associação; a visibilidade protegida indica que os objetos encontrados nessa extremidade não estão acessíveis a qualquer objeto externo à associação, com exceção dos filhos existentes na outra extremidade.

A visibilidade do pacote indica que as classes declaradas no mesmo pacote podem ver o elemento especificado; isso não se aplica a extremidades de associação.

- ➡ A visibilidade pública, protegida e privada é examinada no [Capítulo 9](#).



**Figura 10.4:**  
Visibilidade

**Qualificação** No contexto de uma associação, um dos idiomas mais comuns que você encontrará na modelagem é um problema de busca. Considerado um objeto em uma extremidade da associação, como identificar um objeto ou um conjunto de objetos existente na outra extremidade? Por exemplo, considere o problema de fazer a modelagem de uma bancada em uma fábrica para a qual os itens devolvidos são processados para serem consertados. Conforme mostra a [Figura 10.5](#), seria feita a modelagem de uma associação entre duas classes, Bancada e ItemDevolvido. No contexto de Bancada, haveria um códigoDoServiço que identificaria determinado ItemDevolvido. Nesse sentido, códigoDoServiço é um atributo da associação. Não é uma característica de ItemDevolvido, pois os itens realmente não têm conhecimento de coisas como consertos ou serviços. Assim, considerando um objeto de Bancada e um determinado valor de códigoDoServiço, é possível navegar até zero ou um objetos do ItemDevolvido. Na UML, a modelagem desse idioma seria feita com a utilização de um qualificador, que é um atributo de associação cujos valores particionam o conjunto de objetos relacionados a um objeto da associação. O qualificador é representado como um pequeno retângulo anexo à extremidade da associação, contendo os atributos, conforme mostra a figura. O objeto de origem, juntamente com os valores dos atributos do qualificador, gera um objeto de destino (no caso de a multiplicidade do destino ser limitada a um) ou um conjunto de objetos (se a multiplicidade do destino for igual a muitos).



**Figura 10.5:**  
Qualificação

- ➡ Os atributos são examinados nos [Capítulos 4 e 9](#).

**Composição** Uma agregação passa a ser um conceito simples com uma semântica bem profunda. A agregação simples é inteiramente conceitual e nada faz além de diferenciar o “todo” da “parte”. A agregação simples não

modifica o significado da navegação pela associação entre o todo e suas partes, nem vincula o tempo de vida do todo e suas partes.

- A agregação simples é examinada no [Capítulo 5](#).

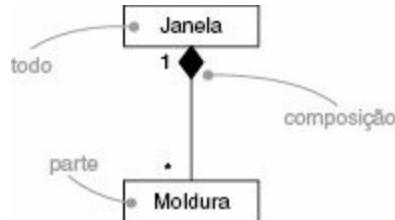
Porém, existe uma variação da agregação simples – a composição – que adiciona alguma semântica importante. A composição é uma forma de agregação, com propriedade bem definida e tempo de vida coincidente como parte do todo. As partes sem multiplicidade fixada podem ser criadas após a composição, mas, uma vez criadas, vivem e morrem com ela. Essas partes também podem ser removidas explicitamente antes da morte do objeto composto.

- Um atributo é essencialmente uma abreviação para a composição; os atributos são examinados nos [Capítulos 4 e 9](#).

Isso significa que, em uma agregação composta, um objeto poderá ser uma parte de somente uma composição em determinado momento. Por exemplo, em um sistema de janelas, uma Moldura pertence exatamente a uma única Janela. Isso está em contraste com uma agregação simples, em que uma parte pode ser compartilhada por vários todos. Por exemplo, no modelo de uma casa, uma Parede pode ser parte de um ou mais objetos Cômodo.

Além disso, em uma agregação composta, o todo é responsável pela disposição de suas partes, significando que o objeto composto deve gerenciar a criação e a destruição de suas partes. Por exemplo, ao criar uma Moldura em um sistema de janelas, você deve anexá-la a uma Janela que a conterá. Da mesma maneira, quando você destrói a Janela, o objeto Janela, por sua vez, deve destruir sua parte Moldura.

Conforme mostra a [Figura 10.6](#), a composição é realmente apenas um tipo especial de associação e é especificada por um adorno adicionado a uma associação pura: um diamante preenchido na extremidade referente ao todo.



**Figura 10.6:**  
Composição

**Nota:** Como uma alternativa, você pode exibir a composição, usando uma classe estruturada e aninhando os símbolos das partes no símbolo da composição. Essa forma é mais útil quando quiser dar ênfase aos relacionamentos entre as partes que se aplicam somente no contexto do todo.

► A estrutura interna é examinada no [Capítulo 15](#).

**Classes de associação** Em uma associação entre duas classes, a própria associação poderá ter propriedades. Por exemplo, em um relacionamento empregador/ empregado entre uma Empresa e uma Pessoa, existe um Serviço que representa as propriedades desse relacionamento que se aplicam a exatamente um único par de Pessoa e Empresa. Não seria adequado fazer a modelagem dessa situação com uma associação de Empresa a Serviço, juntamente com uma associação de Serviço a Pessoa. Isso não ligaria uma determinada instância de Serviço ao par específico de Empresa e Pessoa.

► Os atributos são examinados nos [Capítulos 4 e 9](#).

Na UML, você faria a modelagem desse caso como uma classe de associação, que é um elemento da modelagem com propriedades de associação e de classe. Uma classe de associação pode ser vista como uma associação que também tem propriedades de classe ou como uma classe que também tem propriedades de associação. As classes de associação são representadas com um símbolo de classe anexado a uma associação por uma linha tracejada, conforme mostra a [Figura 10.7](#).



**Figura 10.7:**  
Classes de associação

**Nota:** De vez em quando, você desejará as mesmas propriedades para várias classes de associação diferentes. Porém, não é possível anexar uma classe de associação a mais de uma associação, pois a classe é a própria associação. Para conseguir esse efeito, defina uma classe (`C`) e depois crie cada classe de associação que precisar dessas características herdadas de `C` ou use `C` como o tipo de um atributo.

**Restrições** Essas propriedades simples e avançadas das associações são suficientes para a maioria dos relacionamentos estruturais que você encontrará. Entretanto, se você quiser especificar mais claramente o significado, a UML define cinco restrições a serem aplicadas aos relacionamentos de associações.

- Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).

Primeiro, você pode especificar se os objetos encontrados em uma extremidade da associação (com multiplicidade maior que um) estão ordenados ou não.

- |            |  |
|------------|--|
| 1. ordered | Especifica que o conjunto de objetos em uma das extremidades da associação se encontra em uma ordem explícita. |
|------------|--|

Por exemplo, em uma associação Usuário/Senha, a Senha associada ao Usuário poderá ser mantida na ordem mais recentemente utilizada e seria marcada

como ordered. Se essa senha estiver ausente, os objetos ficam desordenados. Segundo, você pode especificar se os objetos encontrados em uma extremidade da associação são únicos – ou seja, formam um conjunto – ou se não são únicos – ou seja, não formam um conjunto.

2. set	Os objetos são únicos sem duplicatas.
3. bag	Os objetos não são únicos, podem ter duplicatas.
4. ordered set	Os objetos são únicos e ordenados.
5. list of sequence	Os objetos são ordenados, podem ser duplicados. Por fim, existe uma restrição da possibilidade de modificação das instâncias de uma associação.
6. readonly	Um vínculo adicionado a partir de um objeto existente na extremidade oposta da associação não pode ser modificado ou excluído. O padrão na ausência dessa restrição é a possibilidade de modificação ilimitada.

- *Essas propriedades de modificação também são aplicadas aos atributos, conforme é examinado no [Capítulo 9](#); os vínculos são apresentados no [Capítulo 16](#).*

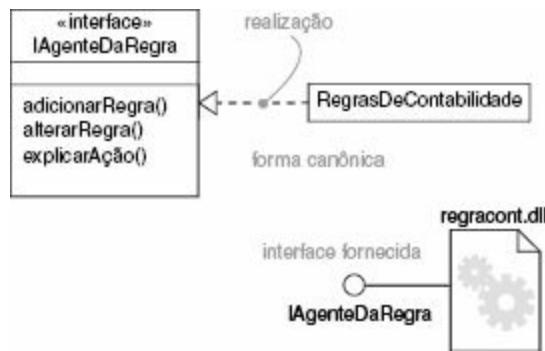
**Nota:** Para ser preciso, ordered e readonly são propriedades de uma extremidade da associação. Porém, são representadas pela mesma notação das restrições.

Realização Uma *realização* é um relacionamento semântico entre classificadores, no qual um dos classificadores especifica um contrato cujo cumprimento é assegurado pelo outro classificador. Uma realização é representada graficamente por uma linha tracejada com uma grande seta vazia apontando para o classificador que especifica o contrato.

A realização é tão diferente dos relacionamentos de dependência, generalização e associação, que é tratada como um tipo separado de relacionamento. Semanticamente, a realização é algo entre a dependência e a generalização e sua notação é uma combinação das notações desses dois tipos de relacionamentos. Use a realização em duas circunstâncias: no contexto de interfaces e no contexto de colaborações.

Na maioria dos casos, a realização será empregada para especificar um relacionamento entre uma interface e a classe ou o componente que proporciona uma operação ou serviço para a interface. Uma interface é uma coleção de operações utilizadas para especificar um serviço de uma classe ou de um componente. Portanto, uma interface especifica um contrato que a classe ou o componente deve executar. Uma interface poderá ser realizada por muitas dessas classes ou componentes e uma classe ou componente poderá realizar muitas interfaces. Talvez o aspecto mais interessante sobre as interfaces seja o fato de permitirem separar a especificação de um contrato (a própria interface) de sua implementação (por uma classe ou um componente). Assim, as interfaces se estendem sobre as partes físicas e lógicas da arquitetura de um sistema. Por exemplo, conforme mostra a [Figura 10.8](#), uma classe (como `RegrasDeContabilidade` em um sistema de entrada de pedidos), na visão de projeto de um sistema, poderá realizar uma determinada interface (como `IAgenteDaRegra`). Essa mesma interface (`IAgenteDaRegra`) também poderá ser realizada por um componente (como `regracont.dll`) na visão de implementação do sistema. Observe que é possível representar a realização de duas maneiras: na forma canônica (usando o estereótipo `interface` é a linha tracejada com uma grande seta vazia) e em uma forma oculta (usando a notação de pirulito das interfaces).

- As *interfaces* são examinadas no [Capítulo 11](#); as *classes*, nos [Capítulos 4 e 9](#); os *componentes*, no [Capítulo 15](#); as cinco visões de uma arquitetura, no [Capítulo 2](#).

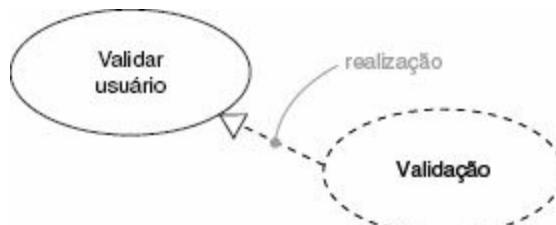


**Figura 10.8:**

Realização de uma interface

Você também usará a realização para especificar o relacionamento entre um caso de uso e a colaboração que realiza esse caso de uso, conforme mostra a [Figura 10.9](#). Nessas circunstâncias, quase sempre serão utilizadas as formas canônicas da realização.

- Os casos de uso são examinados no [Capítulo 17](#); as colaborações são apresentadas no [Capítulo 28](#).



**Figura 10.9:**

Realização de um caso de uso

**Nota:** Quando uma classe ou um componente realiza uma interface, isso significa que os clientes podem confiar na classe ou no componente em relação a uma fiel execução do comportamento especificado pela interface. Isso implica que a classe ou o componente implementa todas as operações da interface, responde a todos os seus sinais e segue, de todas as formas, o protocolo estabelecido pela interface para os clientes que utilizam essas operações ou enviam esses sinais.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE REDES DE RELACIONAMENTOS

Ao fazer a modelagem do vocabulário de um sistema complexo, você poderá encontrar dezenas, se não centenas ou milhares, de classes, interfaces, componentes, nós e casos de uso. É difícil estabelecer um claro limite entre essas abstrações. Estabelecer os inúmeros relacionamentos entre essas abstrações é ainda mais difícil: isso requer que você forme uma distribuição equilibrada de responsabilidades no sistema como um todo, com abstrações individuais altamente coesivas e com relacionamentos expressivos, apesar de fracamente acoplados.

- » *A modelagem do vocabulário de um sistema e a modelagem da distribuição de responsabilidades em um sistema são examinadas no [Capítulo 4](#).*

Ao fazer a modelagem dessas redes de relacionamentos:

- » Não comece de maneira isolada. Aplique casos de uso e cenários para orientar a descoberta de relacionamentos entre um conjunto de abstrações.
- » Casos de uso são discutidos no [Capítulo 17](#).
- » Em geral, comece fazendo a modelagem dos relacionamentos estruturais que estão presentes. Esses relacionamentos refletem a visão estática do sistema e, portanto, são mais tangíveis.
- » A seguir, identifique oportunidades de relacionamentos de generalização/especialização; use heranças múltiplas de modo econômico.
- » Somente após concluir os passos anteriores, você deverá procurar as dependências; elas costumam representar formas mais sutis de conexão semântica.
- » Para cada tipo de relacionamento, comece com a forma básica e aplique as características avançadas apenas quando for absolutamente necessário para expressar seu propósito.

- » Lembre-se de que é indesejável e desnecessário fazer a modelagem de todos os relacionamentos entre um conjunto de abstrações em um mesmo diagrama ou visão. Em vez disso, construa os relacionamentos do seu sistema, considerando diferentes visões sobre o sistema. Destaque conjuntos interessantes de relacionamentos em diagramas individuais.
- » As cinco visões de uma arquitetura são discutidas no Capítulo 2; o Rational Unified Process é resumido no Apêndice B.

O segredo para uma boa modelagem de redes complexas de relacionamentos é fazê-la de maneira incremental. Elabore os relacionamentos à medida que os incluir à estrutura da arquitetura do sistema. Simplifique esses relacionamentos, à medida que descobrir oportunidades para mecanismos comuns. Em cada versão do processo de desenvolvimento, avalie os relacionamentos entre as principais abstrações existentes no sistema.

**Nota:** Na prática – e principalmente se você estiver seguindo um processo de desenvolvimento iterativo e incremental – os relacionamentos existentes em seu modelo serão derivados de decisões explícitas pelo modelador, como também da engenharia reversa de sua implementação.

## DICAS E SUGESTÕES

Ao fazer a modelagem de relacionamentos avançados em UML, lembre-se de que existe uma ampla variedade de blocos de construção à sua disposição, desde associações simples até propriedades mais detalhadas de navegação, qualificação, agregação e assim por diante. Você deve escolher o relacionamento e os detalhes do relacionamento mais adequados à sua abstração. Um relacionamento bem estruturado:

- » Expõe apenas aquelas características necessárias para que os clientes utilizem o relacionamento e oculte todos os demais.
- » Não contém ambiguidades em seu propósito e semântica.

- › Não é superespecificado, que elimine todos os graus de liberdade para os seus implementadores.
- › Não é subespecificado, tornando ambíguo o significado do relacionamento.

Ao estabelecer um relacionamento na UML:

- › Mostre somente aquelas propriedades do relacionamento que são importantes para a compreensão da abstração em seu contexto.
- › Escolha uma versão estereotipada que proporcione a melhor indicação visual para o propósito do relacionamento.

CAPÍTULO

---

**11**

---

# Interfaces, Tipos e Funções

## Neste capítulo

- » *Interfaces, tipos, funções (ou papéis) e realização*
- » *Modelagem das costuras de um sistema*
- » *Modelagem de tipos estáticos e dinâmicos*
- » *Criação de interfaces compreensíveis e acessíveis*

**A**s interfaces definem uma linha entre a especificação do que uma abstração realiza e a implementação do modo como isso é realizado pela abstração. Uma interface é uma coleção de operações utilizadas para especificar um serviço de uma classe ou de um componente.

As interfaces são empregadas para visualizar, especificar, construir e documentar a coesão interna do sistema. Os tipos e as funções (ou papéis) fornecem um mecanismo para você fazer a modelagem da conformação estática e dinâmica da interface em um contexto específico.

Uma interface bem estruturada fornece uma clara separação entre a visão externa e a visão interna de uma abstração, tornando possível que se compreenda e se tenha acesso a uma abstração sem ser preciso se aprofundar nos detalhes de sua implementação.

## PRIMEIROS PASSOS

Não haveria sentido em projetar uma casa que exigisse quebrar a fundação sempre que fosse necessário pintar as paredes. De maneira semelhante, ninguém desejaria morar em um local no qual seria necessário mudar a fiação sempre que uma lâmpada fosse substituída. O proprietário de

um prédio não gostaria de mover portas ou substituir pontos de luz e de telefone sempre que entrasse um novo inquilino.

► *Como definir o projeto de uma casa é examinado no [Capítulo 1](#).*

Centenas de experiências em construção proporcionaram informações de caráter pragmático sobre essa atividade, auxiliando os construtores a evitar esses problemas óbvios – e outros não tão óbvios – que surgem quando uma edificação cresce e é modificada ao longo do tempo. Em termos de software, cuidamos dessa estruturação com uma clara separação de conceitos. Por exemplo, em um prédio bem estruturado, a frente ou fachada da estrutura pode ser modificada ou substituída sem afetar o restante da construção. De modo semelhante, os móveis encontrados dentro do prédio poderão ser movidos facilmente, sem qualquer alteração da infraestrutura. Os serviços que atravessam paredes para instalações elétricas, de aquecimento, de água e esgoto podem ser alterados com uma pequena obra, sem ser preciso modificar a estrutura da construção.

Não só os métodos-padrão de construção auxiliam a erguer prédios que podem ser ampliados com o passar do tempo, mas também existem muitas interfaces-padrão para as quais você pode construir, permitindo a utilização de componentes comuns predefinidos, de grande ajuda para auxiliar a reduzir os custos de construção e de manutenção. Por exemplo, existem tamanhos-padrão de madeira que facilitam a construção de paredes e que sejam múltiplos de um tamanho comum. Existem tamanhos padrão para portas e janelas. Existem padrões até para tomadas elétricas e para conexões de linhas telefônicas (embora possam variar de um país para outro) tornando mais fácil combinar diferentes equipamentos eletrônicos.

Em relação ao software, é importante construir sistemas com uma clara separação de conceitos, de maneira que, à medida que o sistema evoluir, quaisquer modificações de uma parte não afetarão outras partes do sistema. Uma forma importante de alcançar esse grau de separação consiste em especificar costuras claras no sistema, estabelecendo uma linha entre aquelas

partes que poderão ser modificadas de maneira independente. Além disso, escolhendo as interfaces corretas, você pode selecionar componentes-padrão, bibliotecas e frameworks para a implementação dessas interfaces, sem precisar construí-las. À medida que descobrir implementações melhores, você pode substituir as antigas sem perturbar seus usuários.

- Frameworks são examinados no [Capítulo 29](#).

Na UML, as interfaces são utilizadas para a modelagem das costuras do sistema. Uma interface é uma coleção de operações empregadas para especificar o serviço de uma classe ou componente. Declarando a interface, você pode estabelecer o comportamento desejado de uma abstração independente de qualquer implementação dessa interface. Os clientes podem construir em contraposição a essa interface e você pode construir ou comprar qualquer implementação da interface, desde que a implementação satisfaça às responsabilidades e ao contrato denotado pela interface.

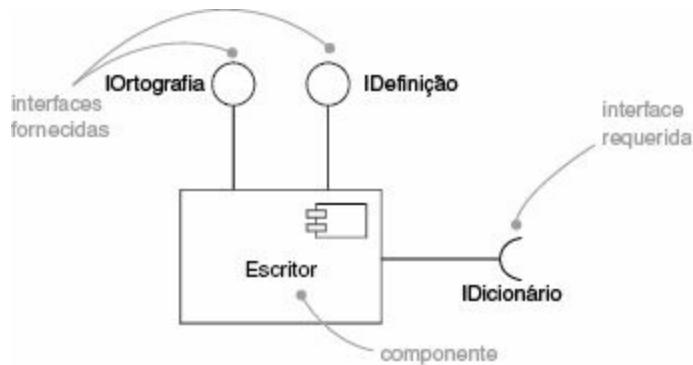
- As classes são examinadas nos [Capítulos 4 e 9](#); os componentes são apresentados no [Capítulo 15](#).

Muitas linguagens de programação têm suporte para o conceito de interfaces, inclusive Java e CORBA IDL. As interfaces não são importantes apenas para dividir a especificação e a implementação de uma classe ou componente, mas, à medida que você passar a trabalhar com sistemas maiores, poderá usar interfaces para especificar a visão externa de um pacote ou subsistema.

- Os pacotes são examinados no [Capítulo 12](#); os subsistemas são apresentados no [Capítulo 32](#).

A UML oferece uma representação gráfica para as interfaces, conforme mostra a [Figura 11.1](#). Essa notação permite visualizar a especificação de uma abstração sem qualquer implementação.

► Os componentes são examinados no [Capítulo 15](#).



**Figura 11.1:**  
Interfaces

## TERMOS E CONCEITOS

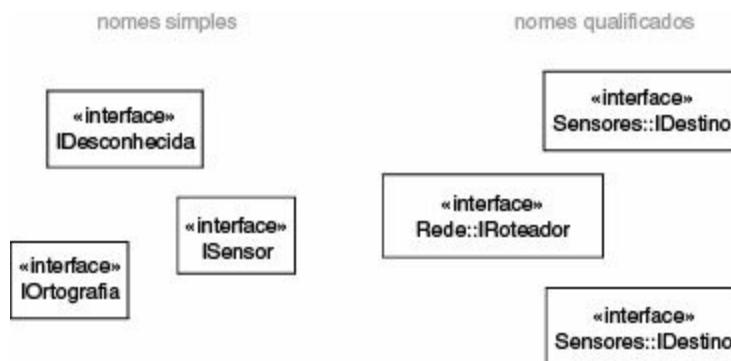
Uma *interface* é uma coleção de operações utilizadas para especificar um serviço de uma classe ou componente. Um *tipo* é um estereótipo de uma classe, utilizado para especificar um domínio de objetos, juntamente com as operações (mas não os métodos) aplicáveis a um objeto. Uma função (ou papel) é o comportamento de uma entidade que participa de um determinado contexto. Uma interface pode ser representada como uma classe estereotipada para expor suas operações e outras propriedades. Para mostrar o relacionamento entre uma classe e suas interfaces, é fornecida uma notação especial. Uma determinada interface (uma que represente os serviços fornecidos pela classe) é exibida como um pequeno círculo anexado à caixa de classes. Uma interface requerida (uma que represente os serviços requeridos por uma classe de outra classe) é exibida como um pequeno semicírculo anexado à caixa de classes.

**Nota:** As interfaces também podem ser empregadas com a finalidade de especificar contratos para casos de uso ou para subsistemas.

## NOMES

Toda interface deve ter um nome que a diferencie das demais interfaces. Um *nome* é uma sequência de caracteres textual. O nome sozinho é conhecido como um *nome simples*; o *nome do caminho* é o nome da interface, tendo como prefixo o nome do pacote em que a interface está armazenada. Uma interface poderá ser incluída, mostrando-se apenas o respectivo nome, conforme mostra a [Figura 11.2](#):

- *O nome de uma interface deve ser único no pacote que a contém, conforme é explicado no [Capítulo 12](#).*



**Figura 11.2:**  
Nomes simples e de caminho

**Nota:** Um nome de interface poderá ser um texto formado por qualquer quantidade de letras, números e determinados sinais de pontuação (exceto o sinal de dois-pontos, utilizado para separar o nome da interface e o nome do pacote que a contém) e poderá se estender por várias linhas. Na prática, os nomes das interfaces são curtos ou expressões nominais definidas a partir do vocabulário do sistema que está sendo modelado.

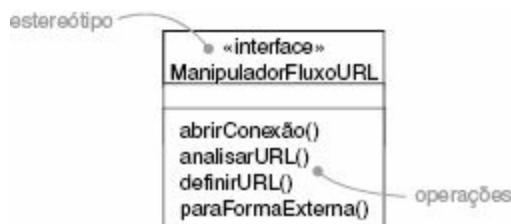
## OPERAÇÕES

Uma interface é uma coleção de operações empregadas para especificar um serviço de uma classe ou de um componente. Ao contrário das classes e dos tipos, as interfaces não especificam qualquer implementação (portanto, poderão não incluir métodos, os quais fornecem a implementação de uma operação). Assim como as classes, uma interface poderá ter qualquer número de operações. Essas operações poderão receber adornos com propriedades de

visibilidade, propriedades de concorrência, estereótipos, valores atribuídos e restrições.

- As operações são examinadas nos Capítulos 4 e 9; os mecanismos de extensibilidade são apresentados no Capítulo 6.

Ao declarar uma interface, você a representa como uma classe estereotipada, listando suas operações no comportamento apropriado. As operações poderão ser representadas mostrando apenas seus nomes ou poderão ser expandidas de modo a serem exibidas suas assinaturas completas e outras propriedades, conforme mostra a [Figura 11.3](#).



**Figura 11.3:**  
Operações

**Nota:** Também é possível associar sinais a uma interface.

- Os eventos são examinados no Capítulo 21.

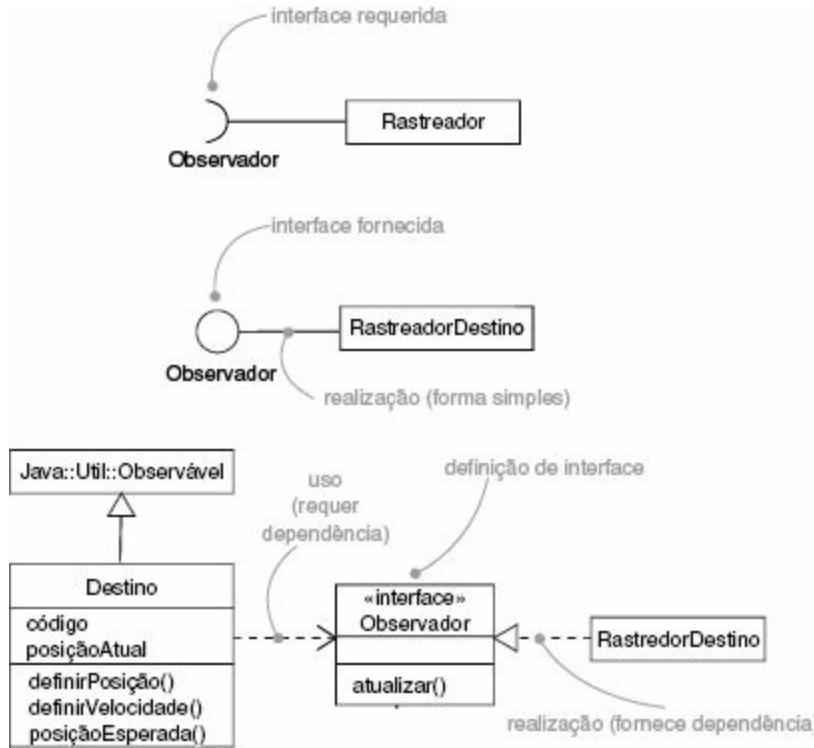
## RELACIONAMENTOS

Assim como as classes, as interfaces podem participar de relacionamentos de generalização, associação e dependência. Além disso, uma interface pode participar de relacionamentos de realização. A realização é um relacionamento semântico entre dois classificadores, em que um classificador especifica um contrato cuja execução é assegurada pelo outro classificador.

- Os relacionamentos são examinados nos Capítulos 5 e 10.

Uma interface especifica o contrato para uma classe ou componente sem determinar sua implementação. Uma classe ou componente poderá realizar várias interfaces. Nesse caso, a classe ou o componente se compromete a executar fielmente todos esses contratos, o que significa que é fornecido um conjunto de métodos capazes de implementar adequadamente as operações definidas na interface. O conjunto de serviços que se combina fornecer é a *interface fornecida*. De modo semelhante, uma classe ou um componente poderão depender de várias interfaces. Nesse caso, espera-se que esses contratos sejam cumpridos por algum conjunto de componentes que os executarão. O conjunto de serviços que uma classe requer de outras classes é a *interface requerida*. É por isso que se diz que uma interface representa uma costura no sistema. Uma interface especifica um contrato, e o cliente e o fornecedor, em cada lado do contrato, poderão fazer alterações de modo independente, de forma que cada um deles continue a atender às suas obrigações em relação ao contrato.

Conforme mostra a [Figura 11.4](#), você pode mostrar que um elemento realiza uma interface de duas maneiras. Em primeiro lugar, pode utilizar a forma simples, na qual a interface e seu relacionamento de realização são representados como um pirulito que se estende até um dos lados de uma classe ou componente. Essa forma é útil quando você apenas desejar expor as costuras existentes em seu sistema. Entretanto, a limitação desse estilo se deve ao fato de você não poder visualizar diretamente as operações ou sinais fornecidos pela interface. Em segundo lugar, você pode usar a forma expandida, em que a interface é representada como uma classe estereotipada, permitindo uma visualização de suas operações e outras propriedades, e depois definir um relacionamento de realização do classificador ou do componente para a interface. Na UML, um relacionamento de realização é representado como uma linha tracejada com uma grande seta aberta apontando a interface. Essa notação é um cruzamento entre a generalização e a dependência.



**Figura 11.4:**  
Realizações

**Nota:** As interfaces são semelhantes às classes abstratas. Por exemplo, não podem ter instâncias diretas. Entretanto, uma classe abstrata pode implementar suas operações concretas. Uma interface é mais como uma classe abstrata na qual todas as operações também são abstratas.

- As classes abstratas são examinadas no [Capítulo 4](#); os componentes são apresentados no [Capítulo 15](#).

## COMPREENDENDO UMA INTERFACE

Ao analisar uma interface recebida, a primeira coisa que você verá será um conjunto de operações que especifica um serviço de uma classe ou componente. Observe com mais cuidado e você verá a assinatura completa dessas operações, juntamente com suas propriedades especiais, como visibilidade, escopo e semântica de concorrência.

- ➡ As operações e suas propriedades são examinadas no Capítulo 9; a semântica de concorrência é apresentada no Capítulo 24.

Essas propriedades são importantes, mas, para interfaces complexas, não são suficientes para auxiliá-lo a compreender a semântica do serviço que representam, muito menos conhecer como utilizar essas operações de modo apropriado. Na ausência de quaisquer outras informações, será necessário considerar alguma abstração que realize a interface para compreender o que cada operação faz e como essas operações devem funcionar em conjunto. Porém, esse procedimento não está de acordo com o propósito de uma interface, que é fornecer uma clara separação de conceitos em um sistema.

Na UML, você pode fornecer muito mais informações a uma interface, com a finalidade de facilitar sua compreensão e utilização. Em primeiro lugar, você poderá anexar pré e pós-condições a cada operação e invariantes à classe ou componente como um todo. Fazendo isso, um cliente que precise utilizar uma interface será capaz de compreender o que a interface faz e como usá-la, sem precisar compreender sua implementação. Caso você precise ser rigoroso, pode empregar a OCL da UML para especificar formalmente a semântica. Em segundo lugar, você pode anexar uma máquina de estados à interface. Essa máquina de estados pode ser utilizada para especificar a solicitação legal parcial das operações de uma interface. Use as colaborações para especificar o comportamento esperado em relação à interface por intermédio de uma série de diagramas de interação.

- ➡ Precondições, pós-condições e invariantes são examinadas no [Capítulo 9](#); as máquinas de estado, no [Capítulo 22](#); as colaborações, no [Capítulo 28](#); a OCL, no [Capítulo 6](#).

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DA COSTURA DE UM SISTEMA

O propósito mais comum para utilização de interfaces é a modelagem da costura de um sistema formado por componentes de software, como Eclipse, .NET ou Java Beans. Você reutilizará componentes de outros sistemas ou adquiridos prontos de terceiros; outros serão criados da estaca zero. Em todos os casos, será necessário escrever um código capaz de reunir esses componentes. Isso exigirá que você compreenda as interfaces fornecidas e requeridas por cada um dos componentes.

- Os componentes são examinados no [Capítulo 15](#); os sistemas são apresentados no [Capítulo 32](#).

Identificar as costuras de um sistema envolve a identificação de linhas claras de demarcação em sua arquitetura. Em cada um dos lados dessas linhas, você encontrará componentes que poderão se modificar de maneira independente, sem afetar os componentes existentes no outro lado, desde que os componentes nos dois lados estejam em conformidade com o contrato especificado por essa interface.

Ao reutilizar componentes de outros sistemas ou adquiridos prontos, provavelmente você terá um conjunto de operações com alguma documentação mínima sobre o significado de cada uma delas. Isso é útil, mas não é suficiente. É mais importante compreender a ordem em que cada operação deverá ser chamada e quais são os mecanismos embutidos que a interface contém. Infelizmente, no caso de um componente com pouca documentação, o melhor a ser feito é construir, por tentativa e erro, um modelo conceitual referente ao modo como a interface funciona. Assim você pode documentar seu entendimento por meio da modelagem dessa costura no sistema, usando interfaces em UML, de maneira que, posteriormente, você e outras pessoas sejam capazes de ter acesso mais facilmente a esse componente. Da mesma forma, ao criar seu próprio componente, você precisará compreender seu contexto, o que significa especificar as interfaces de que ele depende para realizar sua tarefa, como também as interfaces que ele apresenta ao mundo e que poderão ser utilizadas por outros objetos.

► Padrões e frameworks são examinados no [Capítulo 29](#).

**Nota:** Em sua maioria, os sistemas de componentes, como Eclipse, .NET e Enterprise Java Beans, proporcionam uma introspecção de componentes, significando que é possível, de um modo programático, fazer consultas a uma interface com a finalidade de determinar suas operações. Esse é o primeiro passo para compreender a natureza de qualquer componente cuja documentação seja insuficiente.

Para fazer a modelagem da costura de um sistema:

› Na coleção de classes e componentes de seu sistema, desenhe uma linha em volta daqueles que tendem a estar altamente acoplados a outros conjuntos de classes e componentes.

► As colaborações são examinadas no [Capítulo 28](#).

› Aprimore seu agrupamento, levando em consideração o impacto da modificação. As classes ou os componentes que tendem a ser alterados juntos deverão ser agrupados como colaborações.

› Considere as operações e os sinais que ultrapassam essas fronteiras, desde instâncias de um único conjunto de classes ou componentes até instâncias de outros conjuntos de classes e componentes.

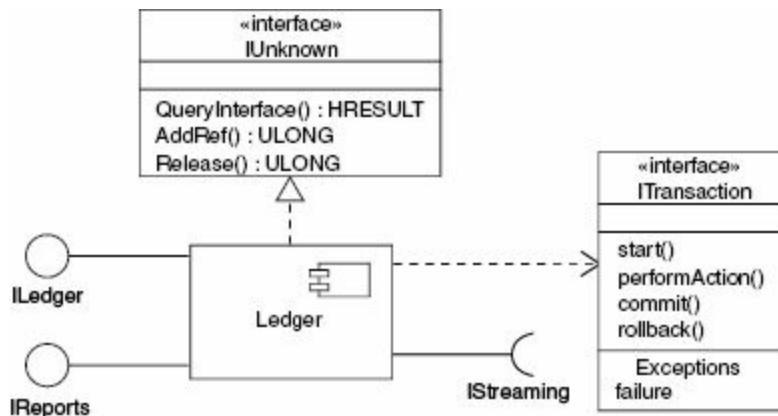
› Crie pacotes de conjuntos logicamente relacionados dessas operações e sinais como interfaces.

› Para cada uma dessas colaborações existentes no sistema, identifique as interfaces de que elas dependem (importam) e aquelas que fornecem (exportam) para outros objetos. Faça a modelagem da importação de interfaces usando os relacionamentos de dependência, e a modelagem da exportação de interfaces usando os relacionamentos de realização.

› Para cada uma dessas interfaces existentes no sistema, documente sua dinâmica, usando pré e pós-condições para cada operação e casos de uso e máquinas de estados para a interface como um todo.

► A modelagem comportamental é examinada nas Partes 4 e 5.

Por exemplo, a [Figura 11.5](#) mostra a costura ao redor de um componente (a biblioteca ledger.dll), definida em um sistema financeiro. Esse componente realiza três interfaces: IUnknown, ILedger e IReports. Nesse diagrama, a interface IUnknown é mostrada em sua forma expandida; as outras duas interfaces aparecem em suas formas simples, representadas como pirulitos. Essas três interfaces são realizadas por Ledger e são exportadas para outros componentes que as utilizam.



**Figura 11.5:**

A modelagem das costuras de um sistema

Conforme esse diagrama também mostra, Ledger importa duas interfaces, **IStreaming** e **ITransaction**, a segunda sendo apresentada em sua forma expandida. Essas duas interfaces são necessárias para a operação adequada do componente **Ledger**. Portanto, no sistema em execução, você deverá fornecer os componentes que realizam essas duas interfaces. Identificando interfaces como **ITransaction**, você estará efetivamente desacoplando os componentes de cada lado da interface, permitindo a utilização de qualquer componente em conformidade com essa interface.

As interfaces como **ITransaction** são mais do que apenas uma pilha de operações. Essa interface particular faz algumas pressuposições sobre a ordem em que as operações deverão ser chamadas. Apesar de não estar

mostrado aqui, pode-se anexar casos de uso a essa interface e enumerar as formas básicas de sua utilização.

- » *Os casos de uso são examinados no [Capítulo 17](#).*

## TIPOS ESTÁTICOS E DINÂMICOS DE MODELAGEM

Em sua maioria, as linguagens de programação orientadas a objetos têm tipos estáticos, significando que o tipo de um objeto está vinculado no momento em que é criado. Dessa maneira, esse objeto provavelmente desempenhará papéis diferentes ao longo do tempo. Isso significa que os clientes que usam esse objeto interagem com ele por meio de conjuntos diferentes de interfaces, representando conjuntos de operações de interesse e que possivelmente se sobreponham.

- » *As instâncias são examinadas no [Capítulo 13](#).*

A modelagem da natureza estática de um objeto pode ser visualizada em um diagrama de classes. Entretanto, quando se faz a modelagem de algo como objetos de negócios, que naturalmente mudam de papel ao longo do fluxo de trabalho, às vezes é útil fazer explicitamente a modelagem da natureza dinâmica desse tipo de objeto. Nessas circunstâncias, um objeto pode ganhar e receber tipos ao longo de sua vida. Você também pode modelar o ciclo de vida do objeto usando uma máquina de estados.

- » *Os diagramas de classes são examinados no [Capítulo 8](#).*

Para fazer a modelagem de um tipo dinâmico:

- » Especifique os diferentes tipos desse objeto, representando cada tipo como uma classe (se a abstração exigir estrutura e comportamento) ou como uma interface (se a abstração exigir apenas comportamento).

Faça a modelagem de todos os papéis que a classe do objeto poderá ter em qualquer momento. Você pode marcá-los com o estereótipo «dinâmico».

(Esse não é um estereótipo predefinido da UML, mas um que você pode adicionar.)

- ➡ As associações e as generalizações são examinadas nos Capítulo 5 e 10; os diagramas de interação, no Capítulo 19; as dependências, nos Capítulos 5 e 10.
- » Em um diagrama de interação, represente cada instância da classe com tipos definidos dinamicamente. Exiba o papel da instância entre ângulos (« ») acima do nome do objeto, exatamente como um estado. (Estamos usando a sintaxe da UML de uma nova maneira, que sentimos ser consistente com o objetivo dos estados.)

Por exemplo, a Figura 11.6 mostra os papéis que as instâncias da classe Pessoa poderão desempenhar no contexto de um sistema de recursos humanos.



Figura 11.6:

A modelagem de tipos estáticos

Esse diagrama especifica que as instâncias da classe Pessoa poderão ser de três tipos – chamados Candidato, Funcionário ou Aposentado.

## DICAS E SUGESTÕES

Ao fazer a modelagem de uma interface na UML, lembre-se de que toda interface deverá representar uma costura do sistema, separando a especificação da implementação. Uma interface bem estruturada:

- » É simples, mas completa, fornecendo todas as operações necessárias, mas suficientes, para especificar um único serviço.

- » É comprehensível, proporcionando informações suficientes para a utilização e realização da interface sem ser preciso examinar uma utilização ou implementação existentes.
- » É acessível, oferecendo informações para guiar o usuário em suas principais propriedades, sem sobrecarregá-lo com os detalhes de uma pilha de operações.

Ao definir uma interface na UML:

- » Use a notação de pirulito ou soquete sempre que for necessário apenas especificar a presença de uma costura do sistema. Na maior parte do tempo, isso será necessário para os componentes, e não para as classes.
- » Use a forma expandida quando for necessário visualizar os detalhes do próprio serviço. Na maior parte do tempo, isso será necessário para especificar as costuras em um sistema anexado a um pacote ou subsistema.

CAPÍTULO

---

12

---

# Pacotes

## Neste capítulo

- › *Pacotes, visibilidade, importação e exportação*
- › *Modelagem de grupos de elementos*
- › *Modelagem de visões da arquitetura*
- › *Aumentando a escala para sistemas grandes*

**V**isualizar, especificar, construir e documentar sistemas grandes envolve a manipulação de quantidades potencialmente grandes de classes, interfaces, componentes, nós, diagramas e outros elementos. À medida que você passar para sistemas como esses, perceberá a necessidade de organizar esses itens em grupos maiores. Na UML, o pacote é um mecanismo de propósito geral para a organização de elementos da modelagem em grupos.

Você usa os pacotes para organizar seus elementos de modelagem em conjuntos maiores que possam ser manipulados como grupos. Você é capaz de controlar a visibilidade desses elementos, permitindo que alguns itens fiquem visíveis fora do pacote, enquanto outros ficarão ocultos. Os pacotes também podem ser empregados para apresentar diferentes visões da arquitetura de seu sistema.

Os pacotes bem estruturados agrupam elementos que estão próximos semanticamente e que tendem a se modificar em conjunto. Portanto, os pacotes bem estruturados são fracamente acoplados e muito coesos, com acesso altamente controlado ao conteúdo do pacote.

## PRIMEIROS PASSOS

As casas de cachorro não são complexas: existem quatro paredes, uma delas com uma passagem do tamanho do cachorro, e um teto. Ao construir uma casa de cachorro, você realmente precisa apenas de uma pequena pilha de madeira. Não há muito mais do que isso para ser estruturado.

- As diferenças entre construir uma casa de cachorro e um prédio com vários andares são examinadas no Capítulo 1.

Casas são mais complexas. Paredes, tetos e pisos são reunidos em abstrações maiores chamadas cômodos. Até esses cômodos são organizados em conjuntos maiores: a área de estar, a área de diversão e assim por diante. Esses grupos maiores poderão não descrever algo diretamente relacionado com a própria casa física, consistindo apenas em nomes que atribuímos a cômodos logicamente relacionados da casa e que aplicamos quando falamos a respeito de como utilizaremos a casa.

Prédios com vários andares são muito complexos. Não existem apenas estruturas elementares, como paredes, tetos e pisos, mas também existem componentes maiores, como áreas públicas, úteis e de trabalho. Provavelmente esses componentes são agrupados em componentes ainda maiores, como espaços a serem alugados e a área de serviço do prédio. Esses componentes maiores poderão não ter qualquer relação com o prédio final, sendo apenas artefatos que utilizamos para organizar nossos planos em relação ao prédio.

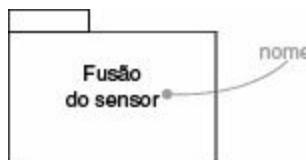
Todo grande sistema é erguido dessa maneira. De fato, a única maneira de compreender um sistema complexo é reunir suas abstrações em grupos cada vez maiores. A maioria dos agrupamentos de tamanho modesto (como um quarto) são, por seu próprio direito, abstrações de algum tipo de classe para a qual existem muitas instâncias. Grande parte dos agrupamentos maiores são puramente conceituais (como áreas úteis) e não são instâncias reais. Nunca se manifestam no sistema físico, mas, ao contrário, existem somente para atender ao propósito de compreender o próprio sistema. Este último tipo de

agrupamento não tem uma identidade no sistema implantado; sua identidade está restrita ao modelo do sistema.

Na UML, os agrupamentos que organizam um modelo são chamados de pacotes. Um pacote é um mecanismo de propósito geral para organizar elementos em grupos. Os pacotes ajudam a organizar os elementos em modelos, de maneira que você seja capaz de compreendê-los com maior facilidade. Os pacotes também permitem controlar o acesso a seus conteúdos, de modo que você possa controlar as costuras existentes na arquitetura do sistema.

- A *arquitetura de um software* é examinada no *Capítulo 2*; a *modelagem da arquitetura de um sistema* é apresentada no [\*Capítulo 32\*](#).

A UML oferece uma representação gráfica para os pacotes, conforme mostra a [\*Figura 12.1\*](#). Essa notação permite a visualização de grupos de elementos que podem ser manipulados como um todo e de uma forma que permite controlar a respectiva visibilidade e o acesso aos elementos individuais.



**Figura 12.1:**

Pacotes

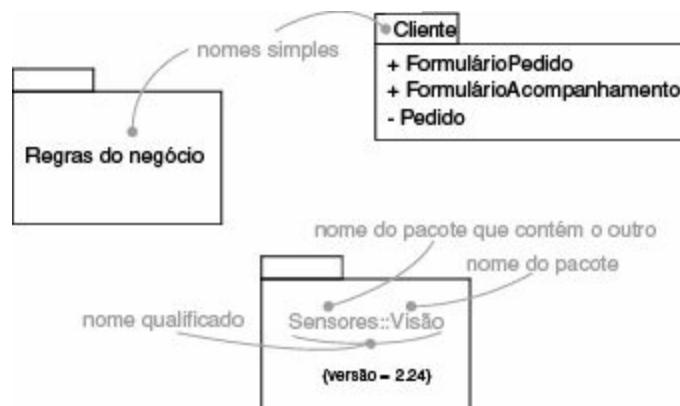
## TERMOS E CONCEITOS

Um *pacote* é um mecanismo de propósito geral para a organização do próprio modelo em uma hierarquia; ele não tem nenhum significado para a execução. Um pacote é representado graficamente como uma pasta com uma guia. O nome do pacote encontra-se na pasta (se o conteúdo não for exibido) ou na guia (se o conteúdo da pasta for exibido).

## NOMES

Todo pacote deve ter um nome que o diferencie de outros pacotes. Um *nome* é uma sequência de caracteres textual. Esse nome sozinho é conhecido como *nome simples*; um *nome qualificado* é um nome de pacote que apresenta como prefixo o nome do pacote que o contém, se houver. Os dois-pontos duplos (::) separam os nomes dos pacotes. Tipicamente os pacotes são definidos exibindo somente seu nome, conforme mostra a [Figura 12.2](#). Assim como acontece com as classes, você poderá estabelecer pacotes com adornos como valores atribuídos ou comportamentos adicionais para expor seus detalhes.

- *O nome do pacote deve ser único em relação ao pacote que o contém.*



**Figura 12.2:**

Nomes de pacotes simples e qualificados

**Nota:** O nome de um pacote poderá ser um texto formado por qualquer quantidade de letras, números e certos sinais de pontuação (com exceção de sinais como os dois-pontos, utilizados para separar o nome do pacote e o nome do outro pacote que o contém) e poderá se estender por várias linhas. Na prática, os nomes dos pacotes são expressões ou substantivos curtos de grupos definidos a partir do vocabulário do modelo.

## ELEMENTOS PERTINENTES

Um pacote pode conter outros elementos, incluindo classes, interfaces, componentes, nós, colaborações, casos de uso, diagramas e até outros pacotes. A propriedade de elementos é um relacionamento composto, significando que os elementos são declarados no pacote. Se o pacote for destruído, os elementos serão destruídos. Cada elemento pertence, de maneira única, a exatamente um pacote.

► *A composição é examinada no [Capítulo 10](#).*

**Nota:** *O pacote contém os elementos do modelo declarados nele. Pode incluir elementos, como classes, associações, generalizações, dependências e notas. Ele não contém elementos que são meramente referenciados em um pacote.*

O pacote forma um espaço de nome, significando que os elementos do mesmo tipo precisam ser declarados de maneira única no contexto do pacote que os contém. Por exemplo, não poderá haver duas classes com o nome Fila pertencentes ao mesmo pacote, mas poderá haver uma classe com o nome Fila no pacote P1 e uma outra (diferente) chamada Fila no pacote P2. As classes P1::Fila e P2::Fila são, de fato, classes distintas e podem ser diferenciadas pelos respectivos nomes de caminho. Tipos diferentes de elementos podem ter o mesmo nome.

**Nota:** *Se possível, é melhor evitar nomes duplicados em diferentes pacotes, para não haver confusão.*

Elementos de tipos diferentes poderão ter o mesmo nome em um pacote. Portanto, pode haver uma classe chamada Cronômetro, assim como um componente também com o nome Cronômetro no mesmo pacote. Na prática, entretanto, para evitar confusões, é melhor nomear os elementos de maneira única em relação a todos os tipos existentes no pacote.

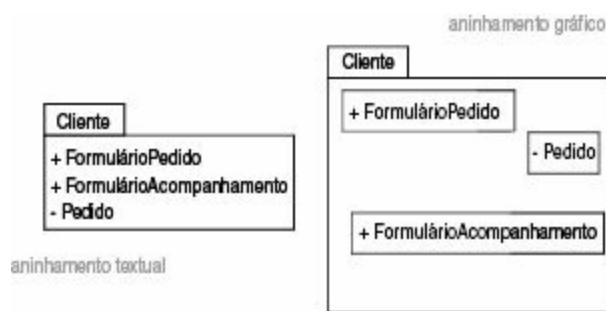
Os pacotes podem conter outros pacotes. Isso significa que é possível decompor os modelos de modo hierarquizado. Por exemplo, você poderá ter uma classe chamada Câmera, pertencente ao pacote Visão, que, por sua vez, pertence ao pacote Sensores. O nome completo dessa classe será

Sensores::Visão::Câmera. Na prática, é melhor evitar pacotes com vários níveis de aninhamento. Dois ou três níveis de aninhamento constituem o limite do que será conveniente utilizar. Mais do que o aninhamento, você usará a importação para a organização de seus pacotes.

► *A importação é examinada em uma seção adiante.*

Essa semântica de propriedade faz com que os pacotes sejam um mecanismo importante para lidar com escala. Sem os pacotes, você acabaria com grandes modelos planos, em que todos os elementos precisariam receber nomes únicos – uma situação impossível de ser administrada, principalmente quando existem classes e outros elementos desenvolvidos por várias equipes. Os pacotes ajudam a controlar os elementos que compõem o sistema, à medida que eles evoluem em taxas distintas ao longo do tempo.

Conforme mostra a [Figura 12.3](#), é possível mostrar explicitamente o conteúdo de um pacote de uma maneira textual ou gráfica. Observe que, ao exibir esses elementos pertinentes, o nome do pacote é colocado na guia. Na prática, dificilmente você desejará mostrar o conteúdo de pacotes dessa maneira. Em vez disso, você utilizará ferramentas para examinar o conteúdo do pacote.



**Figura 12.3:**  
Elementos pertinentes

**Nota:** A UML pressupõe a existência de um pacote-raiz anônimo em um modelo. Como consequência disso, os elementos de cada tipo no nível superior do modelo devem ser declarados de maneira única.

## VISIBILIDADE

Você poderá controlar a visibilidade dos elementos pertencentes a um pacote da mesma forma como é capaz de controlar a visibilidade dos atributos e operações pertencentes a uma classe. Tipicamente, um elemento pertencente a um pacote é público. Isso significa que o elemento está visível em relação ao conteúdo de qualquer pacote que faça a importação do pacote que contém o elemento. De modo inverso, os elementos protegidos somente podem ser vistos por elementos-filho e os elementos-privado não podem ser vistos fora do pacote em que são declarados. Na [Figura 12.3](#), o formulário de pedidos FormulárioPedido é uma parte pública do pacote Cliente e o pedido Pedido é uma parte privada. Qualquer pacote que importe Cliente é capaz de ver FormulárioPedido, mas não Pedido. Ao ser visualizado externamente, o nome inteiramente qualificado de FormulárioPedido seria Cliente::FormulárioPedido.

► A visibilidade é examinada no [Capítulo 9](#).

Para especificar a visibilidade de um elemento pertencente a um pacote, use como prefixo o nome do elemento com o apropriado símbolo de visibilidade. Os elementos públicos são representados com o símbolo + como prefixo de seus nomes, como no caso de FormulárioPedido na [Figura 12.3](#). Coletivamente, as partes públicas de um pacote constituem a interface desse pacote.

Assim como ocorre com as classes, pode-se designar um elemento como protegido ou privado, representado pela inclusão, como prefixo do nome do elemento, do símbolo # ou do símbolo -, respectivamente. Os elementos protegidos podem ser vistos somente pelos pacotes que herdam de um outro pacote; os elementos privados não podem ser vistos de forma alguma fora do pacote.

A visibilidade do pacote indica que uma classe é visível para outras classes declaradas no mesmo pacote, mas é invisível para classes declaradas em outros pacotes. Mostre a visibilidade do pacote com um prefixo ~ no nome da classe.

## IMPORTAÇÃO E EXPORTAÇÃO

Suponha que você tenha duas classes chamadas A e B posicionadas lado a lado. Por serem iguais, A pode ver B e B pode ver A; portanto, uma pode depender da outra. Quando apenas duas classes formam um sistema trivial, realmente não é necessário qualquer tipo de pacote.

Agora, imagine haver algumas centenas dessas classes, posicionadas lado a lado. Não existirá um limite para a complexa rede de relacionamentos que você pode desenvolver. Além disso, não é possível compreender um grupo de classes tão grande e desorganizado. Esse é um problema muito sério para sistemas grandes – o acesso simples e sem restrições não cresce em escala. Para essas situações, é necessário algum tipo de pacote controlado com a finalidade de organizar as abstrações. Suponha que, ao contrário, você colocou A em um pacote e B em outro pacote, estando os dois posicionados lado a lado.

Suponha também que A e B sejam declarados como partes públicas de seus respectivos pacotes. Essa é uma situação bastante diferente. Apesar de A e B serem ambas classes públicas, uma não pode ter acesso à outra, pois os pacotes que as contêm formam uma parede opaca. Porém, se o pacote de A importar o pacote de B, A é capaz de ver B, embora B não possa ver A. A importação assegura uma permissão unilateral para que os elementos de um pacote tenham acesso aos elementos pertencentes ao outro pacote. Na UML, a modelagem de um relacionamento de importação é feita como uma dependência assinalada pelo estereótipo importar como adorno. Formando pacotes em que as abstrações são reunidas em agrupamentos significativos e depois controlando seu acesso por meio da importação, você é capaz de controlar a complexidade de uma grande quantidade de abstrações.

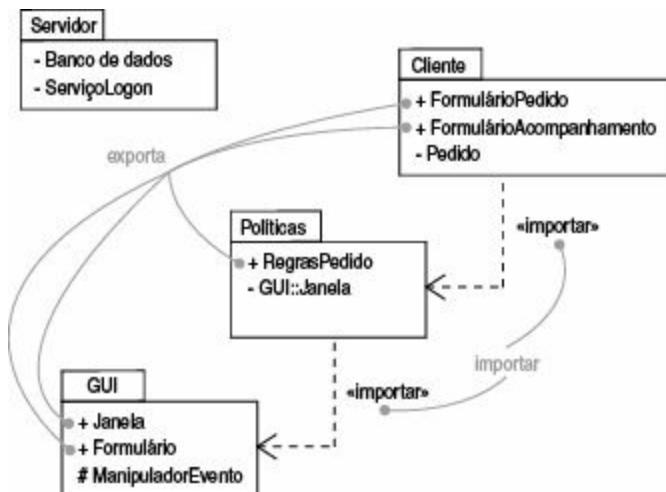
- ➡ Os relacionamentos de dependência são examinados no [Capítulo 5](#); os mecanismos de extensibilidade são apresentados no [Capítulo 6](#).

**Nota:** Na verdade, dois estereótipos se aplicam a esse caso – importar e acessar – e ambos especificam que o pacote de origem tem acesso ao conteúdo do pacote de destino. O estereótipo importar adiciona o conteúdo do destino ao espaço de nome da

origem e, portanto, não é preciso qualificar seus nomes. Isso admite a possibilidade de conflitos entre nomes, que você deve evitar para manter a boa formação do modelo. O estereótipo `acessar` não adiciona o conteúdo do destino e, portanto, é preciso qualificar seus nomes. Na maior parte do tempo, você utilizará importar.

As partes públicas de um pacote são chamadas suas exportações. Por exemplo, na [Figura 12.4](#), o pacote GUI exporta duas classes, Janela e Formulário. ManipuladorEvento não é exportado por GUI; ManipuladorEvento é uma parte protegida do pacote.

► As interfaces são examinadas no [Capítulo 11](#).



**Figura 12.4:**

Importação e exportação

As partes exportadas por um pacote são visíveis para o conteúdo desses pacotes que têm visibilidade para o pacote. Nesse exemplo, Políticas importa o pacote GUI explicitamente. GUI::Janela e GUI::Formulário, portanto, ficam visíveis para o conteúdo do pacote Políticas, usando seus nomes simples, Janela e Formulário. Porém, GUI::ManipuladorEvento não é visível por ser protegido. Como o pacote Servidor não importa GUI, o conteúdo de Servidor pode acessar qualquer conteúdo de GUI, mas deve usar nomes qualificados para fazer isso, por exemplo, GUI::JANELA. De maneira semelhante, o conteúdo de GUI não tem permissão para acessar qualquer conteúdo de Servidor, porque ele é privado; eles são inacessíveis mesmo usando nomes qualificados.

As dependências de importação e de acesso não são transitivas. Nesse exemplo, Cliente importa Políticas e Políticas importa GUI; portanto, Cliente importa GUI transitivamente. Assim, o conteúdo de Cliente tem acesso às exportações de Políticas, bem como tem acesso às exportações de GUI. Se Políticas acessa GUI, em vez de importá-lo, Cliente não adiciona os elementos de GUI ao seu nome de espaço, mas ele ainda pode referenciá-los usando nomes qualificados (como GUI::JANELA).

**Nota:** *Se um elemento é visível em um pacote, ele está visível em todos os pacotes aninhados existentes naquele pacote. Os pacotes aninhados podem ver tudo que os pacotes que os contêm são capazes de ver. Um nome em um pacote aninhado pode ocultar um nome em um pacote que o contém. Nesse caso, é necessário um nome qualificado para referenciá-lo.*

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE GRUPOS DE ELEMENTOS

O propósito mais comum para o qual os pacotes serão utilizados é organizar os elementos da modelagem em grupos que você pode nomear e manipular como um conjunto. Caso esteja desenvolvendo uma aplicação trivial, não será necessário usar pacotes, pois todas as suas abstrações caberão adequadamente em um único pacote. Para qualquer outro sistema, entretanto, você perceberá que muitas de suas classes, interfaces, componentes, nós e até diagramas tendem naturalmente a formar grupos. A modelagem desses grupos é feita como pacotes.

Existe uma importante distinção entre classes e pacotes: as classes são abstrações de itens encontradas no problema ou na solução; os pacotes são mecanismos empregados para organizar os itens no modelo. Os pacotes ficam invisíveis no sistema em execução; eles são estritamente mecanismos para organizar o projeto.

Na maior parte do tempo, você utilizará os pacotes para agrupar o mesmo tipo básico de elementos. Por exemplo, a partir da visão de projeto do sistema, você poderá separar todas as classes e seus correspondentes

relacionamentos em uma série de pacotes, utilizando as dependências de importação da UML para o controle de acesso entre esses pacotes. Todos os componentes da visão de implementação do sistema poderão ser organizados de uma maneira semelhante.

► As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#).

Os pacotes também podem ser utilizados para agrupar diferentes tipos de elementos. Por exemplo, no caso de um sistema que está sendo desenvolvido por uma equipe distribuída geograficamente, você poderá empregar os pacotes como unidade de gerenciamento de configuração, colocando neles todas as classes e diagramas que cada equipe pode examinar separadamente. De fato, é comum o uso de pacotes para agrupar os elementos da modelagem e os diagramas a eles associados.

Para fazer a modelagem de grupos de elementos:

- » Examine os elementos da modelagem em uma determinada visão da arquitetura e procure porções definidas por elementos próximos uns dos outros, conceitual ou semanticamente.
- » Reúna cada uma dessas porções em um pacote.
- » Para cada pacote, diferencie quais elementos deverão estar acessíveis externamente ao pacote. Marque-os como públicos e todos os demais como protegidos ou privados. Em caso de dúvida, oculte o elemento.
- » Conecte explicitamente os pacotes construídos a partir de outros por meio de dependências de importação.
- » No caso de famílias de pacotes, conecte os pacotes especializados à sua parte mais geral usando generalizações.

Por exemplo, a [Figura 12.5](#) mostra um conjunto de pacotes organizando, em uma clássica arquitetura de três camadas, as classes da visão de projeto de um sistema de informações. Os elementos do pacote Serviços do Usuário fornecem a interface visual para apresentar e receber dados. Os elementos

encontrados no pacote Serviços de Dados mantêm, acessam e atualizam os dados. Os elementos do pacote Serviços do Negócio constituem a ponte que une os elementos dos outros dois pacotes e abrangem todas as classes e outros elementos que gerenciam solicitações feitas pelo usuário em relação à execução das tarefas do negócio, incluindo as regras do negócio que determinam as políticas para manipulação de dados.



**Figura 12.5:**

A modelagem de grupos de elementos

Em um sistema trivial, todas as abstrações poderiam ser reunidas em um único pacote. Porém, ao organizar suas classes e outros elementos da visão de projeto do sistema em três pacotes, você não só torna mais compreensível o seu modelo, mas é capaz de controlar o acesso aos elementos desse modelo, ocultando alguns e exportando outros.

**Nota:** Ao representar modelos como esse, tipicamente você desejará expor elementos centrais de cada pacote. Para tornar claro o propósito de cada pacote, você também pode expor um valor atribuído da documentação para cada pacote.

► O valor atribuído da documentação é examinado no Capítulo 6.

## MODELAGEM DE VISÕES DA ARQUITETURA

É importante a utilização de pacotes para agrupar elementos relacionados; sem fazer isso, não é possível desenvolver modelos complexos. Essa solução proporciona bons resultados para a organização de elementos relacionados, como classes, interfaces, componentes, nós e diagramas. À medida que levar em consideração as diferentes visões da arquitetura de um sistema de software, você precisa de agrupamentos cada vez maiores. Você pode usar os pacotes para fazer a modelagem das visões de uma arquitetura.

- » As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#).

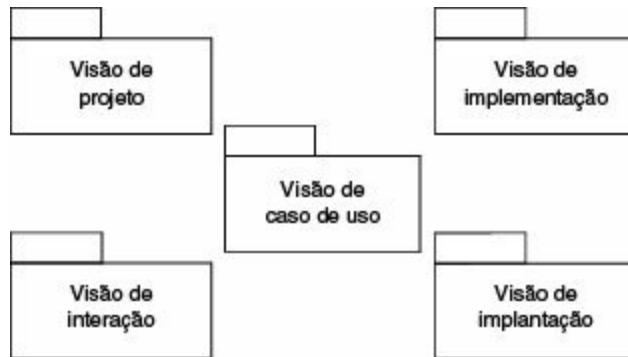
Lembre-se de que uma visão é uma projeção na organização e estrutura de um sistema, focalizada em determinado aspecto desse sistema. Essa definição apresenta duas implicações. Em primeiro lugar, quase sempre é possível decompor um sistema em pacotes ortogonais, cada um dos quais direciona um conjunto de decisões significativas em termos de arquitetura. Por exemplo, você poderá ter uma visão de projeto, de processo, de implementação, de implantação e de caso de uso. Em segundo lugar, esses pacotes contêm todas as abstrações próprias àquela visão. Por exemplo, todos os componentes do seu modelo pertenceriam ao pacote que representa a visão de implementação. Entretanto, os pacotes podem referenciar elementos pertencentes a outros pacotes.

- » As visões estão relacionadas aos modelos, conforme é examinado no [Capítulo 32](#).

Para fazer a modelagem da visão da arquitetura:

- » Identifique o conjunto de visões de arquitetura significativas no contexto do problema. Na prática, isso costuma incluir uma visão de projeto, de interação, de implementação, de implantação e de caso de uso.
- » Coloque os elementos (e os diagramas) necessários e suficientes para visualizar, especificar, construir e documentar a semântica de cada visão no pacote apropriado.

- Conforme seja necessário, agrupe mais esses elementos em seus próprios pacotes. Tipicamente haverá dependências entre os elementos de visões diferentes. Portanto, de uma forma geral, permita que cada visão do nível mais alto do sistema fique aberta a todas as outras do mesmo nível.



**Figura 12.6:**

A modelagem das visões de arquitetura

Por exemplo, a [Figura 12.6](#) mostra uma decomposição canônica no nível superior, apropriada até para o sistema mais complexo que você encontrar.

► A modelagem de sistemas é examinada no [Capítulo 32](#).

## DICAS E SUGESTÕES

Ao fazer a modelagem de pacotes na UML, lembre-se de que eles existem somente para ajudar a organizar os elementos de seu modelo. Caso você tenha abstrações manifestadas como objetos no sistema real, não use pacotes. Em seu lugar, use elementos de modelagem como as classes ou os componentes. Um pacote bem estruturado:

- É coeso, fornecendo uma clara fronteira para um conjunto de elementos relacionados.
- É fracamente acoplado, exportando somente os elementos que outros pacotes realmente precisam enxergar e importando apenas os elementos

necessários e suficientes para os elementos do pacote fazerem suas tarefas.

- › Não contém muitos aninhamentos, por haver limites para a compreensão humana de estruturas com muitos aninhamentos.
- › Tem um conjunto equilibrado de conteúdo; em relação uns com os outros em um sistema, os pacotes não deverão ser muito grandes (divida-os, se necessário) nem muito pequenos (combine elementos que sejam manipulados como um grupo).

Ao definir um pacote na UML:

- › Use a forma simples de ícone de pacote, a menos que seja necessário revelar o conteúdo desse pacote explicitamente.
- › Ao revelar o conteúdo do pacote, mostre somente os elementos necessários para a compreensão do significado do pacote no contexto.
- › Principalmente se estiver usando pacotes para fazer a modelagem de itens sob um gerenciamento de configuração, revele os valores de etiquetas associados com a versão.

CAPÍTULO

---

13

---

# Instâncias

## Neste capítulo

- » *Instâncias e objetos*
- » *Modelagem de instâncias concretas*
- » *Modelagem de instâncias prototípicas*
- » *O mundo das instâncias real e conceitual*

**O**s termos “instância” e “objeto” podem ser, para a maioria dos efeitos, considerados sinônimos e, portanto, poderão ser permutados na maioria dos casos. Uma instância é a manifestação concreta de uma abstração à qual um conjunto de operações poderá ser aplicado e que poderá ter um estado que armazena os efeitos das operações. Você utilizará as instâncias para fazer a modelagem de coisas concretas ou prototípicas que existem no mundo real. Quase todos os blocos de construção da UML participam dessa dicotomia entre classe e objeto. Por exemplo, pode haver casos de uso e instâncias de casos de usos, nós e instâncias de nós, associações e instâncias de associações e assim por diante.

► *Uma discussão sobre a estrutura interna, que é preferível ao lidar com objetos e papéis prototípicos, é apresentada no Capítulo 15.*

## PRIMEIROS PASSOS

Suponha que você tenha decidido construir uma casa para sua família. Dizendo “casa” no lugar de “carro”, você já terá começado a restringir o vocabulário do espaço de sua solução. A casa é a abstração de “uma

habitação permanente ou semipermanente, cujo propósito é fornecer um abrigo”. O carro é “móvel, veículo utilizado com o propósito de transportar pessoas de um lugar para outro”. À medida que for trabalhando para satisfazer os vários requisitos concorrentes que formam seu problema, você desejará aperfeiçoar sua abstração dessa casa. Por exemplo, você poderá escolher “uma casa de três quartos com um porão”, um tipo de casa, apesar de ser um pouco mais especializada.

Quando o construtor finalmente lhe entregar as chaves de sua casa e você e sua família estiverem entrando pela porta da frente, nesse momento você está lidando com algo concreto e específico. Não é mais apenas uma grande casa de três quartos com porão, mas é a “minha casa de três quartos com um porão, localizada na Rua das Flores, 835”. Caso você seja extremamente sentimental, poderá colocar um nome em sua casa, algo como Meu Santuário (ou Sumidouro de Dinheiro).

Existe uma diferença fundamental entre uma casa de três quartos com porão e minha casa de três quartos com o nome de Meu Santuário. A primeira é uma abstração que representa um certo tipo de casa com várias propriedades; a segunda é uma instância concreta dessa abstração, representando alguma coisa que é manifestada no mundo real, com valores reais para cada uma dessas propriedades.

A abstração denota a essência ideal de uma coisa; a instância denota uma manifestação concreta. Você encontrará essa separação entre abstração e instância em tudo cuja modelagem você fizer. Para uma determinada abstração, pode haver inúmeras instâncias. Para uma determinada instância, existe alguma abstração que especifica as características comuns a todas as instâncias como essa.

A UML permite fazer a representação de abstrações e suas instâncias. Quase todos os blocos de construção da UML – principalmente as classes, os componentes, os nós e os casos de uso – poderão ser modelados em termos de sua essência ou em termos de suas instâncias. Na maior parte do tempo, você trabalhará com eles como abstrações. Quando quiser fazer a modelagem

de manifestações concretas, será necessário trabalhar com as respectivas instâncias.

- As classes são apresentadas nos Capítulos 4 e 9; os componentes são examinados no Capítulo 15; os nós são explicados no Capítulo 27; os casos de uso são apresentados no Capítulo 17. A UML realmente usa o termo especificação de instância, mas isso é um requinte de metamodelo.

A UML fornece uma representação gráfica para as instâncias, conforme mostra a [Figura 13.1](#). Esse tipo de notação permite que você visualize as instâncias nomeadas, bem como as instâncias anônimas.



**Figura 13.1:**  
Instâncias

## TERMOS E CONCEITOS

Uma *instância* é a manifestação concreta de uma abstração à qual um conjunto de operações poderá ser aplicado e que tem um estado capaz de armazenar os efeitos das operações. As *instâncias* e os *objetos* são em grande parte sinônimos. As instâncias são representadas graficamente, sublinhando seu nome.

- A dicotomia entre classe e objeto da UML é apresentada no [Capítulo 2](#).

**Nota:** A partir do uso comum, a manifestação concreta de uma classe é chamada *objeto*. Os objetos são instâncias de classes e, portanto, é incomodamente apropriado afirmar que todos os objetos são instâncias, embora algumas instâncias não sejam objetos (por exemplo, a instância de uma associação não é realmente um

*objeto; ela é apenas uma instância, também conhecida como uma conexão). Somente modeladores avançados efetivamente se preocuparão com essa sutil distinção.*

- As associações são apresentadas nos Capítulos 5 e 10; os vínculos são examinados nos Capítulos 14 e 16.

## ABSTRAÇÕES E INSTÂNCIAS

As instâncias não existem sozinhas; quase sempre estão relacionadas a alguma abstração. Muitas instâncias cuja modelagem você fará com a UML serão instâncias de classes (e esses itens são chamados objetos), embora possa haver instâncias de outros itens, como componentes, nós, casos de uso e associações. Na UML, as instâncias são facilmente diferenciadas das abstrações. Para indicar uma instância, basta sublinhar o respectivo nome.

- Os classificadores são apresentados no [Capítulo 9](#).

Em um sentido geral, um objeto é algo que ocupa espaço no mundo real ou conceitual e com o qual você pode fazer coisas. Por exemplo, a instância de um nó é tipicamente um computador que se encontra fisicamente em uma sala; a instância de um componente ocupa algum espaço no sistema de arquivos; a instância do registro de um cliente consome uma certa quantidade de memória física. De maneira semelhante, a instância da carta de voo para um avião é algo que você pode manipular matematicamente.

Você pode utilizar a UML para fazer a modelagem dessas instâncias físicas, mas também pode fazer a modelagem de itens que não sejam tão concretos. Por exemplo, uma classe abstrata, por definição, poderá não ter quaisquer instâncias diretas. Entretanto, é possível fazer a modelagem de instâncias indiretas de classes abstratas para exibir a utilização de uma instância prototípica dessa classe abstrata. Literalmente, nenhum desses objetos poderá existir. Porém, de maneira pragmática, essa instância permite que você nomeie qualquer uma das instâncias potenciais da classe-filha concreta dessa classe abstrata. O mesmo se aplica às interfaces. De acordo com sua definição, as interfaces poderão não ter quaisquer instâncias diretas,

mas você pode fazer a modelagem da instância prototípica de uma interface, representando qualquer uma das instâncias potenciais de classes concretas que realizam essa interface.

- As classes abstratas são apresentadas no [Capítulo 9](#); as interfaces são examinadas no [Capítulo 11](#).

Ao fazer a modelagem de instâncias, você as coloca em diagramas de objetos (caso queira visualizar seus detalhes estruturais) ou em diagramas de interação ou de atividades (caso prefira visualizar sua participação em situações dinâmicas). Embora tipicamente não seja necessário, você pode colocar os objetos em diagramas de classes, se quiser mostrar explicitamente o relacionamento de um objeto e a respectiva abstração.

- Os diagramas de objetos são apresentados no [Capítulo 14](#).

## TIPOS

Cada instância possui um tipo. O tipo de uma instância real deve ser um classificador concreto, mas uma especificação de instância (que não representa uma única instância) pode ter um tipo abstrato. Na notação, o nome da instância é seguido por dois-pontos e pelo tipo, por exemplo, t : Transação.

- Os diagramas de interação são examinados no [Capítulo 19](#); os diagramas de atividades são explicados no [Capítulo 20](#); os tipos definidos dinamicamente aparecem no [Capítulo 11](#); os classificadores são apresentados no [Capítulo 9](#).

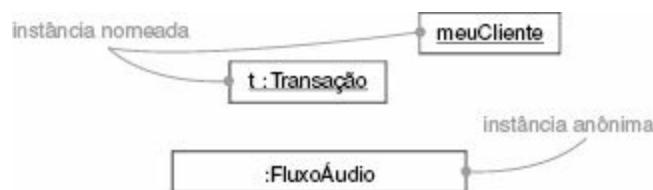
O classificador de uma instância costuma ser estático. Por exemplo, uma vez criada a instância de uma classe, essa classe não poderá ser alterada durante todo o tempo de vida do objeto. Entretanto, em algumas situações de modelagem e em algumas linguagens de programação, existe a possibilidade de alterar a abstração de uma instância. Por exemplo, um objeto Lagarta poderá

se transformar no objeto Borboleta. Será o mesmo objeto, mas de uma abstração diferente.

**Nota:** Durante o desenvolvimento, você também pode ter instâncias sem um classificador associado, que pode ser representado como um objeto, mas sem incluir o nome da abstração correspondente, conforme mostra a [Figura 13.2](#). Você pode introduzir objetos órfãos, como aqueles utilizados quando é necessário fazer a modelagem de comportamentos muito abstratos, embora eventualmente deva-se vincular essas instâncias a uma abstração, caso queira forçar qualquer grau de semântica em relação ao objeto.

## NOMES

Toda instância deve ter um nome que a diferencie das outras instâncias encontradas em seu contexto. Tipicamente, um objeto se encontra no contexto de uma operação, um componente ou um nó. Um *nome* é uma sequência de caracteres textual, como `t` e `meuCliente`, mostrados na [Figura 13.2](#). O nome sozinho é conhecido como *nome simples*. A abstração de uma instância poderá ser um nome simples (como Transação) ou um *nome de caminho* (como Multimídia :: FluxoÁudio), ou seja, o nome da abstração apresentando, como prefixo, o nome do pacote que contém a abstração.



**Figura 13.2:**  
Instâncias nomeadas e anônimas

- As operações são apresentadas nos [Capítulos 4](#) e [9](#); os componentes são examinados no [Capítulo 15](#); os nós são explicados no [Capítulo 27](#).

Ao nomear um objeto explicitamente, você está, de fato, lhe dando um nome (como `meuCliente`) que pode ser utilizado por um ser humano. Você também pode simplesmente nomear um objeto (como em `meuCliente`) e ocultar

sua abstração, caso sua existência em determinado contexto seja óbvia. Em muitos casos, entretanto, o nome real de um objeto é conhecido somente pelo computador no qual o objeto se encontra. Nesse caso, você pode representar um objeto anônimo (como em `:FluxoÁudio`). Cada ocorrência de um objeto anônimo é considerada distinta de todas as outras ocorrências. Caso nem sequer conheça a abstração associada ao objeto, você deve pelo menos atribuir-lhe um nome explícito (como `agente` :). O nome e o tipo de um objeto formam uma sequência de caracteres na notação, por exemplo, `t : Transação`. Em um objeto (ao contrário de um papel em uma classe estruturada), a sequência de caracteres inteira é sublinhada.

► *Papéis e classes estruturadas são apresentados no [Capítulo 15](#).*

**Nota:** *O nome de uma instância poderá ser um texto contendo qualquer quantidade de letras, números e certos sinais de pontuação (exceto sinais como os dois-pontos, utilizados para separar o nome da instância e o nome da respectiva abstração) e poderá se estender por várias linhas. Na prática, os nomes de instâncias são expressões ou nomes curtos definidos a partir do vocabulário do sistema cuja modelagem está sendo feita. Tipicamente, a primeira letra de cada palavra é maiúscula, com exceção da primeira palavra do nome da instância, como em `t` ou `meuCiente`.*

## OPERAÇÕES

Um objeto não é apenas algo que costuma ocupar espaço no mundo real; também é algo com o qual é possível fazer coisas. As operações que podem ser executadas com um objeto são declaradas na abstração do próprio objeto. Por exemplo, se a classe `Transação` define a operação `confirmar`, então dada a instância `t : Transação`, é possível escrever expressões como `t . confirmar ()`. A execução dessa expressão significa que `t` (o objeto) é operado por `confirmar` (a operação). Dependendo da herança associada com `Transação`, essa operação pode ou não ser chamada polimorficamente.

► *As operações são examinadas nos Capítulos 4 e 9; o polimorfismo é apresentado no [Capítulo 9](#).*

## ESTADO

Um objeto também tem estado, que, nesse sentido, envolve todas as propriedades do objeto juntamente com os valores atuais de cada uma dessas propriedades (também inclui vínculos e objetos relacionados, dependendo do seu ponto de vista). Essas propriedades incluem os atributos e as associações do objeto, além de todas as suas partes agregadas. O estado de um objeto é, portanto, dinâmico. Assim, ao visualizar seu estado, você está realmente especificando o valor de seu estado em um determinado momento no tempo e no espaço. É possível exibir o estado de alteração do objeto, mostrando-o várias vezes no mesmo diagrama de interação, mas cada ocorrência representando um estado diferente.

- Os atributos são examinados no [Capítulo 4](#); os diagramas de interação são apresentados no [Capítulo 19](#). Uma outra forma de exibir o estado de alteração de um objeto individual ao longo do tempo é a utilização de máquinas de estado, apresentadas no [Capítulo 22](#).

Ao trabalhar com um objeto, tipicamente você altera seu estado; ao fazer uma consulta a um objeto, seu estado não é alterado. Por exemplo, ao fazer a reserva de um voo (representada pelo objeto `r : Reserva`), você poderá definir o valor de um de seus atributos (por exemplo, `preço = 395,75`). Se você alterar sua reserva, talvez adicionando uma nova escala para o seu itinerário, então seu estado poderá ser modificado (por exemplo, `preço = 1024,86`).

Conforme mostra a [Figura 13.3](#), você pode utilizar a UML para representar o valor dos atributos de um objeto. Por exemplo, `meuCliente` é mostrado com o atributo `id` tendo o valor “432-89-1783”. Nesse caso, o tipo (`SSN`) de código é mostrado explicitamente, apesar de poder ficar oculto (como no caso de `ativo = Verdadeiro`), pois seu tipo pode ser encontrado na declaração de código na classe associada de `meuCliente`.

Uma máquina de estados pode ser associada com uma classe, o que é de grande ajuda, principalmente ao fazer a modelagem de sistemas orientados a

eventos ou do tempo de vida de uma classe. Nesses casos, também é possível mostrar o estado dessa máquina em relação a um determinado objeto em certo momento. Por exemplo, conforme mostra a [Figura 13.3](#), o objeto c (uma instância da classe Telefone) é indicado no estado AguardandoResposta, um estado nomeado que foi definido na máquina de estados de Telefone.



**Figura 13.3:**

O estado do objeto

**Nota:** Como um objeto poderá estar em vários estados simultaneamente, também é possível mostrar uma lista de seus estados atuais.

## OUTRAS CARACTERÍSTICAS

Os processos e os threads são elementos importantes da visão de processo de um sistema; assim, a UML fornece uma indicação visual para diferenciar elementos que se encontram ativos (aqueles que fazem parte de um processo ou thread e representam a raiz de um fluxo de controle) e os que estão passivos. Você pode declarar as classes ativas que retificam processos ou threads e, por sua vez, pode diferenciar uma instância de uma classe ativa, conforme mostra a [Figura 13.4](#).

► Os processos e os threads são examinados no [Capítulo 23](#).



**Figura 13.4:**

Objetos ativos

**Nota:** Com muita frequência, você usará objetos ativos no contexto de diagramas de interação que fazem a modelagem de vários fluxos de controle. Cada objeto ativo representa a raiz de um fluxo de controle e poderá ser utilizado para nomear fluxos distintos.

► Os diagramas de interação são examinados no [Capítulo 19](#).

Dois elementos da UML também poderão ter instâncias. O primeiro é uma associação. Uma instância de uma associação é um vínculo. Os vínculos são conexões semânticas existentes entre uma lista de objetos. Um vínculo é representado como uma linha, assim como uma associação, mas, ao contrário das associações, os vínculos podem conectar apenas objetos. O segundo tipo de instância é um atributo de estática (classe com escopo). Um atributo de estática é, na verdade, um objeto da classe que é compartilhado por todas as instâncias da mesma classe. É, portanto, apresentado em uma declaração de classe como um atributo sublinhado.

► Os vínculos são examinados nos [Capítulos 14 e 16](#); operações e atributos de classes com escopo são apresentados no [Capítulo 9](#).

## ELEMENTOS-PADRÃO

Todos os mecanismos de extensibilidade da UML se aplicam aos objetos. Geralmente, porém, você não cria o estereótipo de uma instância diretamente, nem atribui a ela seus próprios valores atribuídos. Em vez disso, o estereótipo e os valores atribuídos do objeto são derivados do estereótipo e dos valores atribuídos da correspondente abstração associada. Por exemplo, conforme mostra a [Figura 13.5](#), é possível indicar o estereótipo de um objeto explicitamente, assim como sua abstração.

► Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).



**Figura 13.5:**  
Objetos estereotipados

A UML define dois estereótipos-padrão que se aplicam aos relacionamentos de dependência entre objetos e entre classes:

- |                |   |
|----------------|---|
| 1. instanceOf  | Especifica que o objeto cliente é uma instância do classificador do fornecedor. Raramente é exibido graficamente; em geral, apresenta uma notação de texto seguida de dois- pontos. |
| 2. instantiate | Especifica que a classe cliente cria instâncias da classe do fornecedor.  |

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE INSTÂNCIAS CONCRETAS

Ao fazer a modelagem de instâncias concretas, na verdade, você está visualizando algo que existe no mundo real. Não é possível ver exatamente a instância de uma classe CUSTOMER, por exemplo, a menos que o cliente esteja ao seu lado; em um depurador, porém, você poderá ser capaz de ver uma representação desse objeto.

Uma das coisas para as quais os objetos serão utilizados será a modelagem de instâncias concretas que existem no mundo real. Por exemplo, para fazer a modelagem da topologia da rede de uma empresa, você usará diagramas de implantação contendo instâncias dos nós. De modo semelhante, para fazer a modelagem de componentes existentes nos nós físicos dessa rede, você usará diagramas de componentes contendo instâncias desses componentes. Por fim, suponha que exista um depurador conectado ao

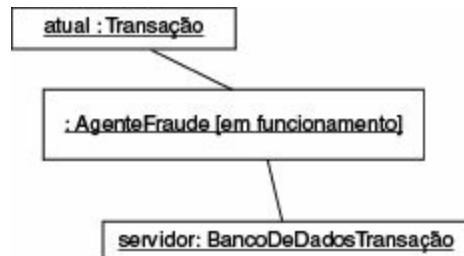
sistema em execução; ele poderá apresentar os relacionamentos estruturais existentes entre as instâncias, representando-os em diagramas de objetos.

- » Os *diagramas de componentes* são examinados no [Capítulo 15](#); os *diagramas de implantação* são apresentados no [Capítulo 31](#); os *diagramas de objetos* são explicados no [Capítulo 14](#).

Para fazer a modelagem de instâncias concretas:

- » Identifique as instâncias necessárias e suficientes para visualizar, especificar, construir ou documentar o problema cuja modelagem você está fazendo.
- » Represente esses objetos na UML como instâncias. Onde for possível, dê um nome a cada objeto. Caso não haja um nome significativo para o objeto, represente-o como um objeto anônimo.
- » Exponha o estereótipo, os valores atribuídos e os atributos (com os respectivos valores) de cada instância necessária e suficiente para a modelagem de seu problema.
- » Represente essas instâncias e seus relacionamentos em um diagrama de objetos ou em qualquer outro diagrama apropriado ao tipo de instância.

Por exemplo, a [Figura 13.6](#) mostra um diagrama de objetos definido a partir da execução de um sistema para validação de cartões de crédito, talvez da forma como seja visto por um depurador que esteja testando o sistema em execução.



**Figura 13.6:**

A modelagem de instâncias concretas

## **MODELAGEM DE INSTÂNCIAS PROTOTÍPICAS**

### **DICAS E SUGESTÕES**

Ao fazer a modelagem de instâncias na UML, lembre-se de que todas as instâncias deverão denotar uma manifestação concreta de alguma abstração, tipicamente uma classe, componente, nó, caso de uso ou associação. Uma instância bem estruturada:

- » Está associada explicitamente a uma abstração específica.
- » Tem um nome único definido a partir do vocabulário do domínio do problema ou domínio da solução.

Ao definir uma instância na UML:

- » Represente o nome da abstração da qual ela é uma instância, a menos que seja óbvia devido ao contexto.
- » Mostre o estereótipo, o papel e o estado da instância somente conforme seja necessário para a compreensão do objeto em seu contexto.
- » Se visível, organize extensas listas de atributos e seus valores, agrupando-os às suas categorias.

CAPÍTULO

---

**14**

---

# Diagramas de Objetos

## Neste capítulo

- » *Modelagem de estruturas de objetos*
- » *Engenharia direta e reversa*

**O**s diagramas de objetos fazem a modelagem de instâncias de itens contidos em diagramas de classes. Um diagrama de objetos mostra um conjunto de objetos e seus relacionamentos em determinado ponto no tempo.

Você usa os diagramas de objetos para fazer a modelagem da visão estática do projeto ou do processo de um sistema. Isso envolverá a modelagem de um retrato do sistema em determinado momento e a representação de um conjunto de objetos, seus estados e relacionamentos. Os diagramas de objetos não são importantes apenas para a visualização, especificação e documentação de modelos estruturais, mas também para a construção de aspectos estáticos de sistemas por meio de engenharia direta e reversa.

## PRIMEIROS PASSOS

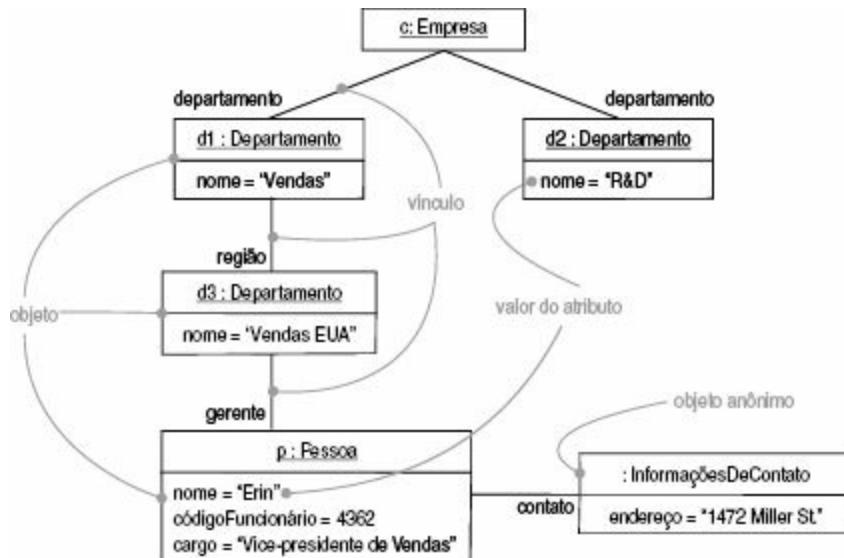
Se você não estiver acostumado com o jogo, o futebol americano parecerá um esporte terrivelmente simples – um movimento descoordenado de pessoas correndo loucamente em um campo, perseguindo uma bola branca. Observando a confusa imagem dos corpos em movimento, dificilmente parece haver algum estilo ou sutileza nesse esporte.

Congele o movimento por um instante e então classifique os jogadores individualmente. Um retrato bem diferente do jogo é percebido. Em vez de um amontoado de seres humanos, você será capaz de distinguir ataque, meio de campo e defesa. Pense um pouco mais e você compreenderá como esses jogadores colaboram uns com os outros, seguindo estratégias para tentar a marcação de um gol, passando a bola, tomando a bola do adversário e atacando. Em um time vencedor, você não encontrará jogadores distribuídos aleatoriamente pelo campo. Em vez disso, em qualquer momento do jogo, você observará que são bem calculados a sua movimentação pelo campo e o seu relacionamento com os demais jogadores.

É algo semelhante tentar visualizar, especificar, construir ou documentar um sistema complexo de software. Se tentar traçar o fluxo de controle de um sistema em execução, você logo perderá a visão global a respeito do modo como as partes do sistema estão organizadas, principalmente se houver múltiplos threads de controle. De maneira semelhante, se você tiver uma estrutura de dados complexa, apenas a observação do estado de um único objeto em determinado momento não é de grande ajuda. Em vez disso, é necessário estudar um retrato do objeto, seus vizinhos e seus relacionamentos com esses vizinhos. Em todos os sistemas orientados a objetos, com exceção dos mais simples, você encontrará uma variedade de objetos presentes, cada um com um relacionamento preciso com os demais. De fato, quando ocorre uma falha em um sistema orientado a objetos, tipicamente isso não ocorre devido a algum erro de lógica, mas por causa de conexões quebradas entre objetos ou ao estado danificado de objetos individuais.

A UML permite a utilização de diagramas de classes para uma visualização dos aspectos estáticos dos blocos de construção do sistema. Você usa os diagramas de interação para visualizar os aspectos dinâmicos de seu sistema, formados por instâncias desses blocos de construção e mensagens enviadas entre eles. Os diagramas de objetos cobrem um conjunto de instâncias dos itens encontrados nos diagramas de classes. O diagrama de objetos, portanto, expressa a parte estática de uma interação, composta pelos objetos que colaboram entre si, mas sem qualquer uma das mensagens

passadas entre eles. Nos dois casos, o diagrama de objetos congela um momento no tempo, conforme mostra a [Figura 14.1](#).



**Figura 14.1:**

Um diagrama de objetos

- Os *diagramas de classes* são examinados no [Capítulo 8](#); as *interações* são apresentadas no [Capítulo 16](#); os *diagramas de interação* são mostrados no [Capítulo 19](#).

## TERMOS E CONCEITOS

Um *diagrama de objetos* é um diagrama que mostra um conjunto de objetos e seus relacionamentos em um ponto no tempo. Graficamente, o diagrama de objetos é uma coleção de vértices e de arcos.

## PROPRIEDADES COMUNS

Um diagrama de objetos é um tipo especial de diagrama e compartilha as mesmas propriedades comuns a todos os outros diagramas – ou seja, um nome e o conteúdo gráfico que formam uma projeção em um modelo. O que distingue um diagrama de objetos de todos os outros tipos de diagramas é o seu conteúdo particular.

- ➡ As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

## CONTEÚDO

Os diagramas de objetos costumam conter o seguinte:

- ➡ Os objetos são examinados no [Capítulo 13](#); os vínculos no [Capítulo 16](#).

» Objetos

» Vínculos

Assim como os demais diagramas, os diagramas de objetos podem conter notas e restrições. Em alguns casos, você também desejará incluir classes em seus diagramas de objetos, principalmente quando quiser visualizar as classes que existem por trás de cada instância.

**Nota:** Um diagrama de objetos relaciona-se a um diagrama de classes: o diagrama de classes descreve a situação geral, e o diagrama de instâncias descreve instâncias específicas derivadas do diagrama de classes. Um diagrama de objetos contém principalmente objetos e vínculos. Os diagramas de implantação também podem ocorrer nas formas genérica e de instâncias: os diagramas de implantação gerais descrevem tipos de nós, e os diagramas de implantação de instâncias descrevem uma configuração concreta das instâncias de nós descritas por esses tipos.

- ➡ Os diagramas de classes são examinados no [Capítulo 8](#); os diagramas de interação são apresentados no [Capítulo 19](#).

## USOS COMUNS

Você usa os diagramas de objetos para fazer a modelagem da visão de projeto estática ou da visão de processo estática de um sistema, da mesma maneira como faz com os diagramas de classes, mas a partir da perspectiva de instâncias reais ou protótipicas. Essa visão atende principalmente aos requisitos funcionais do sistema – ou seja, os serviços que o sistema deverá

proporcionar aos seus usuários finais. Os diagramas de objetos permitem que você faça a modelagem de estruturas de dados estáticos.

- As visões de projeto são examinadas no [Capítulo 2](#).

Ao fazer a modelagem da visão de projeto estática ou da visão de processo estática de um sistema, tipicamente você usa os diagramas de objetos para modelar estruturas dos objetos. A modelagem de estruturas dos objetos envolve um retrato dos objetos de um sistema em um determinado momento. Um diagrama de objetos representa um quadro estático no enredo dinâmico representado por um diagrama de interação. Você usa os diagramas de objetos para visualizar, especificar, construir e documentar a existência de certas instâncias encontradas em seu sistema, juntamente com os relacionamentos entre essas instâncias. Você pode mostrar o comportamento dinâmico e a execução como uma sequência de quadros.

- Os diagramas de interação são examinados no [Capítulo 19](#).

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE ESTRUTURAS DE OBJETOS

Ao construir um diagrama de classes, de componentes ou de implantação, o que você realmente está fazendo é captar um conjunto de abstrações que interessa a você como um grupo e, nesse contexto, expõe sua semântica e seus relacionamentos com outras abstrações existentes no grupo. Esses diagramas mostram somente a potencialidade. Se a classe A tem uma associação de um-para-muitos com a classe B, então para uma única instância de A poderá haver cinco instâncias de B; para outra instância de A poderá haver apenas uma instância de B. Além disso, em um determinado momento, para essa instância de A, juntamente com as instâncias relacionadas de B, cada uma terá certos valores para seus atributos e máquinas de estados.

Congelando o sistema em execução ou apenas imaginando um momento no tempo em um sistema modelado, você encontrará um conjunto de objetos, cada um em um estado específico e em um determinado relacionamento com os demais objetos. Você usa os diagramas de objetos para visualizar, especificar, construir e documentar a estrutura desses objetos. Os diagramas de objetos são de grande ajuda principalmente para a modelagem de estruturas complexas de dados.

Quando você faz a modelagem da visão de projeto de seu sistema, um conjunto de diagramas de classes pode ser utilizado para especificar toda a semântica de suas abstrações e seus relacionamentos. Com os diagramas de objetos, porém, você não é capaz de especificar completamente a estrutura de objetos de seu sistema. Para uma classe individual, poderá haver várias instâncias possíveis e, para um conjunto de classes em relacionamentos um-para-um, muitas vezes poderá haver mais configurações possíveis desses objetos. Portanto, ao utilizar diagramas de objetos, você pode expor significativamente somente conjuntos interessantes de objetos concretos ou prototípicos. Isso é o que significa fazer a modelagem de uma estrutura de objetos – um diagrama de objetos exibe um único conjunto de objetos relacionados uns com os outros em um determinado momento.

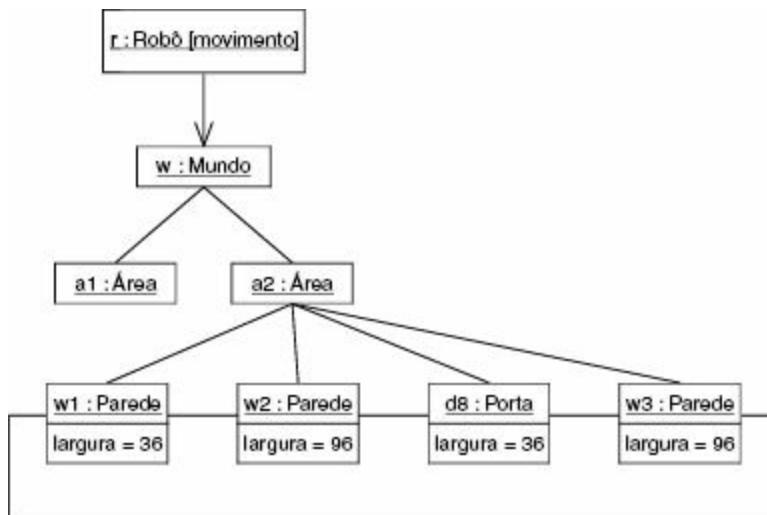
Para fazer a modelagem de uma estrutura de objetos:

- › Identifique o mecanismo cuja modelagem você deseja fazer. Um mecanismo representa alguma função ou comportamento da parte do sistema cuja modelagem você está fazendo e que é resultante da interação de uma sociedade de classes, interfaces e outros itens.
- › Crie uma colaboração para descrever um mecanismo.
- › Para cada mecanismo, identifique as classes, interfaces e outros elementos que participam nessa colaboração; identifique também os relacionamentos entre esses itens.
- › Leve em consideração um único cenário capaz de percorrer esse mecanismo. Congele o cenário em determinado momento e represente

cada objeto que participa do mecanismo.

- › Exponha o estado e os valores dos atributos de cada um desses objetos, conforme seja necessário, para a compreensão do cenário.
  - › De maneira semelhante, exponha os vínculos existentes entre esses objetos, representando instâncias de associações entre eles.
- *Mecanismos como esses costumam estar relacionados a casos de uso, conforme é examinado nos Capítulos 17 e 29.*

Por exemplo, a [Figura 14.2](#) mostra um conjunto de objetos definidos a partir da implementação de um robô autônomo. Essa figura focaliza alguns dos objetos envolvidos no mecanismo utilizado pelo robô para calcular um modelo do mundo em que ele se move. Existem muitos outros objetos envolvidos em um sistema em execução, mas esse diagrama focaliza somente aquelas abstrações que estão diretamente envolvidas na criação dessa visão de mundo. Conforme a figura indica, um objeto representa o próprio robô (`r`, uma instância de Robô) e, no momento, `r` está no estado marcado como movimento. Esse objeto tem um vínculo com `w`, uma instância de Mundo, que representa uma abstração do modelo de mundo do robô.



**Figura 14.2:**

A modelagem de estruturas de objetos

Nesse momento, w está vinculado a duas instâncias de Área. Uma delas (a2) é mostrada com seus próprios vínculos a três objetos Parede e a um objeto Porta. Cada uma dessas paredes está marcada com sua largura atual e cada uma é mostrada vinculada às paredes vizinhas. Conforme sugere esse diagrama de objetos, o robô reconheceu essa área interna, que tem paredes em três lados e uma porta no quarto lado.

## ENGENHARIA DIRETA E ENGENHARIA REVERSA

A engenharia reversa (a criação de um modelo a partir de código) de um diagrama de objetos pode ser útil. De fato, enquanto você está depurando seu sistema, isso será algo que você ou suas ferramentas farão o tempo todo. Por exemplo, ao procurar um vínculo problemático, você desejará, literal ou mentalmente, definir um diagrama de objetos dos objetos afetados para ver em que local, em um determinado momento, o estado de um objeto ou seu relacionamento com outros objetos é interrompido. Para fazer a engenharia reversa de um diagrama de objetos:

- » Escolha o alvo da engenharia reversa. Tipicamente, você definirá seu contexto em uma operação ou relativo a uma instância de uma determinada classe.
- » Utilizando uma ferramenta ou simplesmente percorrendo um cenário, interrompa a execução em um certo momento.
- » Identifique o conjunto de objetos interessantes que colaboram nesse contexto e represente-os em um diagrama de objetos.
- » Conforme seja necessário para a compreensão de sua semântica, exponha os estados desse objeto.
- » Conforme seja necessário para a compreensão de sua semântica, identifique os vínculos existentes entre esses objetos.
- » Caso o diagrama se torne muito complicado, simplifique-o, eliminando objetos que não sejam relevantes às perguntas que você precisa responder a respeito do cenário. Se o diagrama ficar muito simples,

experimente expandir os vizinhos de certos objetos interessantes e expor o estado de cada objeto com maior profundidade.

- » Em geral, você precisará adicionar ou identificar manualmente a estrutura que não está explícita no código de destino. As informações ausentes fornecem o objetivo do projeto que só está implícito no código final.

## DICAS E SUGESTÕES

Ao criar diagramas de objetos na UML, lembre-se de que todo diagrama de objetos é apenas uma representação gráfica da visão estática de projeto ou visão de interação estática de um sistema. Isso significa que num único diagrama de objetos não é preciso captar tudo sobre a visão de processo ou de projeto de um sistema. De fato, em relação à maioria dos sistemas, com exceção dos mais triviais, você encontrará centenas, se não milhares de objetos, muitos deles anônimos. Portanto, é impossível especificar completamente todos os objetos de um sistema ou todas as formas em que esses objetos poderão estar associados. Consequentemente, os diagramas de objetos refletem alguns dos objetos concretos ou protótipicos encontrados no sistema em execução. Um diagrama de objetos bem estruturado:

- » Tem seu foco voltado para comunicar um único aspecto da visão estática, de projeto ou de processo, do sistema.
- » Representa um único quadro no enredo dinâmico representado por um diagrama de interação.
- » Contém somente aqueles elementos essenciais para a compreensão desse aspecto.
- » Fornece detalhes consistentes com seu nível de abstração; você deverá expor apenas os valores de atributo e outros adornos que sejam essenciais para a compreensão.

- › Não é tão minimalista, que acabe informando mal o leitor sobre a semântica que seja importante.

Quando você definir um diagrama de objetos:

- › Dê-lhe um nome capaz de comunicar seu propósito.
- › Distribua seus elementos para minimizar a ocorrência de linhas cruzadas.
- › Organize seus elementos por espaço, para que os itens semanticamente afins apareçam fisicamente próximos.
- › Use anotações e cores como indicações visuais para chamar a atenção em relação a características importantes de seu diagrama.
- › Inclua os valores, o estado e o papel de cada objeto, conforme seja necessário para comunicar suas intenções.

CAPÍTULO

---

15

---

# Componentes

## Neste capítulo

- » *Componentes, interfaces e realização*
- » *Estrutura interna, portas e conectores*
- » *Coneção de subcomponentes*
- » *Modelagem de uma API*

**U**m componente é a parte lógica e substituível de um sistema ao qual se adapta e fornece a realização de um conjunto de interfaces. Bons componentes definem abstrações com interfaces bem definidas, tornando possível substituir facilmente componentes mais antigos por outros compatíveis mais novos. As interfaces funcionam como uma ponte entre os modelos lógico e de projeto. Por exemplo, você pode especificar uma interface para uma classe em um modelo lógico, e essa mesma interface será transferida para algum componente do projeto que a realiza. As interfaces permitem a criação da implementação de um componente usando componentes menores, conectando as portas aos componentes.

## PRIMEIROS PASSOS

Ao construir uma casa, você pode instalar um *home-theater*. É possível comprar uma única unidade que inclua tudo: televisão, sintonizador, VCR, DVD e alto-falantes. Esse sistema é fácil de instalar e funciona bem, se atender às suas necessidades. Entretanto, uma unidade de uma eça só não é muito flexível. Você precisa obter a combinação de recursos que o fabricante fornece. Provavelmente, não será possível ter alto-falantes de boa qualidade.

Se desejar instalar uma nova tela de televisão de alta definição, você terá de jogar fora a unidade toda e substituí-la, incluindo o VCR e o DVD, que ainda podem estar funcionando bem. Se tiver uma coleção de discos (alguns leitores devem lembrar o que é isso), você está sem sorte.

Uma abordagem mais flexível é instalar um *home-theater* com componentes individuais, cada um com uma única funcionalidade. Um monitor exibe a imagem; os alto-falantes individuais fornecem o som e podem ser colocados onde seu espaço e seus ouvidos permitirem; o sintonizador, o VCR e o DVD são unidades separadas, e seus recursos são adequados aos seus requisitos videófilos e ao seu orçamento. Em vez de ligá-los de uma maneira rígida, você coloca cada componente onde deseja e os prende com cabos. Cada cabo tem um tipo específico de plugue que se ajusta à porta correspondente de um componente, de maneira que você não possa ligar o cabo de um alto-falante a uma saída de vídeo. Se quiser, é possível ligar seu velho toca-discos. Quando quiser atualizar o sistema, você pode substituir um componente por vez, sem precisar jogar fora o sistema todo e começar de novo. Os componentes fornecem mais flexibilidade e permitem que você obtenha mais qualidade, se quiser e puder.

- *Você pode plugar a entrada do amplificador na saída de vídeo, porque elas usam os mesmos tipos de plugues. O software tem a vantagem de um número ilimitado de tipos de plugues.*

O software é similar. Você pode construir um aplicativo como uma unidade grande e monolítica, mas será difícil modificá-la conforme as necessidades mudem. Além disso, você não pode aproveitar a vantagem de recursos existentes. Mesmo que um sistema existente tenha muitas funcionalidades de que você precisa, ele também pode ter muitas outras partes que você não deseja, e elas são difíceis ou impossíveis de remover. A solução para sistemas de software é a mesma que para sistemas eletrônicos: crie-os a partir de componentes bem definidos que possam ser conectados de

maneira flexível e substituídos individualmente quando os requisitos mudarem.

## TERMOS E CONCEITOS

Uma *interface* é uma coleção de operações que especifica um serviço fornecido por ou solicitado de uma classe ou componente.

- As *interfaces* são examinadas no [Capítulo 11](#); as *classes*, nos [Capítulos 4 e 9](#).

Um *componente* é uma parte substituível de um sistema ao qual se adapta e fornece a realização de um conjunto de interfaces.

Uma *porta* é uma janela específica em um componente encapsulado que aceita mensagens para e do componente que se adapta às interfaces especificadas.

A *estrutura interna* é a implementação de um componente por meio de um conjunto de partes que são conectadas de uma determinada maneira.

Uma *parte* é a especificação de um papel que compõe parte da implementação de um componente. Em uma instância do componente, há uma instância correspondente à parte.

Um *conector* é um relacionamento de comunicação entre duas partes ou portas no contexto de um componente.

## COMPONENTES E INTERFACES

Uma interface é uma coleção de operações utilizadas para especificar um serviço de uma classe ou de um componente. O relacionamento entre componente e interface é importante. Todas as funcionalidades dos sistemas operacionais mais comuns baseados em componentes (como COM+, CORBA e Enterprise Java Beans) usam interfaces como o que permite reunir os componentes.

► As interfaces são examinadas no [Capítulo 11](#).

Para construir um sistema baseado em componentes, você decompõe o sistema, especificando as interfaces que representam as principais costuras existentes no sistema. Depois você fornece os componentes que realizam essas interfaces, juntamente com outros componentes que têm acesso aos serviços por meio de suas interfaces. Esse mecanismo permite entregar um sistema cujos serviços são, de alguma forma, independentes de localização e, conforme é examinado na próxima seção, substituíveis.

► A modelagem de sistemas distribuídos é examinada no [Capítulo 24](#).

Uma interface realizada por um componente é chamada *interface fornecida*, significando uma interface em que o componente fornece um serviço para outros componentes. Um componente poderá declarar muitas interfaces fornecidas. A interface utilizada pelo componente é chamada *interface requerida*, significando uma interface à qual o componente se adapta quando solicita serviços de outros componentes. Um componente poderá estar em conformidade a muitas interfaces requeridas. Além disso, um componente pode tanto fornecer, como requerer interfaces.

► A realização é examinada no [Capítulo 10](#).

Conforme mostra a [Figura 15.1](#), um componente é exibido como um retângulo com um pequeno ícone de dois pinos no canto superior direito. O nome do componente aparece no retângulo. Um componente pode ter atributos e operações, mas eles são frequentemente eliminados nos diagramas. Um componente pode apresentar uma rede de estrutura interna, como será descrito neste capítulo.

É possível mostrar o relacionamento entre um componente e suas interfaces em uma de duas maneiras. O primeiro (e mais comum) estilo representa a interface em sua forma icônica, oculta. Uma interface fornecida apresenta um círculo anexado ao componente por uma linha (um pirulito).

Uma interface requerida apresenta um semicírculo anexado ao componente por uma linha (um soquete). Nos dois casos, o nome da interface é colocado perto do símbolo. O segundo estilo representa a interface em sua forma expandida, talvez revelando suas operações. O componente que realiza a interface é conectado a ela por meio de um relacionamento de realização pleno. O componente que tem acesso aos serviços do outro componente por meio da interface é conectado a ela usando um relacionamento de dependência.

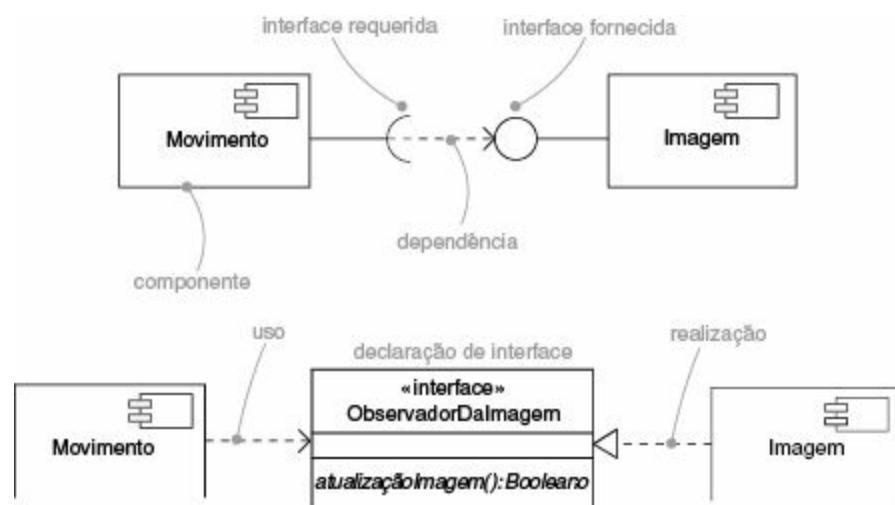


Figura 15.1:

Componentes e interfaces

Uma determinada interface poderá ser fornecida por um componente e requerida por um outro. O fato de essa interface se encontrar entre dois componentes quebra a dependência direta entre os componentes. Um componente que utiliza uma determinada interface funcionará adequadamente, qualquer que seja o componente que a realiza. É claro que um componente pode ser utilizado em um contexto, se e somente se todas as interfaces requeridas forem realizadas pelas interfaces fornecidas de outros componentes.

**Nota:** As interfaces aplicam-se a vários níveis, exatamente como os outros elementos. A interface de nível de projeto utilizada ou realizada por um componente

*será mapeada para uma interface de nível de implementação utilizada ou realizada pelo artefato que implementa o componente.*

## SUBSTITUIÇÃO

O propósito básico de qualquer funcionalidade de um sistema operacional baseado em componentes consiste em permitir a montagem de sistemas a partir de artefatos binários substituíveis. Isso significa que é possível criar um sistema a partir de componentes e implementar esses componentes usando artefatos. Você pode evoluir esse sistema pela adição de novos componentes e pela substituição dos anteriores, sem reconstruir o sistema. As interfaces são a chave para que isso possa acontecer. No sistema executável, você pode usar qualquer artefato que implemente um componente que esteja em conformidade ou que forneça essa interface. Você pode estender o sistema, fazendo com que os componentes ofereçam novos serviços por meio de outras interfaces, as quais, por sua vez, podem ser descobertas e utilizadas por outros componentes. Essa semântica explica o propósito que existe por trás da definição de componentes na UML. Um componente está em conformidade e proporciona a realização de um conjunto de interfaces e permite a substituição tanto no projeto lógico quanto na implementação física baseada nele.

► Os artefatos são examinados no [Capítulo 26](#).

Um componente é *substituível*. É possível substituir um componente por outro que esteja em conformidade com as mesmas interfaces. No momento do projeto, você escolhe um componente diferente. Tipicamente, o mecanismo de inserir ou substituir um artefato em um sistema em tempo de execução é transparente para o usuário do componente e é permitido nos modelos do objeto (como COM+ e Enterprise Java Beans) que requerem pouca ou nenhuma transformação de intervenção ou por ferramentas que automatizam o mecanismo. Um componente é *parte de um sistema*.

Um componente raramente existe sozinho. Em vez disso, um determinado componente colabora com outros componentes e, ao fazer isso, existe no contexto de arquitetura ou da tecnologia em que se pretende utilizá-lo. Um componente é lógica e fisicamente coeso e, portanto, denota uma parte estrutural e/ou comportamental significativa de um sistema maior. Um componente poderá ser reutilizado em muitos sistemas. Portanto, um componente representa um bloco de construção fundamental, no qual os sistemas poderão ser projetados e compostos. Essa definição é recursiva – um sistema em um nível de abstração poderá simplesmente ser um componente em um nível de abstração mais alto.

- *Os sistemas e subsistemas são examinados no [Capítulo 32](#).*

Finalmente, conforme é examinado na seção anterior, um componente *está em conformidade e proporciona a realização de um conjunto de interfaces*.

## ORGANIZAÇÃO DE COMPONENTES

Você pode organizar os componentes, agrupando-os em pacotes da mesma maneira como organiza as classes.

- *Os pacotes são examinados no [Capítulo 12](#).*

Também pode organizar os componentes especificando relacionamentos de dependência, generalização, associação (incluindo agregação) e realização entre eles.

- *Os relacionamentos são examinados nos [Capítulos 5 e 10](#).*

Os componentes podem ser criados de outros componentes. Veja a discussão sobre estrutura interna mais adiante neste capítulo.

## PORtAS

As interfaces são úteis para declarar o comportamento geral de um componente, mas elas não têm identidade individual. A implementação do componente deve garantir meramente que todas as operações em todas as interfaces fornecidas sejam implementadas. As portas podem ser usadas para ter um controle maior sobre a implementação.

Uma *porta* é uma janela explícita em um *componente encapsulado*. Em um componente encapsulado, todas as interações dentro e fora do componente passam pelas portas. O comportamento visível externamente do componente é a soma de suas portas, nem mais nem menos. Além disso, uma porta tem identidade. Outro componente pode se comunicar com o componente por meio de uma porta específica. As comunicações são descritas completamente pelas interfaces que a porta suporta, mesmo que o componente suporte outras interfaces. Na implementação, as partes internas do componente podem interagir por meio de uma porta externa específica; portanto, cada parte pode ser independente dos requisitos das outras partes. As portas permitem que as interfaces de um componente sejam divididas em pacotes discretos e usados independentemente. O encapsulamento e a independência fornecidos pelas portas permitem um grau bem maior de encapsulamento e capacidade de substituição.

Uma porta apresenta um pequeno quadrado abrindo a borda de um componente – ela representa uma abertura para o encapsulamento do componente. Tanto a interface fornecida quanto a requerida podem ser anexadas ao símbolo da porta. Uma interface fornecida representa um serviço que pode ser solicitado por meio dessa porta. Uma interface requerida representa um serviço que a porta precisa obter de algum outro componente. Cada porta tem um nome, de maneira que possa ser identificada exclusivamente dados o componente e o nome da porta. O nome da porta pode ser usado pelas partes internas do componente para identificar a porta por meio da qual se enviam e recebem mensagens. Os nomes do componente e da porta juntos identificam exclusivamente uma porta específica em um determinado componente para uso por outros componentes.

As portas são parte de um componente. As instâncias de portas são

criadas e destruídas junto com a instância do componente ao qual pertencem. As portas também podem ter multiplicidade. Isso indica o número possível de instâncias de uma determinada porta em uma instância do componente. Cada porta em uma instância de componente tem uma matriz de instâncias de porta. Embora as instâncias de porta em uma matriz satisfaçam a mesma interface e aceitem os mesmos tipos de solicitação, elas podem ter estados e valores de dados diferentes. Por exemplo, cada instância de uma matriz pode ter um nível de prioridade diferente, com as instâncias de porta de prioridade mais alta sendo atendidas primeiro.

- As partes também podem ter multiplicidade; portanto, uma parte em um componente pode corresponder a várias instâncias em uma instância do componente.

A Figura 15.2 mostra o modelo de um componente Bilheteiro com portas. Cada porta tem um nome e, opcionalmente, um tipo para informar o tipo de porta. O componente tem portas para venda de bilhetes, atrações e venda com cartão de crédito.



**Figura 15.2:**

Portas de um componente

Há duas portas para venda de bilhetes, uma para clientes normais e outra para clientes prioritários. Ambas têm a mesma interface fornecida de tipo Venda de bilhetes. A porta de processamento de cartão de crédito tem uma

interface requerida. Qualquer componente que forneça os serviços especificados pode satisfazê-la. As portas de atrações têm interfaces fornecidas e requeridas. Usando a interface `Carregar atrações`, um teatro pode inserir shows e outras atrações no banco de dados de bilhetes para venda. Usando a interface `Reserva`, o componente bilheteiro pode consultar a disponibilidade de bilhetes nos teatros e realmente comprar os bilhetes.

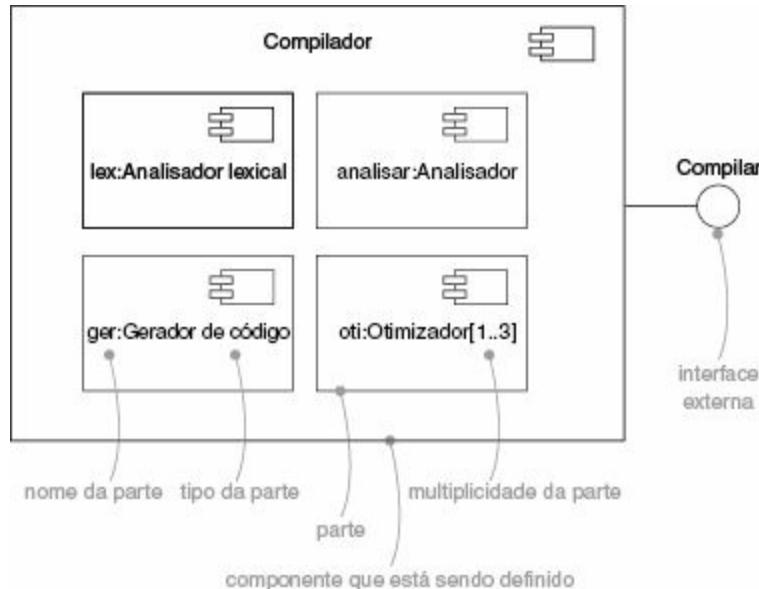
## ESTRUTURA INTERNA

Um componente pode ser implementado como um código simples, mas, em sistemas maiores, é desejável poder construir componentes grandes usando componentes menores como blocos de construção. A estrutura interna de um componente é a parte que compõe a implementação do componente junto com as suas conexões. Em muitos casos, as partes internas podem ser instâncias de componentes menores que foram conectados estaticamente por meio de portas, para fornecer o comportamento necessário sem a necessidade de o modelador especificar uma lógica extra.

Uma *parte* é uma unidade da implementação de um componente. Uma parte tem um nome e um tipo. Em uma instância do componente, há uma ou mais instâncias que correspondem a cada parte que tem o tipo especificado. Uma parte tem uma multiplicidade em seu componente. Se a multiplicidade da parte for maior que um, pode haver mais de uma instância de parte em uma determinada instância de componente. Se a multiplicidade não for um inteiro, o número de instâncias de parte pode variar de uma instância do componente para outra. Uma instância de componente é criada com o número mínimo de partes; partes adicionais podem ser acrescentadas depois. O atributo de uma classe é um tipo de parte: ele tem um tipo e uma multiplicidade, e cada instância da classe tem uma ou mais instâncias do tipo determinado.

A [Figura 15.3](#) mostra um componente compilador construído de quatro tipos de partes. Há um analisador léxico, um analisador sintático, um gerador de código e de um a três otimizadores. Versões mais completas do

compilador podem ser configuradas com diferentes níveis de otimização. Em uma determinada versão, o otimizador apropriado pode ser selecionado em tempo de execução.



**Figura 15.3:**  
Partes dentro de um componente

Observe que uma parte não é o mesmo que uma classe. Cada parte é potencialmente distinguível por seu nome, como cada atributo em uma classe é distinguível. Pode haver mais de uma parte do mesmo tipo, mas você pode separá-las por seus nomes e, presumivelmente, por suas diferentes funções no componente. Por exemplo, na [Figura 15.4](#), um componente Vendas de passagem aérea pode ter as partes Vendas separadas para voos frequentes e clientes regulares. Ambas funcionam da mesma maneira, mas a parte de voos frequentes só está disponível para clientes especiais e envolve menos chance de ficar na fila de espera e pode fornecer benefícios adicionais. Como esses componentes têm o mesmo tipo, eles devem ter nomes para distingui-los. Os outros dois componentes de tipos `AtribuiçãoDeAssento` e `GerenciamentoDeInventário` não exigem nomes, porque só há um de cada tipo no componente Vendas de passagem aérea.

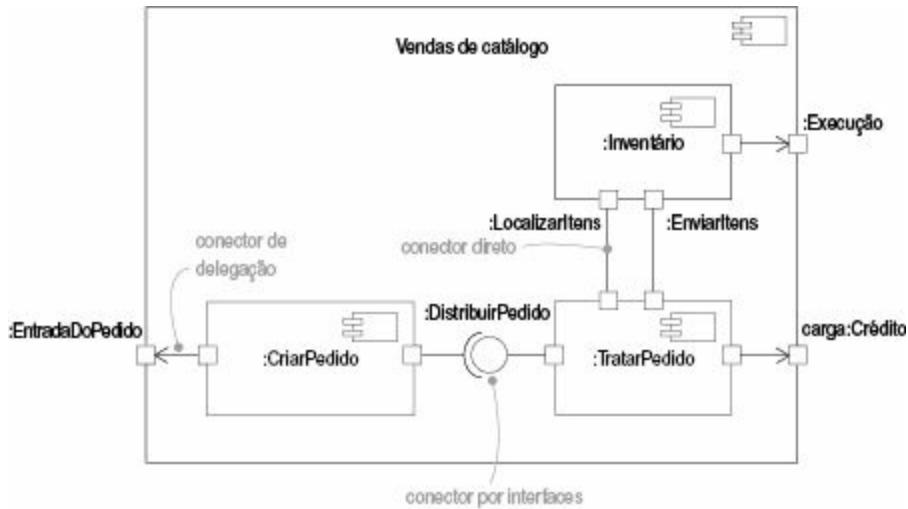


**Figura 15.4:**  
Partes do mesmo tipo

Se as partes são componentes com portas, você pode conectá-las por meio dessas portas. A regra é simples: duas portas podem ser conectadas se uma fornece uma determinada interface que a outra requer. Conectar portas significa que a porta requerida chamará a porta fornecida para obter serviços. A vantagem de portas e interfaces é que não se precisa saber mais nada. Se as interfaces são compatíveis, as portas podem ser conectadas. Uma ferramenta pode gerar automaticamente o código de chamada de um componente para outro. Elas também podem ser reconectadas a outros componentes que fornecem as mesmas interfaces, se novos componentes ficarem disponíveis. Uma conexão entre duas portas chama-se *conector*. Em uma instância do componente geral, isso representa um vínculo ou um vínculo transitório. Um vínculo é uma instância de uma associação simples. Um vínculo provisório representa um relacionamento de uso entre dois componentes. Em vez de uma associação simples, ele pode ser fornecido por um parâmetro de procedimento ou uma variável local que funciona como o destino de uma operação. A vantagem de portas e interfaces é que os dois componentes não precisam ter conhecimento um do outro no momento do projeto, desde que suas interfaces sejam compatíveis.

Você pode mostrar os conectores de duas maneiras ([Figura 15.5](#)). Se dois componentes são conectados explicitamente, diretamente ou por meio de portas, apenas desenhe uma linha entre eles ou suas portas. Por outro lado, se dois componentes estão conectados porque têm interfaces compatíveis, você

pode usar uma notação de círculo e semicírculo para mostrar que não há relacionamento inerente entre os componentes, embora eles estejam conectados dentro desse componente. Você pode substituir algum outro componente que satisfaça a interface.



**Figura 15.5:**

Conectores

Também é possível conectar portas internas a portas externas do componente geral. Esse procedimento chama-se conector de delegação, porque as mensagens na porta externa são delegadas para a porta interna, e é identificado por uma seta que vai de uma porta interna para uma externa. Você pode pensar nisso de duas maneiras: na primeira abordagem, a porta interna é igual à porta externa; ela foi movida da entrada e pode ser observada. Na segunda abordagem, qualquer mensagem para a porta externa é transmitida imediatamente para a porta interna e vice-versa. Na verdade, tanto faz, o comportamento é o mesmo nos dois casos.

A [Figura 15.5](#) mostra um exemplo com portas internas e diferentes tipos de conectores. As solicitações externas na porta EntradaDoPedido são delegadas para a porta interna do subcomponente CriarPedido. Esse componente envia seu resultado para a porta DistribuirPedido. Essa porta está conectada por um símbolo círculo-e-semicírculo com o subcomponente TratarPedido. Esse tipo de conexão implica que não haja conhecimento especial entre os dois componentes. O

resultado pode ser conectado a qualquer componente que atenda à interface DistribuirPedido. O componente TratarPedido comunica-se com o componente Inventário para localizar os itens no estoque. Esse procedimento é mostrado como um conector direto. Como nenhuma interface aparece, isso sugere que a conexão esteja mais acoplada. Após localizar os itens no estoque, o componente TratarPedido acessa um serviço Crédito externo. Isso se vê no conector de delegação com a porta externa carga. Depois que o crédito externo responde, o componente TratarPedido comunica-se com uma porta diferente EnviarItens no componente Inventário para preparar a entrega do pedido. O componente Inventário acessa o serviço Execução para realmente realizar a entrega.

Observe que o diagrama de componentes mostra a estrutura e os caminhos potenciais das mensagens do componente. O diagrama de componentes em si não mostra a sequência das mensagens por meio dos componentes. A sequência e outros tipos de informações dinâmicas podem ser encontrados nos diagramas de interação.

► Os diagramas de interação são examinados no [Capítulo 19](#).

**Nota:** A estrutura interna, incluindo portas, partes e conectores, pode ser usada como a implementação de qualquer classe, não apenas de componentes. De fato, não há muita distinção semântica entre classes e componentes. Entretanto, normalmente é útil usar a convenção de que os componentes são usados para conceitos encapsulados com estrutura interna, particularmente aqueles conceitos que não são mapeados diretamente para uma única classe na implementação.

## TÉCNICAS BÁSICAS DE MODELAGEM

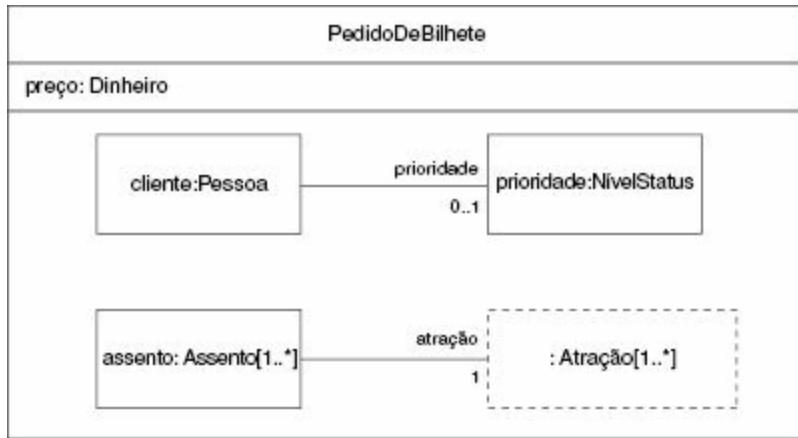
### MODELAGEM DE CLASSES ESTRUTURADAS

Uma classe estruturada pode ser usada para modelar estruturas de dados nas quais as partes têm conexões contextuais que se aplicam somente na classe. Atributos simples ou associações podem definir partes compostas de uma classe, mas as partes não podem se relacionar mutuamente em um

diagrama de classe simples. Uma classe cuja estrutura interna é mostrada com partes e conectores evita esse problema. Para fazer a modelagem de uma classe estruturada:

- › Identifique as partes internas da classe e seus tipos.
- › Forneça a cada parte um nome que indique seu propósito na classe estruturada, e não seu tipo genérico.
- › Desenhe conectores entre as partes que se comunicam ou têm relacionamentos contextuais.
- › Sinta-se à vontade para usar outras classes estruturadas como tipos de partes, mas lembre-se de que você não pode fazer conexões com partes dentro de outra classe estruturada. Conecte-se com suas portas externas.

A [Figura 15.6](#) mostra o projeto da classe estruturada PedidoDeBilhete. Essa classe tem quatro partes e um atributo simples, preço. O cliente é um objeto Pessoa. O cliente pode ter ou não um status prioridade; portanto, a parte prioridade é apresentada com multiplicidade 0..1. O conector de cliente a prioridade também tem a mesma multiplicidade. Há um ou mais assentos reservados; assento tem um valor de multiplicidade. É desnecessário mostrar um conector de cliente para assento, porque eles estão na mesma classe estruturada. Observe que Atração é desenhada com uma borda pontilhada. Isso significa que a parte é uma referência a um objeto que não pertence à classe estruturada. A referência é criada e destruída com uma instância da classe PedidoDeBilhete, mas instâncias de atração são independentes da classe PedidoDeBilhete. A parte assento está conectada à referência atração, porque o pedido pode incluir assentos para mais de uma atração, e cada reserva de assento deve ser conectada a uma atração específica. Observe na multiplicidade do conector que cada reserva de assento está conectada a exatamente um objeto de atração.



**Figura 15.6:**  
Classe estruturada

## MODELAGEM DE UMA API

Se você for um desenvolvedor que monta um sistema a partir de partes de componentes, frequentemente desejará ver as interfaces de programação de aplicações (APIs), que podem ser utilizadas para a montagem dessas partes. As APIs representam as costuras programáticas de seu sistema, que podem ser modeladas pela utilização de interfaces e componentes.

Uma API é essencialmente uma interface realizada por um ou mais componentes. Como um desenvolvedor, você realmente cuidará somente da própria interface; qual componente realiza as operações da interface não é relevante, desde que *algum* componente a realize. Sob a perspectiva do gerenciamento da configuração de um sistema, entretanto, essas realizações são importantes, pois você precisa assegurar que, ao publicar uma API, haverá alguma realização disponível para executar as obrigações da API. Felizmente, com a UML, você pode fazer a modelagem das duas perspectivas.

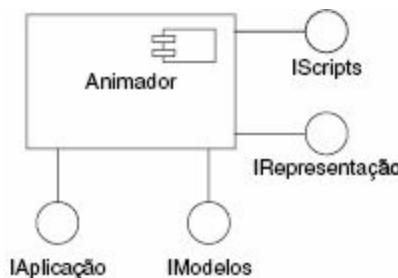
As operações associadas com qualquer API semanticamente rica serão bastante extensivas e, portanto, na maior parte do tempo, você não precisará visualizar todas essas operações de uma vez. Ao contrário, você tenderá a manter as operações na base de seus modelos e usar as interfaces para manipular esses conjuntos de operações. Se quiser construir sistemas executáveis com essas APIs, você precisará adicionar detalhes suficientes de

forma que suas ferramentas de desenvolvimento possam compilar as propriedades de suas interfaces. Juntamente com as assinaturas de cada operação, provavelmente você também desejará incluir casos de uso, explicando como utilizar cada interface.

Para fazer a modelagem de uma API:

- › Identifique as costuras existentes em seu sistema e modele cada costura como uma interface, colecionando os atributos e operações que formam essa parte.
- › Exponha somente as propriedades da interface que são importantes para serem visualizadas em determinado contexto; caso contrário, oculte essas propriedades, mantendo-as na especificação da interface para referência, conforme seja necessário.
- › Faça a modelagem da realização de cada API somente na medida em que seja importante para mostrar a configuração de uma implementação específica.

A [Figura 15.7](#) expõe as APIs de um componente de animação. Existem quatro interfaces que formam a API: IAplicação, IModelos, IRepresentação e IScripts.



**Figura 15.7:**

A modelagem de uma API

## DICAS E SUGESTÕES

Os componentes permitem que você encapsule as partes do sistema para reduzir as dependências, torná-las explícitas e melhorar a capacidade de

substituição e a flexibilidade quando o sistema precisar ser alterado no futuro.  
Um bom componente:

- › Encapsula um serviço que tem uma interface e limites bem definidos.
- › Tem estrutura interna suficiente para merecer ser descrito.
- › Não combina funcionalidade não relacionada em uma única unidade.
- › Organiza seu comportamento externo usando algumas interfaces e portas.
- › Interage somente por meio de portas declaradas.

Se você escolhe mostrar a implementação de um componente usando subcomponentes aninhados:

- › Use um número moderado de subcomponentes. Se houver subcomponentes demais para exibir confortavelmente em uma página, use níveis adicionais de decomposição em alguns dos subcomponentes.
- › Verifique se os subcomponentes interagem somente por meio de portas e conectores definidos.
- › Determine os subcomponentes que interagem diretamente com o mundo externo e modele-os com os conectores de delegação.

Ao definir um componente na UML:

- › Nomeie-o de maneira que indique claramente o seu propósito. Nomeie as interfaces assim também.
- › Nomeie os subcomponentes e as portas, se o seu significado não estiver claro a partir de seus tipos, ou se houver várias partes do mesmo tipo.
- › Oculte os detalhes desnecessários. Você não precisa mostrar todos os detalhes da implementação no diagrama de componentes.
- › Mostre a dinâmica de um componente usando diagramas de interação.

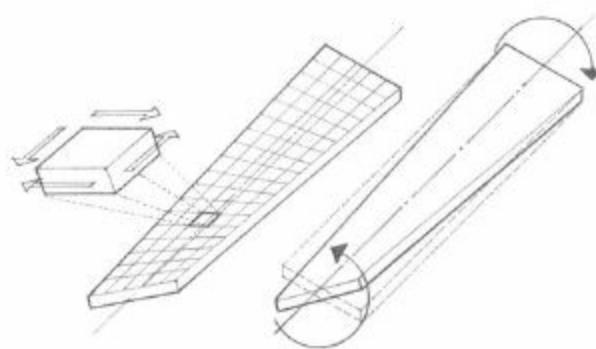
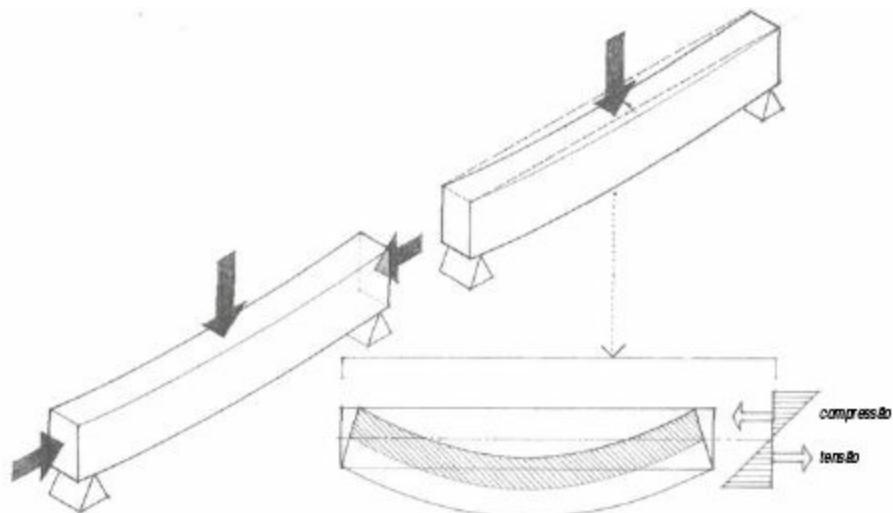
P A R T E

---

4

---

# MODELAGEM BÁSICA DE COMPORTAMENTO



CAPÍTULO

---

**16**

---

# Interações

## Neste capítulo

- » *Papéis, vínculos, mensagens, ações e sequências*
- » *Modelagem de fluxos de controle*
- » *Criação de algoritmos bem estruturados*

**E**m todos os sistemas interessantes, os objetos não permanecem apenas ociosos; eles interagem entre si passando mensagens uns para os outros. A interação é um comportamento que compreende um conjunto de mensagens trocadas entre um conjunto de objetos dentro de um contexto para a execução de um determinado propósito.

As interações são utilizadas para a modelagem do aspecto dinâmico das colaborações, representando sociedades de objetos que executam papéis específicos, todos trabalhando em conjunto para a execução de determinado comportamento que é maior do que a soma dos elementos. Esses papéis são instâncias prototípicas de classes, interfaces, componentes, nós e casos de uso. Seus aspectos dinâmicos são visualizados, especificados, construídos e documentados como fluxos de controle que poderão abranger threads sequenciais simples de um sistema, além de fluxos mais complexos envolvendo ramificações, loops, recursão e concorrência. A modelagem de cada interação pode ser feita de duas maneiras: dando-se ênfase à ordem temporal das mensagens ou dando-se ênfase à sequência das mensagens no contexto de alguma organização estrutural de objetos.

As interações bem estruturadas são semelhantes a algoritmos bem estruturados – eficientes, simples, adaptáveis e comprehensíveis.

## PRIMEIROS PASSOS

Uma edificação é algo vivo. Embora cada prédio seja construído de material estático, como tijolos, cimento, madeira, matéria plástica, vidro e ferro, essas coisas trabalham em conjunto, dinamicamente, para a execução de um comportamento útil para as pessoas que conviverão com a edificação. As portas e janelas se abrem e fecham. As luzes são ligadas e desligadas. Dutos de ventilação, termostatos, ar-condicionado e aquecimento funcionam em conjunto para regular a temperatura ambiente do prédio. Em prédios inteligentes, sensores detectam a presença ou ausência de atividade e ajustam a iluminação, o aquecimento, a refrigeração e a música, de acordo com alterações das condições ambientais. Os prédios são construídos de forma a facilitar o fluxo de pessoas e de material de um local para outro. De um modo mais sutil, as construções se destinam a se adaptar a alterações de temperatura, subindo-a ou baixando-a durante o dia e a noite e ao longo das estações. As construções bem estruturadas são projetadas de forma a ser capazes de reagir a forças dinâmicas, como ventos, tremores de terra e à movimentação de seus ocupantes, sempre mantendo o equilíbrio do prédio.

- As diferenças entre a construção de uma casa de cachorro e de um prédio de vários andares são examinadas no [Capítulo 1](#).

Os sistemas complexos de software são semelhantes. Um sistema de linhas aéreas poderá administrar muitos terabytes de informações que permanecem intocados em um mesmo disco na maior parte do tempo, sendo ativados somente por eventos externos, como o registro de uma reserva, a direção de uma aeronave ou a programação de um voo. Em sistemas reativos, como aqueles encontrados no computador de um forno de microondas, os objetos entram em ação e suas tarefas são executadas quando o sistema é estimulado por eventos como o pressionar de um botão ou no decorrer de determinado tempo.

Na UML, a modelagem dos aspectos estáticos do sistema é feita pela utilização de elementos como diagramas de classes e diagramas de objetos.

Esses diagramas permitem visualizar, especificar, construir e documentar os itens existentes no sistema, incluindo classes, interfaces, componentes, nós e casos de uso e respectivas instâncias, juntamente com a forma como esses itens se encontram relacionados uns com os outros.

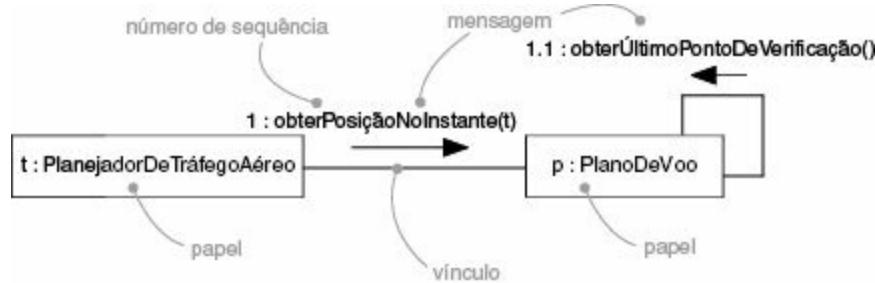
- A modelagem de aspectos estruturais de um sistema é examinada nas Partes 2 e 3; você também pode fazer a modelagem de aspectos dinâmicos de um sistema utilizando máquinas de estados, conforme é apresentado no [Capítulo 22](#); os diagramas de objetos são explicados no [Capítulo 14](#); os diagramas de interação são examinados no [Capítulo 19](#); as colaborações são apresentadas no [Capítulo 28](#).

Na UML, a modelagem dos aspectos dinâmicos do sistema é feita pela utilização das interações. Assim como os diagramas de objetos, uma interação define estaticamente o conjunto de estados para o seu comportamento pela inclusão de todos os objetos que trabalham em conjunto para a execução de determinada ação. Indo além desses diagramas de objetos, porém, as interações também introduzem mensagens que são enviadas de objeto a objeto. Com muita frequência, as mensagens envolvem a chamada a uma operação ou o envio de um sinal; as mensagens ainda poderão abranger a criação e a destruição de outros objetos.

A interação é empregada para a modelagem do fluxo de controle de uma operação, uma classe, um componente, um caso de uso ou do sistema como um todo. Com a utilização de diagramas de interação, é possível analisar esses fluxos de duas maneiras. Primeiro, pode-se focalizar a maneira como as mensagens são enviadas ao longo do tempo. Segundo, pode-se focalizar os relacionamentos estruturais entre os objetos de uma interação e então considerar como as mensagens são passadas no contexto dessa estrutura.

A UML fornece uma representação gráfica das mensagens, conforme mostra a [Figura 16.1](#). Essa notação torna possível visualizar a mensagem de uma maneira que permite dar ênfase às partes mais importantes: seu nome, parâmetros (se houver) e sequência. As mensagens são representadas

graficamente por linhas com uma direção e quase sempre incluem os nomes de suas operações.



**Figura 16.1:**  
Mensagens, vínculos e sequências

## TERMOS E CONCEITOS

Uma *interação* é um comportamento que compreende um conjunto de mensagens trocadas entre um conjunto de objetos em determinado contexto para a realização de um propósito. Uma *mensagem* é a especificação de uma comunicação entre objetos, a qual contém informações relacionadas ao que se espera resultar dessa atividade.

## CONTEXTO

Uma interação pode ser encontrada em qualquer parte em que os objetos estejam vinculados entre si. As interações serão encontradas na colaboração de objetos que existem no contexto do sistema ou subsistema. As interações também serão encontradas no contexto de uma operação. Por fim, as interações serão encontradas no contexto de uma classe.

- Os *diagramas de estrutura interna* mostram a conexão estrutural existente entre papéis, conforme é examinado no [Capítulo 15](#); os objetos são apresentados no [Capítulo 14](#); os sistemas e subsistemas são apresentados no [Capítulo 32](#); as colaborações são explicadas no [Capítulo 27](#).

Com muita frequência, você encontrará interações na colaboração de objetos existentes no contexto do sistema ou subsistema como um todo. Por exemplo, em um sistema para comércio na Web, haverá objetos no cliente (como as instâncias das classes `PedidoDeReserva` e `FormulárioDePedido`) interagindo uns com os outros. Também haverá objetos no cliente (mais uma vez, como as instâncias de `PedidoDeReserva`) interagindo com objetos encontrados no servidor (como as instâncias de `VoltarGerenciadorPedido`). Essas interações, portanto, não só envolvem colaborações localizadas de objetos (como as interações ao redor de `FormulárioDePedido`), mas também poderão percorrer diferentes níveis conceituais do sistema (como as interações ao redor de `VoltarGerenciadorPedido`).

Você também encontrará interações entre objetos na implementação de uma operação. Os parâmetros de uma operação, quaisquer variáveis locais da operação ou quaisquer variáveis globais (mas ainda visíveis à operação) poderão interagir entre si para a execução do algoritmo da implementação dessa operação. Por exemplo, a chamada à operação `moverParaPosição` (`p : Posição`), definida para uma classe de um robô móvel, envolverá a interação de um parâmetro (`p`), um objeto global da operação (como o objeto `posiçãoAtual`) e possivelmente vários objetos locais (como as variáveis locais utilizadas pela operação para o cálculo de pontos intermediários no caminho até a nova posição).

- As operações são examinadas nos Capítulos 4 e 9; a modelagem de operações é apresentada nos Capítulos 20 e 28.

Por fim, você encontrará interações no contexto de uma classe. As interações podem ser empregadas para visualizar, especificar, construir e documentar a semântica de uma classe. Por exemplo, para compreender o significado de uma classe `AgenteRastreamentoRaio`, você poderá criar interações mostrando como os atributos dessa classe colaboram com uma outra (e com objetos globais para as instâncias da classe e com parâmetros definidos nas operações da classe).

- ➡ As classes são examinadas nos Capítulos 4 e 9.

**Nota:** Uma interação também poderá ser encontrada na representação de um componente, um nó ou um caso de uso, cada um dos quais é realmente um tipo de classificador na UML. No contexto de um caso de uso, uma interação representa um cenário que, por sua vez, representa um thread ao longo da ação do caso de uso.

- ➡ Os componentes são examinados no [Capítulo 15](#); os nós são apresentados no [Capítulo 27](#); os casos de uso são explicados no [Capítulo 17](#); a modelagem da realização dos casos de uso é apresentada no [Capítulo 28](#); os classificadores aparecem no [Capítulo 9](#).

## OBJETOS E PAPÉIS

Os objetos que participam em uma interação são itens concretos ou prototípicos. Assim como uma coisa concreta, os objetos representam algo no mundo real. Por exemplo, p, uma instância da classe Pessoa, poderá denotar um determinado ser humano. Alternativamente, como um item prototípico, p poderá representar qualquer instância de Pessoa.

**Nota:** Em uma colaboração, os objetos encontrados são algo prototípico que desempenham determinados papéis, e não objetos específicos do mundo real, embora, às vezes, seja útil descrever colaborações entre determinados objetos.

No contexto de uma interação, você encontrará instâncias de classes, componentes, nós e casos de uso. Apesar de as interfaces e as classes abstratas, por definição, não poderem ter qualquer instância direta, você poderá encontrar instâncias desses itens em uma interação. Essas instâncias não representam instâncias diretas da classe abstrata ou da interface, mas poderão representar, respectivamente, instâncias indiretas (ou prototípicas) de qualquer classe-filha concreta da classe abstrata de alguma classe concreta que realiza essa interface.

- ➡ As classes abstratas são examinadas no [Capítulo 4](#); as interfaces são apresentadas no [Capítulo 11](#).

Você pode pensar em um diagrama de objetos como uma representação do aspecto estático de uma interação, definindo o estágio para a interação pela especificação de todos os objetos que trabalham em conjunto. As interações vão além, introduzindo uma sequência dinâmica de mensagens que poderão ser passadas pelos vínculos que conectam esses objetos.

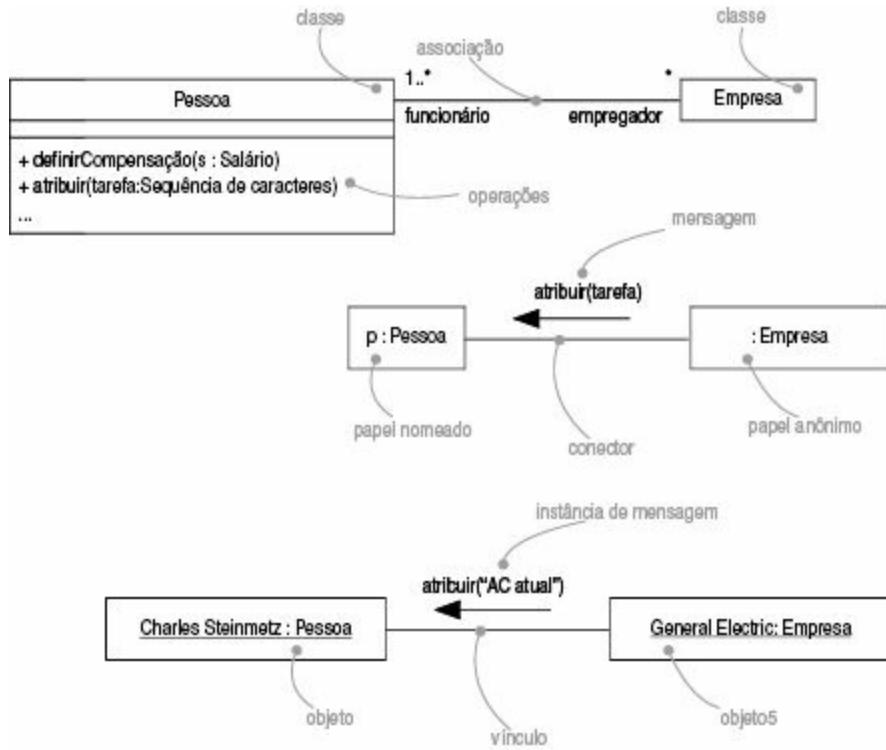
- ➡ As instâncias são examinadas no [Capítulo 13](#); os diagramas de objetos são apresentados no [Capítulo 14](#).

## VÍNCULOS E CONECTORES

Um vínculo é uma conexão semântica existente entre os objetos. Em geral, um vínculo é uma instância de uma associação. Conforme mostra a [Figura 16.2](#), sempre que uma classe tem uma associação com outra classe, poderá haver um vínculo entre as instâncias das duas classes; sempre que existe um vínculo entre dois objetos, um objeto é capaz de enviar uma mensagem ao outro.

- ➡ As associações são examinadas nos [Capítulos 5 e 10](#); os conectores e papéis são apresentados no [Capítulo 15](#).

Um vínculo especifica um caminho pelo qual um objeto pode enviar uma mensagem para outro (ou para o mesmo) objeto. Na maior parte do tempo, é suficiente especificar que esse caminho existe. Caso seja necessário ser mais preciso sobre a maneira como esse caminho existe, você pode incluir um adorno na extremidade apropriada do vínculo, utilizando qualquer uma das seguintes restrições.



**Figura 16.2:**  
Vínculos, associações e conectores

association	Especifica que o objeto correspondente é visível pela associação.
self	Especifica que o objeto correspondente é visível, por ser o emissor da operação.
global	Especifica que o objeto correspondente é visível, por estar no escopo que o contém.
local	Especifica que o objeto correspondente é visível, por estar em um escopo local.
parameter	Especifica que o objeto correspondente é visível, por ser um parâmetro.

**Nota:** Como uma instância de uma associação, um vínculo poderá ser representado com a maioria dos adornos apropriados às associações, como um nome, o nome do papel da associação, a navegação e a agregação. A multiplicidade, entretanto, não se aplica aos vínculos, pois são instâncias de uma associação.

Na maioria dos modelos, estamos mais interessados em objetos prototípicos e vínculos no mesmo contexto do que em objetos e vínculos individuais. Um objeto prototípico chama-se *papel*; um vínculo prototípico chama-se *conector*; o contexto é uma colaboração ou a estrutura interna de um classificador. A multiplicidade de papéis e conectores é definida em relação ao contexto no qual se encontram. Por exemplo, uma multiplicidade de 1 em um papel significa que um objeto representa o papel de cada objeto que representa o contexto. Uma colaboração ou estrutura interna pode ser usada muitas vezes, como uma declaração de classe; cada uso está ligado a um conjunto separado de objetos e vínculos para o contexto, papéis e vínculos.

- Papéis, conectores e estrutura interna são examinados no [Capítulo 15](#); colaborações são apresentadas no [Capítulo 28](#).

A [Figura 16.2](#) mostra um exemplo. A parte superior da figura mostra um diagrama de classes que declara as classes Pessoa e Empresa e a associação funcionário-empregador entre elas. No meio da figura, é exibido o conteúdo de uma colaboração AtribuiçãoDeTarefa que atribui um funcionário a um trabalho. Ele tem dois papéis e um conector entre eles. A parte inferior da figura mostra uma instância dessa colaboração, na qual há objetos e vínculos ligados a papéis e conectores. Uma mensagem concreta na parte inferior representa a declaração de mensagem prototípica na colaboração.

## MENSAGENS

Suponha que você tenha um conjunto de objetos e um conjunto de vínculos conectando esses objetos. Caso isso seja tudo que você tenha, então você terá um modelo completamente estático, que pode ser representado por

qualquer diagrama de objetos. Os diagramas de objetos permitem fazer a modelagem do estado de uma sociedade de objetos em um determinado momento e são de grande ajuda quando se deseja visualizar, especificar, construir ou documentar a estrutura estática de um objeto.

- » *Os diagramas de objetos são examinados no Capítulo 14.*

Suponha que você queira fazer a modelagem do estado de mudança de uma sociedade de objetos ao longo de determinado período. Pense nisso como se fosse filmar um conjunto de objetos, em que cada quadro representa um momento sucessivo. Se esses objetos não estiverem totalmente ociosos, você verá objetos passando mensagens a outros objetos, enviando eventos e chamando operações. Além disso, em cada um dos quadros, é possível visualizar explicitamente o papel e o estado atual de instâncias individuais.

- » *As operações são examinadas nos Capítulos 4 e 9; os eventos são apresentados no Capítulo 21; as instâncias são explicadas no Capítulo 13.*

Uma mensagem é a especificação de uma comunicação entre objetos, que contém informações sobre a expectativa a respeito do resultado dessa atividade. O destinatário da instância de uma mensagem poderá ser considerado uma instância de um evento. (Uma *ocorrência* é o nome da UML para uma instância de um evento.)

Quando uma mensagem é passada, a ação resultante normalmente ocorre no destinatário. Uma ação poderá resultar em uma mudança de estado do objeto de destino e dos objetos acessíveis a partir dele.

Na UML, você poderá fazer a modelagem de vários tipos de mensagens:

» Call

Invoca uma operação em um objeto; o objeto poderá enviar uma mensagem a si mesmo, resultando em chamada local de uma operação.

» Return	Retorna um valor para quem o solicitou.
» Send	Envia um sinal para um objeto.
» Create	Cria um objeto.
» Destroy	Destroi um objeto; o objeto poderá cometer suicídio, destruindo a si próprio.

► As operações são examinadas nos Capítulos 4 e 9; os sinais são apresentados no Capítulo 21.

**Nota:** A UML também permite fazer a modelagem de ações complexas. Além dos cinco tipos básicos de ações relacionados anteriormente, você pode anexar uma sequência arbitrária de caracteres a uma mensagem, contendo expressões complexas. A UML não especifica a sintaxe nem a semântica dessas sequências de caracteres; espera-se que as ferramentas forneçam várias linguagens de ações ou usem a sintaxe de linguagens de programação.

A UML proporciona uma distinção visual entre esses tipos de mensagens, conforme mostra a Figura 16.3.

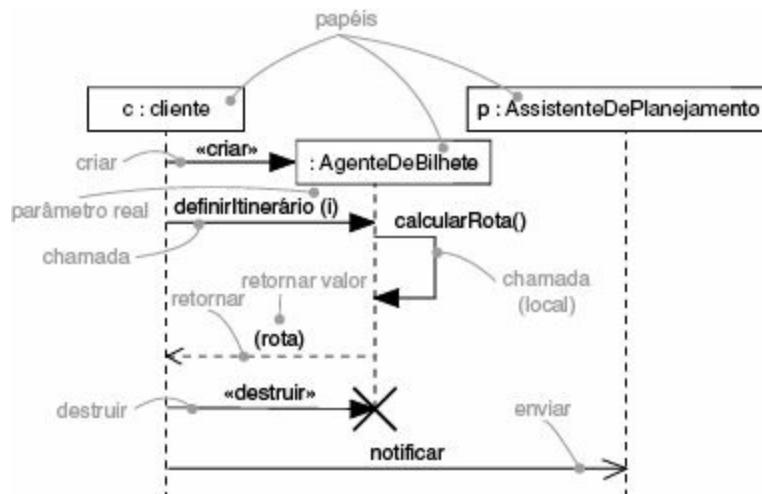


Figura 16.3:  
Mensagens

- *Criar e destruir são visualizados como estereótipos, conforme é examinado no Capítulo 6; a distinção entre mensagens síncronas e assíncronas é mais relevante no contexto da concorrência, conforme é apresentado no Capítulo 23.*

A chamada será o tipo mais comum de mensagem cuja modelagem você fará. Na chamada, um objeto invoca uma operação a outro (ou ao mesmo) objeto. O objeto não pode chamar qualquer operação aleatória. Se um objeto, como c no exemplo anterior, chamar a operação definirItinerário em uma instância da classe AgenteDeBilhete, a operação definirItinerário não deve estar somente definida para a classe AgenteDeBilhete (ou seja, deve estar declarada na classe AgenteDeBilhete ou em uma de suas classes-mãe), mas também deve estar visível ao c que a está chamando.

- *As classes são examinadas nos Capítulos 4 e 9.*

**Nota:** Linguagens como C++ têm tipos definidos estaticamente (apesar de polimórficos). Isso significa que a validade de uma chamada é verificada em tempo de compilação. Linguagens como Smalltalk, entretanto, têm tipos definidos dinamicamente. Isso significa que não é possível determinar se um objeto pode receber adequadamente uma mensagem a não ser em tempo de execução. Na UML, em geral um modelo bem formado pode ser verificado estaticamente por uma ferramenta, porque, em tempo de modelagem, o desenvolvedor tipicamente conhece a intenção da operação.

Quando um objeto chama uma operação ou envia um sinal a outro objeto, é possível fornecer parâmetros atuais para a mensagem. De modo semelhante, quando um objeto retorna o controle a outro objeto, também é possível fazer a modelagem do valor retornado.

As mensagens também podem corresponder ao envio de sinais. Um sinal é um valor de objeto comunicado a um objeto de destino assincronamente. Após o envio de um sinal, o objeto enviado continua sua própria execução. Quando o objeto de destino recebe a mensagem de sinal, ele decide de forma independente o que fazer. Normalmente, os sinais acionam transições na máquina de estados do objeto de destino. Acionar uma transição faz o objeto

de destino executar as ações e mudar para um novo estado. Em um sistema assíncrono de transmissão de mensagens, os objetos de comunicação são executados concorrente e independentemente. Eles compartilham os valores somente passando mensagens, de maneira que não há perigo de conflito na memória compartilhada.

**Nota:** Você também pode qualificar uma operação pela classe ou interface em que é declarada. Por exemplo, a chamada à operação `register` em uma instância de `Student` invocaria polimorficamente qualquer operação com o nome correspondente na hierarquia da classe `Student`; a chamada a `IMember::register` invocaria a operação especificada na interface `IMember` (e realizada por alguma classe adequada, também da hierarquia da classe `Student`).

► As interfaces são examinadas no [Capítulo 11](#).

## SEQUENCIAMENTO

Quando um objeto passa uma mensagem a outro objeto (na verdade, delegando alguma ação ao destinatário), o objeto destinatário poderá, por sua vez, enviar a mensagem a um outro objeto, que poderá enviá-la a um objeto diferente e assim por diante. Esse fluxo de mensagens forma uma sequência. Qualquer sequência deve ter um ponto de partida; o início de toda sequência é a raiz de algum processo ou thread. Além disso, qualquer sequência será continuada durante o tempo de vida do processo ou thread a que ela pertence. Um sistema sem interrupções, como aqueles que você poderá encontrar em controles de dispositivos em tempo real, continuará sendo executado enquanto permanecer ativo o nó em que ele se encontra.

► Os processos e os threads são examinados no [Capítulo 23](#).

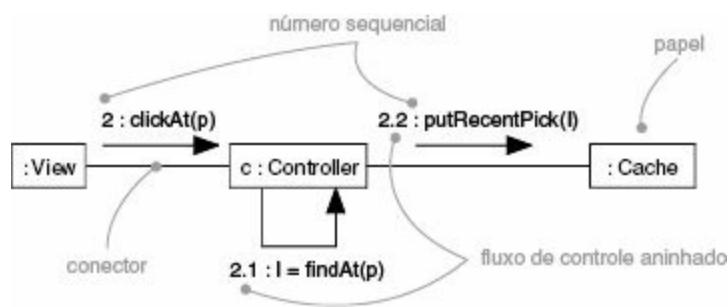
Cada processo e thread em um sistema define um fluxo diferente de controle e, em cada um desses fluxos, as mensagens são ordenadas em sequência de acordo com o tempo. Para uma melhor visualização da sequência de uma mensagem, você pode fazer explicitamente a modelagem da ordem da mensagem em relação ao início da sequência, colocando na

mensagem um prefixo com um conjunto de números da sequência, separados pelo sinal de dois-pontos.

► Os sistemas são examinados no [Capítulo 32](#).

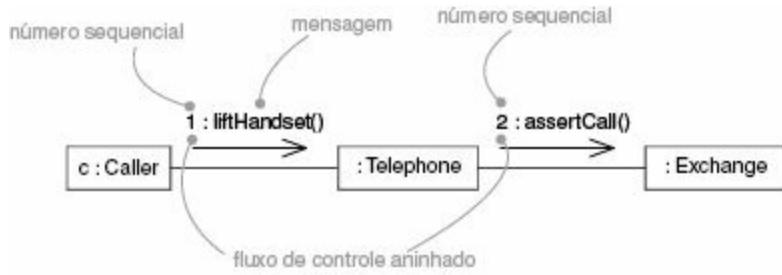
Um diagrama de comunicação mostra o fluxo de mensagens entre papéis em uma colaboração. As mensagens transitam pelas conexões da colaboração, como na [Figura 16.4](#).

Com maior frequência, você pode especificar um fluxo de controle procedural ou aninhado, representado por uma linha cheia e com uma seta, conforme mostra a [Figura 16.4](#). Nesse caso, a mensagem `findAt` é especificada como a primeira mensagem aninhada na segunda mensagem da sequência (2.1).



**Figura 16.4:**  
Sequência procedural

Menos comum, mas também possível, conforme mostra a [Figura 16.5](#), você pode especificar um fluxo de controle contínuo, representado por uma seta fina, com a finalidade de fazer a modelagem da progressão não procedural do controle, passo a passo. Nesse caso, a mensagem `assertCall` é especificada como a segunda mensagem da sequência.



**Figura 16.5:**  
Sequência contínua

**Nota:** A distinção entre sequências assíncronas e procedurais é importante no mundo moderno da computação concorrente. Para mostrar o comportamento geral de um sistema de objetos concorrentes, use a passagem de mensagens assíncrona. Esse é o caso mais geral. Quando os objetos que fazem a chamada podem fazer uma solicitação e aguardar uma resposta, você pode usar o fluxo de controle procedural. O fluxo de controle procedural é familiar das linguagens de programação tradicionais, mas lembre-se de que uma série de chamadas procedurais aninhadas resulta em uma pilha de objetos bloqueados que estão temporariamente incapacitados de fazer qualquer coisa; portanto, não é muito útil se eles representarem servidores ou recursos compartilhados.

Ao fazer a modelagem de interações que envolvem vários fluxos de controle, é muito importante identificar o processo ou thread que enviou uma determinada mensagem. Na UML, é possível diferenciar um fluxo de controle de outro, utilizando, como prefixo, o número da sequência da mensagem, com o nome do processo ou thread encontrado na raiz da sequência. Por exemplo, a expressão:

D5 : ejectHatch(3)

especifica que a operação ejectHatch é iniciada (com o argumento atual 3) como a quinta mensagem na sequência cuja raiz é o processo ou thread chamado D.

- Os processos e threads são examinados no [Capítulo 23](#); fluxos de controle assíncronos também podem ser especificados, representados por uma meia linha com uma seta, conforme é apresentado no [Capítulo 23](#).

Não apenas é possível exibir os argumentos atuais enviados juntamente com uma operação ou com um sinal no contexto de uma interação, mas você também é capaz de mostrar os valores retornados de uma função. Conforme mostra a expressão apresentada a seguir, o valor `p` é retornado pela operação `find`, iniciada com o parâmetro atual “Rachelle”. Essa é uma sequência aninhada, iniciada como a segunda mensagem aninhada na terceira mensagem aninhada na primeira mensagem da sequência. No mesmo diagrama, `p` pode então ser utilizado como um parâmetro atual em outras mensagens.

1.3.2 : `p := find("Rache11e")`

**Nota:** Na UML, também é possível fazer a modelagem de formas mais complexas de sequências, como a iteração, ramificação e as mensagens protegidas. Além disso, para a modelagem de restrições de tempo, como as que você poderá encontrar em sistemas de tempo real, é possível associar marcas de tempo e uma sequência. No caso de outras formas de mensagens, mais exóticas, como as de recusa (balking) e expiração de tempo (timeout), sua modelagem pode ser feita definindo um estereótipo apropriado para a mensagem.

- Iteração, ramificação e mensagens protegidas são examinadas no [Capítulo 19](#); as marcas de tempo são apresentadas no [Capítulo 24](#); os estereótipos e as restrições são explicados no [Capítulo 6](#).

## CRIAÇÃO, MODIFICAÇÃO E DESTRUIÇÃO

Na maior parte do tempo, os objetos que você mostra participando de uma interação existem ao longo de toda a duração da interação. Entretanto, em algumas interações, os objetos poderão ser criados (especificados por uma mensagem `create`) e destruídos (especificados por uma mensagem `destroy`). O mesmo é verdadeiro em relação aos vínculos: os relacionamentos entre objetos poderão ocorrer e desaparecer. Para especificar se um objeto ou vínculo entra e/ou sai durante uma interação, você pode anexar uma nota ao seu papel em um diagrama de comunicação.

Durante uma interação, um objeto tipicamente muda os valores de seus atributos, seu estado ou seus papéis. Você pode representar as modificações

de um objeto em um diagrama de sequência, mostrando o estado ou os valores da linha de vida.

Em um diagrama de sequência, o tempo de vida, a criação e a destruição de objetos ou papéis são explicitamente apresentados pela extensão vertical de suas linhas de vida. Em um diagrama de comunicação, a criação e a destruição devem ser indicadas usando notas. Use diagramas de sequência se for importante mostrar os tempos de vida do objeto.

- As linhas de vida são examinadas no [Capítulo 19](#).

## REPRESENTAÇÃO

Ao fazer a modelagem de uma interação, tipicamente você inclui objetos (cada um desempenhando um papel específico) e mensagens (cada uma representando a comunicação entre os objetos, com alguma ação como resultado).

Esses papéis e mensagens envolvidos em uma interação podem ser visualizados de duas maneiras: com uma ênfase na ordenação temporal de suas mensagens e com uma ênfase na organização estrutural dos papéis que enviam e recebem mensagens. Na UML, o primeiro tipo de representação é chamado diagrama de sequência; o segundo tipo de representação é chamado diagrama de comunicação. Tanto o diagrama de sequência como o diagrama de comunicação são tipos de diagramas de interação. (A UML também tem um tipo mais especializado de diagrama de interação, chamado de *diagrama de temporização*, que mostra os momentos exatos em que as mensagens são trocadas pelos papéis. Esse diagrama não é abordado neste livro. Consulte o *UML Reference Manual* para obter mais informações.)

- Os diagramas de interação são examinados no [Capítulo 19](#).

Os diagramas de sequências e de comunicação são inteiramente isomórficos. Isso significa que um poderá ser transformado no outro sem qualquer perda de informação. Entretanto, existem algumas diferenças

visuais. Em primeiro lugar, os diagramas de sequências permitem fazer a modelagem da linha de vida de um objeto. A linha de vida de um objeto representa a existência desse objeto em um determinado período, possivelmente abrangendo sua criação e destruição. Em segundo lugar, os diagramas de comunicação permitem fazer a modelagem de vínculos estruturais que possam existir entre os objetos em uma interação.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DO FLUXO DE CONTROLE

O propósito mais comum para o qual você utilizará as interações é a modelagem do fluxo de controle que caracteriza o comportamento do sistema como um todo, incluindo casos de uso, padrões, mecanismos e frameworks, ou o comportamento de uma classe ou operação individual. Enquanto as classes, as interfaces, os componentes, os nós e seus relacionamentos permitem a modelagem de aspectos estáticos do sistema, as interações permitem a modelagem dos aspectos dinâmicos.

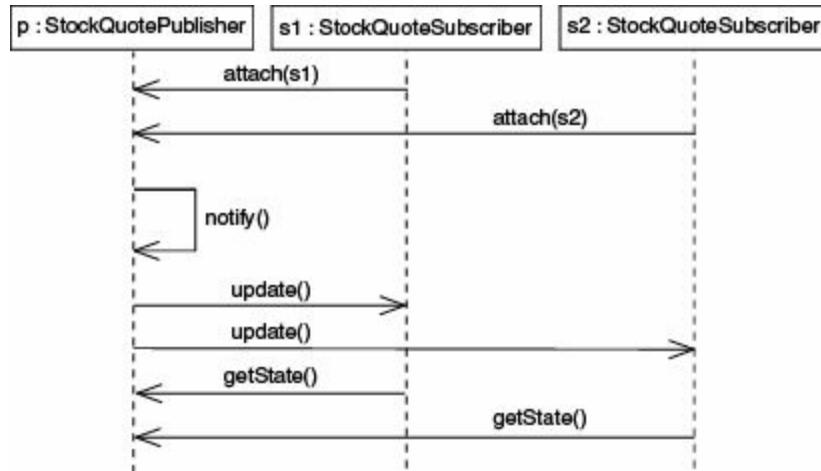
- ➡ *Os casos de uso são examinados no [Capítulo 17](#); os padrões e frameworks são apresentados no [Capítulo 29](#); as classes e as operações são explicadas nos Capítulos [4](#) e [9](#); as interfaces aparecem no [Capítulo 11](#); os componentes, no [Capítulo 15](#); os nós, no [Capítulo 27](#); você também pode fazer a modelagem dos aspectos dinâmicos do sistema utilizando máquinas de estado, conforme é examinado no [Capítulo 22](#).*

Ao fazer a modelagem de uma interação, você essencialmente monta um enredo das ações realizadas em um conjunto de objetos. Técnicas como cartões CRC são de grande ajuda para você descobrir e pensar a respeito dessas interações.

Para fazer a modelagem de um fluxo de controle:

- › Defina o contexto para a interação, seja o sistema como um todo, uma classe ou uma operação individual.
- › Defina o estágio para a interação, identificando quais objetos desempenham algum papel; defina suas propriedades iniciais, incluindo os valores de seus atributos, estado e papel. Dê nome aos papéis.
- › Se o seu modelo estiver dando ênfase à organização estrutural desses objetos, identifique os vínculos que os conectam, relevantes para os caminhos das comunicações realizadas nessa interação. Especifique a natureza dos vínculos, utilizando as restrições e os estereótipos-padrão da UML, conforme seja necessário.
- › Em ordem temporal, especifique as mensagens passadas de um objeto para outro. Conforme necessário, destaque os tipos diferentes de mensagens; inclua parâmetros e valores retornados para o transporte dos detalhes necessários a essa interação.
- › Também para o transporte dos detalhes necessários a essa interação, coloque adornos em cada objeto em todos os momentos com seu estado e papel.

Por exemplo, a [Figura 16.6](#) mostra um conjunto de objetos que interagem no contexto de um mecanismo de publicações e assinaturas (uma instância do padrão de projeto do observador). Essa figura inclui três objetos: `p` (`StockQuote-Pub1isher`), `s1` e `s2` (ambas instâncias de `StockQuoteSubscriber`). Essa figura é o exemplo de um diagrama de sequência, que dá ênfase à ordem temporal das mensagens.



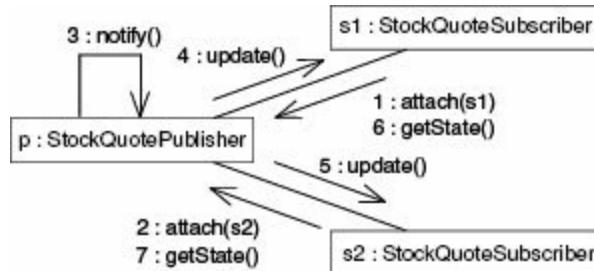
**Figura 16.6:**

Fluxo de controle por tempo

► Os diagramas de sequência são examinados no [Capítulo 19](#).

A [Figura 16.7](#) é semanticamente equivalente à anterior, mas está definida como um diagrama de comunicação, que dá ênfase à organização estrutural dos objetos. Essa figura mostra o mesmo fluxo de controle, mas também proporciona uma visualização dos vínculos existentes entre esses objetos.

► Os diagramas de comunicação são examinados no [Capítulo 19](#).



**Figura 16.7:**

Fluxo de controle pela organização

## DICAS E SUGESTÕES

Ao fazer a modelagem de interações na UML, lembre-se de que toda interação representa o aspecto dinâmico de uma sociedade de objetos. Uma

interação bem estruturada:

- › É simples e deve abranger somente aqueles objetos que trabalham em conjunto para a execução de algum comportamento maior do que a soma de todos esses elementos.
- › Tem um contexto claro e poderá representar a interação de objetos no contexto de uma operação, de uma classe ou do sistema como um todo.
- › É eficiente e deverá desempenhar seu comportamento com ótimo equilíbrio de tempo e de recursos.
- › É adaptável e os elementos de uma interação que provavelmente serão alterados deverão ficar isolados para facilitar suas modificações.
- › É comprehensível e deverá ser objetiva, sem envolver soluções incompletas, efeitos secundários ocultos ou uma semântica obscura.

Ao definir uma interação na UML:

- › Escolha a ênfase para a interação. Você pode dar ênfase à ordem das mensagens ao longo do tempo ou à sequência das mensagens no contexto de alguma organização estrutural de objetos. Não é possível dar ênfase a esses dois aspectos ao mesmo tempo.
- › Observe que os eventos em subsequências separadas são apenas parcialmente ordenados. Cada subsequência é ordenada, mas os tempos relativos de eventos em diferentes subsequências não são fixos.
- › Mostre somente as propriedades de cada objeto (como os valores dos atributos, papéis e estados) que sejam importantes para a compreensão da interação em seu contexto.
- › Mostre somente as propriedades de cada mensagem (como os seus parâmetros, semântica de concorrência e valor retornado) que sejam importantes para a compreensão da interação em seu contexto.

CAPÍTULO

---

17

---

# Casos de Uso

## Neste capítulo

- » Casos de uso, atores, inclusão e extensão
- » Modelagem do comportamento de um elemento
- » Realização de casos de uso com colaborações

**N**enhum sistema existe isoladamente. Todo sistema interessante interage com atores humanos ou autômatos que utilizam esse sistema para algum propósito e esses atores esperam que o sistema se comporte de acordo com as maneiras previstas. Um caso de uso especifica o comportamento de um sistema ou de parte de um sistema e é uma descrição de um conjunto de sequências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator.

Os casos de usos podem ser aplicados para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementado. Os casos de uso fornecem uma maneira para os desenvolvedores chegarem a uma compreensão comum com os usuários finais do sistema e com os especialistas do domínio. Além disso, os casos de uso servem para ajudar a validar a arquitetura e para verificar o sistema à medida que ele evolui durante seu desenvolvimento. À proporção que você implementa o seu sistema, esses casos de uso são realizados por colaborações cujos elementos trabalham em conjunto para a execução de cada caso de uso.

Casos de uso bem estruturados denotam somente o comportamento essencial do sistema ou subsistema e não são amplamente gerais, nem muito específicos.

## PRIMEIROS PASSOS

Uma casa bem projetada é muito mais do que um grupo de paredes reunidas para sustentar um teto como proteção ao tempo. Ao trabalhar com seu arquiteto e projetar sua casa, você dará muita importância ao modo como essa casa será utilizada. Se você gosta de dar festas, pensará no fluxo de pessoas pelos cômodos de maneira a facilitar a conversação e evitar pontos sem saída que resultem em congestionamentos. Caso pretenda preparar refeições para sua família, desejará ter certeza de que sua cozinha está projetada para armazenar mantimentos e utensílios de modo eficiente. Até o planejamento do caminho a ser percorrido do carro à cozinha com a finalidade de descarregar mantimentos afetará o modo como eventualmente os cômodos estarão conectados uns com os outros. Caso sua família seja grande, será preciso pensar na utilização dos banheiros. Planejar o número adequado e a posição correta dos banheiros logo no início do projeto reduzirá significativamente o risco de confusões pela manhã, quando sua família precisar sair para o trabalho e para a escola. Se você tiver filhos adolescentes, essa será uma questão de alto risco, pois o custo emocional de um mau planejamento será muito grande.

Pensar a respeito da maneira como você e sua família utilizarão a casa é um exemplo de uma análise baseada em caso de uso. Você considera os vários modos como a casa será utilizada e esses casos de uso orientam a arquitetura. Muitas famílias terão os mesmos tipos de casos de uso – usamos casas para comer, dormir, criar os filhos e acumular memórias. Toda família também terá seus próprios casos de uso especiais ou variações e alguns casos básicos. As necessidades de uma família grande, por exemplo, são diferentes das necessidades de um adulto solteiro que tenha acabado de concluir a faculdade. São essas variações que têm o maior impacto na forma final da casa.

Um fator importante para a criação de casos de uso como esses está relacionado ao fato de você criá-los sem especificar o modo como os casos de uso são implementados. Por exemplo, você pode especificar como um

sistema de caixa eletrônico deve funcionar, definindo, em casos de uso, como os usuários deverão interagir com o sistema; você não precisa saber nada sobre o que deverá acontecer dentro do caixa eletrônico. Os casos de uso especificam o comportamento desejado; eles não determinam como esse comportamento será executado. A vantagem disso é permitir que você (como usuário final e especialista do domínio) se comunique com seus desenvolvedores (que constroem sistemas satisfazendo aos seus requisitos) sem se preocupar com detalhes. Esses detalhes aparecerão, mas os casos de uso permitem que você focalize os pontos que considera de maior risco.

Na UML, todos esses comportamentos são modelados como casos de uso que poderão ser especificados independentemente de suas realizações. Um caso de uso é uma descrição de um conjunto de sequências de ações, incluindo variantes que um sistema realiza para produzir um resultado observável do valor de um ator. Existem várias partes importantes para essa definição.

No nível do sistema, um caso de uso descreve um conjunto de sequências, cada uma representando a interação de itens externos ao sistema (seus atores) com o próprio sistema (e suas principais abstrações). Esses comportamentos, na verdade, são funções em nível de sistema, que você utiliza para visualizar, especificar, construir e documentar o comportamento pretendido do sistema durante a captação e análise de requisitos. Um caso de uso representa um requisito funcional do sistema como um todo. Por exemplo, um caso de uso central em um banco é o processamento de empréstimos.

- ➡ As interações são examinadas no [Capítulo 16](#); os requisitos são apresentados no [Capítulo 6](#).

Um caso de uso envolve a interação dos atores com o sistema. Um ator representa um conjunto coerente de papéis que os usuários dos casos de uso desempenham quando interagem com esses casos. Os atores podem ser humanos ou sistemas automatizados. Por exemplo, na modelagem de um

banco, o processamento de um empréstimo envolve, entre outras coisas, a interação entre o cliente e quem concede o empréstimo.

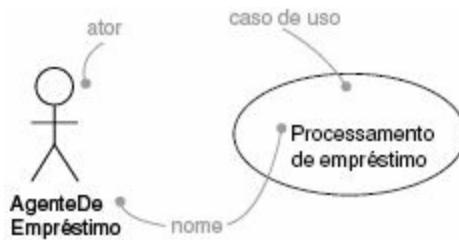
Um caso de uso poderá ter variantes. Qualquer sistema interessante terá casos de uso que são versões especializadas de outros casos de uso, que são incluídos como parte de outro caso de uso e que estendem o comportamento de outro caso de uso básico. Você pode fatorar o comportamento reutilizável e comum de um conjunto de casos de uso, organizando-os de acordo com esses três tipos de relacionamentos. Por exemplo, na modelagem de um banco, você encontrará muitas variações do caso de uso básico de processamento de empréstimos, como a diferença entre o processamento do financiamento de um avião e o empréstimo para uma pequena empresa. Em cada situação, entretanto, esses casos de uso compartilham algum grau de comportamento comum, como o caso de uso de qualificação do cliente para o empréstimo, um comportamento que faz parte do processamento de qualquer tipo de empréstimo.

Um caso de uso executa alguma quantidade tangível de trabalho. Sob a perspectiva de um determinado ator, um caso de uso realiza algo que é de valor para tal ator, como o cálculo de um resultado, a geração de um novo objeto ou a modificação do estado de outro objeto. Por exemplo, na modelagem de um banco, o processamento de um empréstimo resulta na entrega de um empréstimo aprovado, manifestada como uma pilha de dinheiro entregue nas mãos do cliente.

Você poderá aplicar os casos de uso a todo o seu sistema. Também pode aplicá-los a uma parte do sistema, incluindo subsistemas e até interfaces e classes individuais. Em cada situação, os casos de uso não apenas representam o comportamento desejado desses elementos, mas também podem ser utilizados como a base de casos de teste para esses elementos, à medida que evoluem durante o desenvolvimento. Casos de uso aplicados aos subsistemas são excelentes fontes de testes de regressão; casos de uso aplicados a todo o sistema são excelentes fontes de testes de sistema e de integração. A UML fornece a representação gráfica de um caso de uso e de um ator, conforme mostra a [Figura 17.1](#). Essa notação permite visualizar um

caso de uso em separado de sua realização e no contexto com outros casos de uso.

- Os subsistemas são examinados no [Capítulo 32](#); as classes são apresentadas nos [Capítulos 4 e 9](#); as interfaces são explicadas no [Capítulo 11](#).



**Figura 17.1:**

Atores e casos de uso

## TERMOS E CONCEITOS

Um *caso de uso* é uma descrição de um conjunto de sequências de ações, inclusive variantes, que um sistema executa para produzir um resultado de valor observável por um ator. Graficamente, o caso de uso é representado como uma elipse.

- A notação para os casos de uso é semelhante à das colaborações, conforme é apresentado no [Capítulo 28](#).

## ASSUNTO

O *assunto* é uma classe descrita por um conjunto de casos de uso. Normalmente, a classe é um sistema ou subsistema. Os casos de uso representam aspectos do comportamento da classe. Os atores representam aspectos de outras classes que interagem com o assunto. Juntos, os casos de uso descrevem o comportamento completo do assunto.

## NOMES

Todo caso de uso deve ter um nome que o diferencie dos demais casos de uso. Um nome é uma sequência de caracteres textual. Esse nome sozinho é conhecido como um *nome simples*; um *nome de caminho* é o nome do caso de uso, cujo prefixo é o nome do pacote em que o caso de uso se encontra. Tipicamente, um caso de uso é definido exibindo somente seu nome, conforme mostra a [Figura 17.2](#).



**Figura 17.2:**  
Nomes simples e qualificados

- *Um nome de caso de uso deve ser único dentro do pacote que o contém, conforme é apresentado no [Capítulo 12](#).*

**Nota:** Um nome de caso de uso poderá ser um texto contendo qualquer quantidade de letras, números e a maioria dos sinais de pontuação (exceto sinais como os dois-pontos, utilizado para separar um nome de classe e o nome do pacote que a contém) e poderá continuar por várias linhas. Na prática, os nomes de casos de uso são breves expressões verbais ativas, nomeando algum comportamento encontrado no vocabulário do sistema cuja modelagem você está fazendo.

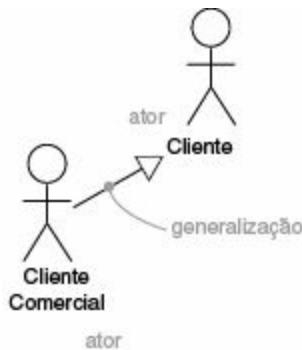
## CASOS DE USO E ATORES

Um ator representa um conjunto coerente de papéis que os usuários de casos de uso desempenham quando interagem com esses casos de uso. Tipicamente, um ator representa um papel que um ser humano, um dispositivo de hardware ou até outro sistema desempenha com o sistema. Por exemplo, se você trabalha em um banco, poderá ser um AgenteDeEmpréstimo. Se utiliza o serviço de banco pessoal, também desempenhará o papel de Cliente. Uma instância de um ator, portanto, representa uma interação individual com

o sistema de uma maneira específica. Embora você utilize atores em sua modelagem, os atores não são, de fato, parte do sistema. Eles residem fora do sistema.

Em um sistema em execução, os atores não precisam existir como entidades separadas. Um objeto pode representar o papel de vários atores. Por exemplo, uma Pessoa pode ser AgenteDeEmpréstimo e Cliente. Um ator representa um aspecto de um objeto.

Conforme mostra a [Figura 17.3](#), os atores são representados como figuras esquematizadas. Você poderá definir grupos gerais de atores (como em Cliente) e especializá-los (como em ClienteComercial), utilizando a generalização de relacionamentos.



**Figura 17.3:**

Atores

- A generalização é apresentada nos [Capítulos 5 e 10](#).

**Nota:** Os mecanismos de extensibilidade da UML podem ser utilizados para criar o estereótipo de um ator, com a finalidade de proporcionar um ícone diferente, capaz de oferecer uma melhor indicação visual para seus propósitos.

- Os estereótipos são apresentados no [Capítulo 6](#).

Os atores poderão estar conectados aos casos de uso somente pela associação. A associação entre o ator e um caso de uso indica que o ator e o caso de uso se comunicam entre si, cada um com a possibilidade de enviar e receber mensagens.

- ➡ Os relacionamentos por associação são apresentados nos Capítulos 5 e 10; as mensagens são apresentadas no Capítulo 16.

## CASOS DE USO E FLUXO DE EVENTOS

Um caso de uso descreve *o que* um sistema (ou um subsistema, classe ou interface) faz, mas ele não especifica *como* isso é feito. Ao fazer uma modelagem, é importante manter clara a separação de questões entre a visão interna e externa.

Você pode especificar o comportamento de um caso de uso pela descrição do fluxo de eventos no texto de maneira suficientemente clara para que alguém de fora possa compreendê-lo facilmente. Ao escrever o fluxo de eventos, você deverá incluir como e quando o caso de uso se inicia e termina, quando o caso de uso interage com os atores e quais objetos são transferidos e o fluxo básico e fluxo alternativo do comportamento.

Por exemplo, no contexto de um sistema de caixa eletrônico, você poderá descrever o caso de uso `ValidarUsuário` da seguinte maneira:

*Fluxo de eventos principal:* o caso de uso começa quando o sistema solicitar ao *Cliente* um número PIN, seu número de identificação pessoal (em inglês, Personal Identification Number). O Cliente agora pode digitar o número PIN via teclado numérico. O Cliente confirma a entrada, pressionando a tecla Enter. O sistema então verifica o número PIN para saber se é válido. Se o número PIN for válido, o sistema reconhece a entrada, finalizando o caso de uso.

*Fluxo excepcional de eventos:* o Cliente pode cancelar uma transação a qualquer momento, pressionando o botão Cancelar, reiniciando assim o caso de uso. Nenhuma alteração é realizada na conta do Cliente.

*Fluxo excepcional de eventos:* o Cliente pode limpar o número PIN a qualquer momento antes de submetê-lo e redigitar um novo número PIN.

*Fluxo excepcional de eventos:* se o Cliente entrar um número PIN inválido, o caso de uso é reiniciado. Se isso ocorrer três vezes seguidas, o sistema cancela toda a transação, impedindo o Cliente de interagir com o caixa eletrônico por 60 segundos.

**Nota:** Você pode especificar o fluxo de eventos de um caso de uso de diversas maneiras, incluindo texto estruturado informal (como no exemplo anterior), texto estruturado formal (com pré e pós-condições), máquinas de estado (particularmente para sistemas reativos), diagramas de atividades (particularmente para fluxos de trabalho) e pseudocódigo.

## CASOS DE USO E CENÁRIOS

Tipicamente, primeiro você descreverá o fluxo de eventos para um caso de uso no texto. Além de aperfeiçoar sua compreensão dos requisitos do sistema, entretanto, você desejará utilizar os diagramas de interação para especificar esses fluxos graficamente. Além disso, você também utilizará um diagrama de sequência para especificar o fluxo principal de um caso de uso e variações deste diagrama para especificar os fluxos excepcionais.

- ➡ Os diagramas de interação, inclusive os diagramas de sequência e os diagramas de colaboração, são apresentados no [Capítulo 19](#).

Isso é recomendável para separar os fluxos alternativos do fluxo principal, porque um caso de uso descreve um conjunto de sequências e não apenas uma sequência isolada e seria impossível expressar todos os detalhes de um caso de uso interessante em apenas uma sequência. Por exemplo, em um sistema de recursos humanos, você poderá encontrar o caso de uso Contratar empregado. Essa função de negócio geral poderá ter muitas variações possíveis. Você poderá contratar uma pessoa de outra empresa (o cenário mais comum); poderá transferir uma pessoa de um departamento para outro (comum em empresas internacionais); ou poderá contratar um estrangeiro residente no local (o que incluirá suas próprias regras especiais). Cada uma dessas variantes pode ser expressa em uma sequência diferente.

Esse caso de uso (Contratar empregado) realmente descreve um conjunto de sequências, em que cada sequência no conjunto representa um possível fluxo nessas variações. Cada sequência é chamada cenário. Um cenário é uma sequência específica de ações que ilustra o comportamento. Os cenários estão para os casos de uso assim como as instâncias estão para as classes. Isso significa que o cenário é basicamente uma instância de um caso de uso.

- As instâncias são apresentadas no [Capítulo 13](#).

*Nota: Existe um fator de expansão dos casos de uso para os cenários. Um sistema modestamente complexo poderá ter algumas dúzias de casos de uso captando esse comportamento e cada caso de uso poderá ser expandido em várias dúzias de cenários. Para cada caso de uso, você encontrará cenários primários (que definem as sequências essenciais) e cenários secundários (que definem as sequências alternativas).*

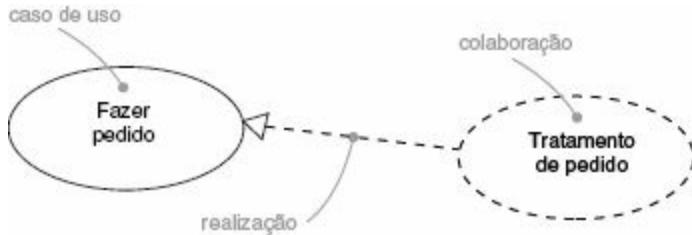
## CASOS DE USO E COLABORAÇÕES

Um caso de uso captura o comportamento pretendido do sistema (ou subsistema, classe ou interface) que você está desenvolvendo, sem ser preciso especificar como esse comportamento é implementado. Essa é uma separação importante, porque a análise de um sistema (que especifica o comportamento) deveria, tanto quanto possível, não ser influenciada por questões referentes à implementação (que especificam como esse comportamento é executado). Por fim, entretanto, é necessário implementar seus casos de uso e isso é feito pela criação de uma sociedade de classes e de outros elementos que trabalham em conjunto para a implementação do comportamento desse caso de uso. Essa sociedade de elementos, incluindo as estruturas estática e dinâmica, é modelada na UML como uma colaboração.

- As colaborações são apresentadas no [Capítulo 28](#).

Conforme mostra a [Figura 17.4](#), pode-se especificar explicitamente a realização de um caso de uso por meio de uma colaboração. Na maior parte do tempo, entretanto, um determinado caso de uso é realizado por exatamente

uma colaboração, não sendo necessário, portanto, fazer a modelagem desse relacionamento explicitamente.



**Figura 17.4:**

Casos de uso e colaborações

- A realização é apresentada nos [Capítulos 9 e 10](#).

**Nota:** Embora você não possa visualizar esse relacionamento explicitamente, as ferramentas que você utiliza para gerenciar suas modelagens provavelmente manterão esse relacionamento.

**Nota:** Encontrar o conjunto mínimo de colaborações bem estruturadas que satisfazem aos fluxos de evento especificados em todos os casos de uso de um sistema é o foco da arquitetura de um sistema.

- A arquitetura é apresentada no [Capítulo 2](#).

## ORGANIZAÇÃO DOS CASOS DE USO

Você pode organizar os casos de uso agrupando-os em pacotes do mesmo modo como são organizadas as classes.

- Os pacotes são apresentados no [Capítulo 12](#).

Os casos de uso também podem ser organizados pela especificação de relacionamentos de generalização, inclusão e extensão existentes entre eles. Você aplica esses relacionamentos com a finalidade de fatorar o comportamento comum (obtendo esse comportamento a partir de outros casos de uso que ele inclui) e de fatorar variantes (obtendo esse comportamento em outros casos de uso que o estendem).

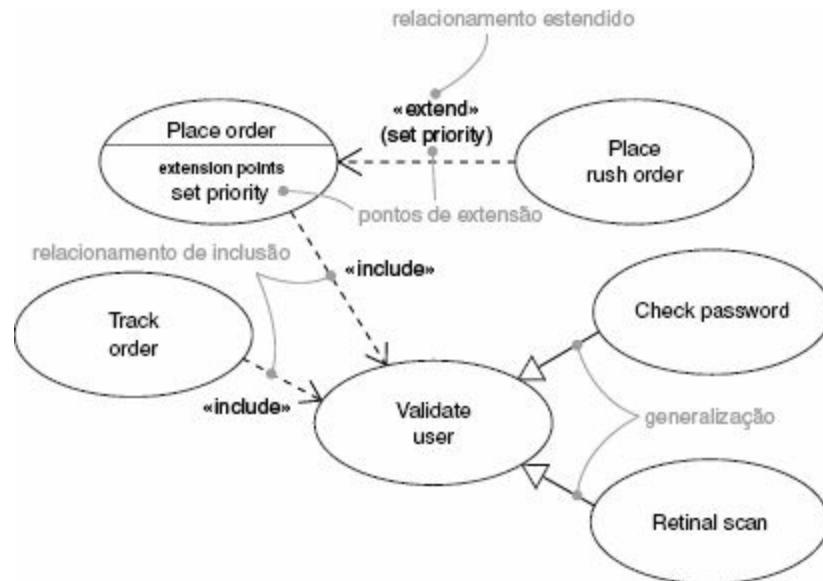
A generalização entre casos de uso é semelhante à generalização existente entre as classes. Aqui a generalização significa que o caso de uso filho herda o comportamento e o significado do caso de uso pai; o filho deverá acrescentar ou sobrescrever o comportamento de seu pai; e o filho poderá ser substituído em qualquer local no qual o pai apareça (ambos, o pai e o filho poderão ter instâncias concretas). Por exemplo, em um sistema bancário, você poderá ter o caso de uso Validar Usuário, responsável pela verificação da identidade do usuário. Poderá então haver dois filhos especializados nesse caso de uso (Verificar senha e Analisar retina), ambos se comportam da mesma maneira como Validar usuário e podem ser aplicados em qualquer lugar em que Validar usuário apareça, ainda que ambos acrescentem seu próprio comportamento (o primeiro, pela verificação da senha textual; o último, pela verificação dos padrões únicos da retina do usuário). Conforme mostra a [Figura 17.5](#), a generalização entre casos de uso é representada como uma linha cheia direcionada com uma seta aberta, exatamente como ocorre com a generalização entre as classes.

► A generalização é examinada nos Capítulos [5](#) e [10](#).

Um relacionamento de inclusão entre casos de uso significa que o caso de uso base incorpora explicitamente o comportamento de outro caso de uso em uma localização especificada na base. O caso de uso incluído nunca permanece isolado, mas é apenas instanciado como parte de alguma base maior que o inclui. Você pode pensar na inclusão como o caso de uso base que obtém o comportamento a partir do fornecedor do caso de uso.

Você utiliza um relacionamento de inclusão para evitar descrever o mesmo fluxo de eventos várias vezes, incluindo o comportamento comum em um caso de uso próprio (o caso de uso incluído em um caso de uso básico). O relacionamento de inclusão é essencialmente um exemplo de delegação – você coleta um conjunto de responsabilidades do sistema e o captura em um único local (o caso de uso incluído); depois, permite que outras partes do

sistema (outros casos de uso) incluem a nova agregação de responsabilidades sempre que precisarem utilizar essa funcionalidade.



**Figura 17.5:**  
Generalização, inclusão e extensão

Um relacionamento de inclusão pode ser representado como uma dependência, estereotipada como `include`. Para especificar a localização em um fluxo de eventos no qual o caso de uso base inclui o comportamento de um outro, simplesmente escreva `include`, seguido pelo nome do caso de uso que deseja incluir, como no seguinte fluxo para `Track order`:

Track order:  
 obtain and verify the order number;  
 include ‘Validar usu’;  
 for each part in the order,  
 query the order status;  
 report overall status to user.

- *Os relacionamentos de dependência são examinados nos Capítulos 5 e 10; os estereótipos são apresentados no Capítulo 6.*

Um relacionamento estendido entre casos de uso significa que o caso de uso base incorpora implicitamente o comportamento de um outro caso de uso em um local especificado indiretamente pelo caso de uso estendido. O caso

de uso base poderá permanecer isolado, mas, sob certas condições, seu comportamento poderá ser estendido pelo comportamento de um outro caso de uso. Esse caso de uso base poderá ser estendido somente em determinados pontos chamados, sem qualquer surpresa, seus pontos de extensão. Considere extensão como caso de uso estendido que envia um comportamento para o caso de uso base.

Um relacionamento estendido é utilizado para a modelagem da parte de um caso de uso que o usuário poderá considerar como um comportamento opcional do sistema. Desse modo, separa-se o comportamento opcional do comportamento obrigatório. Um relacionamento estendido também poderá ser empregado para a modelagem de um subfluxo separado, que é executado somente sob determinadas condições. Por fim, um relacionamento estendido poderá ser utilizado para a modelagem de vários fluxos que poderão ser inseridos em um certo ponto, de acordo com a interação explícita com um ator. Você também pode usar um relacionamento estendido para distinguir as partes configuráveis de um sistema de implementação; a implicação é que o sistema pode existir com ou sem as várias extensões.

O relacionamento estendido é representado como uma dependência, estereotipada como `extend`. Os pontos de extensão do caso de uso base poderão ser relacionados em um compartimento extra. Esses pontos de extensão são apenas rótulos que poderão aparecer no fluxo do caso de uso base. Por exemplo, o fluxo para `Place order` (`Colocar pedido`) poderá ser lido da seguinte forma:

```
Fazer pedido:  
include 'Validar usuário';  
collect the user's order items;  
set priority: extension point;  
submit the order for processing.
```

- *Os relacionamentos de dependência são examinados nos Capítulos 5 e 10; os estereótipos e compartimentos extras são apresentados no Capítulo 6.*

Nesse exemplo, set priority (definir prioridade) é um ponto de extensão. Um caso de uso poderá ter vários pontos de extensão (que poderão aparecer mais de uma vez) e esses pontos sempre serão considerados pelos seus nomes. Em circunstâncias normais, esse caso de uso base será executado independentemente da prioridade de ordem. Se, por outro lado, for uma instância de uma ordem de prioridade, o fluxo para esse caso base será executado, conforme foi descrito acima. Entretanto, no ponto de extensão (set priority), o comportamento do caso de uso estendido (Place rush order – Fazer pedido urgente) será realizado e depois o fluxo prosseguirá. Se houver vários pontos de extensão, o caso de uso estendido simplesmente permanecerá em seu fluxo de ordem.

***Nota:** Organizar os casos de uso, extraindo o comportamento comum (por meio de relacionamentos de inclusão) e diferenciando as variantes (por relacionamentos estendidos), é uma parte importante para a criação de um conjunto de casos de uso simples, equilibrado e compreensível, para o seu sistema.*

## OUTRAS CARACTERÍSTICAS

Os casos de uso são classificadores e, portanto, poderão ter atributos e operações que você poderá representar da mesma maneira como faz com as classes. Considere esses atributos como os objetos encontrados no caso de uso cujo comportamento externo você precisará descrever. De modo semelhante, considere essas operações como as ações do sistema cujo fluxo de eventos será necessário descrever. Esses objetos e operações poderão ser utilizados em diagramas de interação para especificar o comportamento do caso de uso.

- ➡ Os atributos e as operações são examinados no [Capítulo 4](#); as máquinas de estados são examinadas no [Capítulo 22](#).

Por serem classificadores, você também pode anexar máquinas de estados aos casos de uso. As máquinas de estados podem ser uma outra forma de descrever o comportamento representado por um caso de uso.

# TÉCNICAS BÁSICAS DE MODELAGEM

## MODELAGEM DO COMPORTAMENTO DE UM ELEMENTO

A modelagem do comportamento de um elemento será a situação mais comum em que os casos de uso serão aplicados, seja esse elemento o sistema como um todo, um subsistema ou uma classe. Ao fazer a modelagem do comportamento desses elementos, é importante que você focalize o que o elemento faz e não como faz.

- *Os sistemas e subsistemas são examinados no [Capítulo 32](#); as classes são apresentadas nos [Capítulos 4 e 9](#).*

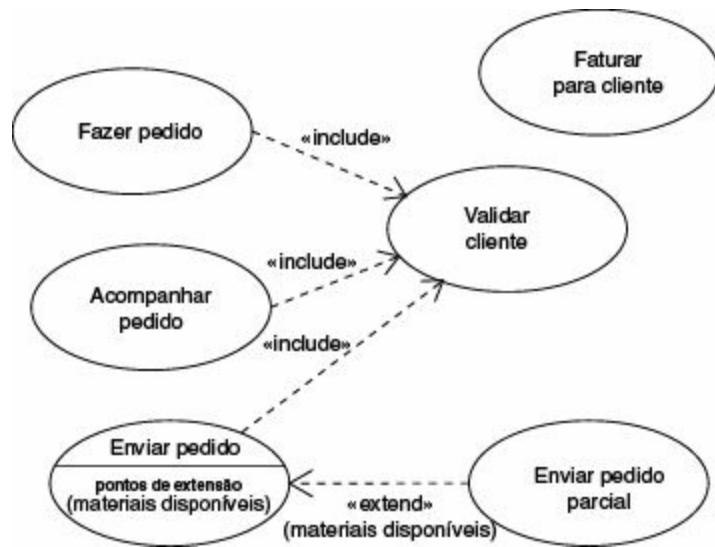
A aplicação dos casos de uso aos elementos dessa maneira é importante por três razões. Primeiro, a partir da modelagem do comportamento de um elemento por meio de casos de uso, você proporciona aos especialistas do domínio uma maneira de especificar sua visão externa em um grau suficiente para que os desenvolvedores construam sua visão interna. Os casos de uso fornecem um fórum para que seus especialistas de domínios, usuários finais e desenvolvedores possam se comunicar uns com os outros. Segundo, os casos de uso oferecem uma forma de os desenvolvedores abordarem um elemento e compreendê-lo. Um sistema, subsistema ou classe poderão ser complexos e cheios de operações e outras partes. Especificando casos de uso do elemento, você ajuda os usuários desses elementos a abordá-los de uma maneira direta, de acordo com a forma com que provavelmente os utilizarão. Na ausência desses casos de uso, os usuários precisam descobrir, por conta própria, como utilizar os elementos. Os casos de uso permitem que o autor de um elemento comunique sua intenção a respeito de como o elemento deverá ser utilizado. Terceiro, os casos de uso servem como base para testar cada elemento, à medida que ele evolui durante seu desenvolvimento. Testando continuamente cada elemento em relação a seus casos de uso, você validará continuamente a sua implementação. Esses casos de uso não só fornecem uma fonte de testes de regressão, mas, sempre que incluir um novo caso de uso com um

elemento, você será obrigado a reconsiderar sua implementação para assegurar que esse elemento admite modificações. Se não admitir, deve ajustar sua arquitetura de maneira apropriada.

Para fazer a modelagem do comportamento de um elemento:

- › Identifique os atores que interagem com o elemento. Candidatos a atores incluem grupos que exigem determinado comportamento para a realização de suas tarefas ou que são necessários direta ou indiretamente para a realização de funções do elemento.
- › Organize os atores, identificando papéis gerais e mais especializados.
- › Para cada ator, considere as formas primárias em que o ator interage com o elemento. Considere também as interações que alteram o estado do elemento ou seu ambiente ou que envolvam uma resposta a algum evento.
- › Considere também as formas excepcionais em que cada ator interage com o elemento.
- › Organize esses comportamentos como casos de uso, aplicando relacionamentos de inclusão e estendidos com a finalidade de fazer a fatoração do comportamento comum e diferenciar o comportamento excepcional.

Por exemplo, um sistema de vendas a varejo interage com os clientes que incluem e acompanham seus pedidos. Por sua vez, o sistema remeterá os pedidos e cobrará dos clientes. Conforme mostra a [Figura 17.6](#), é possível fazer a modelagem do comportamento de um sistema como esse, declarando esses comportamentos como casos de uso (Fazer pedido, Acompanhar pedido, Enviar pedido e Faturar para cliente). O comportamento comum pode ser fatorado (Validar cliente) e variantes (Enviar pedido parcial) também podem ser identificadas. Para cada um desses casos de uso, você poderia incluir uma especificação do comportamento, por meio de um texto, máquina de estados ou interações.



**Figura 17.6:**

A modelagem do comportamento de um elemento

À medida que seus modelos crescem, você perceberá que muitos casos de uso tendem a se reunir em grupos relacionados conceitual e semanticamente. Na UML, você pode usar os pacotes para fazer a modelagem desses agrupamentos de classes.

» *Os pacotes são examinados no Capítulo 12.*

## DICAS E SUGESTÕES

Ao fazer a modelagem de casos de uso na UML, todos os casos deverão representar algum comportamento distinto e identificável do sistema ou de parte do sistema. Um caso de uso bem estruturado:

- » Nomeia um comportamento do sistema ou de parte do sistema, que seja único, identificável e razoavelmente atômico.
- » Faz a fatoração do comportamento comum, obtendo esse comportamento a partir de outros casos de uso que ele inclui.
- » Faz a fatoração das variantes, aplicando esse comportamento a outros casos de uso que o estendem.

- » Descreve o fluxo de eventos de maneira suficientemente clara para que alguém de fora seja capaz de compreendê-lo com facilidade.
- » É descrito por um conjunto mínimo de cenários que especificam a semântica normal e variante do caso de uso.

Ao definir um caso de uso na UML:

- » Mostre somente os casos de uso que são importantes para a compreensão do comportamento do sistema ou de parte do sistema em seu contexto.
- » Mostre somente os atores que estão relacionados com esses casos de uso.

CAPÍTULO

---

18

---

# Diagramas de Casos de Uso

## Neste capítulo

- » Modelagem do contexto de um sistema
- » Modelagem dos requisitos de um sistema
- » Engenharia direta e reversa

**O**s diagramas de casos de uso são um dos diagramas disponíveis na UML para a modelagem de aspectos dinâmicos de sistemas (diagramas de atividades, diagramas de gráfico de estados, diagramas de sequências e diagramas de comunicação são quatro outros tipos de diagramas da UML para a modelagem de aspectos dinâmicos de sistemas). Os diagramas de casos de uso têm um papel central para a modelagem do comportamento de um sistema, de um subsistema ou de uma classe. Cada um mostra um conjunto de casos de uso e atores e seus relacionamentos.

- ➡ Os diagramas de atividades são examinados no [Capítulo 20](#); os diagramas de gráfico de estados são apresentados no [Capítulo 25](#); os diagramas de comunicação e de sequências são explicados no [Capítulo 19](#).

Você aplica os diagramas de casos de uso para fazer a modelagem da visão de caso de uso do sistema. Na maior parte, isso envolve a modelagem do contexto do sistema, subsistema ou classe ou a modelagem dos requisitos do comportamento desses elementos.

Os diagramas de casos de uso são importantes para visualizar, especificar e documentar o comportamento de um elemento. Esses diagramas fazem com

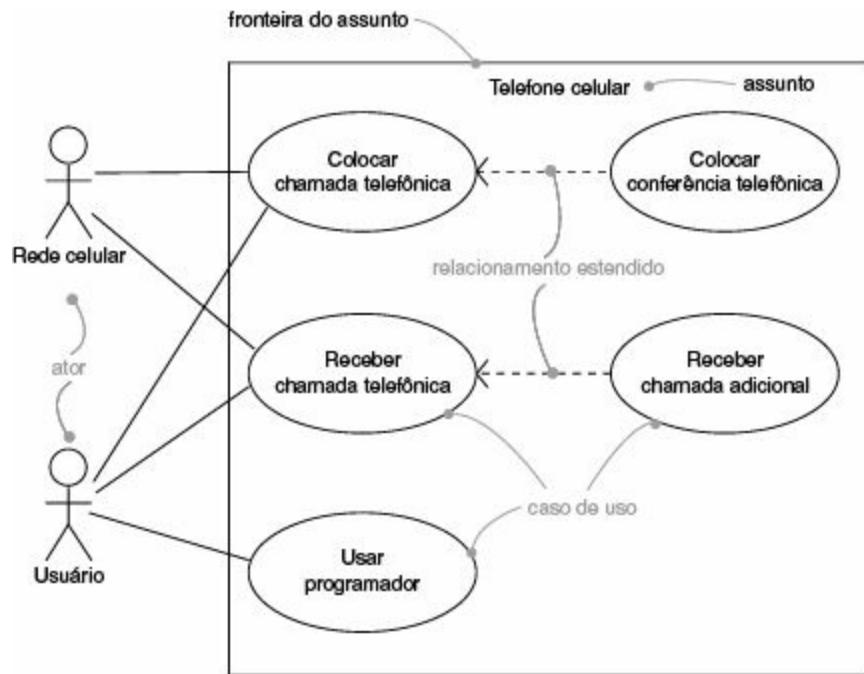
que sistemas, subsistemas e classes fiquem acessíveis e compreensíveis, por apresentarem uma visão externa sobre como esses elementos podem ser utilizados no contexto. Os diagramas de casos de uso também são importantes para testar sistemas executáveis por meio de engenharia direta e para compreendê-los por meio de engenharia reversa.

## PRIMEIROS PASSOS

Suponha que alguém lhe entregue uma caixa. De um lado da caixa, existem alguns botões e um pequeno painel LCD. Além disso, a caixa não contém qualquer descrição; você não tem qualquer pista sobre como utilizá-la. Você poderia apertar os botões aleatoriamente e ver o que acontece, mas você foi bastante pressionado a tentar compreender o que a caixa faz ou como utilizá-la sem gastar muito tempo com o processo de tentativa e erro.

Sistemas complexos de software podem ser semelhantes a essa situação. Se você for um usuário, poderá receber uma aplicação que deverá utilizar. Se a aplicação obedecer às convenções normais do sistema em operação com o qual você está acostumado, talvez você seja capaz de conseguir fazer algo útil após algumas tentativas, mas, dessa maneira, nunca compreenderá seu comportamento mais complexo e sutil. De modo semelhante, se você for um desenvolvedor, poderá receber uma aplicação ou um conjunto de componentes com a incumbência de utilizá-lo sendo pressionado a saber como utilizar esses elementos até o modelo conceitual estar formado para conhecer como usá-lo.

Com a UML, você pode aplicar diagramas de casos de uso para visualizar o comportamento de um sistema, subsistema ou classe, para que os usuários possam entender como utilizar esse elemento e os desenvolvedores possam implementá-lo. Conforme mostra a [Figura 18.1](#), você pode fornecer um diagrama de caso de uso ao fazer a modelagem do comportamento dessa caixa – que a maioria das pessoas chama de telefone celular.



**Figura 18.1:**

Um diagrama de caso de uso

## TERMOS E CONCEITOS

Um *diagrama de caso de uso* é um diagrama que mostra um conjunto de casos de uso e atores e seus relacionamentos.

## PROPRIEDADES COMUNS

Um diagrama de caso de uso é apenas um tipo especial de diagrama e compartilha as mesmas propriedades comuns a todos os demais diagramas – um nome e um conteúdo gráfico que são uma projeção em um modelo. O que diferencia um diagrama de caso de uso dos outros tipos de diagramas é o seu conteúdo particular.

- ➡ As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

## CONTEÚDO

Os diagramas de casos de uso costumam conter o seguinte:

- » Cenário
- » Casos de uso
- » Atores
- » Relacionamentos de dependência, generalização e associação

→ *Os casos de uso e os atores são examinados no Capítulo 17; os relacionamentos são apresentados nos Capítulos 5 e 10; os pacotes aparecem no Capítulo 12; as instâncias são explicadas no Capítulo 13.*

Assim como os outros diagramas, os diagramas de casos de uso podem conter notas e restrições.

Os diagramas de caso de uso também podem conter pacotes, utilizados para agrupar elementos do modelo em conjuntos maiores. Ocassionalmente, você também desejará incluir instâncias de casos de uso em seus diagramas, sobretudo quando quiser visualizar um sistema específico em execução.

## NOTAÇÃO

O cenário é exibido como um retângulo que contém um conjunto de elipses de casos de uso. O nome do objeto é colocado no retângulo. Os atores são apresentados como figuras de palito colocadas fora do retângulo; seus nomes são colocados embaixo deles. Linhas conectam ícones de atores a elipses de casos de uso com os quais se comunicam. Os relacionamentos entre casos de uso (como extensão e inclusão) são desenhados dentro do retângulo.

## USOS COMUNS

Você aplica os diagramas de casos de uso para fazer a modelagem da visão do caso de uso de um cenário, como um sistema. Essa visão

proporciona suporte principalmente para o comportamento de um sistema – os serviços externamente visíveis que o sistema fornece no contexto de seu ambiente.

- ➡ As visões de casos de uso são apresentadas no [Capítulo 2](#).

Ao fazer a modelagem da visão de caso de uso de um cenário, tipicamente você aplicará os diagramas de casos de uso em uma de duas maneiras.

### 1. Fazer a modelagem do contexto de um cenário.

A modelagem do contexto de um cenário envolve desenhar uma linha ao redor de todo o sistema e declarar quais atores ficam fora do cenário e como eles interagem. Aqui você aplicará os diagramas de casos de uso para especificar os atores e o significado de seus papéis.

### 2. Fazer a modelagem dos requisitos de um sistema.

- ➡ Os requisitos são examinados nos [Capítulos 4 e 6](#).

A modelagem dos requisitos de um cenário envolve a especificação do que esse cenário deverá fazer (sob um ponto de vista externo ao cenário), independentemente de como o cenário deverá fazê-lo. Aqui, você aplicará os diagramas de casos de uso para especificar o comportamento desejado do cenário. Dessa maneira, um diagrama de caso de uso permite que você visualize todo o cenário como uma caixa preta; é possível ver o que está fora do cenário e como ele reage a algo externo, mas não é possível ver como o cenário funciona internamente.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DO CONTEXTO DO SISTEMA

Dado um sistema – qualquer sistema – algumas coisas se encontrarão dentro do sistema, algumas coisas se encontrarão fora dele. Por exemplo, em um sistema de validação de cartão de crédito, dentro do sistema serão encontradas coisas como contas, transações e agentes de detecção de fraudes.

De modo semelhante, fora do sistema serão encontradas coisas como clientes de cartão de crédito e instituições de vendas a varejo. As coisas que se encontram dentro do sistema são responsáveis pela execução do comportamento que aquelas que estão fora esperam que seja fornecido pelo sistema. Todas essas coisas externas que interagem com o sistema constituem o contexto do sistema. Esse contexto define o ambiente em que esse sistema existe.

Na UML, é possível fazer a modelagem do contexto de um sistema com um diagrama de caso de uso, dando ênfase aos atores que estão ao redor do sistema. Decidir o que é incluído como ator é importante, pois, ao fazê-lo, você especifica a classe de coisas que interagem com o sistema. Decidir o que não está incluído como ator é igualmente, se não ainda mais, importante, pois restringirá o ambiente do sistema para incluir somente os atores que são necessários na vida do sistema.

► *Os sistemas são examinados no [Capítulo 32](#).*

Para fazer a modelagem do contexto de um sistema:

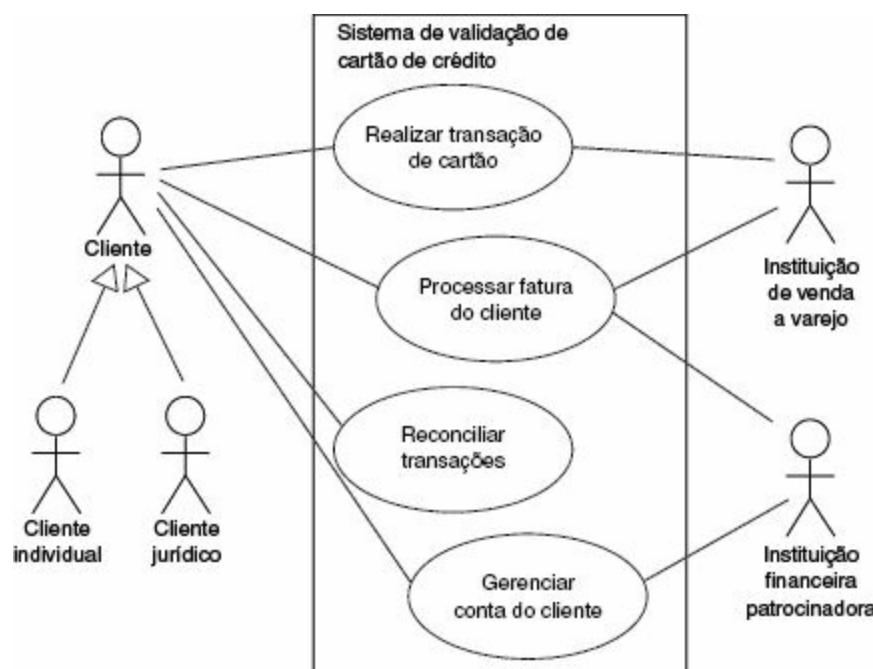
- » Identifique os limites do sistema decidindo quais comportamentos são parte dele e quais são realizados por entidades externas. Isso define o cenário.
- » Identifique os atores que se encontram ao redor do sistema, considerando quais grupos precisam de ajuda do sistema para a realização de suas tarefas; quais grupos são necessários para a execução de funções do sistema; quais grupos interagem com algum hardware externo ou outros sistemas de software; e quais grupos realizam funções secundárias de administração e de manutenção.
- » Organize os atores semelhantes em uma hierarquia de generalização/especialização.
- » Ofereça um estereótipo para cada ator, para ajudar a compreensão.

Preencha um diagrama de caso de uso com esses atores e especifique os

caminhos de comunicação de cada ator até os casos de uso do sistema.

Por exemplo, a [Figura 18.2](#) mostra o contexto de um sistema de validação de cartões de crédito, com uma ênfase nos atores ao redor do sistema. Você encontrará Clientes, dos quais existem dois tipos (Cliente individual e Cliente jurídico). Esses atores são os papéis desempenhados pelos seres humanos quando interagem com o sistema. Nesse contexto, também existem atores que representam outras instituições, como a Instituição de venda a varejo (com a qual o Cliente realiza transação com o cartão para comprar qualquer item ou serviço) e a Instituição financeira patrocinadora (que serve como carteira de compensação para a conta de cartão de crédito). No mundo real, esses dois últimos atores são semelhantes aos próprios sistemas complexos de software.

Essa mesma técnica se aplica à modelagem do contexto de um subsistema. Um sistema em um nível de abstração costuma ser um subsistema de um sistema maior em um nível mais alto de abstração. A modelagem do contexto de um subsistema é, portanto, útil quando você está construindo sistemas a partir de outros sistemas interconectados.



**Figura 18.2:**

A modelagem do contexto de um sistema

## MODELAGEM DOS REQUISITOS DE UM SISTEMA

Um requisito é uma característica de projeto, uma propriedade ou um comportamento de um sistema. Ao estabelecer os requisitos do sistema, você está declarando um contrato, estabelecido entre as coisas externas ao sistema e o próprio sistema, declarando o que se espera que seja feito pelo sistema. Na maior parte, você não precisa se preocupar com o que sistema faz, você deve apenas cuidar *para que* ele o faça. Um sistema bem-comportado executará todos os seus requisitos de maneira fiel, previsível e confiável. Quando você construir um sistema, é importante iniciar com um consenso a respeito do que o sistema deverá fazer, apesar de que certamente você evoluirá sua compreensão a respeito desses requisitos, à medida que iterativa e incrementalmente implementar o sistema. De modo semelhante, ao receber um sistema a ser utilizado, conhecer como ele se comporta é essencial para empregá-lo adequadamente.

Os requisitos podem ser expressos de várias formas, desde um texto não estruturado a expressões em uma linguagem formal e tudo o mais entre esses extremos. A maioria, se não todos, dos requisitos funcionais de um sistema pode ser expressa como casos de uso e os diagramas de casos de uso da UML são essenciais para o gerenciamento desses requisitos.

- » As notas podem ser utilizadas para estabelecer os requisitos, conforme é examinado no [Capítulo 6](#).

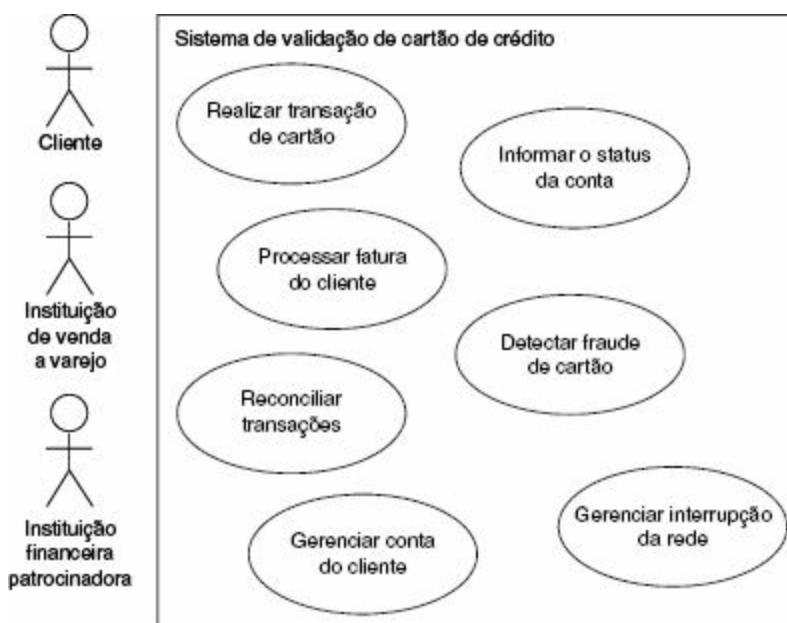
Para fazer a modelagem dos requisitos de um sistema:

- » Estabeleça o contexto do sistema, identificando os atores ao seu redor.
- » Para cada ator, considere o comportamento que cada um espera ou requer que o sistema proporcione.
- » Nomeie esses comportamentos comuns como casos de uso.
- » Faça a fatoração do comportamento comum em novos casos de uso utilizados pelos outros; faça a fatoração do comportamento variante em novos casos de uso que estendem os fluxos da linha principal.

› Faça a modelagem desses casos de uso, atores e seus relacionamentos em um diagrama de caso de uso.

› Inclua adornos nesses casos de uso com notas declarando requisitos não funcionais; poderá ser necessário anexar alguns deles a todo o sistema.

A [Figura 18.3](#) expande o diagrama de caso de uso anterior. Embora oculte os relacionamentos existentes entre os atores e os casos de uso, são incluídos casos de uso adicionais que, de alguma forma, são invisíveis ao cliente médio, por não serem comportamentos essenciais do sistema. Esse diagrama é valioso, por oferecer um ponto de partida comum para usuários finais, especialistas de domínio e desenvolvedores para a visualização, especificação, construção e documentação de suas decisões sobre os requisitos funcionais desse sistema. Por exemplo, Detectar fraude de cartão é um comportamento importante tanto para a Instituição de venda a varejo como para a Instituição financeira patrocinadora. De modo semelhante, Informar o status da conta é outro comportamento solicitado pelo sistema pelas várias instituições de seu contexto.



**Figura 18.3:**

A modelagem dos requisitos de um sistema

O requisito modelado pelo caso de uso Gerenciar interrupção da rede é um pouco diferente de todos os outros, por representar um comportamento secundário do sistema, necessário para sua operação contínua e confiável.

- ➡ A modelagem da dinâmica para o equilíbrio de carga e a reconfiguração de rede são examinadas no [Capítulo 24](#).

Uma vez que a estrutura do caso de uso é determinada, você deve descrever o comportamento de cada caso de uso. Normalmente, é preciso escrever um ou mais diagramas de sequências para cada caso da linha principal. Em seguida, escreva os diagramas de sequências para os casos variantes. Finalmente, escreva pelo menos um diagrama de sequências para ilustrar cada tipo de erro ou condição de exceção; o tratamento de erros é parte do caso de uso e deve ser planejado junto com o comportamento normal.

A mesma técnica se aplica à modelagem dos requisitos de um subsistema.

- ➡ Os subsistemas são examinados no [Capítulo 32](#).

## ENGENHARIA DIRETA E REVERSA

A maioria dos demais diagramas da UML, incluindo os diagramas de classes, de componente e de gráficos de estados são claros candidatos para a engenharia direta e reversa, pois cada um tem um análogo no sistema executável. Os diagramas de caso de uso são um pouco diferentes, porque refletem mais do que especificam a implementação do sistema, subsistema ou classe. Os casos de uso descrevem como um elemento se comporta e não como esse comportamento é implementado; portanto, eles não são incluídos diretamente em uma engenharia direta ou reversa.

- ➡ Os diagramas são examinados no [Capítulo 7](#); os casos de uso são apresentados no [Capítulo 17](#).

*Engenharia direta* é o processo de transformar um modelo em código pelo mapeamento em uma linguagem de implementação. O diagrama de caso de uso pode ser incluído na engenharia direta para formar testes destinados ao elemento ao qual ele se aplica. Cada caso em um diagrama de caso de uso especifica um fluxo de eventos (e as variantes desses fluxos). Esses fluxos especificam o comportamento esperado desse elemento – isso é algo que vale a pena ser testado. Um caso de uso bem estruturado até especifica pré e pós-condições que poderão ser utilizadas para definir o estado inicial de um teste e o correspondente critério de sucesso. Para cada caso de um diagrama de caso de uso, você poderá criar um caso de teste a ser executado sempre que liberar uma nova versão desse elemento, confirmando, assim, que o elemento está funcionando conforme é exigido antes de outros elementos dependerem dele.

Para fazer a engenharia direta de um diagrama de caso de uso:

- › Identifique os objetos que interagem com o sistema. Tente identificar os vários papéis que cada objeto externo pode desempenhar.
- › Crie um ator para representar cada papel de interação diferente.
- › Para cada caso de uso do diagrama, identifique seu fluxo de eventos e seu fluxo de eventos excepcional.
- › Dependendo da profundidade escolhida para o teste, gere o roteiro de um teste para cada fluxo, utilizando as precondições do fluxo como o estado inicial do teste e suas pós-condições como critério de sucesso.
- › De acordo com a necessidade, gere níveis de teste para representar cada ator que interage com o caso de uso. Os atores que passam informações ao elemento ou são ativados pelo elemento, ou poderão ser simulados ou substituídos por seus equivalentes do mundo real.
- › Use ferramentas para executar esses testes cada vez que liberar o elemento ao qual o diagrama de caso de uso se aplica.

*Engenharia reversa* é o processo de transformar código em um modelo pelo mapeamento a partir de uma linguagem de implementação específica. Aplicar automaticamente a engenharia reversa a um diagrama de caso de uso está bem longe do estado da arte, simplesmente por haver uma perda de informações quando se passa da especificação do comportamento de um elemento para o modo como ele é implementado. Entretanto, é possível estudar um sistema existente e discernir seu comportamento pretendido, que você pode então colocar sob a forma de um diagrama de caso de uso. Na verdade, é exatamente isso que é necessário fazer quando se recebe uma porção de software não documentada. Os diagramas de casos de uso da UML simplesmente fornecem uma linguagem padrão e expressiva para você declarar o que descobriu.

Para fazer a engenharia reversa de um diagrama de caso de uso:

- › Identifique cada ator que interage com o sistema.
- › Para cada ator, considere a maneira como esse ator interage com o sistema, altera o estado do sistema ou seu ambiente, ou responde a algum evento.
- › Trace o fluxo de eventos do sistema executável relativo a cada ator. Inicie com os fluxos primários e somente depois considere os caminhos alternativos.
- › Agrupe os fluxos relacionados, declarando um caso de uso correspondente. Considere a modelagem de variantes, usando relacionamentos do tipo estendido e considere a modelagem de fluxos comuns pela aplicação de relacionamentos de inclusão.
- › Represente esses atores e casos de uso em um diagrama de caso de uso e estabeleça seus relacionamentos.

## DICAS E SUGESTÕES

Ao criar diagramas de casos de uso na UML, lembre-se de que todo diagrama de caso de uso é apenas uma apresentação gráfica da visão estática do caso de uso de um sistema. Dessa forma, podemos dizer que nenhum diagrama único de caso de uso necessita captar tudo a respeito da visão de caso de uso do sistema. Em conjunto, todos os diagramas de casos de uso de um sistema representam a visão estática completa do caso de uso do sistema; individualmente, cada um representa apenas um aspecto.

Um diagrama de caso de uso bem estruturado:

- › Tem como foco comunicar um aspecto da visão estática de caso de uso do sistema.
- › Contém somente os casos de uso e atores essenciais à compreensão desse aspecto.
- › Fornece detalhes consistentes com seu nível de abstração; deverão ser expostos somente os adornos (como os pontos de extensão) essenciais à compreensão.
- › Não é tão minimalista, que informe mal o leitor sobre a semântica que é importante.

Ao definir um diagrama de caso de uso:

- › Dê-lhe um nome capaz de comunicar seu propósito.
- › Distribua seus elementos para minimizar o cruzamento de linhas.
- › Organize seus elementos espacialmente, de maneira que os comportamentos e papéis semanticamente relacionados apareçam próximos.
- › Use notas e cores como indicações visuais e para chamar a atenção para características importantes do diagrama.
- › Tente não mostrar muitos tipos de relacionamentos. Em geral, se você tiver relacionamentos de inclusão e estendido complicados, coloque esses elementos em outro diagrama.

CAPÍTULO

---

**19**

---

# Diagramas de Interação

## Neste capítulo

- » Modelagem dos fluxos de controle por ordem temporal
- » Modelagem dos fluxos de controle por organização
- » Engenharia direta e reversa

**O**s diagramas de sequência e os diagramas de comunicação – chamados de diagramas de interação – são dois dos diagramas utilizados na UML para a modelagem dos aspectos dinâmicos de sistemas. Um diagrama de interação mostra uma interação, formada por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que poderão ser enviadas entre eles. Um diagrama de sequências é um diagrama de interação que dá ênfase à ordenação temporal das mensagens; o diagrama de comunicação é um diagrama de interação que dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens.

- ➡ Os diagramas de atividades, diagramas de estados e diagramas de casos de uso são outros três tipos de diagramas utilizados na UML para a modelagem de aspectos dinâmicos de sistemas; os diagramas de atividades são examinados no [Capítulo 20](#); os diagramas de estados são apresentados no [Capítulo 25](#); os diagramas de casos de uso são explicados no [Capítulo 18](#).

Os diagramas de interação são utilizados para fazer a modelagem dos aspectos dinâmicos do sistema. Na maior parte, isso envolve a modelagem de instâncias concretas ou protótipicas de classes, interfaces, componentes e nós,

juntamente com as mensagens que são trocadas entre eles, tudo isso no contexto de um cenário que ilustra um comportamento. Diagramas de interações podem aparecer sozinhos para visualizar, especificar, construir e documentar a dinâmica de uma determinada sociedade de objetos ou podem ser utilizados para fazer a modelagem de um determinado fluxo de controle de um caso de uso.

Os diagramas de interação não são importantes somente para a modelagem de aspectos dinâmicos do sistema, mas também para a construção de sistemas executáveis por meio de engenharia direta e reversa.

## PRIMEIROS PASSOS

Quando você assiste a um filme projetado na tela do cinema ou transmitido pela televisão, sua mente realmente prega uma peça em você. Em vez do movimento contínuo que seria observado na ação ao vivo, na verdade você vê uma série de imagens estáticas reproduzidas muito rapidamente para dar a ilusão de movimento contínuo.

Quando os diretores e os animadores planejam um filme, eles usam a mesma técnica, mas com uma menor fidelidade. Fazendo o storyboard, roteiro gráfico dos principais quadros, eles constroem um modelo de cada cena, com detalhes suficientes para comunicar sua intenção a todos os participantes da equipe de produção. De fato, a criação desse roteiro é a principal atividade do processo de produção, ajudando a equipe a visualizar, especificar, construir e documentar um modelo do filme, à medida que ele evolui da concepção para a construção e, por fim, a entrega.

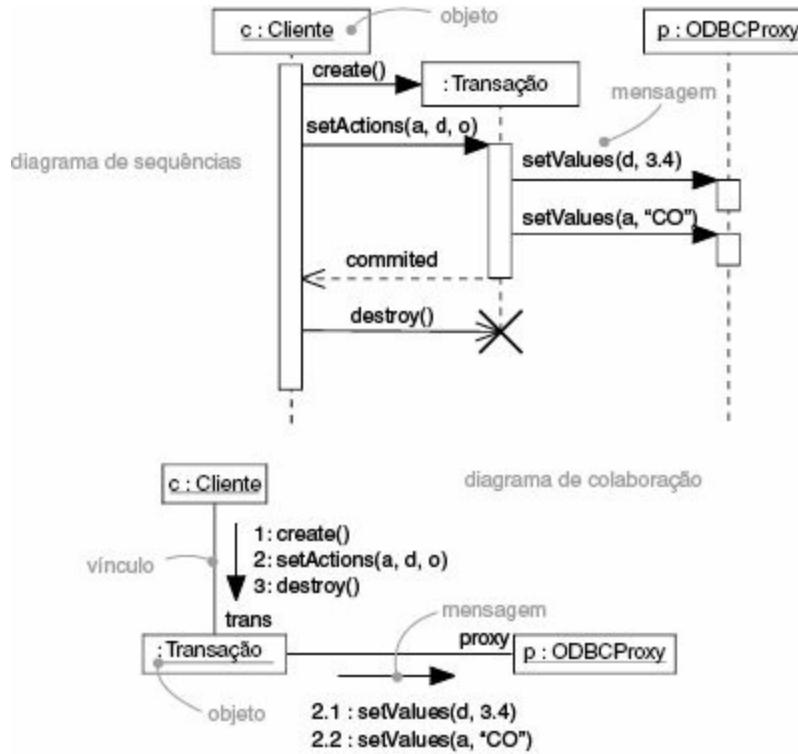
Na modelagem de sistemas complexos de software, encontra-se um problema semelhante: como você fará a modelagem de seus aspectos dinâmicos? Imagine, por um momento, como poderá visualizar um sistema em execução. Caso você tenha um depurador interativo anexado ao sistema, será capaz de assistir a uma seção da memória e observar como ela modifica seu conteúdo ao longo do tempo. Com um pouco mais de foco, poderá até monitorar vários objetos de interesse. Ao longo do tempo, você verá a criação

de alguns objetos, as alterações dos valores de seus atributos e depois a destruição de alguns deles.

- A modelagem de aspectos estruturais de um sistema é examinada nas Partes 2 e 3.

O valor da visualização de aspectos dinâmicos de um sistema dessa maneira é bastante limitado, principalmente no caso de um sistema distribuído, com vários fluxos de controle concorrentes. Você também poderia tentar compreender o sistema circulatório humano, observando o sangue que passa em um ponto de uma artéria ao longo do tempo. Uma forma melhor de fazer a modelagem dos aspectos dinâmicos de um sistema consiste em construir roteiros de cenários, envolvendo a interação de certos objetos de interesse e as mensagens que são trocadas entre eles.

Na UML, a modelagem desses roteiros é feita com a utilização de diagramas de interação. Conforme mostra a [Figura 19.1](#), esses roteiros podem ser construídos de duas maneiras: dando ênfase à ordenação temporal das mensagens e dando ênfase aos relacionamentos estruturais entre os objetos que interagem uns com os outros. Em qualquer uma dessas maneiras, os diagramas são semanticamente equivalentes; é possível converter um no outro sem qualquer perda de informação.



**Figura 19.1:**

Diagramas de interação

## TERMOS E CONCEITOS

Um *diagrama de interação* mostra uma interação formada por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que poderão ser trocadas entre eles. Um *diagrama de sequências* é um diagrama de interação que dá ênfase à ordenação temporal de mensagens. Graficamente, um diagrama de sequências é uma tabela que mostra objetos distribuídos no eixo X e mensagens, em ordem crescente no tempo, no eixo Y. Um *diagrama de comunicação* é um diagrama de interação que dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens. Graficamente, um diagrama de comunicação é uma coleção de vértices e arcos.

## PROPRIEDADES COMUNS

Um diagrama de interação é apenas um tipo especial de diagrama e compartilha as mesmas propriedades comuns de todos os demais diagramas – um nome e um conteúdo gráfico que são a projeção em um modelo. O que diferencia um diagrama de interação dos outros tipos de diagramas é o seu conteúdo particular.

- As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

## CONTEÚDO

Os diagramas de interação costumam conter o seguinte:

- › Papéis ou objetos
- › Comunicações ou vínculos
- › Mensagens

- Os objetos são examinados no [Capítulo 13](#); os vínculos são apresentados nos [Capítulos 14 e 16](#); as mensagens e interações são explicadas no [Capítulo 16](#); a estrutura interna aparece no [Capítulo 15](#); as colaborações são examinadas no [Capítulo 28](#).

**Nota:** Um diagrama de interação é basicamente uma projeção dos elementos encontrados em uma interação. A semântica de um contexto da interação, objetos e papéis, vínculos, mensagens e sequências se aplica aos diagramas de interação.

Como os outros diagramas, os diagramas de interação podem conter notas e restrições.

## DIAGRAMAS DE SEQUÊNCIAS

Um diagrama de sequências dá ênfase à ordenação temporal das mensagens. Conforme mostra a [Figura 19.2](#), um diagrama de sequências é formado colocando primeiro os objetos que participam da interação no nível superior do diagrama, ao longo do eixo X. Tipicamente, o objeto que inicia a

interação é colocado à esquerda e objetos mais subordinados vão crescendo à direita. A seguir, as mensagens que esses objetos enviam e recebem são colocadas ao longo do eixo Y, em ordem crescente de tempo, de cima para baixo. Isso proporciona ao leitor uma clara indicação visual do fluxo de controle ao longo do tempo.

Os diagramas de sequências têm duas características que os diferenciam dos diagramas de comunicação. Primeiro, existe linha de vida do objeto. A linha de vida do objeto é a linha tracejada vertical que representa a existência de um objeto em um período de tempo. Muitos objetos que aparecem em um diagrama de interação terão existência igual à duração da interação; assim, esses objetos são alinhados na parte superior do diagrama, com suas linhas de vida desenhadas de cima para baixo no diagrama. Objetos poderão ser criados durante a interação. Suas linhas de vida se iniciam com o destinatário da mensagem, estereotipado como *create* (desenhado na caixa no início da linha da vida). Objetos poderão ser destruídos durante a interação. Suas linhas de vida terminam com o destinatário da mensagem estereotipado como *destroy* (e são fornecidas as indicações visuais de um grande x, marcando o fim de suas vidas).

Se a interação representa a história de objetos específicos individuais, os símbolos de objetos com nomes sublinhados serão colocados no início das linhas da vida. Na maior parte do tempo, entretanto, você apresentará interações protótipicas. As linhas da vida não representam objetos específicos; elas representam papéis protótipicos que representam diferentes objetos em cada instância da interação. Nesse caso normal, você não sublinha os nomes, já que eles não são objetos específicos.

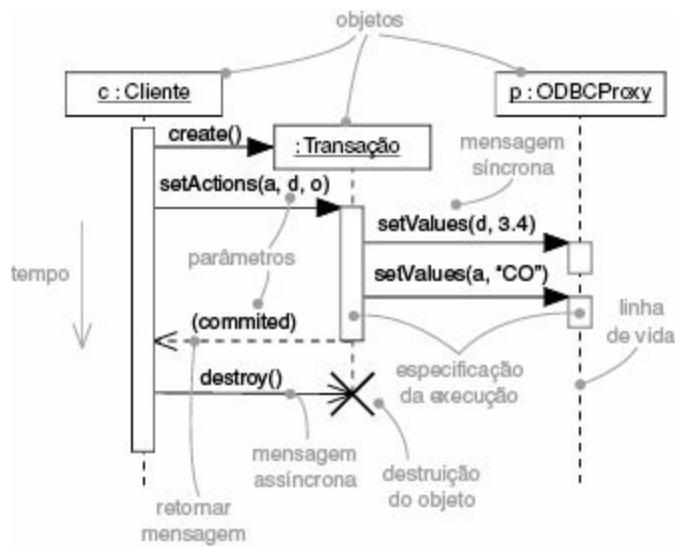


Figura 19.2:

Diagrama de sequências

**Nota:** Se um objeto mudar os valores de seus atributos, estado ou papéis, você pode colocar uma cópia do ícone do objeto no ponto de sua linha da vida em que a alteração ocorrer, indicando essas modificações.

Segundo, existe o foco de controle. O foco de controle é um retângulo alto e estreito, que mostra o período durante o qual um objeto está desempenhando uma ação, diretamente ou por meio de um procedimento subordinado. A parte superior do retângulo é alinhada com o início da ação; a parte inferior é alinhada com sua conclusão (e pode ser marcada por uma mensagem de retorno). Você pode exibir o aninhamento de um foco de controle (causado por recursão, uma chamada a uma autooperação ou pela chamada feita por outro objeto) mantendo um outro foco de controle ligeiramente à direita de seu foco pai (e pode fazer isso em uma profundidade arbitrária). Para ser especialmente preciso em relação ao local em que se encontra o foco de controle, você também pode sombrear a região do retângulo durante a qual o método do objeto está realmente sendo computado (sem que o controle tenha sido passado a outro objeto).

O principal conteúdo em um diagrama de sequências é o conjunto de mensagens. Uma mensagem é apresentada por uma seta de uma linha da vida para outra. A seta aponta para o destinatário. Se a mensagem é assíncrona, a

linha tem uma seta fina. Se a mensagem é síncrona (uma chamada), a linha tem uma seta triangular cheia. Uma resposta a uma mensagem síncrona (um retorno de chamada) é exibida por uma linha tracejada com uma seta fina. A mensagem de retorno pode ser omitida, já que há um retorno implícito após qualquer chamada, mas frequentemente é útil mostrar os valores de retorno.

A ordenação temporal em uma linha de vida única é significante. Normalmente, a distância exata não interessa; as linhas da vida só mostram sequências relativas; portanto, a linha da vida não é um diagrama de escala de tempo. Normalmente, as posições das mensagens em pares separados de linhas da vida não implicam nenhum sequenciamento de informações; as mensagens podem ocorrer em qualquer ordem. O conjunto inteiro de mensagens em linhas da vida separadas forma uma ordenação parcial. Entretanto, uma série de mensagens estabelece uma cadeia de causalidade, de maneira que qualquer ponto em outra linha da vida no fim da cadeia deve sempre seguir o ponto na linha da vida original no início da cadeia.

## **CONTROLE ESTRUTURADO NOS DIAGRAMAS DE SEQUÊNCIAS**

Uma sequência de mensagens é boa para mostrar uma única sequência linear, mas frequentemente precisamos mostrar condicionais e loops. Às vezes, é preciso mostrar a execução concorrente de várias sequências. O tipo de controle de alto nível pode ser apresentado com operadores de controle estruturado nos diagramas de sequências.

Um operador de controle é apresentado como uma região retangular no diagrama de sequências. Ele tem uma tag – um rótulo de texto dentro de um pequeno pentágono no canto superior esquerdo – para informar o tipo de operador de controle. O operador aplica-se às linhas da vida que o atravessam. Isso é considerado o corpo do operador. Se uma linha da vida não se aplica ao operador, ela pode ser interrompida no topo do operador de controle e retomada na base. Os tipos de controle mais comuns são os seguintes:

Execução opcional. A tag é `opt`. O corpo do operador de controle é executado se uma condição de guarda for verdadeira na entrada do operador. A condição de guarda é uma expressão booleana que pode aparecer entre colchetes na parte superior de qualquer linha da vida no corpo, e pode fazer referência a atributos desse objeto.

Execução condicional. A tag é `alt`. O corpo do operador de controle é dividido em várias sub-regiões por linhas horizontais tracejadas. Cada sub-região representa um ramo de um condicional. Cada sub-região tem uma condição de guarda. Se a condição de guarda de uma sub-região for verdadeira, a sub-região será executada. Entretanto, no máximo uma sub-região pode ser executada; se mais de uma condição de guarda for verdadeira, a escolha de sub-região não será determinística, podendo variar de execução para execução. Se nenhuma condição de guarda for verdadeira, o controle continua passando pelo operador de controle. Uma sub-região pode ter uma condição de guarda especial `[else]`; essa sub-região será executada, se nenhuma das outras condições de proteção for verdadeira.

Execução paralela. A tag é `par`. O corpo do operador de controle é dividido em várias sub-regiões por linhas horizontais tracejadas. Cada sub-região representa uma computação paralela (concorrente). Na maioria dos casos, cada sub-região contém diferentes linhas da vida. Quando o operador de controle entra, todas as sub-regiões são executadas paralelamente. A execução das mensagens em cada sub-região é sequencial, mas a ordem relativa das mensagens em sub-regiões paralelas é completamente arbitrária. Esse construto não deve ser usado se houver interação de computações diferentes. Entretanto, há várias situações do mundo real que se decompõem em atividades paralelas independentes; por isso, esse é um operador muito útil.

Execução de loop (iterativa). A tag é `loop`. Uma condição de guarda aparece na parte superior de uma linha da vida no corpo. O corpo do loop é executado repetidamente enquanto a condição de guarda é verdadeira, antes de cada iteração. Quando a condição de guarda é falsa na parte superior do corpo, o controle passa fora do operador de controle.

Há muitos outros tipos de operadores, mas esses são os mais úteis.

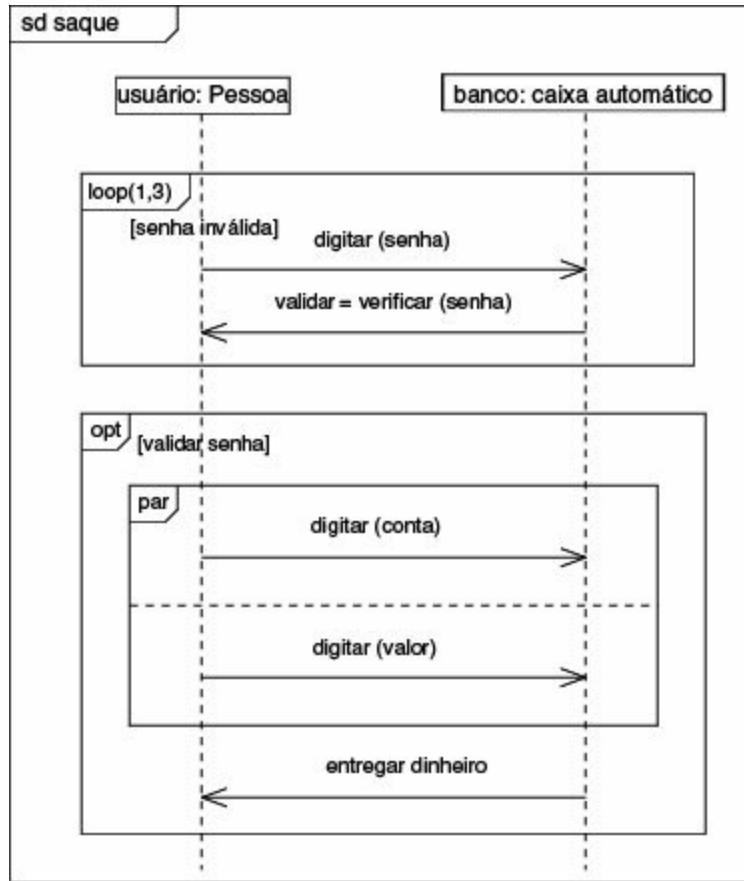
Para fornecer uma indicação clara dos limites, um diagrama de sequências pode estar contido em um retângulo, com uma tag no canto superior esquerdo. A tag é `sd`, que pode ser seguida pelo nome do diagrama.

A [Figura 19.3](#) mostra um exemplo simplificado que ilustra alguns operadores de controle. O usuário inicia a sequência. O primeiro operador é um loop. Os números entre parênteses (1,3) indicam o número de vezes mínimo e máximo que o corpo do loop deve ser executado. Como o mínimo é um, o corpo é sempre executado pelo menos uma vez antes de a condição ser testada. No loop, o usuário informa a senha e o sistema a verifica. O loop continua enquanto a senha estiver incorreta. Entretanto, após três tentativas, o loop termina de qualquer maneira.

O próximo operador é um opcional. O corpo do opcional será executado se a senha for válida; caso contrário, o resto do diagrama de sequências será ignorado. O corpo do operador opcional contém um operador paralelo. Os operadores podem ser aninhados da maneira apresentada.

O operador paralelo tem duas sub-regiões: uma permite que o usuário informe uma conta, e a outra permite que o usuário informe um montante. Como elas são paralelas, não há ordem obrigatória para fazer as duas entradas; elas podem ocorrer em qualquer ordem. Esse procedimento enfatiza que a concorrência nem sempre implica execução simultânea fisicamente. A concorrência realmente significa que duas ações são descoordenadas e podem ocorrer em qualquer ordem. Se forem realmente ações independentes, elas podem se sobrepor; se forem ações sequenciais, podem ocorrer em qualquer ordem.

Após a realização das duas ações, o operador paralelo estará completo. Na próxima ação no operador opcional, o banco entrega dinheiro ao usuário. O diagrama de sequências agora está completo.

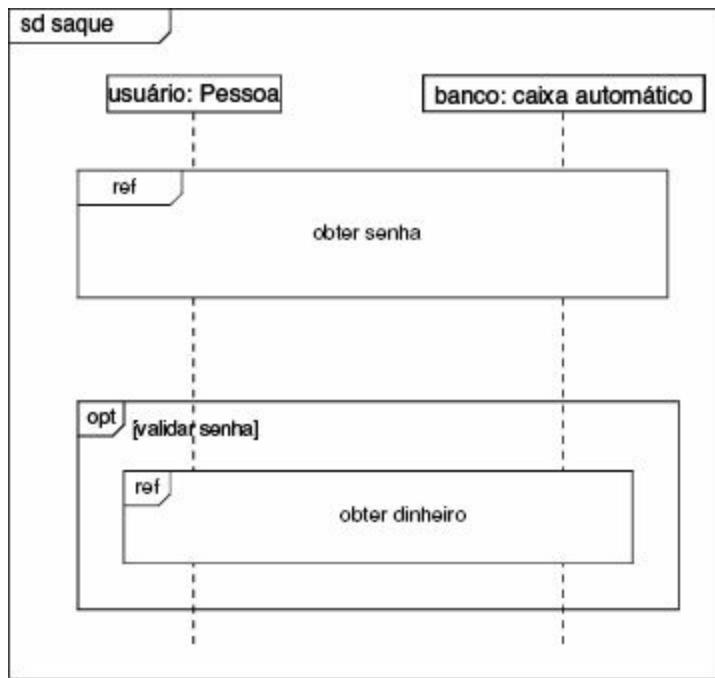


**Figura 19.3:**  
Operadores de controle estruturado

## DIAGRAMAS DE ATIVIDADES ANINHADAS

Os diagramas de atividades que são muito grandes podem ser difíceis de entender. As seções estruturadas de uma atividade podem ser organizadas em uma atividade subordinada, especialmente se a atividade subordinada é usada mais de uma vez na atividade principal. A atividade principal e as atividades subordinadas são apresentadas em diagramas separados. O diagrama de atividade principal mostra o uso de uma atividade subordinada por um retângulo com a tag `ref` em seu canto superior esquerdo; o nome do comportamento subordinado é apresentado na caixa. O comportamento subordinado não se restringe a um diagrama de atividades; ele também pode ser uma máquina de estados, um diagrama de sequências ou outra especificação comportamental. A [Figura 19.4](#) mostra o diagrama da [Figura](#)

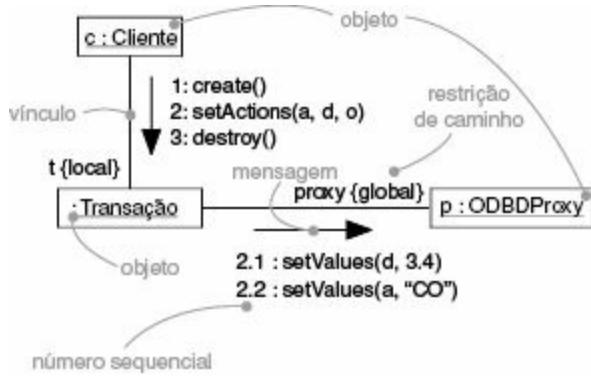
[19.3](#) reorganizado: duas seções foram movidas para diagramas de atividades separados e suas referências encontram-se no diagrama principal.



**Figura 19.4:**  
Diagrama de atividade aninhada

## DIAGRAMAS DE COMUNICAÇÃO

Um diagrama de comunicação dá ênfase à organização dos objetos que participam de uma interação. Conforme mostra a [Figura 19.5](#), você forma o diagrama de comunicação colocando primeiro os objetos que participam da interação como os vértices de um gráfico. A seguir, representa os vínculos que conectam esses objetos como os arcos do gráfico. Os vínculos podem ter nomes de papéis para identificá-los. Por fim, adorna esses vínculos com as mensagens que os objetos enviam e recebem. Isso proporciona ao leitor uma clara indicação visual do fluxo de controle no contexto da organização estrutural dos objetos que colaboram.



**Figura 19.5:**

Diagrama de comunicação

**Nota:** Ao contrário do diagrama de sequências, você não mostra a linha da vida de um objeto explicitamente em um diagrama de comunicação, embora possa mostrar as mensagens *create* e *destroy*. Além disso, você não mostra o foco do controle explicitamente em um diagrama de comunicação, embora o número da sequência de cada mensagem possa indicar aninhamento.

Os diagramas de comunicação têm duas características que os diferenciam dos diagramas de sequências. Primeiro, existe o caminho. Você representa um caminho que corresponde a uma associação. Também representa caminhos correspondentes a variáveis locais, parâmetros, variáveis globais e autoacesso. Um caminho representa uma fonte de conhecimento para um objeto.

Segundo, existe o número de sequência. Para indicar a ordem temporal de uma mensagem, use um número como prefixo da mensagem (iniciando com a mensagem numerada como 1), aumentando unitariamente para cada nova mensagem no fluxo de controle (2, 3 e assim por diante). Para exibir um aninhamento, você pode usar a numeração decimal de Dewey (1 é a primeira mensagem; 1.1 é a primeira mensagem aninhada na mensagem 1; 1.2 é a segunda mensagem aninhada na mensagem 1; e assim por diante). Você pode exibir o aninhamento até uma profundidade arbitrária. Observe também que, no mesmo vínculo, é possível exibir muitas mensagens (possivelmente enviadas a partir de diferentes direções) e cada uma terá um número de sequência único.

- ➡ Uma forma avançada de números de sequência para diferenciar fluxos de controle concorrentes é examinada no [Capítulo 23](#); os estereótipos de caminho são apresentados no [Capítulo 18](#); a iteração e a ramificação complexa podem ser especificadas mais facilmente em diagramas de atividade, conforme é explicado no [Capítulo 19](#).

Na maior parte do tempo, você fará a modelagem de fluxos de controle diretos e sequenciais. Entretanto, também pode fazer a modelagem de fluxos mais complexos, envolvendo iteração e ramificação. Uma iteração representa uma sequência repetida de mensagens. Para fazer a modelagem de uma iteração, use como prefixo do número de sequência de uma mensagem uma expressão de iteração como  $* [i := 1..n]$  (ou apenas  $*$ , caso queira indicar a iteração, mas sem especificar seus detalhes). A iteração indica que a mensagem (e qualquer outra mensagem aninhada) será repetida de acordo com a expressão dada. De modo semelhante, uma condição representa uma mensagem cuja execução é contingente à avaliação de uma condição booleana. Para fazer a modelagem de uma condição, use, como prefixo do número de sequência de uma mensagem, uma expressão condicional, como  $[x > 0]$ . Os caminhos alternativos de uma ramificação terão o mesmo número de sequência, mas cada caminho deve ser diferenciado de maneira única por uma condição que não se sobreponha a outra.

Tanto para a iteração como para a ramificação, a UML não prescreve o formato da expressão colocada entre colchetes; você pode utilizar qualquer pseudocódigo ou a sintaxe de uma determinada linguagem de programação.

**Nota:** Os vínculos entre objetos não são mostrados explicitamente em um diagrama de sequências. Também o número de sequência de uma mensagem não é exibido explicitamente em um diagrama de sequências: ele está implícito na ordenação física das mensagens, de cima para baixo no diagrama. Entretanto a iteração e a ramificação podem ser mostradas, usando as estruturas de controle do diagrama de sequências.

## EQUIVALÊNCIA SEMÂNTICA

Como ambos são derivados das mesmas informações de um metamodelo da UML, o diagrama de sequências e o diagrama de comunicação são semanticamente equivalentes. Como resultado, o diagrama de uma forma pode ser convertido no outro sem qualquer perda de informação, conforme você pode observar nas duas figuras anteriores, que são semanticamente equivalentes. Entretanto, isso não significa que os dois diagramas visualizarão as mesmas informações explicitamente. Por exemplo, nas duas figuras anteriores, o diagrama de comunicação mostra como os objetos estão vinculados (observe os estereótipos {local} e {global}), enquanto isso não ocorre no diagrama de sequências correspondente. De modo semelhante, o diagrama de sequências mostra o retorno da mensagem (observe o valor de retorno committed na [Figura 19.2](#)), mas o mesmo não ocorre com o diagrama de comunicação correspondente. Nos dois casos, os dois diagramas compartilham o mesmo modelo subjacente, mas cada um poderá representar coisas que o outro não é capaz. Entretanto, um modelo inserido em um formato pode perder algumas informações mostradas no outro formato; portanto, embora o modelo subjacente possa incluir os dois tipos de informações, os dois tipos de diagrama podem conduzir a diferentes modelos.

## USOS COMUNS

Os diagramas de interação são utilizados para a modelagem dos aspectos dinâmicos de um sistema. Esses aspectos dinâmicos podem envolver a interação de qualquer tipo de instância em qualquer visão da arquitetura de um sistema, incluindo instâncias de classes (e as classes ativas), interfaces, componentes e nós.

- As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#); as instâncias são apresentadas no [Capítulo 13](#); as classes aparecem nos [Capítulos 4 e 9](#); as classes ativas são explicadas no [Capítulo 23](#); as interfaces são examinadas no [Capítulo 11](#); os componentes são discutidos no [Capítulo 15](#); os nós são apresentados no [Capítulo 27](#); os sistemas e subsistemas são explicados no [Capítulo 32](#); as

*operações são examinadas nos Capítulos 4 e 9; os casos de uso aparecem no Capítulo 17; as colaborações são explicadas no Capítulo 28.*

Quando você usa um diagrama de interação para fazer a modelagem de algum aspecto dinâmico de um sistema, você o faz no contexto do sistema como um todo, um subsistema, uma operação ou uma classe. Você também pode anexar os diagramas de interação aos casos de uso (para fazer a modelagem de um cenário) e às colaborações (para fazer a modelagem dos aspectos dinâmicos de uma sociedade de objetos).

Ao fazer a modelagem dos aspectos dinâmicos de um sistema, você tipicamente utiliza os diagramas de interação de duas maneiras.

### 1. Para a modelagem dos fluxos de controle por ordenação temporal.

Aqui você utilizará os diagramas de sequências. A modelagem do fluxo de controle pela ordenação temporal dá ênfase à passagem de mensagens, à medida que ocorrem com o passar do tempo, o que é uma forma particularmente útil de visualizar o comportamento dinâmico no contexto de um cenário de caso de uso. Os diagramas de sequências fazem um trabalho melhor, para a visualização de uma ramificação e iteração simples, do que os diagramas de comunicação.

### 2. Para a modelagem de fluxos de controle por organização.

Aqui você utilizará os diagramas de comunicação. A modelagem do fluxo de controle pela organização dá ênfase aos relacionamentos estruturais existentes entre as instâncias da interação, juntamente com as mensagens que poderão ser passadas.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE FLUXOS DE CONTROLE POR ORDENAÇÃO TEMPORAL

Considere os objetos que se encontram no contexto de um sistema, subsistema, operação ou classe. Considere também os objetos e papéis que

participam em um caso de uso ou em uma colaboração. Para fazer a modelagem do fluxo de controle referente a esses objetos e papéis, você utiliza um diagrama de interação; para dar ênfase à passagem das mensagens como ocorre ao longo do tempo, utilize um diagrama de sequências, um tipo de diagrama de interação.

- ➡ *Os sistemas e subsistemas são examinados no Capítulo 32; as operações e as classes são apresentadas nos Capítulos 4 e 9; os casos de uso são discutidos no Capítulo 17; as colaborações são mostradas no Capítulo 28.*

Para fazer a modelagem do fluxo de controle por ordenação temporal:

- › Defina o contexto para a interação; se é um sistema, subsistema, operação ou classe ou um cenário de caso de uso ou colaboração.
- › Defina o estágio para a interação, identificando quais objetos desempenham um papel na interação. Distribua-os no diagrama de sequências da esquerda para a direita, colocando os objetos mais importantes à esquerda e seus objetos vizinhos à direita.
- › Defina a linha de vida para cada objeto. Na maioria dos casos, os objetos persistirão ao longo de toda a interação. Para aqueles objetos criados e destruídos durante a interação, defina suas linhas de vida, conforme seja apropriado, e indique explicitamente seu nascimento e morte com as mensagens adequadamente estereotipadas.
- › Começando com as mensagens que iniciam a interação, distribua cada mensagem subsequente de cima para baixo entre as linhas de vida, mostrando as propriedades de cada mensagem (como seus parâmetros), conforme necessário para explicar a semântica da interação.
- › Se for necessário visualizar o aninhamento das mensagens ou dos pontos no tempo quando a computação real estiver sendo realizada, adorne a linha de vida de cada objeto com seu foco de controle.

- » Se for necessário especificar restrições de tempo ou espaço, adorne cada mensagem com uma marca de tempo e anexe as restrições de tempo ou de espaço adequadas.
- » Se for necessário especificar o fluxo de controle de modo mais formal, anexe pré e pós-condições a cada mensagem.
  - ➡ *Marcas do tempo são discutidas no Capítulo 24; pré e pós-condições são discutidas no Capítulo 4; pacotes são discutidos no Capítulo 12.*

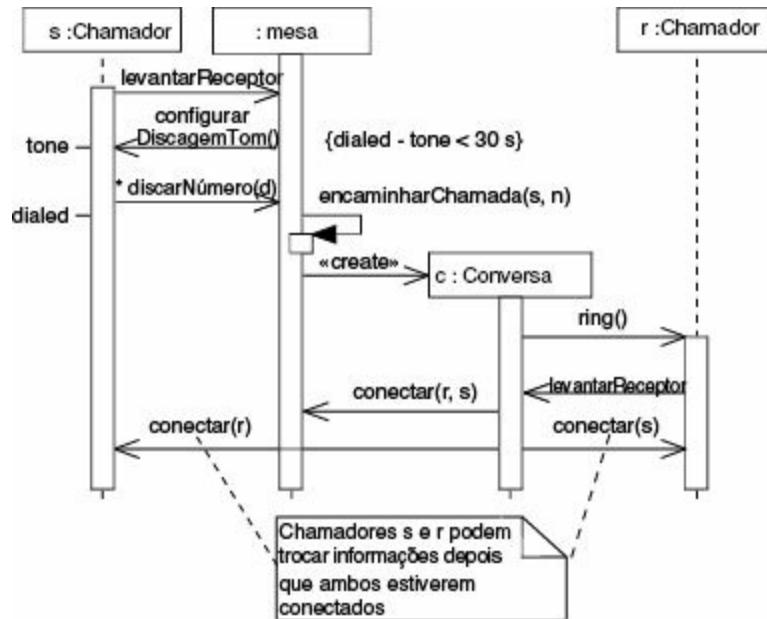
Um único diagrama de sequências é capaz de mostrar somente um fluxo de controle (embora você possa exibir variações simples, utilizando a notação da UML para a iteração e a ramificação). Tipicamente, você terá vários diagramas de interação, alguns dos quais são básicos e outros que mostram caminhos alternativos ou condições de exceção. Os pacotes podem ser empregados para organizar essas coleções de diagramas de sequências, dando a cada diagrama um adequado para diferenciá-lo de seus irmãos.

Por exemplo, a [Figura 19.6](#) mostra um diagrama de sequência, especificando o fluxo de controle envolvido ao iniciar uma simples ligação telefônica entre dois participantes. Nesse nível de abstração, existem quatro objetos envolvidos: dois Chamadores (*s* e *r*), um telefone Mesa sem nome e *c*, a retificação da Conversa entre dois participantes.

A sequência começa com um Chamador (*s*) enviando um sinal (*levantarReceptor*) para o objeto Mesa. Por sua vez, o objeto Mesa chama *configurarDiscagemTom* em Chamador e Chamador itera a mensagem *discarNúmero*. Observe que essa mensagem tem uma restrição de tempo (seu *tempoExecução* deve ser menor do que 30 segundos). Esse diagrama não indica o que acontece se a restrição de tempo for violada. Com essa finalidade, poderia ser incluída uma ramificação ou um outro diagrama de sequências completamente separado. O objeto Mesa então chama a si próprio com a mensagem *encaminhadaChamada*. A seguir, é criado o objeto Conversa (*c*) ao qual o trabalho restante é delegado. Apesar de isso não ser mostrado nessa interação, (*c*) teria a responsabilidade adicional de ser um participante no mecanismo da

cobrança da chamada (o que seria expresso em outro diagrama de interação). O objeto Conversa (c) chama o Chamador (r), que envia assincronamente a mensagem levantarReceptor. O objeto Conversa então instrui o Mesa a realizar a operação conectar e a se conectar à chamada e depois instrui os dois objetos Chamador a conectar; depois disso, os dois objetos poderão trocar informações, conforme é indicado pela nota anexada.

- Os sinais são examinados no [Capítulo 21](#); as marcas de tempo são apresentadas no [Capítulo 24](#); as restrições são explicadas no [Capítulo 6](#); as responsabilidades são discutidas no [Capítulo 4](#); as notas são examinadas no [Capítulo 6](#).



**Figura 19.6:**

A modelagem de fluxos de controle pela ordenação temporal

## MODELAGEM DE FLUXOS DE CONTROLE POR ORGANIZAÇÃO

Considere os objetos que se encontram no contexto de um sistema, subsistema, operação ou classe. Considere também os objetos e papéis que participam em um caso de uso ou em uma colaboração. Para fazer a modelagem do fluxo de controle referente a esses objetos e papéis, você utiliza um diagrama de interação; para dar ênfase à passagem das mensagens

como ocorre ao longo do tempo, utilize um diagrama de comunicação, um tipo de diagrama de interação.

- ➡ Os sistemas e subsistemas são examinados no [Capítulo 32](#); as operações e as classes são apresentadas nos [Capítulos 4 e 9](#); os casos de uso são discutidos no [Capítulo 17](#); as colaborações são mostradas no [Capítulo 28](#).

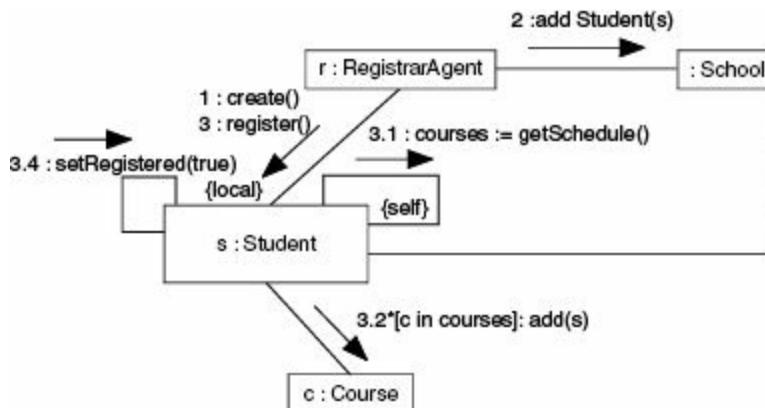
Para fazer a modelagem do fluxo de controle por organização:

- › Defina o contexto para a interação; se é um sistema, subsistema, operação ou classe ou um cenário de caso de uso ou colaboração.
- › Defina o estágio para a interação, identificando quais objetos desempenham um papel na interação. Distribua-os no diagrama de comunicação da esquerda para a direita, colocando os objetos mais importantes à esquerda e seus objetos vizinhos à direita.
- › Especifique os vínculos entre esses objetos, juntamente com as mensagens que poderão ser passadas.
  1. Distribua primeiro os vínculos da associação; esses são os mais importantes, por representarem conexões estruturais.
  2. A seguir distribua os outros vínculos e adorne-os com estereótipos de caminhos adequados (como global e local) para especificar explicitamente como esses objetos estão relacionados uns com os outros.
- ➡ Relacionamentos de dependência são discutidos nos [Capítulos 5 e 10](#); restrições de caminho são explicados no [Capítulo 16](#).
- › Começando com a mensagem que inicia essa interação, anexe cada mensagem subsequente ao vínculo apropriado, definindo seu número de sequência, conforme seja adequado. Mostre o aninhamento, utilizando a numeração decimal de Dewey.

- » Se for necessário especificar restrições de tempo ou espaço, adorne cada mensagem com uma marca de tempo e anexe as restrições de tempo ou espaço adequadas.
- » Se for necessário especificar o fluxo de controle de modo mais formal, anexe pré e pós-condições a cada mensagem.
  - ➡ *Marcas do tempo são discutidas no [Capítulo 24](#); pré e pós-condições são discutidas no [Capítulo 4](#); pacotes são discutidos no [Capítulo 12](#).*

Assim como nos diagramas de sequências, um único diagrama de comunicação é capaz de mostrar somente um fluxo de controle (embora você possa exibir variações simples, utilizando a notação da UML para a iteração e a ramificação). Tipicamente, você terá vários diagramas de interação, alguns dos quais são básicos e outros que mostram caminhos alternativos ou condições de exceção. Os pacotes podem ser empregados para organizar essas coleções de diagramas de sequências, dando a cada diagrama um nome adequado para diferenciá-lo de seus irmãos.

Por exemplo, a [Figura 19.7](#) mostra um diagrama de comunicação, especificando o fluxo de controle envolvido ao matricular um novo aluno na escola, com uma ênfase nos relacionamentos estruturais entre esses objetos. Há quatro papéis: um RegistrarAgent (r) (Agente de Matrícula), um Student (s) (Estudante), um Course (c) (Curso) e um papel School (Escola) sem nome. O fluxo de controle é numerado explicitamente. A ação começa com o RegistrarAgent criando um objeto Student, adicionando o aluno à escola (a mensagem addStudent (adicionar estudante)) e solicitando ao objeto Student que se matricule. O objeto Student usa o getSchedule (obtém programa) em si mesmo, obtendo um conjunto de objetos Course para os quais deve se matricular. Para cada objeto curso no conjunto, o objeto Student se adiciona a cada objeto Course.



**Figura 19.7:**

A modelagem de fluxos de controle por organização

## ENGENHARIA DIRETA E REVERSA

A *engenharia direta* (a criação de código a partir de um modelo) é possível tanto para diagramas de sequências como de colaboração, principalmente se o contexto do diagrama for uma operação. Por exemplo, usando o diagrama de colaboração anterior, uma ferramenta de engenharia direta razoavelmente inteligente poderia gerar o seguinte código em Java para a operação `register`, anexada à classe `Student`.

```

public void register() {
    CourseCollection c = getSchedule();
    for (int i = 0; i < c.size(); i++)
        c.item(i).add(this);
    this.registered = true;
}

```

“Razoavelmente inteligente” significa que a ferramenta precisaria compreender que `getSchedule` retorna um objeto `CourseCollection`, o que poderia ser determinado pela observação da assinatura da operação. Percorrendo o conteúdo desse objeto utilizando um idioma de iteração padrão (que a ferramenta poderia conhecer implicitamente), o código então poderia ser generalizado para qualquer número de cursos oferecidos.

A *engenharia reversa* (a criação de um modelo a partir do código) também é possível tanto para diagramas de sequências quanto para os de colaboração, principalmente se o contexto do diagrama for o corpo de uma

operação. Os segmentos do diagrama anterior poderiam ter sido produzidos por uma ferramenta a partir da execução prototípica da operação register.

*Nota: A engenharia direta é objetiva; a engenharia reversa é complicada. É fácil obter muitas informações a partir de uma engenharia reversa simples e, assim, a parte complicada é saber quais detalhes deverão ser aproveitados.*

Entretanto, mais interessante do que fazer a engenharia reversa de um modelo a partir de um código é a animação de um modelo em relação à execução de um sistema entregue. Por exemplo, considerando o diagrama anterior, uma ferramenta poderia animar as mensagens apresentadas no diagrama como se tivessem sido transmitidas em um sistema em execução. Ainda melhor, com essa ferramenta sob o controle de um depurador, seria possível controlar a velocidade da execução, provavelmente definindo pontos de parada para interromper a ação em pontos de interesse e examinar os valores dos atributos de objetos individuais.

## DICAS E SUGESTÕES

Ao criar diagramas de interação na UML, lembre-se de que os diagramas de sequências e de comunicação são projeções do mesmo modelo de aspectos dinâmicos de um sistema. Nenhum único diagrama de interação é capaz de captar tudo sobre os aspectos dinâmicos do sistema. Em vez disso, use muitos diagramas de interação para fazer a modelagem da dinâmica do sistema como um todo, como também de seus subsistemas, operações, classes, casos de uso e colaborações. Um diagrama de interação bem estruturado:

- » Está voltado para comunicar um único aspecto da dinâmica do sistema.
- » Contém somente os elementos essenciais à compreensão desse aspecto.
- » Fornece detalhes consistentes com seu nível de abstração e expõe somente os adornos essenciais à compreensão.
- » Não é tão minimalista, que informe mal ao leitor sobre a semântica que é importante.

Ao definir um diagrama de interação:

- › Dê-lhe um nome capaz de comunicar o seu propósito.
- › Use o diagrama de sequências, se quiser dar ênfase à ordenação temporal das mensagens. Use o diagrama de colaboração, se quiser dar ênfase à organização dos objetos envolvidos na interação.
- › Distribua os elementos para minimizar o cruzamento de linhas.
- › Use notas e cores como indicações visuais para chamar atenção para características importantes do diagrama.
- › Use a ramificação com cautela; você pode representar bem melhor as ramificações complexas, utilizando os diagramas de atividades.

CAPÍTULO

---

**20**

---

# Diagramas de Atividades

## Neste capítulo

- » Modelagem do fluxo de trabalho
- » Modelagem de uma operação
- » Engenharia direta e reversa

**O**s diagramas de atividades são um dos cinco diagramas disponíveis na UML para a modelagem de aspectos dinâmicos de sistemas. Um diagrama de atividade é essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra. Ao contrário de um gráfico de fluxo tradicional, um diagrama de atividades mostra a concorrência, bem como as ramificações de controle.

► *Os diagramas de sequências, os diagramas de comunicação, os diagramas de gráficos de estados e os diagramas de casos de uso também fazem a modelagem dos aspectos dinâmicos de sistemas; os diagramas de sequências e os de comunicação são examinados no Capítulo 19; os diagramas de gráficos de estados são apresentados no Capítulo 25; os diagramas de casos de uso são explicados no Capítulo 18; as ações aparecem no Capítulo 16.*

Os diagramas de atividades serão empregados para fazer a modelagem de aspectos dinâmicos do sistema. Na maior parte, isso envolve a modelagem das etapas sequenciais (e possivelmente concorrentes) em um processo computacional. Com um diagrama de atividade, você também pode fazer a modelagem do fluxo de um objeto, à medida que ele passa de um estado para

outro em pontos diferentes do fluxo de controle. Os diagramas de atividades poderão permanecer isolados para visualizar, especificar, construir e documentar a dinâmica de uma sociedade de objetos, ou poderão ser utilizados para fazer a modelagem do fluxo de controle de uma operação. Enquanto os diagramas de interação dão ênfase ao fluxo de controle de um objeto para outro, os diagramas de atividades dão ênfase ao fluxo de controle de uma etapa para outra. Uma atividade é uma execução estruturada em andamento de um comportamento. A execução de uma atividade finalmente é expandida na execução de ações individuais, e cada uma dessas pode mudar o estado do sistema ou comunicar mensagens.

Os diagramas de atividades não são importantes somente para a modelagem dos aspectos dinâmicos de um sistema, mas também para a construção de sistemas executáveis por meio de engenharia direta e reversa.

## PRIMEIROS PASSOS

Considere o fluxo de trabalho associado à construção de uma casa. Primeiro, você seleciona um local. A seguir, contrata um arquiteto para projetar sua casa. Uma vez definida a planta, seu desenvolvedor determina os custos da casa. Após concordar com um preço e com uma forma de pagamento, a construção pode começar. As licenças são tiradas, o terreno é cavado, a fundação é cimentada, as estruturas são erguidas e assim por diante até tudo ficar pronto. Você então recebe as chaves e um certificado de habite-se e toma posse da casa.

Embora seja uma grande simplificação do que realmente acontece em um processo de construção, essa descrição capta o percurso crítico do fluxo de trabalho correspondente. Em um projeto real, existem várias atividades paralelas em vários setores. Eletricistas podem estar trabalhando ao mesmo tempo que encanadores e carpinteiros, por exemplo. Também haverá condições e ramificações. Por exemplo, dependendo de resultados de testes no solo, poderá ser necessário explodir, escavar ou planar. Poderá até haver

loops. Por exemplo, uma vistoria na obra poderá revelar alguma violação à legislação, resultando em desfazer e recomeçar o trabalho.

No mercado da construção, técnicas como os gráficos de Gantt e de Pert costumam ser utilizadas para visualizar, especificar, construir e documentar o fluxo de trabalho do projeto.

Na modelagem de sistemas complexos de software, existe um problema semelhante. Qual a melhor maneira de fazer a modelagem de um fluxo de trabalho ou de uma operação, que são aspectos da dinâmica do sistema? A resposta é que existem duas opções básicas, semelhantes ao uso dos gráficos de Gantt e de Pert.

- A modelagem de aspectos estruturais de um sistema é examinada nas Partes 2 e 3; os diagramas de interação são apresentados no [Capítulo 19](#).

Por um lado, você pode preparar roteiros de cenários, envolvendo a interação de certos objetos de interesse e as mensagens que poderão ser trocadas entre eles. Na UML, você pode fazer a modelagem desses roteiros de duas maneiras: com ênfase à ordenação temporal das mensagens (utilizando diagramas de sequências) ou com ênfase aos relacionamentos estruturais existentes entre os objetos que interagem (utilizando diagramas de colaboração). Diagramas de interação como esses são semelhantes aos gráficos de Gantt, que focalizam os objetos (recursos) que executam alguma atividade ao longo do tempo.

Por outro lado, a modelagem desses aspectos dinâmicos pode ser feita com a utilização de diagramas de atividades, que primeiro focalizam as atividades que ocorrem entre objetos, conforme mostra a [Figura 20.1](#). Em relação a isso, os diagramas de atividades são semelhantes aos gráficos de Pert. Um diagrama de atividades é essencialmente um fluxograma que dá ênfase à atividade que ocorre ao longo do tempo. Você pode considerar um diagrama de atividades como um diagrama de interação cujo interior é revelado. Um diagrama de interação observa os objetos que passam

mensagens; um diagrama de atividades observa as operações passadas entre os objetos. A diferença semântica é sutil, mas resulta em formas distintas de observar o mundo.

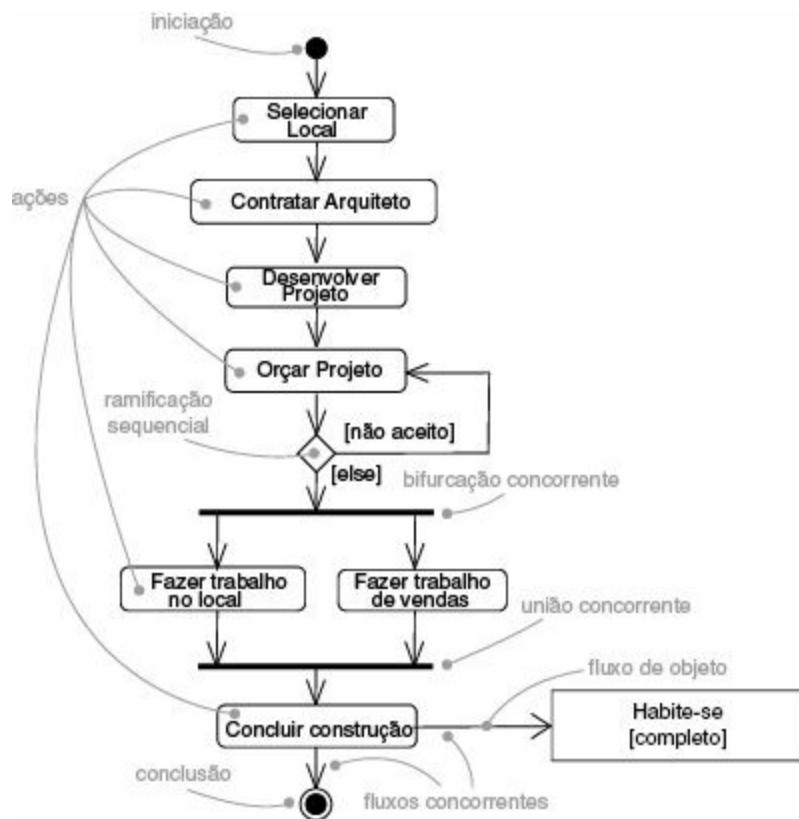


Figura 20.1:

Diagramas de atividades

## TERMOS E CONCEITOS

Um *diagrama de atividades* mostra o fluxo de uma atividade para outra. Uma *atividade* é uma execução em andamento não atômica em uma máquina de estados. As atividades efetivamente resultam em alguma *ação*, formada pelas computações executáveis atômicas que resultam em uma mudança de estado do sistema ou o retorno de um valor. As ações abrangem a chamada a outras operações, enviando um sinal, criando ou destruindo um objeto ou alguma computação pura, como o cálculo de uma expressão. Graficamente, o diagrama de atividades é uma coleção de nós e arcos.

## PROPRIEDADES COMUNS

O diagrama de atividades é apenas um tipo especial de diagrama e compartilha as mesmas propriedades comuns de todos os demais diagramas – um nome e um conteúdo gráfico que são a projeção em um modelo. O que diferencia um diagrama de interação dos demais tipos de diagramas é o seu conteúdo particular.

- ➡ As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

## CONTEÚDO

Os diagramas de atividades costumam conter o seguinte:

- › Ações
- › Nós de atividade
- › Fluxos
- › Valores de objetos

- ➡ Os estados, transições e máquinas de estados são examinados no [Capítulo 22](#); os objetos são apresentados no [Capítulo 13](#).

Como todos os outros diagramas, os diagramas de atividades poderão conter notas e restrições.

## AÇÕES E NÓS DE ATIVIDADES

No fluxo de controle modelado por um diagrama de atividade, as coisas acontecem. É possível calcular uma expressão que defina o conjunto de valor de um atributo ou que retorne algum valor. Alternativamente, você poderá chamar uma operação em um objeto, enviar um sinal a um objeto ou até criar ou destruir um objeto. Essas computações atômicas executáveis são chamadas ações. Conforme mostra a [Figura 20.2](#), você representa uma ação,

utilizando uma caixa arredondada. Dentro dessa forma, você poderá escrever qualquer expressão.

- Os atributos e as operações são examinados nos Capítulos 4 e 9; os sinais são apresentados no [Capítulo 21](#); a criação e a destruição de objetos são explicadas no [Capítulo 16](#); os estados e as máquinas de estados são examinados no [Capítulo 22](#).

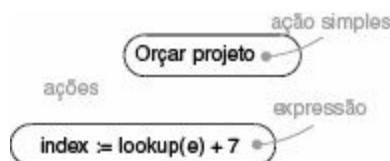


Figura 20.2:

Ações

**Nota:** A UML não prescreve uma linguagem para essas expressões. De modo abstrato, você poderá simplesmente usar texto estruturado; mas de modo concreto, poderá usar a sintaxe e a semântica de uma linguagem de programação específica.

As ações não podem ser decompostas. Além disso, são atômicas, significando que os eventos poderão ocorrer, mas o comportamento interno da ação não é visível. Você não pode executar parte de uma ação; ou a executa completamente ou não a executa. Por fim, o trabalho de uma ação é geralmente considerado como ocupando um tempo de execução insignificante, mas algumas ações podem ter duração substancial.

**Nota:** No mundo real, é claro, toda computação consome alguma quantidade de tempo e de espaço. Principalmente no caso de sistemas complexos em tempo real, é importante fazer a modelagem dessas propriedades.

- A modelagem de tempo e espaço é examinada no [Capítulo 24](#).

Um *nó de atividade* é uma unidade organizacional dentro de uma atividade. Em geral, os nós de atividades são grupos de ações aninhados ou outros nós de atividades aninhados. Além disso, os nós de atividades têm uma subestrutura visível e, em geral, são considerados como tomado algum

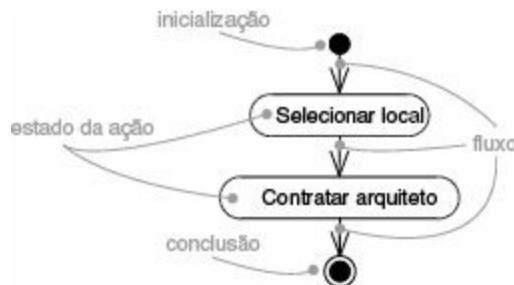
tempo para serem completados. Pense em um estado de ação como um caso especial de um nó de atividade. Uma ação é um nó de atividade que não pode mais ser decomposto. De modo semelhante, pense em um nó de atividade como um estado composto, cujo fluxo de controle é formado por outros nós de atividades e ações. Observando os detalhes de um nó de atividade, você encontrará outro diagrama de atividades. Conforme mostra a [Figura 20.3](#), não existe uma distinção notacional entre ações e nós de atividades, exceto pelo fato de um nó de atividade admitir partes adicionais, que normalmente serão mantidas no plano de fundo por uma ferramenta de edição.



**Figura 20.3:**  
Nós de atividades

## FLUXOS DE CONTROLE

Quando a ação ou o nó de atividade está completo, o fluxo de controle passa imediatamente à próxima ação ou nó de atividade. Você especifica esse fluxo utilizando setas de fluxo para mostrar o caminho de uma ação ou nó de atividade para o seguinte. Na UML, um fluxo é representado como uma seta simples da ação predecessora para a sucessora, sem um rótulo de evento, conforme mostra a [Figura 20.4](#).



**Figura 20.4:**  
Transições de parada

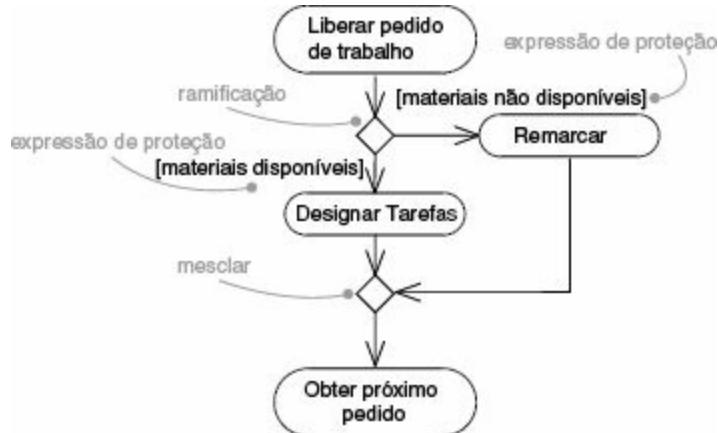
Na verdade, um fluxo de controle inicia e termina em algum lugar (a menos, é claro, que seja um fluxo infinito e, nesse caso terá um início, mas não um fim). Portanto, conforme mostra a figura, você pode especificar esse estado inicial (uma bola cheia) e o estado de parada (a bola cheia e no interior de um círculo) como símbolos especiais.

## RAMIFICAÇÃO

Transições sequenciais simples são comuns, mas não são o único tipo de caminho necessário para fazer a modelagem de um fluxo de controle. Assim como em um fluxograma, pode-se incluir uma ramificação, especificando caminhos alternativos, tomados com base em alguma expressão booleana. Conforme mostra a [Figura 20.5](#), uma ramificação é representada como um diamante. A ramificação poderá ter uma transição de entrada e duas ou mais de saída. Em cada transição de entrada, é colocada uma expressão booleana, avaliada somente após a entrada na ramificação. Ao longo dessas transições de saída, as proteções não poderão se sobrepor (caso contrário, o fluxo de controle seria ambíguo), mas deverão cobrir todas as possibilidades (caso contrário, o fluxo de controle congelaria).

- ➡ As ramificações são uma conveniência notacional, equivalentes semanticamente a várias transições com proteção, conforme é examinado no [Capítulo 22](#).

Como uma conveniência, você pode usar a palavra reservada `else` para marcar uma transição de saída, representando o caminho a ser seguido, caso nenhuma outra expressão de guarda seja avaliada como verdadeira.



**Figura 20.5:**  
Ramificação

Quando dois caminhos de controle fundem-se novamente, você também pode usar um símbolo de diamante com duas setas de entrada e uma seta de saída. Não é necessário ter proteção na fusão.

O efeito de iteração pode ser obtido utilizando uma ação definindo o valor de um iterador, outra ação incrementando o iterador e uma ramificação avaliando se a iteração é concluída. A UML inclui tipos de nós para loops, mas normalmente eles podem ser expressos mais facilmente em texto do que em gráfico.

- *Ramificação e iteração são possíveis nos diagramas de interação, conforme discutido no [Capítulo 19](#).*

**Nota:** A UML não prescreve a linguagem para essas expressões. De modo abstrato, você poderá simplesmente usar texto estruturado; mas de modo concreto, poderá usar a sintaxe e a semântica de uma linguagem de programação específica.

## BIFURCAÇÃO E UNIÃO

Transições sequenciais com ramificações e simples são os caminhos mais comuns a serem encontrados em diagramas de atividades. Entretanto – principalmente quando estiver fazendo a modelagem de fluxos de trabalho de processos de negócios – você encontrará fluxos concorrentes. Na UML, a barra de sincronização é empregada para especificar a bifurcação e a união

desses fluxos paralelos de controle. A barra de sincronização é representada como uma linha fina, horizontal ou vertical.

- *Cada fluxo de controle concorrente se encontra no contexto de um objeto ativo independente, que é tipicamente modelado como um processo ou um thread, conforme é examinado no Capítulo 23; os nós são apresentados no Capítulo 27.*

Por exemplo, considere os fluxos concorrentes envolvidos no controle de um dispositivo audioanimatrônico, capaz de reproduzir a fala e gestos humanos. Conforme mostra a Figura 20.6, uma bifurcação representa a divisão de um mesmo fluxo de controle em dois ou mais fluxos de controle concorrentes. A bifurcação poderá ter uma única transição de entrada e duas ou mais transições de saída, cada uma das quais representa um fluxo de controle independente. Abaixo da bifurcação, as atividades associadas com cada um desses caminhos prosseguem em paralelo. Conceitualmente, as atividades de cada um desses fluxos são, de fato, concorrentes, embora, em um sistema em execução, esses fluxos poderão ser realmente concorrentes (no caso de um sistema distribuído em vários nós) ou sequenciais apesar de intercalado (no caso de um sistema funcionando em um único nó), dando assim somente a ilusão de uma concorrência verdadeira.

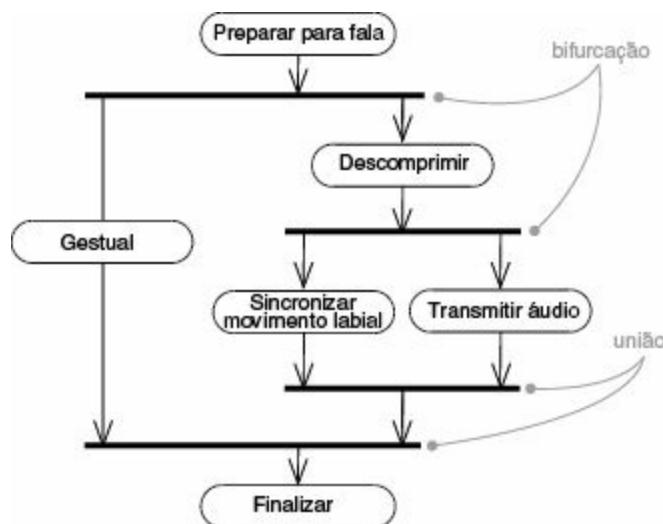


Figura 20.6:

## Bifurcação e união

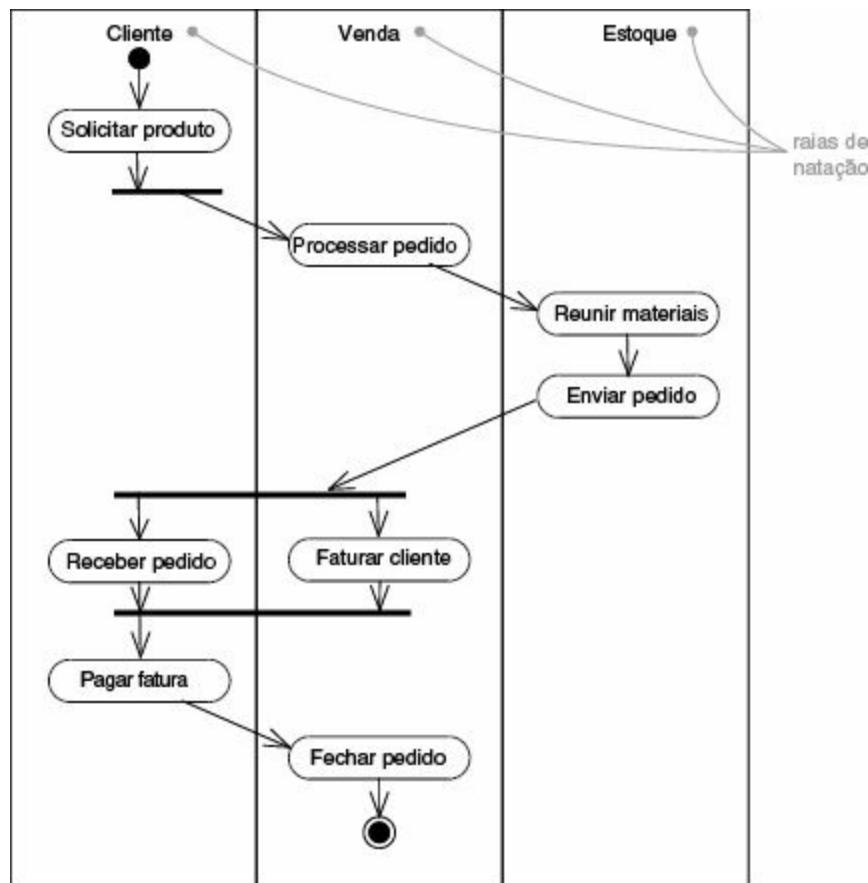
Conforme a figura também mostra, uma união representa a sincronização de dois ou mais fluxos de controle concorrentes. A união poderá ter duas ou mais transições de entrada e uma única transição de saída. Acima da união, as atividades associadas com cada um desses caminhos prosseguem paralelamente. Na união, os fluxos concorrentes são sincronizados, significando que cada um aguarda até que todos os fluxos de entrada tenham alcançado a união no ponto em que um dos fluxos de controle prossegue abaixo da união.

***Nota:** Deverá haver um equilíbrio entre uniões e bifurcações, significando que o número de fluxos que deixam uma bifurcação deve ser equivalente ao número de fluxos que entram na união correspondente. Também as atividades que se encontram em fluxos de controle paralelos poderão se comunicarumas com as outras, enviando sinais. Esse estilo de processo de comunicação sequencial é chamado uma corrotina. Na maioria das vezes, você faz a modelagem desse estilo de comunicação utilizando objetos ativos.*

- ➡ Os objetos ativos são examinados no [Capítulo 23](#); os sinais são apresentados no [Capítulo 21](#).

## RAIAS DE NATAÇÃO

Você considerará útil esse recurso, principalmente na modelagem de fluxos de trabalho de processos de negócio, para dividir em grupos os estados de atividades de um diagrama de atividades, cada grupo representando a organização de negócio responsável por essas atividades. Na UML, cada grupo é chamado uma raia de natação, pois, visualmente, os grupos ficam separados de seus vizinhos por uma linha cheia vertical, conforme mostra a [Figura 20.7](#). A raia de natação especifica um conjunto de atividades que compartilham algumas propriedades organizacionais.



**Figura 20.7:**

Raias de natação

Cada raia de natação tem um nome único em seu diagrama. A raia de natação realmente não tem uma semântica profunda, exceto por poder representar alguma entidade do mundo real. Cada raia de natação representa uma responsabilidade de alto nível relativa a uma parte da atividade geral de um diagrama de atividade e cada uma eventualmente poderá ser implementada por uma ou mais classes. Em um diagrama de atividades dividido em raias de natação, cada atividade pertence exatamente a uma única raia, mas as transições poderão cruzar as raias.

- ➡ Uma raia de natação é um tipo de pacote; os pacotes são examinados no [Capítulo 12](#); as classes são apresentadas nos [Capítulos 4 e 9](#); os processos e threads são explicados no [Capítulo 23](#).

**Nota:** Existe uma ligeira conexão entre as raias de natação e os fluxos de controle concorrentes. Conceitualmente, as atividades de cada raia de natação são geralmente – mas nem sempre – consideradas separadas das atividades das raias vizinhas. Isso faz sentido, pois, no mundo real, as organizações comerciais que costumam ser mapeadas nessas raias são independentes e concorrentes.

## FLUXO DE OBJETOS

Os objetos poderão estar envolvidos no fluxo de controle associado com um diagrama de atividade. Por exemplo, no fluxo de trabalho de processamento de um pedido, como na figura anterior, o vocabulário do espaço do problema também incluirá classes como Pedido e Cobrança. As instâncias dessas duas classes serão produzidas por determinadas atividades (Processar pedido criará um objeto Pedido, por exemplo); outras atividades poderão alterar esses objetos (por exemplo, Enviar pedido modificará o estado do objeto Pedido a ser preenchido).

- ➡ Os objetos são examinados no [Capítulo 13](#); a modelagem do vocabulário de um sistema é apresentada no [Capítulo 4](#).

Conforme mostra a [Figura 20.8](#), é possível especificar as coisas que estão envolvidas em um diagrama de atividades, incluindo esses objetos no diagrama, conectados por setas às ações que os produzem ou consomem.

Isso é chamado de fluxo de objetos, pois representa o fluxo de um valor de objeto de uma ação para outra. Um fluxo de objetos implica fluxo de controle (você não pode executar uma ação que requeira um valor sem o valor), portanto, é desnecessário desenhar um fluxo de controle entre ações conectadas por fluxos de objetos.

- ➡ Os relacionamentos de dependência são examinados nos [Capítulos 5 e 10](#).

Além de mostrar o fluxo de um objeto por meio de um diagrama de atividades, você também pode exibir como seu estado muda. Conforme

mostra a figura, o estado de um objeto é representado colocando o nome do estado entre colchetes abaix do nome do objeto.

- Os valores e o estado de um objeto são examinados no [Capítulo 13](#); os atributos são apresentados nos [Capítulos 4 e 9](#).

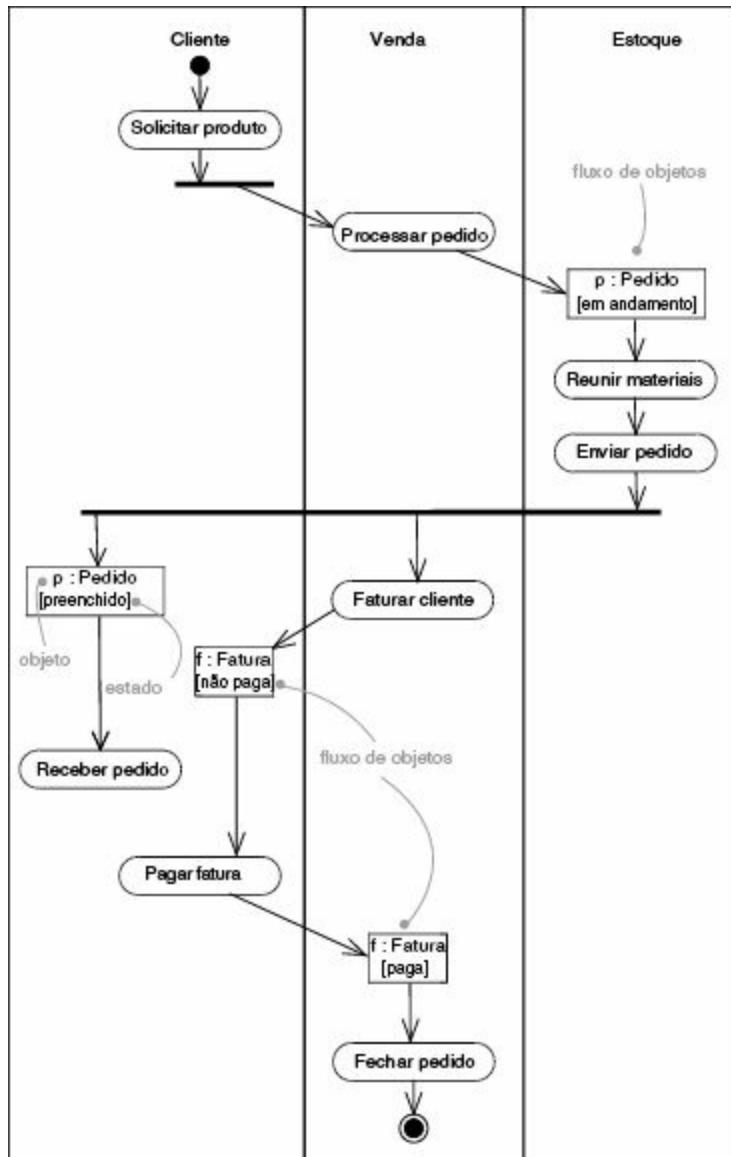


Figura 20.8:  
Fluxo de objetos

## REGIÕES DE EXPANSÃO

Frequentemente, a mesma operação deve ser realizada nos elementos de um conjunto. Por exemplo, se um pedido abrange um conjunto de itens, o manipulador do pedido deve realizar a mesma operação para cada item: verificar a disponibilidade, examinar o custo, verificar se o tipo de item é tributável, etc. As operações nas listas são normalmente modeladas como loops, mas o modelador deve iterar os itens, extraí-los um de cada vez, realizar a operação, montar os resultados em uma matriz de saída, incrementar o índice e verificar a conclusão. A mecânica de execução do loop obscurece o significado real da operação. Esse padrão extremamente comum pode ser modelado diretamente usando uma *região de expansão*.

Uma região de expansão representa um fragmento de modelo de atividade que é realizado nos elementos de uma lista ou conjunto. Ele é apresentado em um diagrama de atividades por meio de um desenho de linha tracejada em torno de uma região no diagrama. As entradas para a região e as saídas da região são coleções de valores, como os itens em um pedido. As entradas e saídas da coleção são apresentadas como uma linha de pequenos quadrados juntos (para sugerir uma matriz de valores). Quando um valor de matriz chega a uma entrada de coleção em uma região de expansão do resto do modelo de atividade, ele é dividido em valores individuais. A região de execução é executada uma vez para cada elemento na matriz. É desnecessário modelar a iteração; ela está implícita na região de expansão. As execuções diferentes podem ser realizadas concorrentemente, se possível. Quando cada execução da região de expansão é concluída, seu valor de saída (se houver) é colocado em uma matriz de saída na mesma ordem da entrada correspondente. Em outras palavras, uma região de expansão realiza uma operação geral nos elementos de uma matriz para criar uma nova matriz.

No caso mais simples, uma região de expansão tem uma entrada de matriz e uma saída de matriz, mas pode ter uma ou mais matrizes de entrada e zero ou mais matrizes de saída. Todas as matrizes devem ter o mesmo tamanho, mas elas não precisam conter o mesmo tipo de valor. Os valores das posições correspondentes são executados juntos para produzir valores de saída na mesma posição. A região pode ter nenhuma saída, se todas as operações

forem realizadas como efeitos colaterais diretamente nos elementos da matriz.

As regiões de expansão permitem que as operações sobre coleções e os elementos individuais das coleções sejam apresentados no mesmo diagrama, sem a necessidade de mostrar todos os detalhes, exceto a maquinaria de iteração simples.

A [Figura 20.9](#) mostra um exemplo de uma região de expansão. No corpo principal do diagrama, um pedido é recebido. Isso produz um valor do tipo Pedido, que consiste em uma matriz de valores Item. O valor de Pedido é a entrada para uma região de expansão. Cada execução da região de expansão funciona sobre um elemento da coleção Pedido. Portanto, dentro da região, o tipo de valor de entrada corresponde a um elemento da matriz Pedido, um Item. A atividade da região de expansão bifurca- se em duas ações: uma ação localiza o Produto e o adiciona à entrega, e a outra ação computa o custo de um item. Não é necessário que os Itens sejam colocados em ordem; as diferentes execuções da região de expansão podem proceder concorrentemente. Quando todas as execuções da região de expansão estão completas, os itens são formados em uma Entrega (uma coleção de Produtos) e as despesas são formadas em uma Fatura (uma coleção de valores de Dinheiro). O valor da Entrega é a entrada para a ação EntregarPedido, e o valor Fatura é a entrada para a ação EnviarFatura.

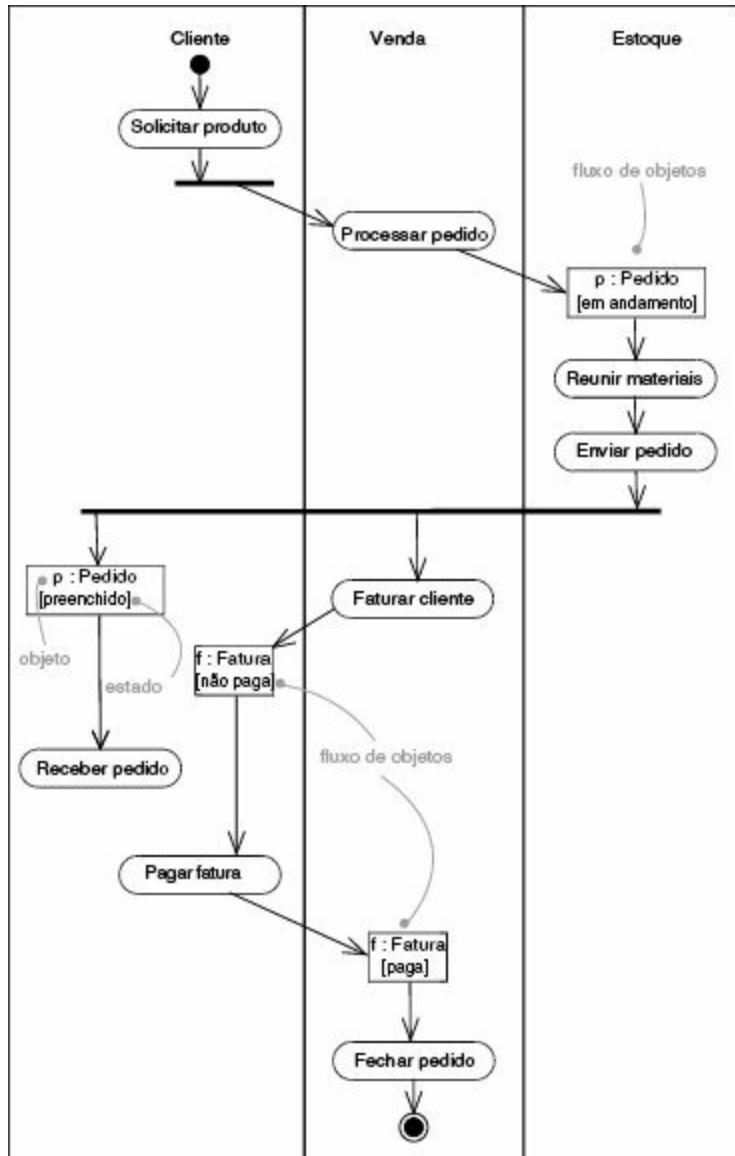


Figura 20.9:

Região de expansão

## USOS COMUNS

Os diagramas de atividades são utilizados para a modelagem dos aspectos dinâmicos de um sistema. Esses aspectos dinâmicos poderão envolver a atividade de qualquer tipo de abstração em qualquer visão da arquitetura do sistema, incluindo classes (o que inclui as classes ativas), interfaces, componentes e nós.

- ➡ As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#); as classes são apresentadas nos [Capítulos 4 e 9](#); as classes ativas são discutidas no [Capítulo 23](#); as interfaces são explicadas no [Capítulo 11](#); as operações são explicadas nos [Capítulos 4 e 9](#); os casos de uso e os atores são discutidos no [Capítulo 17](#); os componentes são discutidos no [Capítulo 15](#); os nós são examinados no [Capítulo 27](#); os sistemas e subsistemas são apresentados no [Capítulo 32](#).

Ao utilizar um diagrama de atividades para fazer a modelagem de algum aspecto dinâmico do sistema, você pode fazê-lo no contexto virtual de qualquer elemento da modelagem. Tipicamente, entretanto, os diagramas de atividades serão empregados no contexto do sistema como um todo, de um subsistema, de uma operação ou de uma classe. Os diagramas de atividades também podem ser anexados a casos de uso (para a modelagem de um cenário) e a colaborações (para a modelagem de aspectos dinâmicos de uma sociedade de objetos).

Ao fazer a modelagem dos aspectos dinâmicos de um sistema, tipicamente você utilizará os diagramas de atividades de duas maneiras.

## 1. Para fazer a modelagem de um fluxo de trabalho.

Aqui as atividades são focalizadas, conforme são visualizadas pelos atores que colaboram com o sistema. Os fluxos de trabalho muitas vezes se encontram nos limites de sistemas complexos de software e são utilizados para visualizar, especificar, construir e documentar processos de negócios que envolvem o sistema que está sendo desenvolvido. Nesse uso de diagramas de atividades, a modelagem do fluxo de objetos é particularmente importante.

## 2. Para fazer a modelagem de uma operação.

Aqui os diagramas de atividades são empregados como fluxogramas, para a modelagem de detalhes de uma computação. Nesse uso de diagramas de atividades, a modelagem de ramificações, bifurcações e estados de união é particularmente importante. O contexto de um diagrama de atividade

utilizado dessa maneira envolve os parâmetros da operação e seus objetos locais.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE FLUXO DE TRABALHO

Nenhum sistema complexo de software existe isoladamente; sempre existe contexto em que se encontra o sistema e esse contexto sempre abrange atores que interagem com o sistema. Principalmente no caso de missões críticas, software corporativo, você encontrará sistemas automatizados funcionando no contexto de processos de negócios de mais alto nível. Esses processos de negócios são tipos de fluxos de trabalho, por representarem o fluxo de trabalho e de objetos no próprio negócio. Por exemplo, em uma empresa de venda a varejo, haverá alguns sistemas automatizados (por exemplo, sistemas de pontos de venda que interagem com sistemas de marketing e de estoque), assim como sistemas humanos (as pessoas que trabalham em cada loja e em departamentos de televendas, marketing, compras e entrega). Você pode fazer a modelagem dos processos de negócios em relação às formas como esses vários sistemas automatizados e humanos colaboram com o emprego de diagramas de atividades.

- ➡ A modelagem do contexto de um sistema é examinada no [Capítulo 18](#).

Para fazer a modelagem de um fluxo de trabalho:

- » Estabeleça um foco para o fluxo de trabalho. Para sistemas não triviais, é impossível mostrar todos os fluxos de trabalho relevantes em um mesmo diagrama.
- » A modelagem de vocabulário de um sistema é discutida no [Capítulo 4](#); pré e pós-condições são examinadas no [Capítulo 9](#).

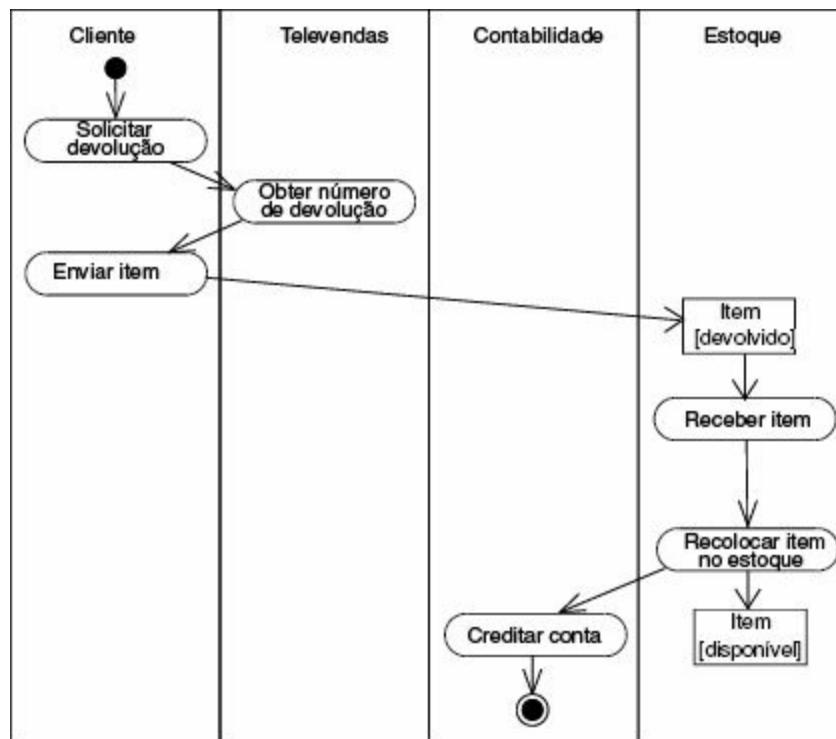
- › Selecione os objetos de negócio que têm responsabilidades de alto nível para partes do fluxo de trabalho geral. Esses objetos poderão ser coisas reais definidas a partir do vocabulário do sistema ou poderão ser mais abstratos. Em qualquer caso, crie uma raia de natação para cada objeto de negócio importante.
- › Identifique as precondições do estado inicial do fluxo de trabalho e as pós-condições do estado final do fluxo de trabalho. Isso é importante para ajudar a modelagem das fronteiras do fluxo de trabalho.
- › Começando no estado inicial do fluxo de trabalho, especifique as ações realizadas ao longo do tempo e represente-as no diagrama de atividades.
- › Para ações complexas ou para conjuntos de ações que aparecem várias vezes, resuma esses casos em chamadas para um diagrama de atividades separado.
- › Represente os fluxos que conectam essas ações e nós de atividade. Inicie com os fluxos sequenciais existentes no primeiro fluxo de trabalho; a seguir, considere as ramificações e somente então leve em consideração as bifurcações e uniões.
- › Caso existam objetos importantes envolvidos no fluxo de trabalho, represente-os também no diagrama de atividades. Mostre seu estado e valores modificados, conforme seja necessário para comunicar o propósito do fluxo de objetos.

Por exemplo, a [Figura 20.10](#) mostra um diagrama de atividades para uma empresa de varejo, que especifica o fluxo de trabalho envolvido quando um cliente devolve um item de um pedido postal. O trabalho começa com a ação Solicitar devolução do Cliente e depois flui por Televendas (Receber número de devolução), retorna ao Cliente (Enviar item) e, a seguir, ao Depósito (Receber item e depois Incluir item novamente no estoque) e, por fim, termina em Contabilidade (Creditar conta). Conforme o diagrama indica, um objeto significativo (i, uma instância de Item)

também acompanha o fluxo do processo, mudando do estado devolvido para o estado disponível.

**Nota:** Os fluxos de trabalho costumam ser processos de negócio, mas nem sempre. Por exemplo, você também pode usar diagramas de atividades para especificar processos de desenvolvimento de software, como o processo de gerenciamento de configuração. Além disso, os diagramas de atividades podem ser utilizados para fazer a modelagem de sistemas que não são de software, como o fluxo de pacientes em um sistema de assistência médica.

Nesse exemplo, não existem ramificações, bifurcações ou uniões. Você encontrará essas características em fluxos de trabalho mais complexos.



**Figura 20.10:**

A modelagem de um fluxo de trabalho

## A MODELAGEM DE UMA OPERAÇÃO

Os diagramas de atividades podem ser anexados a qualquer elemento da modelagem, com o propósito de visualizar, especificar, construir e documentar o comportamento desse elemento. Os diagramas de atividades

podem ser anexados a classes, interfaces, componentes, nós, casos de uso e colaborações. A operação é o elemento mais comum ao qual o diagrama de atividade será anexado.

- As classes e as operações são examinadas nos Capítulos 4 e 9; as interfaces são apresentadas no Capítulo 11; os componentes são examinados no Capítulo 15; os nós são apresentados no Capítulo 27; os casos de uso são explicados no Capítulo 17; as colaborações são discutidas no Capítulo 28; as precondições, as pós-condições e as invariantes são examinadas no Capítulo 9; as classes ativas são discutidas no Capítulo 23.

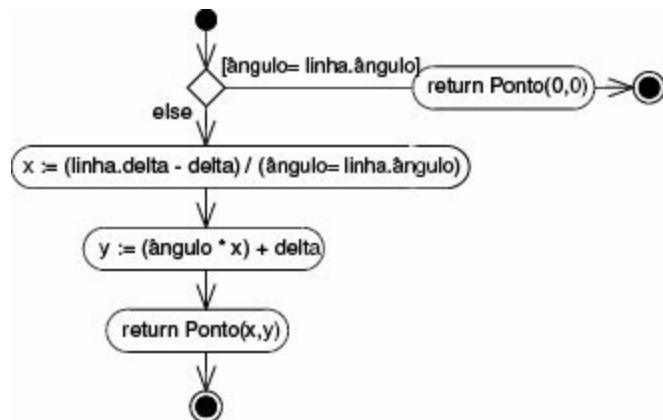
Utilizado dessa maneira, um diagrama de atividades é simplesmente um fluxograma das ações de uma operação. Uma vantagem básica do diagrama de atividades é que todos os elementos apresentados no diagrama estão semanticamente relacionados a um rico modelo subjacente. Por exemplo, qualquer outra operação ou sinal referenciados por um estado de ação podem ter seu tipo verificado em relação à classe do objeto de destino.

Para fazer a modelagem de uma operação:

- » Colecione as abstrações envolvidas na operação. Isso inclui os parâmetros da operação (incluindo seu tipo de retorno, se houver), os atributos da classe que a contém e certas classes vizinhas.
- » Identifique as precondições no estado inicial da operação e as pós-condições no estado final da operação. Identifique também quaisquer invariantes da classe que a contém e que precisem ser mantidas durante a execução da operação.
- » Começando no estado inicial da operação, especifique as atividades e ações realizadas ao longo do tempo e represente-as no diagrama de atividades como estados de atividades ou estados de ação.
- » Use ramificações, conforme seja necessário para especificar caminhos condicionais e iterações.

» Somente se a operação pertencer a uma classe ativa, use bifurcações e uniões, conforme seja necessário para especificar fluxos de controle paralelos.

Por exemplo, no contexto da classe Linha, a [Figura 20.11](#) mostra um diagrama de atividades que especifica o algoritmo da operação `interseção`, cuja assinatura inclui um único parâmetro (`linha`, da classe `Linha`) e um único valor de retorno (da classe `Ponto`). A classe `Linha` tem dois atributos de interesse: `ângulo` (que armazena a inclinação da linha) e `delta` (que armazena o deslocamento da linha em relação à sua origem).



**Figura 20.11:**

A modelagem de uma operação

O algoritmo dessa operação é simples, conforme é mostrado no seguinte diagrama de atividades. Em primeiro lugar, existe uma condição que testa se a inclinação da linha atual é a mesma que a inclinação do parâmetro `linha`. Se for a mesma, não haverá uma interseção das linhas e um ponto em  $(0, 0)$  será retornado. Caso contrário, a operação primeiro calcula um valor  $x$  para o ponto de interseção e depois um valor  $y$ ;  $x$  e  $y$  são objetos locais à operação. Por fim, um ponto em  $(x, y)$  é retornado.

► *Se uma operação envolve a interação de uma sociedade de objetos, você também pode fazer a realização dessa operação, utilizando as colaborações, conforme é descrito no [Capítulo 28](#).*

**Nota:** A utilização de diagramas de atividades para a elaboração do fluxograma de uma operação se encontra no limite de tornar a UML uma linguagem de programação visual. Você pode fazer o fluxograma de qualquer operação, mas, pragmaticamente, não desejará fazê-lo. Escrever o corpo de uma operação em uma linguagem de programação específica costuma ser mais direto. Você desejará empregar os diagramas de atividades para a modelagem de uma operação quando o comportamento dessa operação for complexo e, portanto, difícil de ser compreendido apenas pela observação do código. O exame de um fluxograma revelará algo sobre o algoritmo que poderia não ser notado pela simples observação do código.

## ENGENHARIA DIRETA E REVERSA

A *engenharia direta* (a criação de código a partir de um modelo) é possível para diagramas de atividades, especialmente se o contexto do diagrama é uma operação. Por exemplo, tomando como base o diagrama de atividades mostrado na [Figura 20.11](#), uma ferramenta de engenharia direta poderia gerar o seguinte código em C++ para a operação intersection (interseção).

```
Point Line::intersection (1 : Line) {
    if (slope == l.slope) return Point(0,0);
    int x = (l.delta - delta) / (slope - l.slope);
    int y = (slope * x) + delta;
    return Point(x, y);
}
```

Existe aqui uma solução inteligente, envolvendo a declaração de duas variáveis locais. Uma ferramenta menos sofisticada poderia ter declarado primeiro as duas variáveis e depois definido seus valores.

A *engenharia reversa* (a criação de um modelo a partir de um código) também é possível para os diagramas de atividades, especialmente se o contexto do código é o corpo de uma operação. Em particular, o diagrama anterior poderia ter sido gerado a partir da implementação da classe Line.

Mais interessante do que fazer a engenharia reversa de um modelo a partir do código é a animação de um modelo em relação à execução de um sistema entregue. Por exemplo, considerando o diagrama anterior, uma ferramenta poderia animar os estados de ação mostrados no diagrama como se estivessem em andamento em um sistema em execução. Ainda melhor, com essa ferramenta também sob o controle de um depurador, você poderia

controlar a velocidade da execução, possivelmente definindo pontos de parada para interromper a ação em pontos de interesse no tempo e verificar os valores de atributos de objetos individuais.

## DICAS E SUGESTÕES

Ao criar diagramas de atividades na UML, lembre-se de que esses diagramas são apenas projeções no mesmo modelo de aspectos dinâmicos de um sistema. Nenhum único diagrama de atividades será capaz de captar tudo sobre os aspectos dinâmicos do sistema. Em vez disso, você desejará utilizar muitos diagramas de atividades para fazer a modelagem da dinâmica de um fluxo de trabalho ou de uma operação.

Um diagrama de atividade bem estruturado:

- › Está voltado para comunicar um aspecto da dinâmica do sistema.
- › Contém somente os elementos essenciais para a compreensão desse aspecto.
- › Oferece detalhes consistentes com seu nível de abstração; você expõe somente os adornos essenciais à compreensão.
- › Não é tão minimalista que informe mal o leitor sobre a semântica importante.

Ao definir um diagrama de atividade:

- › Dê-lhe um nome capaz de comunicar seu propósito.
- › Inicie com a modelagem do fluxo primário. Inclua ramificações, concorrências e fluxos de objetos como considerações secundárias, possivelmente em diagramas separados.
- › Distribua seus elementos de forma a minimizar o cruzamento de linhas.
- › Use notas e cores como indicações visuais com a finalidade de chamar a atenção para as características importantes de seu diagrama.

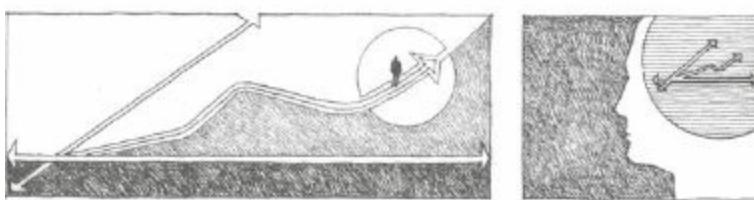
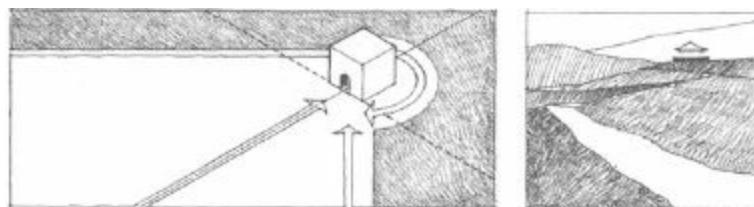
P A R T E

---

5

---

# **MODELAGEM COMPORTAMENTAL AVANÇADA**



CAPÍTULO

---

21

---

# Eventos e Sinais

## Neste capítulo

- › *Eventos de sinal, eventos de chamada, eventos de tempo, eventos de alteração*
- › *Modelagem de uma família de sinais*
- › *Modelagem de exceções*
- › *Manipulação de eventos em objetos ativos e passivos*

**N**o mundo real, as coisas acontecem. Não somente as coisas acontecem, mas muitas coisas podem acontecer ao mesmo tempo e nos momentos mais inesperados. “As coisas que acontecem” são chamadas eventos e cada uma representa a especificação de uma ocorrência significativa que tem uma localização no tempo e no espaço.

No contexto das máquinas de estados, os eventos são utilizados para a modelagem da ocorrência de um estímulo capaz de ativar a transição de um estado. Os eventos podem incluir sinais, chamadas, a contagem de tempo ou uma alteração do estado.

Os eventos podem ser síncronos ou assíncronos; por isso a modelagem de eventos está relacionada à modelagem de processos e threads.

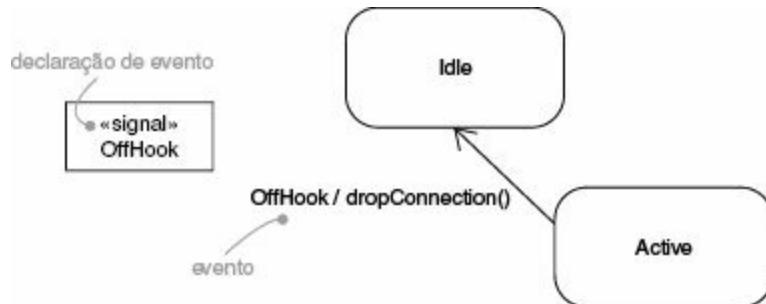
## PRIMEIROS PASSOS

Um sistema perfeitamente estático é extremamente desinteressante, pois nada acontece. Todos os sistemas reais têm alguma dimensão dinâmica e essas dinâmicas são ativadas pelas coisas que acontecem externa ou internamente. Em uma máquina de caixa eletrônico, a ação é iniciada quando

o usuário pressiona um botão para começar a transação. Em um robô autônomo, a ação é iniciada pelo robô batendo em um objeto. Em um roteador de rede, a ação é iniciada pela detecção de uma sobrecarga dos buffers de mensagens. Em uma planta química, a ação é iniciada pela contagem de tempo suficiente para uma reação química.

Na UML, cada coisa que acontece é modelada como um evento. Um evento é a especificação de uma ocorrência significativa que tem uma localização no tempo e no espaço. Um sinal, a contagem de tempo e a alteração de estado são eventos assíncronos, representando eventos que podem ocorrer em momentos arbitrários. As chamadas geralmente são eventos síncronos, representando a chamada a uma operação.

A UML fornece uma representação gráfica para os eventos, conforme mostra a [Figura 21.1](#). Essa notação permite que você visualize a declaração de eventos (como a indicação de telefone fora do gancho, o sinal OffHook), assim como a utilização de eventos para ativar a transição de um estado. O sinal OffHook causa a transição do estado Active (ativo) para o estado Idle (ocioso) da mesma forma que a execução da ação dropConnection (derruba a conexão).



**Figura 21.1:**  
Eventos

## TERMOS E CONCEITOS

Um *evento* é a especificação de uma ocorrência significativa que tem uma localização no tempo e no espaço. No contexto da máquina de estados, o evento é uma ocorrência de um estímulo capaz de ativar a transição de um

estado. Um *sinal* é um tipo de evento que representa a especificação de um estímulo assíncrono comunicado entre instâncias.

## TIPOS DE EVENTOS

Os eventos podem ser externos ou internos. Os eventos externos são aqueles passados entre o sistema e seus atores. Por exemplo, o pressionar de um botão e a interrupção de um sensor de colisão são exemplos de eventos externos. Os eventos internos são aqueles passados entre os objetos que vivem no interior do sistema. Uma exceção de sobrecarga é um exemplo de evento interno.

- ➡ Os atores são examinados no [Capítulo 17](#); os sistemas são apresentados no [Capítulo 32](#).

Na UML, você pode fazer a modelagem de quatro tipos de eventos: sinais, chamadas, contagem de tempo e alteração no estado.

- ➡ A criação e destruição de objetos também são tipos de sinais, conforme é apresentado no [Capítulo 16](#).

## SINAIS

Uma mensagem é um objeto nomeado que é enviado assincronamente por um objeto e recebido por outro. Um sinal é um classificador de mensagens; é um tipo de mensagem.

Os sinais têm muito em comum com as classes puras. Por exemplo, os sinais podem ter instâncias, embora, em geral, não seja necessário fazer a modelagem deles explicitamente. Os sinais também podem estar envolvidos em relacionamentos de generalização, permitindo que você faça a modelagem de hierarquias de eventos, alguns dos quais são gerais (por exemplo, o sinal NetworkFailure, que indica uma falha na rede) e alguns que são específicos (por exemplo, uma especialização de NetworkFailure chamada WarehouseServerFailure indica especificamente a falha de um servidor do estoque). Assim como as

classes, os sinais poderão ter atributos e operações. Antes de ser enviado por um objeto ou após ser recebido por outro, um sinal é apenas um objeto de dados simples.

- As classes são examinadas nos Capítulos 4 e 9; a generalização é apresentada nos Capítulos 5 e 10.

**Nota:** Os atributos de um sinal servem como seus parâmetros. Por exemplo, para indicar uma colisão enviando o sinal Collision, você pode também especificar, através de parâmetros, valores para seus atributos, como em Collision(5.3).

Um sinal poderá ser transmitido como a ação de transição de um estado em uma máquina de estados. Ele pode ser modelado como uma mensagem entre dois papéis em uma interação. A execução de um método também pode enviar sinais. De fato, quando você faz a modelagem de uma classe ou de uma interface, uma parte importante da especificação do comportamento desse elemento consiste em especificar os sinais que suas operações podem enviar.

- As máquinas de estados são examinadas no Capítulo 22; as interações são apresentadas no Capítulo 16; as interfaces são explicadas no Capítulo 11; as dependências são discutidas no Capítulo 5; os estereótipos são examinados no Capítulo 6.

Na UML, conforme mostra a Figura 21.2, você faz a modelagem de sinais como classes estereotipadas. Você pode utilizar uma dependência estereotipada como send, para indicar que uma operação envia um determinado sinal.

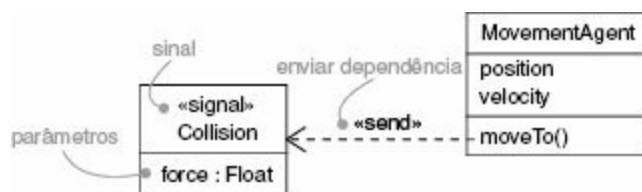


Figura 21.2:

Sinais

## EVENTOS DE CHAMADA

Da mesma forma que o evento de sinal representa a ocorrência de um sinal, um evento de chamada representa o recebimento por um objeto de uma solicitação de chamada para uma operação no objeto. Um evento de chamada pode ativar uma transição de estado em uma máquina de estados ou pode chamar um método no objeto de destino. A escolha é especificada na definição de classe para a operação.

► As máquinas de estados são apresentadas no Capítulo 22.

Enquanto um sinal é um evento assíncrono, um evento de chamada é, em geral, síncrono. Isso significa que, quando um objeto chama uma operação de outro objeto que tem uma máquina de estados, o controle passa do emissor para o receptor, a transição é ativada pelo evento, a operação é completada, o receptor passa a um novo estado e o controle retorna ao emissor. Nesses casos em que quem faz a chamada não precisa aguardar uma resposta, uma chamada pode ser especificada como assíncrona.

Conforme mostra a Figura 21.3, a modelagem do evento de chamada não se diferencia da modelagem do evento de sinal. Em ambos os casos, você mostra o evento, juntamente com seus parâmetros, como a ativação da transição de um estado.



Figura 21.3:

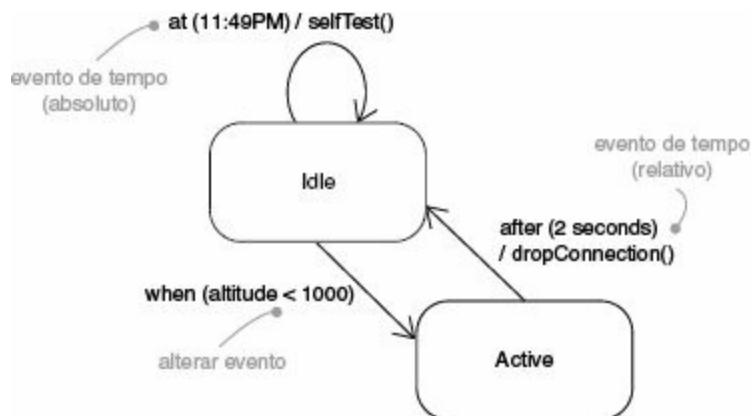
Os eventos de chamada

**Nota:** Embora não existam indicações visuais para distinguir um evento de sinal de um evento de chamada, a diferença é clara na base de seu modelo. O receptor de um evento saberá a diferença, é claro (pela declaração da operação em sua lista de operações). Tipicamente, o sinal será manipulado pela respectiva máquina de estados e o evento de chamada será manipulado por um método. Você pode utilizar suas ferramentas para navegar do evento até o sinal ou a operação.

## EVENTOS DE TEMPO E ALTERAÇÃO

Um evento de tempo é um evento que representa a contagem de tempo. Conforme mostra a [Figura 21.4](#), na UML você faz a modelagem do evento de tempo utilizando a palavra reservada `after` seguida por alguma expressão calculada que estima um período de tempo. Essas expressões podem ser simples (para representar um evento que acontece daqui a dois segundos, use `after 2 seconds`) ou complexas (para dizer que algo deve acontecer depois de um milissegundo após deixar a condição de ociosidade, use `after 1 ms since exiting Idle`). A menos que você o especifique explicitamente, o tempo inicial dessa expressão é o tempo considerado a partir da entrada no estado atual. Para indicar um evento de tempo que ocorre em um tempo absoluto, use a palavra-chave `at`. Por exemplo, o evento de tempo `at (1 jan 2005, 1200 UT)` especifica um evento que ocorre ao meio-dia (UT significa Time, horário universal) no Dia de Ano-Novo de 2005.

Um evento de alteração é um evento que representa uma mudança no estado ou a satisfação de alguma condição. Conforme mostra a [Figura 21.4](#), na UML você faz a modelagem do evento de alteração utilizando a palavra reservada `when` seguida por alguma expressão booleana. Essas expressões poderão ser empregadas para o teste contínuo de uma expressão (por exemplo, para representar um evento que ocorre quando a altitude for menor que 1.000, use `when altitude < 1000`).



**Figura 21.4:**

Eventos de tempo e alteração

Um evento de alteração ocorre quando o valor da condição muda de falso para verdadeiro. Isso não acontece quando o valor da condição muda de verdadeiro para falso. O evento não se repete enquanto o valor permanecer verdadeiro.

**Nota:** Embora o evento de alteração faça a modelagem de uma condição testada de forma contínua, tipicamente você pode analisar a situação para verificar quando deverá testar a condição em pontos discretos no tempo.

## EVENTOS ENVIAR E RECEBER

Os eventos de sinal e de chamada contêm pelo menos dois objetos: o objeto que envia o sinal ou chama a operação e o objeto para onde o evento é direcionado. Como os sinais são assíncronos e as próprias chamadas assíncronas são sinais, a semântica dos eventos interage com a semântica dos objetos ativos e passivos.

► Os processos e os threads são apresentados no [Capítulo 23](#).

Qualquer instância de qualquer classe pode enviar um sinal ou chamar uma operação do objeto receptor. Quando um objeto envia um sinal, o emissor transmite esse sinal e então prossegue em seu fluxo de controle, sem esperar qualquer retorno do destinatário. Por exemplo, se um ator interagindo com um sistema de caixa eletrônico enviar o sinal `pushButton`, o ator poderá prosseguir seu caminho independentemente do sistema para onde o sinal foi enviado. Ao contrário, quando um objeto chama uma operação, o emissor transmite a operação e então espera pelo receptor. Por exemplo, em um sistema comercial, uma instância da classe `Trader` poderá chamar a operação `confirmTransaction` em alguma instância da classe `Trade`, afetando assim o estado do objeto `Trade`. Se essa for uma chamada síncrona, o objeto `Trader` esperará até que a operação seja finalizada.

► As instâncias são apresentadas no [Capítulo 13](#).

**Nota:** Em algumas situações, você poderá querer exibir um objeto enviando um sinal para um conjunto de objetos (difusão seletiva) ou para qualquer objeto existente no sistema que seja capaz de ouvir (difusão geral). Para fazer a modelagem da difusão seletiva, você deverá exibir um objeto enviando um sinal para a coleção que contém um conjunto de receptores. Para fazer a modelagem da difusão geral, você deverá exibir um objeto enviando um sinal para outro objeto que representa o sistema como um todo.

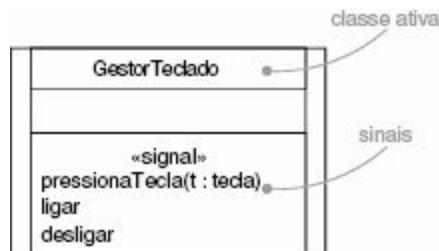
Qualquer instância de qualquer classe pode receber um evento de chamada ou um sinal. Caso seja um evento de chamada síncrono, então o emissor e o receptor estarão em um *rendez-vous* (revezamento) durante toda a duração da operação. Isso significa que o fluxo de controle do emissor é colocado em uma etapa de espera juntamente com o fluxo de controle do receptor até que a atividade da operação seja executada. Se isso for um sinal, então o emissor e o receptor não permanecem em *rendez-vous*: o emissor transmite o sinal, mas não aguarda uma resposta do receptor. Em outro caso, esse evento poderá ser perdido (se nenhuma resposta ao evento for especificada), poderá ativar a máquina do receptor (se houver) ou poderá simplesmente invocar uma chamada normal a um método.

- As máquinas de estados são apresentadas no [Capítulo 22](#); os objetos ativos são examinados no [Capítulo 23](#).

**Nota:** Uma chamada pode ser assíncrona. Nesse caso, quem faz a chamada continua imediatamente após emitir-a. A transmissão da mensagem para o receptor e a sua execução pelo receptor ocorrem simultaneamente com a execução subsequente de quem chama. Quando a execução do método está completa, ela é encerrada. Se o método tentar retornar valores, eles serão ignorados.

Na UML, é possível fazer a modelagem de eventos de chamada que um objeto poderá receber como operações na classe do objeto. Na UML, também é possível fazer a modelagem de sinais nomeados que um objeto poderá receber, nomeando-os em um compartimento extra da classe, conforme mostra a [Figura 21.5](#).

- As operações são examinadas no [Capítulo 4](#); os compartimentos de classe extras são apresentados no [Capítulo 4](#).



**Figura 21.5:**

Sinais e classes ativas

**Nota:** Os sinais nomeados também podem ser anexados a uma interface da mesma maneira. Em qualquer um dos casos, os sinais relacionados nesse compartimento extra não são as declarações de um sinal, mas somente a utilização do sinal.

- As interfaces são examinadas no [Capítulo 11](#); as operações assíncronas são apresentadas no [Capítulo 23](#).

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE UMA FAMÍLIA DE SINAIS

Na maioria dos sistemas orientados a eventos, os eventos de sinal são hierarquizados. Por exemplo, um robô autônomo poderá diferenciar sinais externos, como uma Collision (colisão), e outros internos, como uma HardwareFault (falha de hardware). Entretanto, os sinais externos e internos não precisam ser separados. Apesar dessas duas classificações tão amplas, você poderá encontrar especializações. Por exemplo, os sinais de HardwareFault poderão ser ainda mais especializados, como BatteryFault (falha de bateria) e MovementFault (falha de movimentação). Até esses poderão admitir maior especialização, como MotorStall (travamento de motor), um tipo de MovementFault.

- A generalização é examinada nos Capítulos [5](#) e [10](#).

Modelando as hierarquias de sinais dessa maneira, você pode especificar eventos polimórficos. Por exemplo, considere uma máquina de estados cuja transição é ativada somente pelo receptor de um MotorStall. Como um sinal do tipo folha nessa hierarquia, a transição somente poderá ser ativada por esse sinal e, portanto, ela não será polimórfica. Por outro lado, suponha ter feito a modelagem da máquina de estados com uma transição ativada pelo receptor de um HardwareFault. Nesse caso, a transição é polimórfica e pode ser ativada por um HardwareFault ou qualquer uma de suas especializações, incluindo BatteryFault, MovementFault e MotorStall.

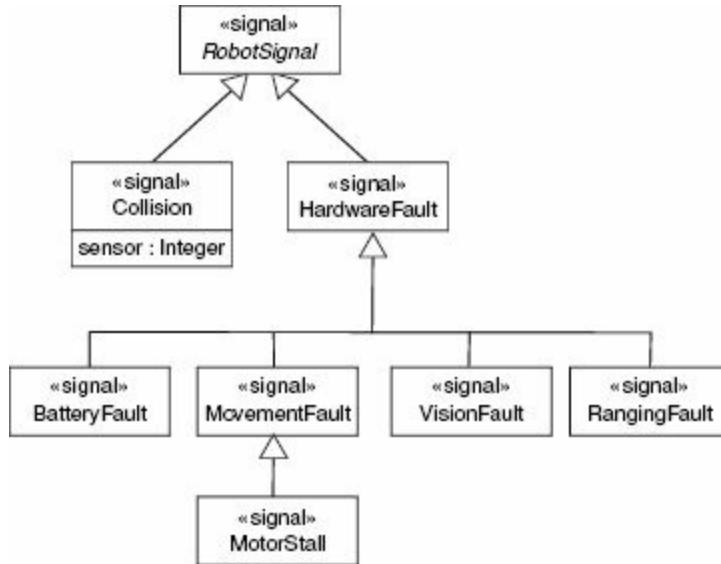
► As máquinas de estados são apresentadas no [Capítulo 22](#).

Para fazer a modelagem de uma família de sinais:

- » Considere todos os tipos diferentes de sinais ao quais um determinado conjunto de objetos ativos poderá responder.
- » Procure os tipos comuns de sinais e coloque-os em uma hierarquia de generalização/especialização, utilizando herança. Eleve os mais gerais e rebaixe os mais especializados.
- » Procure oportunidades para o polimorfismo na máquina de estados desses objetos ativos. Onde encontrar o polimorfismo, ajuste a hierarquia conforme seja necessário, introduzindo sinais abstratos intermediários.

A [Figura 21.6](#) mostra a modelagem de uma família de sinais que poderão ser manipulados por um robô autônomo. Observe que o sinal raiz do robô (RobotSignal1) é abstrato, significando que não poderá haver instâncias diretas. Esse sinal apresenta duas especializações concretas imediatas (Collision e HardwareFault), uma das quais (HardwareFault) é mais especializada. Observe que o sinal Collision tem um único parâmetro.

► As classes abstratas são examinadas nos [Capítulos 5 e 9](#).



**Figura 21.6:**

Modelagem de famílias de sinais

## A MODELAGEM DE EXCEÇÕES

Uma parte importante de visualizar, especificar e documentar o comportamento de uma classe ou interface é a especificação das exceções que suas operações podem determinar. Se você receber uma classe ou uma interface, as operações a serem chamadas serão claras, mas o mesmo não acontecerá com as exceções para cada operação, a menos que sua modelagem seja feita explicitamente.

- As classes são apresentadas nos Capítulos 4 e 9; as interfaces são examinadas no Capítulo 11 os estereótipos aparecem no Capítulo 6.

Na UML, as exceções são tipos de sinais que podem ser modelados como classes estereotipadas. As exceções poderão ser anexadas a operações de especificação. A modelagem de exceções é, de alguma forma, o inverso da modelagem de uma família geral de sinais. Você faz a modelagem de uma família de sinais principalmente para especificar os tipos de sinais que um objeto ativo é capaz de receber; a modelagem de exceções é feita principalmente para especificar os tipos de exceções que um objeto poderá causar em suas operações.

Para fazer a modelagem de exceções:

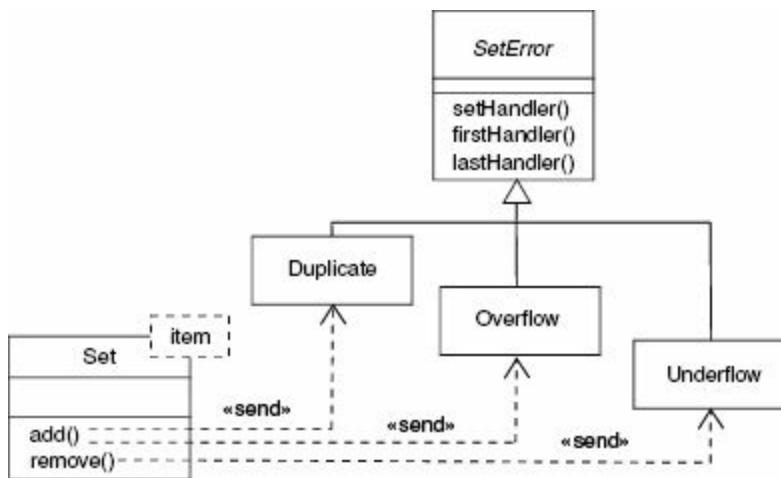
- › Para cada classe e interface e para cada operação desses elementos, considere as condições de exceção que poderão ocorrer. Pense no que pode dar errado e faça a modelagem dessas situações como sinais entre objetos.
- › Organize os sinais em uma hierarquia. Eleve as mais gerais, rebaixe as especializadas e introduza exceções intermediárias, conforme necessário.
- › Para cada operação, especifique as exceções que podem ocorrer. Isso pode ser feito explicitamente (mostrando dependências send de uma operação em relação a seus sinais) ou você pode usar diagramas de sequências que ilustram vários cenários.

A [Figura 21.7](#) apresenta a modelagem de uma hierarquia de exceções que poderá ser causada por uma biblioteca padrão de classes-repositório, como a classe template `Set`, que definirá os tipos de exceções. Essa hierarquia é encabeçada pelo sinal abstrato `Exception` e inclui três exceções especializadas: `Duplicate`, `Overflow` e `Underflow` (respectivamente duplicidade, sobrecarga e subcarga). Conforme é mostrado, a operação de soma `add` gera as exceções `Duplicate` e `Overflow`, enquanto a operação `remove` causa somente a exceção `Underflow`. Alternativamente, essas dependências poderiam ser colocadas em segundo plano, sendo nomeadas na especificação de cada operação. Em qualquer um desses modos, conhecendo as exceções que cada operação poderá enviar, você é capaz de criar clientes que utilizem a classe `Set` corretamente.

► As classes template são examinadas no [Capítulo 9](#).

**Nota:** Os sinais, incluindo os de ocorrência de exceções, são eventos assíncronos entre objetos. A UML também inclui exceções, como aquelas encontradas nas linguagens de programação Ada ou C++. As exceções são condições que fazem o caminho de execução principal ser abandonado e um caminho de execução secundário ser executado em seu lugar. As exceções não são sinais; em vez disso,

*elas são um mecanismo conveniente para especificar um fluxo de controle alternado em um thread de execução síncrono único.*



**Figura 21.7:**  
Modelagem de condições de erro

## DICAS E SUGESTÕES

Ao fazer a modelagem de um evento:

- › Construa hierarquias de sinais, de maneira a explorar as propriedades comuns de sinais relacionados.
- › Certifique-se de dispor de uma máquina de estados ajustável por trás de cada elemento que poderá receber o evento.
- › Certifique-se de fazer a modelagem não somente dos elementos que poderão receber eventos, mas também dos elementos que poderão enviá-los.

Ao definir um evento na UML:

- › Em geral, faça a modelagem de hierarquias de eventos explicitamente, mas faça o modelo de seu uso na base de cada classe ou operação que envia ou recebe esse evento.

CAPÍTULO

---

22

---

# Máquinas de Estados

## Neste capítulo

- » *Estados, transições e atividades*
- » *Modelagem do tempo de vida de um objeto*
- » *Criação de algoritmos bem estruturados*

Usando uma interação, você pode fazer a modelagem do comportamento de uma sociedade de objetos que trabalham em conjunto. Usando a máquina de estados, pode fazer a modelagem do comportamento de um objeto individual. Uma máquina de estados é um comportamento que especifica as sequências de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos.

- ➡ As interações são examinadas no [Capítulo 16](#); os objetos são apresentados no [Capítulo 13](#).

As máquinas de estados são empregadas para a modelagem dos aspectos dinâmicos de um sistema. Na maior parte, isso envolve a especificação do tempo de vida das instâncias de uma classe, um caso de uso ou um sistema inteiro. Essas instâncias poderão responder a eventos como sinais, operações ou a passagem de tempo. Quando um evento ocorre, alguma atividade acontecerá, dependendo do estado atual do objeto. Uma atividade é uma execução não atômica em andamento em uma máquina de estados. As atividades acabam resultando em alguma ação, formada por computações atômicas executáveis que resultam em uma alteração do estado do modelo ou

no retorno de um valor. O estado de um objeto é a condição ou situação durante a vida de um objeto durante a qual ele satisfaz alguma condição, realiza alguma atividade ou aguarda algum evento.

- As classes são examinadas nos Capítulos 4 e 9; os casos de uso são apresentados no Capítulo 17; os sistemas são explicados no Capítulo 32; os diagramas de atividades aparecem no Capítulo 20; e os diagramas de estados são examinados no Capítulo 25.

A dinâmica de execução pode ser visualizada de duas maneiras: dando ênfase ao fluxo de controle de uma atividade para outra (utilizando diagramas de atividades) ou dando ênfase aos estados potenciais dos objetos e às transições entre esses estados (utilizando diagramas de gráficos de estados).

Máquinas de estados bem estruturadas são como algoritmos bem estruturados: são eficientes, simples, adaptáveis e compreensíveis.

## PRIMEIROS PASSOS

Considere a vida do termostato de sua casa em um agradável dia de outono. Nas primeiras horas da manhã, as coisas estão bastante calmas para o termostato. A temperatura da casa está estável e, exceto por um vento ocasional ou uma chuva rápida, a temperatura fora da casa também está estável. Chegando o amanhecer, entretanto, as coisas ficam mais interessantes. O sol começa a surgir no horizonte, aumentando um pouco a temperatura ambiente. Os membros da família começam a acordar; alguém poderá sair da cama e mexer no controle do termostato. Esses dois eventos serão significativos para o sistema de aquecimento e refrigeração da casa. O termostato começará a se comportar como qualquer outro termostato, comandando o aquecimento da casa (para aumentar a temperatura interna) ou o ar-condicionado (para abaixar a temperatura interna).

Depois de todos terem saído para a escola ou o trabalho, as coisas se acalmam e a temperatura da casa se estabiliza mais uma vez. Entretanto, um

programa automático poderá então intervir, comandando o termostato para abaixar a temperatura a fim de economizar energia elétrica e gás. O termostato volta a trabalhar. Mais tarde no mesmo dia, o programa entra em ação novamente, agora comandando o termostato para aumentar a temperatura, com a finalidade de a família chegar e encontrar uma casa aconchegante.

Ao anoitecer, com a casa cheia de corpos quentes, além do calor da cozinha, o termostato precisa trabalhar bastante para manter a temperatura, controlando calor e frio de maneira eficiente. Por fim, de noite, as coisas retornam a um estado de calma.

Vários sistemas complexos de software se comportam como esse termostato. Um marca-passo é executado continuamente, mas se adapta às alterações que ocorrem na atividade ou na pressão sanguínea. O roteador de uma rede também é executado de forma contínua, guiando silenciosamente fluxos assíncronos de bits, de vez em quando precisando adaptar seu comportamento em resposta a comandos provenientes do administrador da rede. Já um telefone celular funciona sob demanda, respondendo às entradas do usuário e às mensagens de celulares locais.

Na UML, pode-se fazer a modelagem dos aspectos estáticos de um sistema, utilizando elementos como diagramas de classes e de objetos. Esses diagramas permitem visualizar, especificar, construir e documentar os itens existentes no sistema, incluindo classes, interfaces, componentes, nós e casos de uso e respectivas instâncias, juntamente com a forma como esses itens estão relacionados uns com os outros.

- *A modelagem de aspectos estruturais de um sistema é examinada nas Partes 2 e 3.*

Na UML, pode-se fazer a modelagem dos aspectos dinâmicos de um sistema, utilizando as máquinas de estados. Enquanto uma interação modela uma sociedade de objetos que trabalham em conjunto para a execução de alguma ação, a máquina de estados modela o tempo de vida de um único

objeto, seja ele a instância de uma classe, um caso de uso ou até o sistema inteiro. Durante sua vida, o objeto poderá estar exposto a vários eventos, como um sinal, a chamada de uma operação, a criação ou destruição do objeto, a passagem do tempo ou a alteração de alguma condição. Em resposta a esses eventos, o objeto reage com alguma ação, que representa uma computação atômica e resulta na alteração do estado do objeto ou no retorno de um valor. O comportamento desse objeto é, portanto, afetado pelo passado. Um objeto poderá receber um evento, responder com uma ação e então modificar seu estado. Um objeto poderá receber outro evento e sua resposta ser diferente, dependendo de seu estado atual em resposta ao evento anterior.

- A modelagem dos aspectos dinâmicos de um sistema também pode ser feita com a utilização de interações, conforme é examinado no [Capítulo 16](#); os eventos são apresentados no [Capítulo 21](#).

As máquinas de estados são empregadas para a modelagem do comportamento de qualquer elemento do modelo, apesar de que, mais frequentemente, esse elemento será uma classe, um caso de uso ou um sistema inteiro. As máquinas de estados podem ser visualizadas utilizando diagramas de gráficos de estados. Você pode focalizar o comportamento ordenado por eventos de um objeto, o que é de grande ajuda para a modelagem de sistemas reativos.

- Os diagramas de atividades são examinados no [Capítulo 20](#); os diagramas de estados são apresentados no [Capítulo 25](#).

A UML fornece uma representação gráfica de estados, transições, eventos e ações, conforme mostra a [Figura 22.1](#). Essa notação permite visualizar o comportamento de um objeto de modo a enfatizar os elementos importantes da vida desse objeto.

# TERMOS E CONCEITOS

Uma *máquina de estados* é um comportamento que especifica as sequências de estados pelas quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos. Um *estado* é uma condição ou situação na vida de um objeto durante a qual ele satisfaz alguma condição, realiza alguma atividade ou aguarda um evento. Um *evento* é a especificação de uma ocorrência significativa que tem uma localização no tempo e no espaço. No contexto de uma máquina de estados, um evento é a ocorrência de um estímulo capaz de ativar uma transição de estado. Uma *transição* é um relacionamento entre dois estados, indicando que um objeto no primeiro estado realizará certas ações e entrará no segundo estado quando um evento especificado ocorrer e condições especificadas estão satisfeitas. Uma *atividade* é uma execução não atômica em andamento em uma máquina de estados. Uma *ação* é uma computação atômica executável que resulta na alteração do estado do modelo ou no retorno de um valor. Graficamente, um estado é representado como um retângulo com cantos arredondados. A transição é representada com uma linha cheia direcionada ou caminho do estado original para o novo estado.

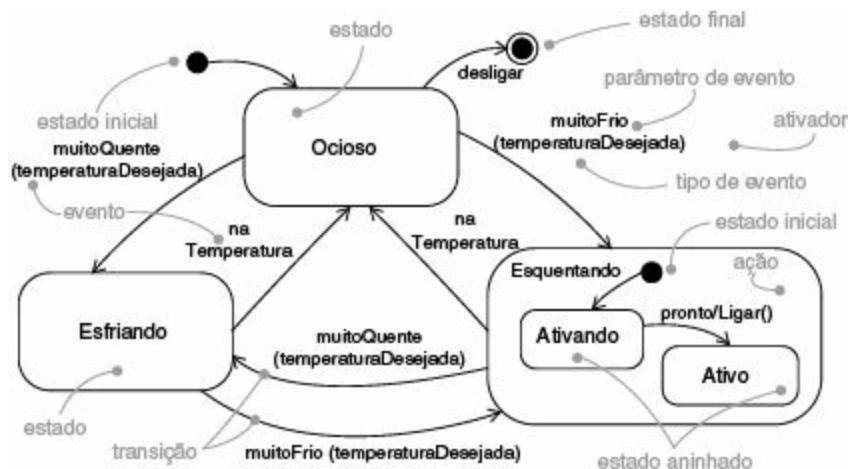


Figura 22.1:  
Máquinas de estados

## CONTEXTO

Todo objeto tem um tempo de vida. Na criação, o objeto nasce; na destruição, o objeto deixa de existir. Entre esses dois momentos, o objeto poderá agir sobre outros objetos (enviando-lhes mensagens), assim como também poderão agir sobre ele (quando é o destinatário de uma mensagem). Em muitos casos, essas mensagens serão simples chamadas síncronas a operações. Por exemplo, uma instância da classe Cliente poderá invocar a operação pegarSaldo ou uma instância da classe ContaCorrente. Objetos como esses não necessitam de uma máquina de estados para especificar seu comportamento, pois o comportamento atual não depende de seu passado.

- Os objetos são examinados no [Capítulo 13](#); as mensagens são apresentadas no [Capítulo 16](#).

Em outros tipos de sistemas, você encontrará objetos que precisam responder a sinais, que são mensagens assíncronas comunicadas entre instâncias. Por exemplo, um telefone celular precisa responder a chamadas telefônicas aleatórias (de outros telefones), a eventos no teclado numérico (do cliente iniciando uma ligação) e os eventos da rede (quando o telefone passa de uma chamada para outra). De modo semelhante, você encontrará objetos cujo comportamento atual depende de seu comportamento passado. Por exemplo, o comportamento de um sistema de orientação de mísseis aéreos dependerá de seu estado atual, como NãoVoando (não é uma boa ideia lançar um míssil, enquanto ele se encontra em uma aeronave que ainda está em terra) ou Procurando (o míssil não deverá ser armado até você ter uma boa ideia do que servirá como alvo).

- Os sinais são examinados no [Capítulo 21](#).

O comportamento de objetos que devem responder a mensagens assíncronas ou cujo comportamento atual depende de seu passado é mais bem especificado pela utilização de uma máquina de estados. A máquina de estados é capaz de abranger instâncias de classes que podem receber sinais, incluindo muitos objetos ativos. De fato, um objeto que recebe um sinal, mas

não dispõe de uma máquina de estados, simplesmente ignorará esse sinal. A máquina de estados também pode ser empregada para a modelagem do comportamento do sistema inteiro, especialmente de sistemas reativos, que devem responder aos sinais de atores externos ao sistema.

- Os objetos ativos são examinados no [Capítulo 23](#); a modelagem de sistemas reativos é apresentada no [Capítulo 25](#); os casos de uso e atores são discutidos no [Capítulo 17](#); as interações aparecem no [Capítulo 16](#); as interfaces são explicadas no [Capítulo 11](#).

**Nota:** Na maior parte do tempo, você usará as interações para fazer a modelagem do comportamento de um caso de uso, mas também pode aplicar máquinas de estados para o mesmo propósito. De modo semelhante, pode aplicar máquinas de estados para a modelagem do comportamento de uma interface. Embora uma interface possa não ter qualquer instância direta, uma classe que realize essa interface poderá ter. Essa classe deve estar em conformidade com o comportamento especificado pela máquina de estados dessa interface.

## ESTADOS

Um estado é uma condição ou situação na vida de um objeto durante a qual o objeto satisfaz alguma condição, realiza alguma atividade ou aguarda um evento. Um objeto permanece em um estado por uma quantidade finita de tempo. Por exemplo, um Aquecedor em uma casa poderá estar em um dos quatro estados: Ocioso (aguardando um comando para começar a aquecer a casa), Ativando (o gás está ligado, mas está aguardando chegar à temperatura), Ativo (o gás e o circulador estão ligados) e Desligando (seu gás está desligado, mas o circulador continua distribuindo o calor restante do sistema).

Quando a máquina de estados de um objeto está em um determinado estado, o objeto é dito estar nesse estado. Por exemplo, uma instância de Aquecedor poderá ser Ocioso ou talvez Desligando.

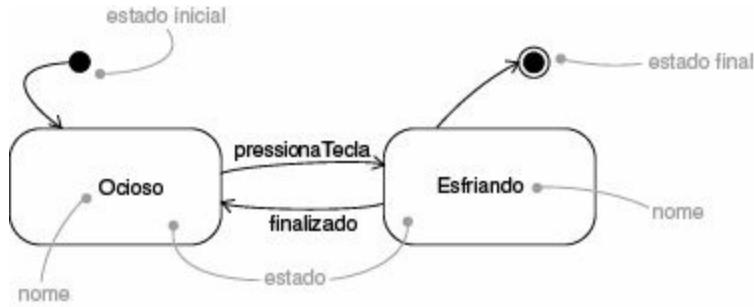
- O estado de um objeto em uma interação pode ser visualizado, conforme é examinado no [Capítulo 13](#); as últimas quatro partes de um estado são examinadas em seções posteriores deste capítulo.

Um estado tem várias partes.

1. Nome                          Uma sequência de caracteres textual que diferencia o estado de outros estados; um estado poderá ser anônimo, significando que não tem nome.
2.                                  Ações de entrada/saída Ações executadas na entrada e saída do estado, respectivamente.
3. Transições internas        Transições manipuladas sem causar alteração do estado.
4. Subestados                    A estrutura aninhada de um estado, envolvendo subestados disjuntos (sequencialmente ativos) ou concorrentes (concorrentemente ativos).
5. Eventos adiados             Uma lista de eventos que não são manipulados nesse estado, mas, em vez disso, são adiados e colocados em fila para serem manipulados pelo objeto em outro estado.

**Nota:** Um nome de estado poderá ser texto formado por qualquer número de letras, números e determinados sinais de pontuação (exceto sinais como os dois-pontos) e poderá se estender por várias linhas. Na prática, os nomes de estados são expressões nominais ou nomes curtos, definidos a partir do vocabulário do sistema cuja modelagem está sendo feita. Tipicamente, é maiúscula a primeira letra de cada palavra de um nome de estado, como em Ociozo OU Desligando.

Conforme mostra a [Figura 22.2](#), o estado é representado como um retângulo com cantos arredondados.



**Figura 22.2:**

Estados

Estados inicial e final Conforme mostra a figura, existem dois estados especiais que poderão ser definidos para a máquina de estados de um objeto. Primeiro, existe o estado inicial, indicando o local de início padrão para a máquina de estados ou subestado. Um estado inicial é representado como um círculo preto preenchido. Segundo, existe o estado final, indicando que a execução da máquina de estados ou do estado que a contém foi concluída. Um estado final é representado como um círculo preto preenchido com outro círculo não preenchido ao seu redor.

**Nota:** Os estados *inicial* e *final* são realmente pseudoestados. Podem não ter as partes usuais de um estado normal, exceto o nome. A transição de um estado inicial para um estado final poderá ter todos os complementos de características, incluindo uma ação e uma condição de guarda (mas não um evento de ativação).

## TRANSIÇÕES

Uma transição é um relacionamento entre dois estados, indicando que um objeto no primeiro estado realizará certas ações e entrará no segundo estado quando um evento especificado ocorrer e as condições especificadas forem satisfeitas. Nessa alteração de estado, a transição é instruída a se iniciar. Até a transição se iniciar, o objeto é considerado estar no estado de origem; após ser iniciada, é dito que a transição está no estado de destino. Por exemplo, um Aquecedor poderá fazer a transição do estado Ocioso para o estado Ativando, quando um evento como muitoFrio (com o parâmetro temperaturaDesejada) ocorre.

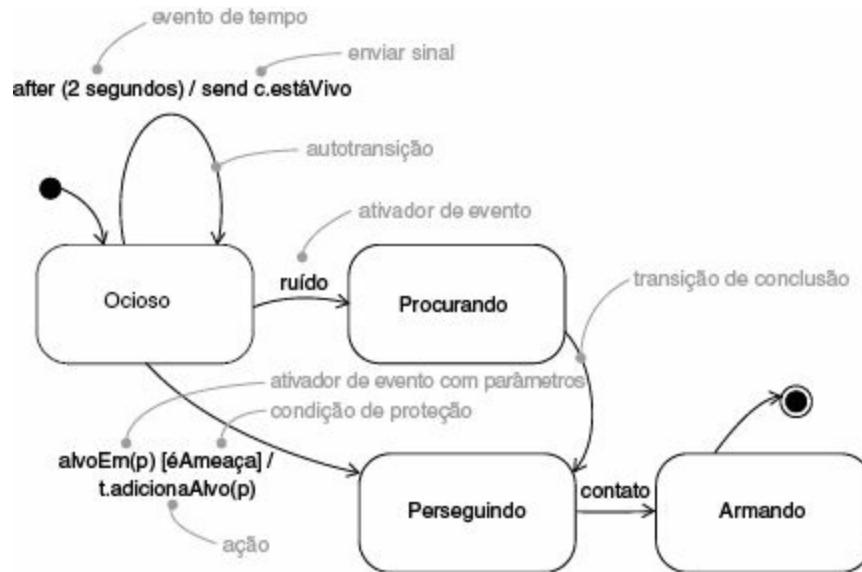
Uma transição tem cinco partes.

1. Estado de origem	O estado afetado pela transição; se um objeto estiver no estado de origem, uma transição de saída poderá ser iniciada quando o objeto receber o evento de ativação da transição e se a condição de guarda, se houver, for satisfeita.
2. Evento de ativação	O evento cuja recepção pelo objeto no estado de origem faz com que a transição possa ser escolhida para ser ativada, desde que sua condição de guarda seja satisfeita.
3. Condição de guarda	Uma expressão booleana que é avaliada quando a transição é iniciada pela recepção do evento de ativação; se a expressão for avaliada como verdadeira, a transição será escolhida para se iniciar; se a expressão for avaliada como falsa, a transição não será iniciada e, caso não haja outra transição que possa ser iniciada pelo mesmo evento, esse evento é perdido.
4. Efeito	Um comportamento executável, como uma ação, que poderá agir diretamente no objeto ao qual a máquina de estados pertence e indiretamente em outros objetos que são visíveis ao objeto.
5. Estado destino	O estado que está ativo após a conclusão da transição.

► Os eventos são examinados no [Capítulo 21](#).

Conforme mostra a [Figura 22.3](#), uma transição é representada como uma linha cheia direcionada do estado de origem para o de destino. Uma

autotransição é uma transição cujos estados de origem e de destino são o mesmo.



**Figura 22.3:**  
Transições

**Nota:** Uma transição poderá ter várias origens (nesse caso, ela representa a união de vários estados concorrentes), assim como vários destinos (nesse caso, ela representa uma bifurcação para vários estados concorrentes). Consulte a discussão posterior sobre subestados sequenciais.

**Evento de ativação** Um evento é a especificação de uma ocorrência que tem uma localização no tempo e no espaço. No contexto das máquinas de estados, um evento é uma ocorrência de um estímulo capaz de ativar uma transição de estado. Conforme mostrado na figura anterior, os eventos podem incluir sinais, chamadas, a passagem de tempo ou a alteração de um estado. Um sinal ou uma chamada podem ter parâmetros, cujos valores estão disponíveis para a transição, incluindo expressões para a ação e a condição de guarda.

► Eventos são discutidos no [Capítulo 21](#).

Também é possível haver uma transição de parada, representada por uma transição sem um evento de ativação. A transição de parada, também chamada transição de conclusão, é iniciada implicitamente, quando seu estado de origem completa seu comportamento, se houver.

**Nota:** Um evento de iniciação poderá ser polimórfico. Por exemplo, caso você tenha especificado uma família de sinais, então uma transição cujo evento de ativação seja  $s$  poderá ser iniciada por  $s$ , assim como por qualquer filho de  $s$ .

- A especificação de uma família de sinais é examinada no [Capítulo 21](#); várias condições de guarda, que não se sobrepõem, formam uma ramificação, conforme é apresentado no [Capítulo 20](#).

Condição de guarda Conforme mostra a figura anterior, a condição de guarda é representada como uma expressão booleana entre colchetes e colocada antes do evento de ativação. Uma condição de guarda é avaliada somente após o evento de ativação para sua transição ocorrer. Portanto, é possível haver várias transições a partir do mesmo estado de origem e com o mesmo evento de ativação, desde que essas condições não se sobreponham.

Uma condição de guarda é avaliada apenas uma vez para cada transição no momento em que o evento ocorre, mas poderá ser avaliada novamente, se a transição for ativada mais uma vez. Na expressão booleana, pode-se incluir condições sobre o estado de um objeto (por exemplo, a expressão `aHeater in Idle`, que é avaliada como verdadeira, caso o objeto que corresponde ao aquecedor, Heater, esteja atualmente em estado de ociosidade, `Idle`). Se a condição não for verdadeira, quando testada, o evento não ocorrerá posteriormente, quando a condição tornar-se verdadeira. Use um evento de alteração para modelar esse tipo de comportamento.

**Nota:** Apesar de a condição de guarda ser avaliada somente uma vez quando sua transição é ativada, um evento de alteração é potencialmente avaliado de modo contínuo.

- Os eventos de alteração são examinados no [Capítulo 20](#).

**Efeito** Um efeito é um comportamento executado quando uma transição é acionada. Os efeitos podem incluir computação na linha, chamadas a operações (ao objeto ao qual a máquina de estados pertence, assim como qualquer outro objeto visível), a criação ou destruição de outro objeto ou o envio de um sinal a um objeto. Para indicar o envio de um sinal, você pode prefixar o nome do sinal com a palavra reservada `send` como uma indicação visual.

Transições só ocorrem quando a máquina de estados está em repouso, ou seja, quando não está executando um efeito de uma transição anterior. A execução do efeito de uma transição e quaisquer entradas associadas e efeitos de saída concorrem para a conclusão antes de qualquer evento adicional poder causar transições adicionais. Esse é um contraste com uma atividade (descrita posteriormente neste capítulo), que pode ser interrompida por eventos.

- As atividades são examinadas em uma seção adiante neste capítulo; as dependências são apresentadas nos Capítulos 5 e 10.

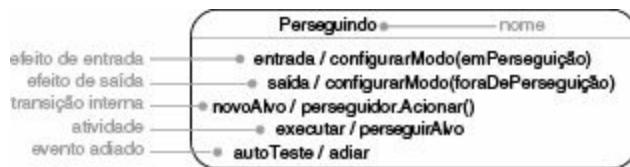
**Nota:** Você pode exibir explicitamente o objeto ao qual um sinal é enviado, utilizando uma dependência estereotipada como `send`, cuja origem é o estado e cujo destino é o objeto.

## TRANSIÇÕES E ESTADOS AVANÇADOS

Você pode fazer a modelagem de uma ampla variedade de comportamentos, utilizando somente as características básicas de estados e transições na UML. Utilizando essas características, você acabará tendo máquinas de estados planas, o que significa que seus modelos comportamentais serão formados por nada mais do que arcos (transições) e vértices (estados).

Entretanto, as máquinas de estados da UML têm muitas características avançadas que auxiliam o gerenciamento de modelos comportamentais complexos. Essas características costumam reduzir a quantidade de estados e transições necessários e codificam vários idiomas comuns e de alguma

maneira complexos que, caso contrário, você encontraria, se utilizasse máquina de estados planas. Algumas dessas características avançadas incluem ações de entrada e saída, transições internas, atividades e eventos adiados. Essas características são apresentadas como sequências de texto em um compartimento de texto do símbolo de estado, como mostra a [Figura 22.4](#).



**Figura 22.4:**

Transições e estados avançados

Ações de entrada e saída Em várias situações de modelagem, você desejará realizar a mesma ação sempre que entrar em um estado, qualquer que seja a transição que o leve até ele. De modo semelhante, ao deixar um estado, você desejará realizar a mesma ação, qualquer que seja a transição que o faça sair. Por exemplo, em um sistema de orientação de mísseis, você poderá querer anunciar explicitamente que o sistema está `emPerseguicao` sempre que estiver no estado `Perseguido` e `foraDePerseguicao` sempre que estiver fora desse estado. Utilizando máquinas de estados planas, é possível obter esse efeito, colocando essas ações em todas as transições de entrada e de saída, conforme seja adequado. Entretanto, isso estará de alguma forma sujeito a erros; é necessário lembrar-se de adicionar essas ações sempre que acrescentar uma nova transição. Além disso, a modificação dessa ação significará que você precisará alterar todas as transições vizinhas.

Conforme mostra a [Figura 22.4](#), a UML oferece um atalho para esse idioma. No símbolo para o estado, pode-se incluir uma ação de entrada (marcada pelo evento da palavra reservada `entry`) e uma ação de saída (marcada pelo evento da palavra reservada `exit`), juntamente com a açãopropriada. Sempre que você entrar no estado, sua ação de entrada será iniciada; sempre que deixar o estado, a ação de saída será iniciada.

Os efeitos de entrada e saída podem não ter argumentos ou condições de proteção. Entretanto, o efeito de entrada que se encontra no nível superior de uma máquina de estados para uma classe pode ter parâmetros representando os argumentos que a máquina recebe quando o objeto é criado.

Transições internas Uma vez em um estado, você encontrará eventos que desejará manipular sem sair desse estado. Esses casos são chamados transições internas e são sutilmente diferentes das autotransições. Em uma autotransição, como aquela mostrada na [Figura 22.3](#), um evento ativa a transição, você sai do estado, uma ação (se houver) é iniciada e depois você entra novamente no mesmo estado. Como essa transição sai e depois entra no estado, uma autotransição realiza a ação de saída do estado e depois inicia a ação de autotransição; por fim, ela ativa a ação de entrada no estado.

Entretanto, suponha que você queira manipular o evento, mas sem realizar as ações de entrada e saída do estado. A UML oferece um atalho para esse idioma usando uma transição interna. Uma *transição interna* é uma transição que responde a um evento realizando um efeito, mas não um estado de alteração. Na [Figura 22.4](#), o evento novoAlvo é o rótulo de uma transição interna; se esse evento ocorre enquanto o objeto está no estado Persguindo, a ação perseguidor.Acionar é executada, mas o estado permanece o mesmo, e nenhuma ação de entrada ou saída é executada. Você indica uma transição interna incluindo uma sequência de transição (incluindo um nome de evento, uma condição de guarda adicional e um efeito) dentro do símbolo de um estado em vez de uma seta de transição. Observe que as palavras-chave entry, exit e do são palavras reservadas que não podem ser usadas como nomes de eventos. Sempre que você estiver no estado e esse evento for ativado, a ação correspondente é iniciada sem sair e depois retornar ao estado. Portanto, o evento é manipulado sem a realização das ações de entrada e saída do estado.

**Nota:** As transições internas admitem eventos com parâmetros e condições de proteção.

Atividades Quando um objeto está em um estado, ele geralmente permanece ocioso, aguardando que um evento ocorra. De vez em quando,

entretanto, você desejará fazer a modelagem de uma atividade em andamento. Enquanto se encontra em um estado, o objeto continuará realizando o mesmo trabalho até ser interrompido por um evento. Por exemplo, se o objeto estiver no estado Perseguindo, ele poderá executar perseguirAlvo enquanto permanecer nesse estado. Conforme mostra a [Figura 22.4](#), na UML Reference Manual, a transição especial do é utilizada para especificar o trabalho a ser realizado em estado após a ação de entrada ter sido ativada. Você também poderá especificar um comportamento, como uma sequência de ações – por exemplo, do / op1(a); op2(b); op3(c). Se a ocorrência de um evento causa uma transição que força uma saída do estado, qualquer atividade em andamento do estado é imediatamente interrompida.

**Nota:** *Uma atividade é equivalente a um efeito de entrada que inicia a atividade quando se entra no estado, e a um efeito de saída que para a atividade quando se sai do estado.*

Eventos adiados Considere um estado como Perseguindo. Conforme mostra a [Figura 22.3](#), suponha existir somente uma transição conduzindo para fora desse estado, ativada pelo evento contato. Enquanto se encontra no estado Perseguindo, qualquer evento que não seja contato e que não seja manipulado pelos seus subestados será perdido. Isso significa que o evento poderá ocorrer, mas será adiado e nenhuma ação resultará devido à presença desse evento.

► Os eventos são examinados no [Capítulo 21](#).

Em qualquer situação de modelagem, você desejará reconhecer alguns eventos e ignorar outros. Você inclui os que deseja reconhecer como eventos que ativam transições; os que deseja ignorar você simplesmente deixa de fora. Entretanto, em algumas situações de modelagem, você desejará reconhecer alguns eventos, mas adiando uma resposta a eles para mais tarde. Por exemplo, enquanto se encontra no estado Perseguindo, você poderá querer

adiar uma resposta a sinais como `autoTeste`, talvez enviados por algum agente de manutenção do sistema.

Na UML, esse comportamento pode ser especificado, utilizando eventos adiados. Um evento adiado é uma lista de eventos cuja ocorrência no estado é adiada até um estado em que os eventos listados não são adiados e se tornam ativos; nesse momento eles ocorrem e podem ativar transições como se tivessem acabado de ocorrer. Conforme é possível observar na figura anterior, um evento adiado pode ser especificado, listando o evento com a ação especial `defer`. Nesse exemplo, os eventos `autoTeste` poderão acontecer enquanto estiverem no estado `Perseguido`, mas são mantidos até que o objeto esteja no estado `Armando`, quando ele aparece como se tivesse acabado de ocorrer.

**Nota:** A implementação de eventos adiados requer a presença de uma fila interna de eventos. Se um evento ocorrer, mas estiver listado como adiado, ele entrará na fila. Os eventos são removidos dessa fila assim que o objeto entra em um estado que não adia esses eventos.

**Submáquinas** Uma máquina de estados pode ser referenciada em outra máquina de estados. Essa máquina de estados referenciada chama-se *submáquina*. Elas são úteis para construir grandes modelos de estado de uma maneira estruturada. Consulte o *Manual de referência da UML* para obter detalhes.

## SUBESTADOS

Essas características avançadas de estados e transições solucionam vários problemas comuns de modelagem de máquinas de estados. Entretanto, existe mais uma característica das máquinas de estados da UML – os subestados – que ajudam ainda mais a simplificar a modelagem de comportamentos complexos. Um subestado é um estado aninhado em outro estado. Por exemplo, um `Heater` (aquecedor) poderá estar no estado `Heating` (aquecendo), mas também, enquanto se encontra nesse estado, poderá haver um estado

aninhado, chamado *Activating* (ativando). Nesse caso, é adequado dizer que o objeto está tanto *Heating* como *Activating*.

Um estado simples é um estado que não tem subestrutura. Um estado que contém subestados – ou seja, estados aninhados – é chamado um estado composto. Um estado composto pode conter outros subestados concorrentes (ortogonais) ou sequenciais (disjuntos). Na UML, um estado composto é representado da mesma maneira que um estado simples, mas com um compartimento gráfico opcional, mostrando uma máquina de estados aninhada. Os subestados podem estar aninhados em qualquer nível.

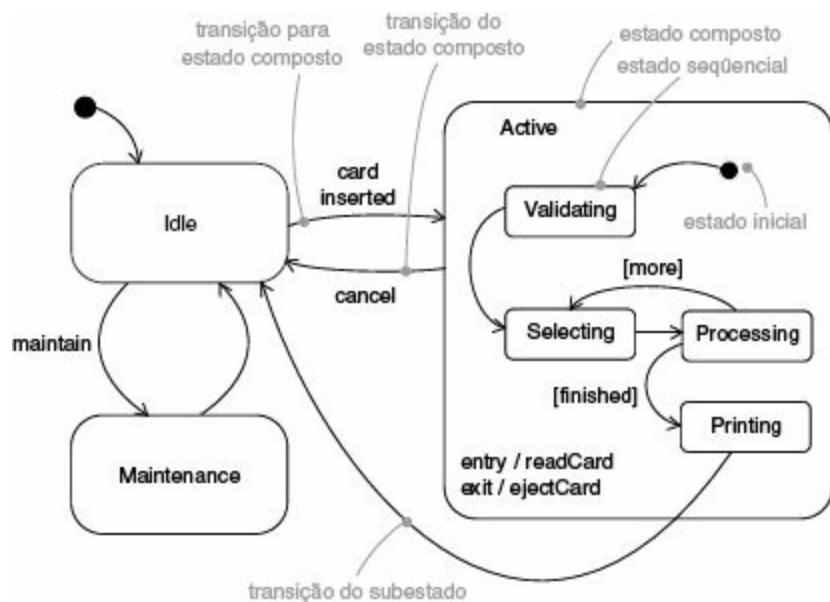
- *Os estados compostos têm uma estrutura aninhada semelhante à composição, conforme é examinado nos Capítulos 5 e 10.*

Subestados sequenciais Considere o problema da modelagem do comportamento de um caixa eletrônico. Existem três estados básicos em que esse sistema poderá estar: *Idle* (em estado de ociosidade, aguardando a interação com o cliente), *Active* (ativo, manipulando a transação do cliente) e *Maintenance* (em manutenção, talvez repondo o estoque de dinheiro). Enquanto no estado *Active*, o comportamento do caixa eletrônico segue um caminho simples: valida o cliente, seleciona uma transação, processa a transação e depois imprime um recibo. Após a impressão, o caixa eletrônico retorna ao estado *Idle*. Esses estágios de comportamento poderão ser representados como os estados *Validating*, *Selecting*, *Processing* e *Printing* (respectivamente *Validando*, *Selecionando*, *Processando* e *Imprimindo*). Seria até conveniente permitir que o cliente selecionasse e processasse várias transações depois de *Validating* a conta e antes de *Printing* o recibo final.

O problema aqui é que, em qualquer estágio nesse comportamento, o cliente poderá decidir cancelar a transação, retornando o caixa eletrônico a seu estado *Idle*. Utilizando uma máquina de estados plana, é possível obter esse efeito, mas será quase inviável. Como o cliente poderá cancelar a transação em qualquer ponto, será necessário incluir uma transição ajustável a partir de todos os estados da sequência *Active*. Isso será inviável, pois será

fácil esquecer a inclusão dessas transições nos locais certos e muitos desses eventos de interrupção significam que você acabará com uma variedade de transições zerando no mesmo estado de destino a partir de várias origens, mas com o mesmo evento de ativação, condição de guarda e ação.

Utilizando subestados sequenciais, existe uma forma mais simples de fazer uma modelagem desse problema, conforme mostra a [Figura 22.5](#). Aqui, o estado Active tem uma subestrutura, contendo os subestados Validating, Selecting, Processing e Printing. O estado do caixa eletrônico de Idle para Active quando o cliente coloca o cartão de crédito na máquina. Entrando no estado Active, a ação de entrada correspondente à leitura do cartão do cliente, readCard, é realizada. A partir do estado inicial da subestrutura, o controle passa para o estado Validating, depois para o estado Selecting e depois para o estado Processing. Depois de Processing, o controle poderá retornar a Selecting (caso o cliente tenha selecionado outra transação) ou poderá passar à impressão. Após o estado Printing, existe uma transição não ativada que faz retornar ao estado Idle. Observe que o estado Active tem uma ação de saída, que ejeta o cartão de crédito do cliente.



**Figura 22.5:**  
Subestados sequenciais

Observe também a transição do estado Active para o estado Idle, iniciada pelo evento de cancelamento, cancel. Em qualquer subestado de Active, o cliente poderá cancelar a transação e isso fará com que o caixa eletrônico retorne ao estado Idle (mas somente após ejetar o cartão de crédito do cliente, que é a ação de saída ativada ao sair do estado Active, independentemente do que tenha causado a transição para fora daquele estado). Sem subestados, seria necessária uma transição ativada por cancel em todos os estados da subestrutura.

Subestados, como Validating e Processing, são chamados sequenciais ou disjuntos. Dado um conjunto de subestados disjuntos no contexto de um estado composto que o contém, o objeto é considerado estar no estado composto e em somente um desses subestados (ou o estado final) em um momento. Portanto, os subestados sequenciais dividem o espaço de estado do estado composto em estados disjuntos.

A partir de uma origem externa ao estado composto que a contém, uma transição poderá ter como destino o estado composto ou um subestado. Se o destino for o estado composto, máquina de estados aninhada precisará incluir um estado inicial, ao qual o controle passará depois de entrar no estado composto e depois de ativar sua ação de entrada (se houver). Se o destino for o estado aninhado, o controle passará ao estado aninhado, após iniciar a ação de entrada (se houver) do estado composto e depois a ação de entrada (se houver) do subestado. A transição que conduz para fora de um estado composto poderá ter, como sua origem, o estado composto ou um subestado. Em qualquer um desses casos, o controle primeiro sai do estado aninhado (e sua ação de saída, se houver, é ativada), depois sai do estado composto (e sua ação de saída, se houver, é ativada). Uma transição cuja origem é o estado composto essencialmente reduz (interrompe) a atividade da máquina de estados aninhada. A transição de parada de um estado composto ocorre quando o controle atinge o subestado final no estado composto.

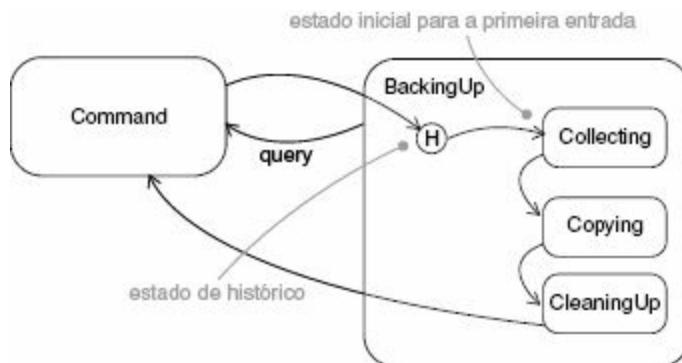
**Nota:** Uma máquina de estados sequencial aninhada poderá ter, no máximo, um único estado inicial e um único estado final.

Estados de histórico Uma máquina de estados descreve os aspectos dinâmicos de um objeto cujo comportamento atual depende de seu passado. Uma máquina de estados especifica a ordenação legal dos estados pelos quais um objeto poderá passar durante seu tempo de vida.

A menos que seja especificado o contrário, quando uma transição entra em um estado composto, a ação da máquina de estados aninhada começa novamente em seu estado inicial (a menos é claro, que, a transição tenha como destino um subestado diretamente). Entretanto, em certos casos, você desejará fazer a modelagem de um objeto, de forma que ele lembre o último subestado que se encontrava ativo antes de deixar o estado composto. Por exemplo, para a modelagem do comportamento de um agente que faz um backup, não assistido, de computadores em uma rede, você desejará que ele lembre onde se encontrava no processo, caso seja interrompido, por exemplo, por uma consulta realizada pelo operador.

Utilizando máquinas de estados planas, é possível fazer a modelagem disso, mas ela não será útil. Para cada subestado sequencial, será necessário fazer com que sua ação de saída coloque um valor em alguma variável local para o estado composto. Então o estado inicial para esse estado composto precisaria de uma transição para todos os subestados com uma condição de guarda, consultando a variável. Dessa forma, a saída do estado composto faria com que o último subestado fosse lembrado; a entrada no estado composto faria a transição ao subestado apropriado. Isso não funcionará, pois requer que você se lembre de alterar todos os subestados e definir uma ação de saída apropriada. Isso deixaria você com várias transições a partir de um mesmo estado inicial para diferentes subestados de destino com condições de guarda muito semelhantes (mas diferentes).

Na UML, uma forma mais simples de fazer a modelagem desse idioma consiste em utilizar os estados de histórico. Um estado de histórico permite a existência de um estado composto contendo subestados sequenciais para lembrar o último subestado que nele estava ativo antes da transição do estado composto. Conforme mostra a [Figura 22.6](#), você representa um estado de histórico como um pequeno círculo contendo o símbolo  $H$ .



**Figura 22.6:**  
Estado de histórico

Se quiser uma transição para ativar o último subestado, mostre uma transição externa ao estado composto diretamente para o estado de histórico. Na primeira vez que você entrar no estado composto, ele não terá um histórico. Esse é o significado de uma única transição do estado de histórico para um subestado sequencial de coleta de dados, como Collecting. O destino dessa transição especifica o estado inicial da máquina de estados aninhada no qual se entra pela primeira vez. Suponha que, enquanto no estado de armazenamento de informações BackingUp e no estado de cópia das mesmas Copying, ocorra o evento query. O controle deixa Copying e BackingUp (ativando suas ações de saída, se necessário) e retorna ao estado anterior para a execução do comando, Command. Quando a ação de Command for concluída, a transição não ativada retornará ao estado de histórico do estado composto BackingUp. Dessa vez, como existe um histórico para a máquina de estados aninhada, o controle retorna ao estado Copying – ignorando, portanto, o estado Collecting, pois Copying foi o último subestado ativo antes da transição a partir de BackingUp.

**Nota:** O símbolo  $\text{h}$  designa um histórico superficial, capaz de lembrar somente o histórico da máquina de estados aninhada imediata. Você também poderá especificar um histórico profundo, mostrado como um pequeno círculo contendo o símbolo  $\text{h}$ . O histórico profundo é capaz de lembrar o estado aninhado mais interno em qualquer profundidade. Caso você tenha somente um nível de aninhamento, os estados de histórico superficial e profundo são semanticamente equivalentes. Caso

*você tenha mais de um nível de aninhamento, o histórico superficial lembra somente o estado aninhado mais externo; o histórico profundo lembra o estado aninhado mais interno em qualquer profundidade.*

Em qualquer caso, se uma máquina de estados aninhada chegar a um estado final, ela perde o histórico armazenado e se comporta como se ainda não tivesse entrado pela primeira vez.

Subestados concorrentes Subestados sequenciais são o tipo mais comum de máquina de estados aninhada que você encontrará. Em certas situações de modelagem, entretanto, você desejará especificar subestados concorrentes. Eses subestados permitem especificar duas ou mais máquinas de estados que são executadas em paralelo no contexto do objeto que o contém.

Por exemplo, a [Figura 22.7](#) mostra uma expansão do estado de manutenção, Maintenance, em relação à [Figura 22.5](#). Esse estado é decomposto em dois subestados concorrentes, Testing e Commanding (respectivamente de Teste e de Comando), mostrados pelo seu aninhamento no estado Maintenance, mas separados um do outro por uma linha tracejada. Cada um desses subestados concorrentes é ainda decomposto em subestados sequenciais. Quando o controle passa do estado Idle para o estado Maintenance, o controle então bifurca-se em dois fluxos concorrentes – o objeto que os contém estará no estado Testing e no estado Commanding. Além disso, enquanto no estado Commanding, o objeto que os contém estará no estado de espera, Waiting, ou no estado de execução de comando, Command.

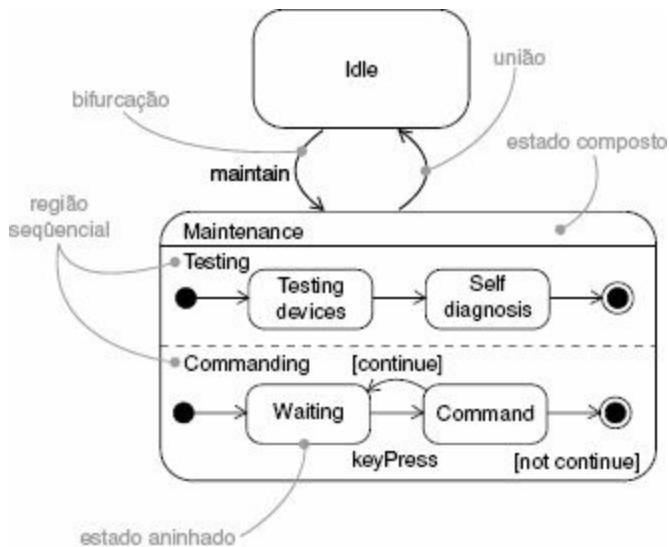


Figura 22.7:

Subestados concorrentes

**Nota:** Isso é o que diferencia subestados sequenciais e subestados concorrentes. Dados dois ou mais subestados sequenciais em um mesmo nível, um objeto estará em um desses subestados. Dados dois ou mais subestados concorrentes em um mesmo nível, um objeto estará em um estado sequencial a partir de cada um dos subestados concorrentes.

A execução desses dois subestados concorrentes continua sendo realizada em paralelo. No fim, cada máquina de estados aninhada alcança seu estado final. Se um dos subestados concorrentes alcançar seu estado final antes do outro, o controle nesse subestado aguarda em seu estado final. Quando as duas máquinas de estados aninhadas alcançam seus estados finais, os controles dos dois subestados concorrentes se unem novamente em um único fluxo.

Sempre que existir uma transição de um estado composto que é decomposto em subestados concorrentes, o controle se bifurcará em tantos fluxos concorrentes quantos forem os subestados concorrentes. De modo semelhante, sempre que houver uma transição de um subestado composto que é decomposto em subestados concorrentes, o controle se unirá em um único fluxo. Isso é verdadeiro em todos os casos. Se todos os subestados concorrentes alcançarem seus estados finais ou se houver uma transição

explícita para fora do estado composto que os contém, o controle se unirá novamente em um único fluxo.

**Nota:** Cada região concorrente pode ter um estado inicial, final ou de histórico.

**Bifurcação e união** Normalmente, a entrada para um estado composto com regiões concorrentes vai para o estado inicial de cada região concorrente. Isso também é possível na transição de um estado externo diretamente para um ou mais estados concorrentes. Esse procedimento chama-se bifurcação, porque o controle passa de um estado único para vários estados concorrentes. A bifurcação é apresentada como uma linha preta forte com uma seta de entrada e várias setas de saída, cada uma para um dos estados concorrentes. Deve haver pelo menos um estado de destino em cada região concorrente. Se uma ou mais regiões concorrentes não tiverem estados de destino, o estado inicial dessas regiões será escolhido implicitamente. Uma transição para um estado concorrente único em um estado composto também é uma bifurcação implícita; os estados iniciais de todas as outras regiões concorrentes são implicitamente parte da bifurcação.

De maneira semelhante, uma transição de qualquer estado em um estado composto com regiões concorrentes força uma saída de todas as regiões concorrentes. Essa transição frequentemente representa uma condição de erro que força a conclusão da computação paralela.

Uma união é uma transição com duas ou mais setas de entrada e uma seta de saída. Cada seta de entrada deve vir de um estado em uma diferente região concorrente do mesmo estado composto. A união pode ter um evento de ativação. A transição de união só é eficiente se todos os estados de origem estiverem ativos; o status das outras regiões concorrentes no estado composto é irrelevante. Se o evento ocorrer, o controle deixa todas as regiões concorrentes no estado composto, não apenas as que têm setas partindo delas.

A [Figura 22.8](#) mostra uma variação do exemplo anterior com transições de bifurcação e união explícitas. A transição `maintain` (entrar em manutenção) para o estado composto `Maintenance` ainda é uma bifurcação implícita nos estados

iniciais padrão das regiões concorrentes. Neste exemplo, entretanto, há também uma bifurcação explícita de `Idle` para os dois estados aninhados `Self diagnosis` (autodiagnóstico) e o estado final da região `Commanding` (um estado final é um estado real e pode ser o destino de uma transição). Se ocorrer um evento de erro enquanto o estado `Self diagnose` estiver ativo, a transição de união implícita para `Repair` é ativada: tanto o estado `Self diagnose` quanto qualquer outro estado ativo na região `Commanding` são encerrados. Há também uma transição de união explícita para o estado desconectado, `Offline`. Essa transição é ativada somente se o evento de desconexão, `disconnect`, ocorrer enquanto o estado `Testing devices` (Testando dispositivos) e o estado final da região `Commanding` estiverem ativos; se os dois estados não estiverem ativos, o evento não tem efeito.

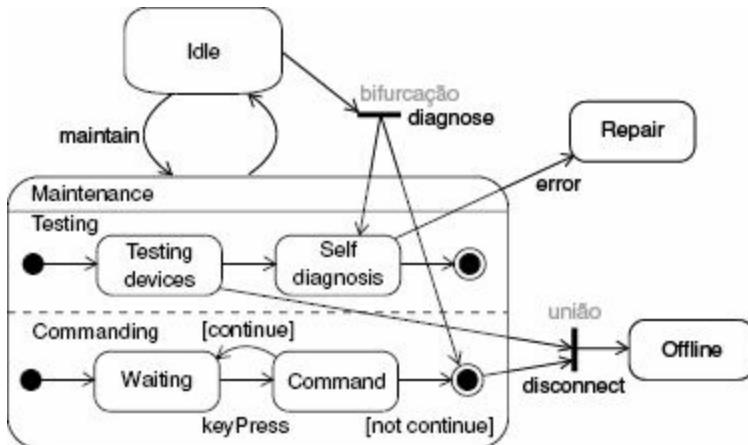


Figura 22.8:

Transições de bifurcação e união

Objetos ativos Outra forma de fazer a modelagem da concorrência consiste na utilização de objetos ativos. Portanto, em vez de dividir a máquina de estados de um objeto em dois (ou mais) subestados concorrentes, você poderia definir dois objetos ativos, cada um dos quais responsável pelo comportamento de um dos subestados concorrentes. Se o comportamento de um desses fluxos concorrentes for afetado pelo estado do outro, você desejará fazer essa modelagem, utilizando subestados concorrentes. Se o comportamento de um desses fluxos concorrentes for afetado por mensagens

enviadas de um e para o outro, você desejará fazer essa modelagem, utilizando objetos ativos. Caso haja pouca ou nenhuma comunicação entre os fluxos concorrentes, então a solução a ser escolhida é uma mera questão de preferência, embora, na maior parte do tempo, a utilização de objetos ativos torne mais óbvias as suas decisões de projeto.

- ➡ *Os objetos ativos são examinados no [Capítulo 23](#).*

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DO TEMPO DE VIDA DE UM OBJETO

O propósito mais comum para o qual a máquina de estados será utilizada é a modelagem do tempo de vida de um objeto, especialmente instâncias de classes, casos de uso e o sistema como um todo. Enquanto as interações modelam o comportamento de uma sociedade de objetos trabalhando em conjunto, as máquinas de estados modelam o comportamento de um único objeto em seu tempo de vida, como você encontrará com interfaces para o usuário, controladores e dispositivos.

- ➡ *Os objetos são examinados no [Capítulo 13](#); as classes são apresentadas nos [Capítulos 4 e 9](#); os casos de uso são explicados no [Capítulo 17](#); os sistemas aparecem no [Capítulo 32](#); as interações são discutidas no [Capítulo 16](#); as colaborações são examinadas no [Capítulo 28](#); as pré e pós-condições são examinadas no [Capítulo 10](#); as interfaces são apresentadas no [Capítulo 11](#).*

Ao fazer a modelagem do tempo de vida de um objeto, você especifica três itens essencialmente: os eventos aos quais o objeto pode responder, a resposta a esses eventos e o impacto do passado no comportamento atual. A modelagem do tempo de vida de um objeto também envolve uma decisão a respeito da ordem em que o objeto pode responder a eventos de maneira

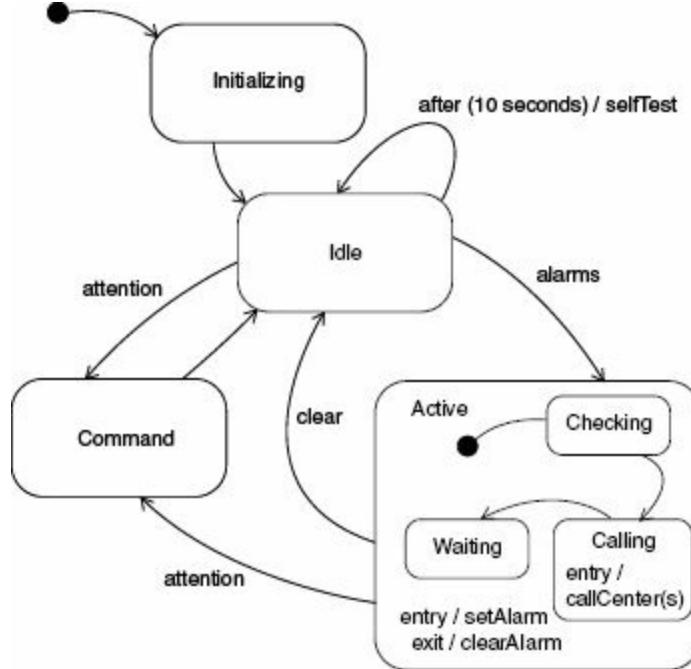
significativa, começando no momento da criação do objeto e continuando até sua destruição.

Para fazer a modelagem do tempo de vida de um objeto:

- › Defina o contexto para a máquina de estados, seja uma classe, um caso de uso ou o sistema como um todo.
  - Se o contexto é uma classe ou um caso de uso, coleciona as classes vizinhas, incluindo qualquer classe-mãe da classe e quaisquer classes que possam ser alcançadas por associações ou dependências. Esses vizinhos são prováveis alvos de ações e são candidatos à inclusão de condições de guarda.
  - Se o contexto é o sistema como um todo, restrinja seu foco a um único comportamento do sistema. Teoricamente, todos os objetos existentes no sistema poderão ser um participante em um modelo do tempo de vida do sistema e, excetuando os sistemas mais triviais, um modelo completo seria inviável.
- › Estabeleça os estados inicial e final para o objeto. Para orientar o restante de seu modelo, possivelmente defina as pré e pós-condições dos estados inicial e final, respectivamente.
- › Decida a respeito dos eventos aos quais esse objeto poderá responder. Caso já tenham sido especificados, você os encontrará nas interfaces do objeto; caso ainda não tenham sido especificados, será necessário considerar quais objetos poderão interagir com o objeto em seu contexto e depois quais eventos eles serão capazes de possivelmente ativar.
- › A partir do estado inicial até o estado final, distribua os estados de nível superior em que o objeto poderá estar. Conecte esses estados com transições ativadas pelos eventos apropriados. Continue adicionando ações a essas transições.
- › Identifique qualquer entrada ou ação de saída (especialmente se você concluir que o idioma por elas coberto é utilizado na máquina de estados).

- › Expanda esses estados, conforme seja necessário, utilizando subestados.
- › Verifique se todos os eventos mencionados na máquina de estados correspondem a eventos esperados pela interface do objeto. De modo semelhante, verifique se todos os eventos esperados pela interface do objeto são manipulados pela máquina de estados. Por fim, observe os locais em que você deseja ignorar eventos explicitamente.
- › Verifique se todas as ações mencionadas na máquina de estados são sustentadas pelos relacionamentos, métodos e operações do objeto que os contém.
- › Faça o acompanhamento da máquina de estados, manualmente ou utilizando ferramentas, para verificá-la em relação às sequências de eventos esperadas e suas respostas. Seja especialmente diligente ao procurar estados que não podem ser alcançados, nos quais a máquina poderá travar.
- › Após reorganizar sua máquina de estados, verifique-a mais uma vez em relação às sequências esperadas para assegurar que não tenha modificado a semântica do objeto.

Por exemplo, a [Figura 22.9](#) mostra a máquina de estados para o controlador de um sistema de segurança doméstico, responsável pela monitoração de vários sensores ao redor do perímetro da casa.



**Figura 22.9:**

A modelagem do tempo de vida de um objeto

No tempo de vida da classe desse controlador, existem quatro estados principais: *Initializing* (o controlador está iniciando), *Idle* (o controlador está pronto e aguardando alarmes ou comando do usuário), *Command* (o controlador está processando comandos do usuário) e *Active* (o controlador está processando uma condição de alarme). Quando o objeto controlador é criado pela primeira vez, ele primeiro passa ao *Initializing* e depois, incondicionalmente, ao estado *Idle*. Os detalhes desses dois estados não são mostrados, com exceção da autotransição com o evento de tempo no estado *Idle*. Esse tipo de evento de tempo é comum em sistemas embutidos, que costumam dispor de um cronômetro central para fazer uma verificação periódica da saúde do sistema.

O controle passa do estado *Idle* para o estado *Active* no destinatário de um evento *alarm* (que inclui o parâmetro *s*, identificando o sensor que foi acionado). Ao entrar no estado *Active*, *setAlarm* é ativado como a ação de entrada e o controle então passa primeiro ao estado *Checking* (validando o alarme), depois ao estado *Calling* (chamando a empresa do alarme para registrar a ocorrência) e, por fim, ao estado *Waiting*. Os estados *Active* e *Waiting*

são abandonados somente após `clear` no alarme ou por uma sinalização do usuário ao controlador para `attention`, presumivelmente para emitir um comando.

Observe que não existe um estado final. Isso também é comum em sistemas embutidos, que são projetados para serem executados continuamente.

## DICAS E SUGESTÕES

Ao fazer a modelagem de uma máquina de estados na UML, lembre-se de que toda máquina de estados representa os aspectos dinâmicos de um objeto individual, tipicamente representando uma instância de uma classe, um caso de uso ou o sistema como um todo. Uma máquina de estados bem estruturada:

- › É simples e, portanto, não deverá conter estados ou transições supérfluos.
- › Tem um contexto claro e, portanto, poderá ter acesso a todos os objetos visíveis ao objeto que a contém (esses vizinhos deverão ser utilizados somente conforme seja necessário para a execução do comportamento especificado pela máquina de estados).
- › É eficiente e, portanto, deverá executar seu comportamento com ótimo equilíbrio de tempo e recursos, conforme seja requerido pelas ações que ela ativa.
- › É compreensível e, portanto, deverá nomear seus estados e transições a partir do vocabulário do sistema.
- › Não é aninhada muito profundamente (o aninhamento de subestados em um ou dois níveis manipulará os comportamentos mais complexos).
- › Utiliza subestados concorrentes com cautela, já que o uso de classes ativas costuma ser uma alternativa melhor.

- ➡ A modelagem do vocabulário de um sistema é examinada no [Capítulo 4](#).

Ao definir uma máquina de estados na UML:

- › Evite transições que se cruzem.
- › Expanda estados compostos no local, somente conforme seja necessário para fazer com que o diagrama seja comprehensível.

CAPÍTULO

---

23

---

# Processos e Threads

## Neste capítulo

- » *Objetos ativos, processos e threads*
- » *Modelagem de vários fluxos de controle*
- » *Modelagem de comunicações entre processos*
- » *Construção de abstrações de thread seguras*

O mundo real não é apenas um lugar rude e implacável, mas é também um lugar muito ocupado. Eventos podem acontecer e várias coisas podem ocorrer ao mesmo tempo. Portanto, ao fazer a modelagem de um sistema do mundo real, é necessário levar em conta sua visão de processo, abrangendo os threads e processos que formam os mecanismos de concorrência e sincronização do sistema.

- ➡ As visões de interação no contexto da arquitetura de software são examinadas no [Capítulo 2](#).

Na UML, cada fluxo de controle independente é modelado como um objeto ativo que representa um processo ou thread capaz de iniciar a atividade de controle. Um processo é, tipicamente, o ambiente de um programa em execução, com seus recursos e espaço de memória. Uma thread é, essencialmente, um processo mais leve, que compartilha recursos e memória com outras threads dentro do ambiente de execução de um programa. Cada processo tem ao menos uma thread. Processos e threads podem ser executados de forma concorrente, com a diferença de que um processo pode ser executado concorrentemente a outro processo e uma thread apenas pode

ser executada concorrentemente a outras threads dentro do mesmo processo ao qual elas pertencem.

É difícil construir abstrações de modo que funcionem de maneira segura na presença de vários fluxos de controle. Em particular, é preciso considerar como abordar a comunicação e a sincronização que são mais complexas do que sistemas sequenciais. Também é necessário ser cuidadoso para não haver um excesso de engenharia em sua visão de processo (muitos fluxos concorrentes e o sistema acaba falhando), nem uma engenharia escassa (uma concorrência insuficiente não otimiza tempo de resposta do sistema).

## PRIMEIROS PASSOS

Na vida de um cachorro e sua casa, o mundo é um local bem simples e sequencial. Comer. Dormir. Perseguir uns gatos. Comer mais alguma coisa. Sonhar com perseguições aos gatos. Usar a casa para dormir ou como abrigo para a chuva nunca é um problema, pois o cachorro, e somente o cachorro, precisa entrar e sair pela porta de sua casa de cachorro. Nunca existe contenção de recursos.

► *A modelagem de casas de cachorro e de altos prédios é examinada no [Capítulo 1](#).*

Na vida de uma família e sua casa, o mundo não é tão simples. Sem ser muito metafísico, cada membro da família vive sua própria vida, mas interage com os demais membros da família (no jantar, vendo televisão, participando de jogos, fazendo faxina). Os membros da família compartilham determinados recursos. As crianças poderão dividir um mesmo quarto; toda a família poderá usar um único aparelho de telefone ou um mesmo computador. Os membros da família também compartilham deveres. O pai vai à lavanderia e ao supermercado; a mãe paga as contas e limpa o quintal; as crianças ajudam na limpeza e na cozinha. Poderá ser um desafio a contenção entre esses recursos compartilhados e a coordenação desses

deveres independentes. Compartilhar um único banheiro, quando todo mundo está se aprontando para sair para a escola ou para o trabalho, poderá ser problemático; o jantar não será servido, enquanto o pai não voltar do supermercado.

Na vida de um alto prédio e seus ocupantes, o mundo é realmente complexo. Centenas, se não milhares, de pessoas poderão trabalhar em um mesmo prédio, cada uma seguindo sua própria agenda. Todas precisam passar por um conjunto limitado de entradas. Todas usam os mesmos elevadores. Todas compartilham as mesmas facilidades de aquecimento, refrigeração, água, eletricidade, sanitários e estacionamento. Para que tudo isso funcione em conjunto de maneira harmoniosa, é necessário que todos se comuniquem e sincronizem suas interações apropriadamente.

Na UML, cada fluxo de controle independente é modelado como um objeto ativo. Um objeto ativo é um processo ou thread capaz de iniciar uma atividade de controle. Assim como qualquer tipo de objeto, um objeto ativo é uma instância de uma classe. Nesse caso, o objeto ativo é uma instância de uma classe ativa. Também como qualquer tipo de objeto, os objetos ativos podem se comunicar uns com os outros pela passagem de mensagens, embora aqui a passagem de mensagens deva ser estendida com certa concorrência semântica, para ajudá-lo a sincronizar as interações entre fluxos independentes.

► Os objetos são examinados no [Capítulo 13](#).

Em software, muitas linguagens de programação têm suporte direto para o conceito de objeto ativo. Java, Smalltalk e Ada têm a concorrência incluída em suas implementações. O C++ tem suporte para a concorrência a partir de várias bibliotecas que constroem mecanismos de concorrência em um sistema operacional hospedeiro. A utilização da UML para visualizar, especificar, construir e documentar essas abstrações é importante; caso contrário, é quase impossível pensar sobre questões de concorrência, comunicação e sincronização.

A UML fornece uma representação gráfica de uma classe ativa, conforme mostra a [Figura 23.1](#). As classes ativas são tipos de classes e, portanto, têm todos os compartimentos usuais para nome, atributos e operações da classe. As classes ativas costumam receber sinais, que tipicamente são enumerados em um compartimento extra.

- As classes são examinadas nos Capítulos 4 e 9; os sinais são apresentados no [Capítulo 21](#).



**Figura 23.1:**  
Classe ativa<sup>4</sup>

## TERMOS E CONCEITOS

Um *objeto ativo* é um objeto que possui um processo ou thread e é capaz de iniciar uma atividade de controle. Uma *classe ativa* é uma classe cujas instâncias são objetos ativos.

Um processo é, tipicamente, o ambiente de um programa em execução, com seus recursos e espaço de memória. Uma thread é, essencialmente, um processo mais leve, que compartilha recursos e memória com outras threads dentro do ambiente de execução de um programa. Cada processo tem ao menos uma thread. Processos e threads podem ser executados de forma concorrente, com a diferença de que um processo pode ser executado concorrentemente a outro processo e uma thread apenas pode ser executada concorrentemente a outras threads dentro do mesmo processo ao qual elas pertencem.

Graficamente, uma classe ativa é representada como um retângulo com linhas grossas. Os processos e threads são representados como classes ativas

estereotipadas (e também aparecem como sequências em diagramas de interação).

- *Diagrams de interação são discutidos no [Capítulo 19](#).*

## FLUXO DE CONTROLE

Em um sistema puramente sequencial, existe um único fluxo de controle. Isso significa que uma coisa, e somente uma coisa, pode ocorrer por vez. Quando um programa sequencial é iniciado, o controle tem como raiz o começo do programa e as operações são ativadas uma depois da outra. Mesmo que existam coisas concorrentes acontecendo entre os atores externos ao sistema, o programa sequencial processará somente um evento por vez, colocando em fila ou descartando quaisquer eventos externos concorrentes.

- *Os atores são apresentados no [Capítulo 17](#).*

Essa é a razão pela qual isso é chamado de um fluxo de controle. Se você acompanhar a execução de um programa sequencial, verá o local da execução fluir de uma instrução para outra, em ordem sequencial. Você poderá ver ações que se ramificam, fazem loops e saltam de uma parte para outra e, caso haja alguma recursão ou iteração, verá o círculo do fluxo retornar para si próprio. Entretanto, em um sistema sequencial, haverá um único fluxo de execução.

- *As ações são examinadas no [Capítulo 16](#).*

Em um sistema concorrente, existe mais de um fluxo de controle – ou seja, mais de uma coisa pode ocorrer ao mesmo tempo. Em um sistema concorrente, existem vários fluxos de controle simultâneos, cada um com sua raiz no princípio de um processo ou thread independente. Se tirar uma foto de um sistema concorrente enquanto ele estiver em execução, você verá logicamente vários locais de execução.

Na UML, as classes ativas são utilizadas para representar um processo ou thread que é a raiz de um fluxo de controle independente e que é concorrente com todos os fluxos de controle irmãos.

- Os nós são examinados no [Capítulo 27](#).

**Nota:** A verdadeira concorrência pode ser obtida de uma de três maneiras: primeiro, distribuindo os objetos ativos em vários nós; segundo, colocando os objetos ativos em nós com vários processadores; e terceiro, combinando os dois métodos anteriores.

## CLASSE E EVENTOS

As classes ativas são apenas classes, mas com uma propriedade muito especial. Uma classe ativa representa um fluxo de controle independente, enquanto uma classe plena não incorpora esse fluxo. Em contraste com as classes ativas, as classes plenas são implicitamente chamadas passivas, pois não são capazes de iniciar uma atividade de controle independentemente.

- As classes são examinadas nos [Capítulos 4 e 9](#).

As classes ativas são utilizadas para a modelagem de famílias comuns de processos ou threads. Em termos técnicos, isso significa que um objeto ativo – uma instância de uma classe ativa – reifica (é a manifestação de) um processo ou thread. Com a modelagem de sistemas concorrentes a partir de objetos ativos, um nome é atribuído a cada fluxo de controle independente. Quando um objeto ativo é criado, o fluxo de controle a ele associado é iniciado; quando o objeto ativo é destruído, o fluxo de controle a ele associado é terminado.

- Os objetos são examinados no [Capítulo 13](#); os atributos e operações são apresentados no [Capítulo 4](#); os relacionamentos aparecem nos [Capítulos 4 e 10](#); os mecanismos de extensibilidade são explicados no [Capítulo 6](#); as interfaces são examinadas no [Capítulo 11](#).

As classes ativas compartilham as mesmas propriedades de todas as outras classes. As classes ativas podem ter instâncias, atributos e operações. As classes ativas podem participar de relacionamentos de dependência, generalização e associação (incluindo a agregação). As classes ativas podem fazer uso de qualquer mecanismo de extensibilidade da UML, incluindo estereótipos, valores atribuídos e restrições. As classes ativas podem ser a realização de interfaces. As classes ativas podem ser realizadas por colaborações e o comportamento de uma classe ativa poderá ser especificado pela utilização de máquinas de estados. Classes ativas podem participar em colaborações.

Em seus diagramas, os objetos ativos poderão aparecer sempre que houver objetos passivos. Você pode fazer a modelagem da colaboração de objetos ativos e passivos, utilizando diagramas de interação (incluindo diagramas de colaboração e de sequências). Um objeto ativo poderá aparecer como o destino de um evento em uma máquina de estados.

- As máquinas de estados são examinadas no [Capítulo 22](#); os eventos são apresentados no [Capítulo 21](#).

Em relação às máquinas de estados, tanto os objetos ativos como os passivos podem enviar e receber eventos de sinal e eventos de chamada.

**Nota:** O uso de classes ativas é opcional. Elas não adicionam realmente muito à semântica.

## COMUNICAÇÃO

Quando os objetos colaboram uns com os outros, eles interagem pela passagem de mensagens de um para o outro. Em um sistema com objetos ativos e passivos, existem quatro combinações possíveis de interação que você deve levar em consideração.

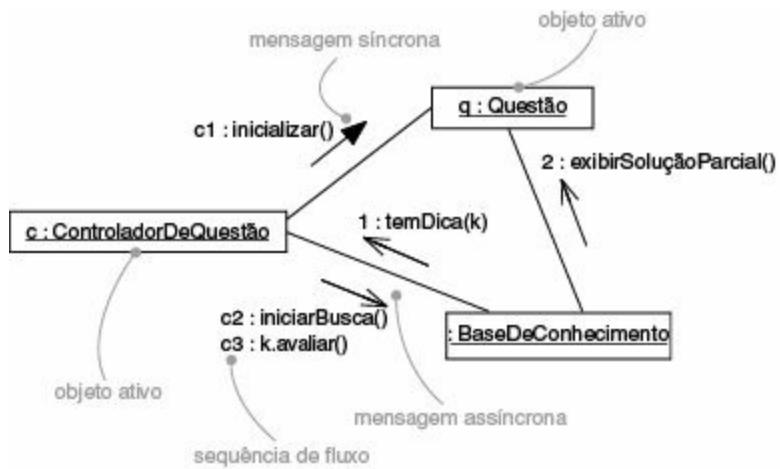
- As interações são examinadas no [Capítulo 16](#).

Primeiro, uma mensagem poderá ser enviada de um objeto passivo para outro. Assumindo que existe somente um fluxo de controle passando por esses objetos por vez, essa interação nada mais é do que a invocação simples de uma operação.

Segundo, uma mensagem poderá ser passada de um objeto ativo para outro. Quando isso acontece, ocorre uma comunicação entre processos e existem dois estilos possíveis de comunicação. Primeiro, um objeto ativo poderá chamar uma operação de outro objeto. Esse tipo de comunicação tem a semântica de um *rendez-vous*, significando que o objeto chama a operação; aguarda que o destinatário aceite a chamada; a operação é invocada; um objeto de retorno (se houver) é passado de volta ao objeto que fez a chamada; e então os dois continuam em seus caminhos independentes. Durante a chamada, os dois fluxos de controle ficam em passo de espera. Segundo, um objeto ativo poderá assincronamente enviar um sinal ou uma chamada a uma operação de outro objeto. Esse tipo de comunicação tem uma semântica de caixa postal, significando que o objeto que fez a chamada envia o sinal ou chama a operação e depois continua seu caminho independente. Enquanto isso, o destinatário aceita o sinal ou chamada sempre que estiver pronto (com eventos intervenientes ou chamadas colocadas em uma fila) e prossegue seu caminho após concluir o que foi solicitado. Isso é chamado de caixa postal, porque os dois objetos não estão sincronizados; em vez disso, um objeto envia uma mensagem para o outro.

- Os eventos de sinal e os eventos de chamada são examinados no [Capítulo 21](#).

Na UML, as mensagens síncronas são representadas por uma seta cheia e as mensagens assíncronas como meia seta, conforme mostra a [Figura 23.2](#).



**Figura 23.2:**  
Comunicação

Terceiro, uma mensagem poderá ser passada de um objeto ativo para um objeto passivo. Uma dificuldade ocorrerá, caso mais de um objeto a cada tempo passe seu fluxo de controle por um objeto passivo. Nessa situação, é preciso fazer a modelagem da sincronização desses dois fluxos com muito cuidado, conforme é examinado na próxima seção.

Quarto, uma mensagem poderá ser passada de um objeto passivo para outro ativo. À primeira vista, isso poderá parecer ilegal, mas, se você lembrar que todo fluxo de controle tem como raiz algum objeto ativo, compreenderá que um objeto passivo enviando uma mensagem a um objeto ativo tem a mesma semântica que um objeto ativo enviando uma mensagem para um objeto ativo.

► As restrições são examinadas no [Capítulo 6](#).

**Nota:** É possível fazer a modelagem da passagem de variações de mensagens síncronas e assíncronas, utilizando as restrições. Por exemplo, para fazer a modelagem de rendez-vous, como aqueles encontrados na linguagem Ada, você usaria uma mensagem síncrona com uma restrição como `{wait = 0}`, dizendo que o objeto que faz a chamada não aguardará o destinatário. De modo semelhante, é possível a modelagem de tempo esgotado, utilizando uma restrição como `{wait = 1 ms}`, dizendo que o objeto que faz a chamada não aguardará mais de um milissegundo para o destinatário aceitar a mensagem.

## SINCRONIZAÇÃO

Visualize por um momento os vários fluxos de controle que convivem em um sistema concorrente. Quando um fluxo passa por uma operação, diz-se que, em determinado momento, o local do controle está na operação. Se essa operação estiver definida para alguma classe, também se pode dizer que, em determinado momento, o local do controle está em uma instância específica dessa classe. Pode haver vários fluxos de controle em uma mesma operação (e, portanto, em um único objeto), e pode haver diferentes fluxos de controle em diferentes operações (mas ainda resulta em vários fluxos de controle em um único objeto).

O problema surge quando mais de um fluxo de controle está em um único objeto simultaneamente. Se você não tiver cuidado, mais de um fluxo pode modificar o mesmo atributo, corrompendo o estado do objeto ou ocasionando perda de informação. Esse é o problema clássico de exclusão mútua. Um erro ao lidar com esse problema de maneira adequada causará todos os tipos de interferências e de condições de conflito que fazem com que os sistemas concorrentes apresentem falhas de formas misteriosas e imprevisíveis.

A chave para solucionar esse problema em sistemas orientados a objetos consiste em tratar um objeto como uma região crítica. Existem três alternativas para essa abordagem, cada uma das quais envolve a inclusão de certas propriedades de sincronização às operações definidas em uma classe. Na UML, você pode fazer a modelagem de todas as três soluções.

### 1. Sequencial

Quem faz a chamada precisa coordenar externamente o objeto, de maneira que somente um fluxo exista no objeto por vez. Na presença de vários fluxos de controle, a semântica e a integridade do objeto não podem ser garantidas.

### 2. Protegida

A semântica e a integridade do objeto são garantidas na presença de vários fluxos de controle pela sequencialização de todas as

chamadas para todas as operações protegidas do objeto. Como efeito, exatamente uma operação por vez pode ser invocada ao objeto, reduzindo isso a uma semântica sequencial. Há um perigo de impasse caso não se tome cuidado.

### 3. Concorrente

A semântica e a integridade do objeto são garantidas na presença de vários fluxos de controle, porque vários fluxos de controle acessam conjuntos de dados disjuntos ou somente dados de leitura. Essa situação pode ser organizada por regras de projeto cuidadosas.

Algumas linguagens de programação têm suporte para essas construções diretamente. Java, por exemplo, tem a propriedade `synchronized`, que é equivalente à propriedade `concurrent` da UML. Em toda linguagem com suporte à concorrência, você pode construir suporte para todas essas propriedades, construindo-as como semáforos.

Conforme mostra a [Figura 23.3](#), essas propriedades podem ser anexadas a uma operação, representada na UML utilizando a notação de restrição. Observe que a concorrência deve ser declarada separadamente para cada operação e para o objeto inteiro. Declarar a concorrência para uma operação significa que várias chamadas dessa operação podem ser executadas ao mesmo tempo sem perigo. Declarar a concorrência para um objeto significa que as chamadas de diferentes operações podem ser executadas concurrentemente sem perigo; essa é uma condição mais rígida.

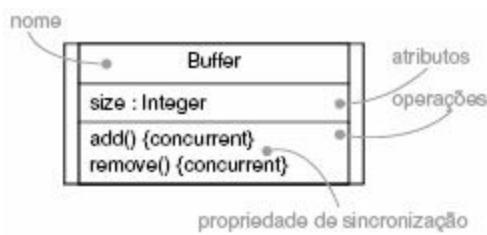


Figura 23.3:

## Sincronização

- As restrições são examinadas no [Capítulo 6](#).

**Nota:** É possível fazer a modelagem das variações dessas primitivas de sincronização, utilizando as restrições. Por exemplo, você poderá modificar a propriedade concurrent, permitindo vários leitores simultâneos, mas somente um escritor.

# TÉCNICAS BÁSICAS DE MODELAGEM

## MODELAGEM DE VÁRIOS FLUXOS DE CONTROLE

É difícil a construção de um sistema abrangendo vários fluxos de controle. Não é só necessário decidir sobre a melhor maneira de dividir o trabalho pelos objetos ativos concorrentes, mas, após fazer isso, também é preciso criar os mecanismos corretos para comunicação e sincronização dos objetos ativos e passivos do sistema, assegurando que se comportarão adequadamente na presença de vários fluxos. Por essa razão, é de grande ajuda visualizar a forma como esses fluxos interagem uns com os outros. Você poderá fazê-lo na UML, aplicando diagramas de classes (para capturar sua semântica estática) e diagramas de interação (para capturar sua semântica dinâmica) contendo objetos e classes ativas.

- Os mecanismos são examinados no [Capítulo 29](#); os diagramas de classes são apresentados no [Capítulo 8](#); os diagramas de interação são explicados no [Capítulo 19](#).

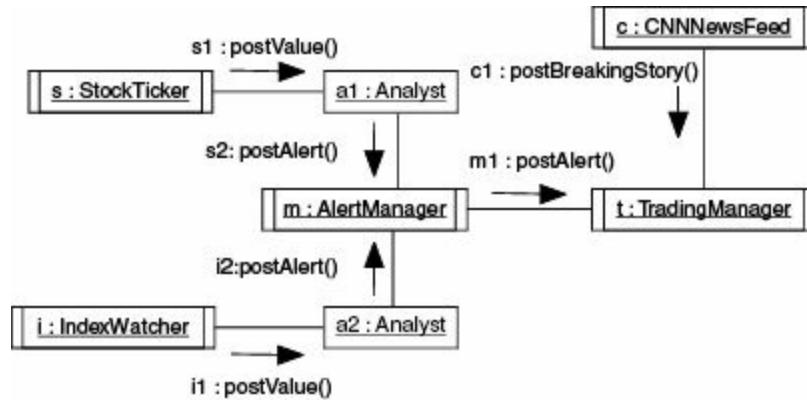
Para fazer a modelagem de vários fluxos de controle:

- » Identifique oportunidades para ação concorrente e verifique cada fluxo como uma classe ativa. Generalize conjuntos comuns de objetos ativos em uma única classe ativa. Tenha cuidado para não exagerar a engenharia da visão de processo do sistema, introduzindo muita concorrência.

- › Considere uma distribuição equilibrada de responsabilidades entre essas classes ativas e depois examine as demais classes ativas e passivas com as quais cada uma colabora estaticamente. Assegure que cada classe ativa seja altamente coesiva e levemente acoplada em relação a essas classes vizinhas e que cada uma tenha o conjunto correto de atributos, operações e sinais.
  - › Capture essas decisões estáticas em diagramas de classes, destacando cada classe ativa explicitamente.
  - › Considere como cada grupo de classes colabora com outro de forma dinâmica. Capture essas decisões em diagramas de interação. Mostre explicitamente os objetos ativos como a raiz desses fluxos. Identifique cada sequência relacionada, atribuindo-lhe o nome do objeto ativo.
  - › Preste bastante atenção à comunicação existente entre os objetos ativos. Aplique mensagens síncronas e assíncronas, conforme seja apropriado.
  - › Preste bastante atenção à sincronização entre esses objetos ativos e os objetos passivos com os quais colaboram. Aplique uma semântica de operação sequencial, protegida ou concorrente, conforme seja apropriado.
- *Visões de processo são discutidas no Capítulo 19; classes são discutidas nos Capítulos 4 e 9; relacionamentos são discutidos nos Capítulos 5 e 10.*

Por exemplo, a [Figura 23.4](#) mostra parte da visão de processo de um sistema de negociação de ações. Existem três objetos que colocam informações no sistema de forma concorrente: StockTicker, IndexWatcher e CNNNewsFeed (Valor da Ação, Mostrador de Índice e o Canal de Notícias da CNN, nomeados como `s`, `i` e `c`, respectivamente). Dois desses objetos (`s` e `i`) se comunicam com suas próprias instâncias de modelo de analista de mercado, Analyst (`a1` e `a2`). Ao menos no escopo desse modelo, o Analyst pode ser projetado sob a consideração simplista de que somente um fluxo de controle estará

ativo em suas instâncias de cada vez. As duas instâncias de Analyst, entretanto, se comunicam simultaneamente com um AlertManager (Gestor de Alerta, nomeado como  $m$ ). Portanto,  $m$  deve ser projetado para preservar sua semântica na presença de vários fluxos. Tanto  $m$  quanto  $c$  se comunicam simultaneamente com  $t$ , um TradingManager (Gerente de Negociação). Cada fluxo recebe um número sequencial que é diferenciado pelo fluxo de controle ao qual pertence.



**Figura 23.4:**

A modelagem de fluxos de controle

**Nota:** Diagramas de interação como esse são úteis para ajudar a visualizar onde dois fluxos de controle poderão ter caminhos cruzados e, portanto, onde se deve prestar uma atenção especial a problemas de comunicação e de sincronização. São permitidas ferramentas para oferecer indicações visuais ainda mais distintivas, como colorir cada fluxo de uma maneira diferente.

Em diagramas como esse, também é comum anexar as máquinas de estados correspondentes, com estados ortogonais mostrando o comportamento detalhado de cada objeto ativo.

► As máquinas de estados são examinadas no [Capítulo 22](#).

## MODELAGEM DE COMUNICAÇÃO ENTRE PROCESSOS

Como uma parte da incorporação de vários fluxos de controle em seu sistema, você também precisará levar em consideração os mecanismos pelos quais os objetos que vivem em fluxos separados se comunicam uns com os

outros. Com as threads (que vivem no mesmo espaço de endereço), os objetos podem se comunicar via sinais ou eventos de chamada, os últimos sendo capazes de exibir uma semântica assíncrona ou síncrona. Com os processos (que vivem em espaços de endereço separados), geralmente é necessário utilizar mecanismos diferentes.

- » *Os sinais e os eventos de chamada são examinados no [Capítulo 21](#).*

O problema da comunicação entre processos se deve ao fato de, em sistemas distribuídos, os processos poderem viver em nós separados. Classicamente, existem duas soluções para a comunicação entre processos: passagem de mensagem e chamada remota a procedimentos (RPC). Na UML, ainda é possível fazer a modelagem desses eventos assíncronos ou síncronos, respectivamente. Entretanto, por não serem mais simples que as chamadas de processos, é necessário incluir adornos em seus projetos com mais informações.

- » *A modelagem da locação é examinada no [Capítulo 24](#).*

Para fazer a modelagem da comunicação entre processos:

- » Faça a modelagem de vários fluxos de controle.
- » Faça a modelagem das mensagens, utilizando a comunicação assíncrona; modele as chamadas remotas a procedimentos, utilizando a comunicação síncrona.
- » Especifique informalmente o mecanismo subjacente para a comunicação, utilizando notas ou, mais formalmente, colaborações.

- » *Os estereótipos são examinados no [Capítulo 6](#); as notas são apresentadas no [Capítulo 6](#); as colaborações são explicadas no [Capítulo 28](#); os nós aparecem no [Capítulo 27](#).*

A [Figura 23.5](#) mostra um sistema de reservas distribuído, contendo processos espalhados por quatro nós. Cada objeto é marcado pelo estereótipo

process. Cada objeto também é assinalado pelo valor atribuído location, que especifica sua localização física. A comunicação entre ReservationAgent, TicketingManager e HotelAgent (respectivamente Agente de Reserva, Gerente de Cobrança e Agente de Hotel) é assíncrona. A nota no modelo descreve a comunicação como a construção de um serviço de mensagem de Java Beans. A comunicação entre TripPlanner e ReservationSystem (respectivamente Agente de Viagem e Sistema de Reserva) é síncrona. A semântica de sua interação se encontra na colaboração nomeada como CORBA ORB. A TripPlanner atua como cliente e ReservationAgent como servidor. Observando a colaboração, você encontrará os detalhes de como colaboram esse servidor e cliente.

## DICAS E SUGESTÕES

Classes ativas e objetos ativos bem estruturados:

- Representam um fluxo de controle independente, maximizando o potencial para uma verdadeira concorrência no sistema.

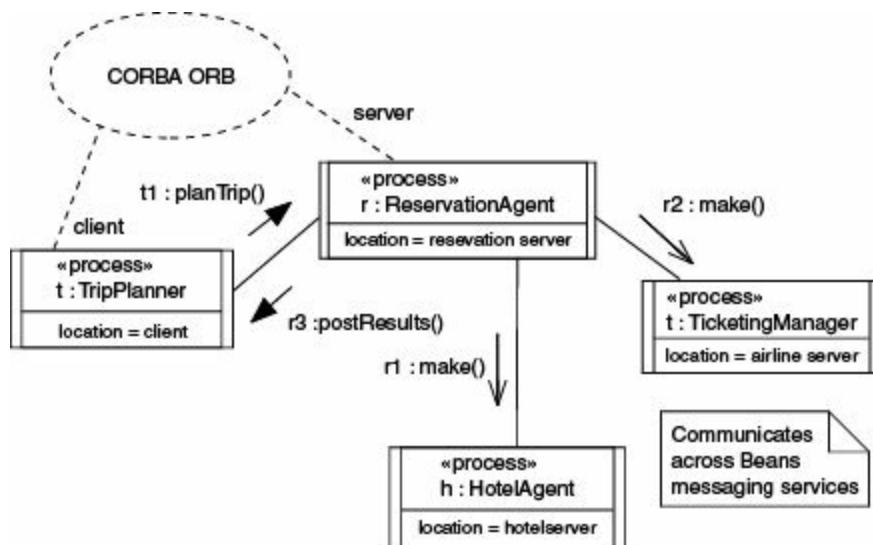


Figura 23.5:

A modelagem da comunicação entre processos

- Não são tão pequenos que exijam vários outros elementos ativos, o que poderá resultar em excesso de engenharia e em uma arquitetura de

processo frágil.

- › Gerenciam cuidadosamente a comunicação entre pares de elementos ativos, escolhendo entre mensagens assíncronas e síncronas.
- › Tratam cuidadosamente cada objeto como uma região crítica, utilizando propriedades de sincronização ajustáveis, para preservar sua semântica na presença de vários fluxos de controle.

Ao definir uma classe ativa ou um objeto ativo na UML:

- › Mostre somente os atributos, operações e sinais importantes para a compreensão da abstração em seu contexto e oculte os outros, usando uma capacidade de filtragem, se a ferramenta de modelagem permitir.
- › Mostre explicitamente todas as propriedades de sincronização de operações.

---

<sup>4</sup> Esta classe ativa representada na [Figura 23.1](#) é uma simplificação do padrão de arquitetura Blackboard, que será retomado no [capítulo 29](#).

CAPÍTULO

---

**24**

---

# Tempo e Espaço

## Neste capítulo

- » *Tempo, duração e localização*
- » *Modelagem de restrições de tempo*
- » *Modelagem da distribuição de objetos*
- » *Modelagem de objetos que migram*
- » *Lidando com sistemas distribuídos e em tempo real*

O mundo real é um lugar rude e implacável. Os eventos podem acontecer em momentos imprevisíveis, mas exigem respostas específicas em instantes específicos. Poderá ser necessário distribuir os recursos de um sistema pelo mundo – e alguns desses recursos poderão até ser movidos – levantando questões de latência, sincronização, segurança e qualidade do serviço.

A modelagem de tempo e espaço é um elemento essencial de qualquer sistema distribuído e/ou em tempo real. Várias características da UML incluem marcas de tempos, expressões de tempo, restrições e valores atribuídos, para visualizar, especificar, construir e documentar esses sistemas.

É difícil lidar com sistemas distribuídos e em tempo real. Bons modelos sempre revelam as propriedades necessárias e suficientes das características de tempo e espaço de um sistema.

## PRIMEIROS PASSOS

Ao começar a fazer a modelagem da maioria de sistemas de software, em geral você pode assumir um ambiente sem atritos – as mensagens são enviadas em tempo zero, as redes nunca caem, as estações de trabalho nunca

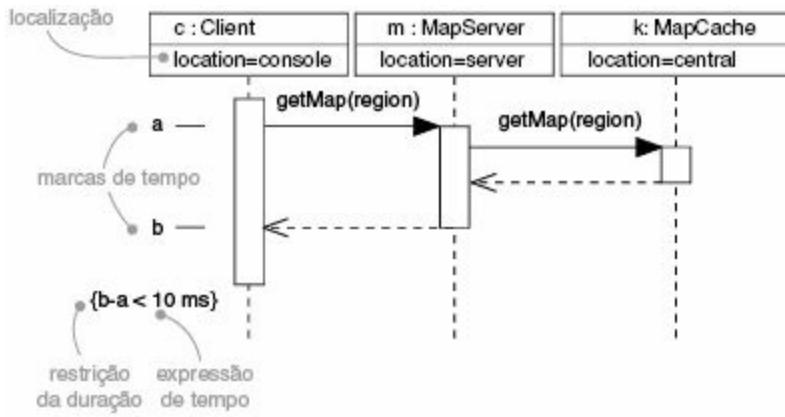
falham, a carga na rede sempre está equilibrada. Infelizmente, o mundo real não funciona dessa maneira – as mensagens demoram a ser enviadas (e, às vezes, nunca chegam a ser enviadas), as redes caem, as estações de trabalho falham e a carga da rede geralmente está desequilibrada. Portanto, quando você encontra sistemas que precisam funcionar no mundo real, é preciso levar em consideração as questões de tempo e espaço.

Um sistema em tempo real é aquele em que certo comportamento deve ser executado em um tempo preciso, absoluto ou relativo, e dentro de uma duração previsível e quase sempre restrita. Em extremo, esses sistemas poderão de fato ser em tempo real, exigindo um comportamento completo e repetitivo em nanossegundos ou milissegundos. No outro extremo, os modelos poderão estar próximos do tempo real e também exigir um comportamento previsível, mas na ordem de segundos ou mais.

Um sistema distribuído é aquele em que os componentes podem estar distribuídos fisicamente pelos nós. Esses nós podem representar diferentes processadores localizados fisicamente em uma mesma caixa ou podem até representar computadores localizados a meio mundo de distância um do outro.

- ➡ Os componentes são examinados no [Capítulo 26](#); os nós são apresentados no [Capítulo 27](#).

Para representar as necessidades de modelagem de sistemas distribuídos e em tempo real, a UML oferece uma representação gráfica para marcas de tempo, expressões de tempo, restrições de tempo e localização, conforme mostra a [Figura 24.1](#).



**Figura 24.1:**

Restrições de tempo e localização

## TERMOS E CONCEITOS

Uma *marca de tempo* é a denotação do tempo em que um evento ocorre. Graficamente, uma marca de tempo é formada como uma expressão a partir do nome dado à mensagem (que tipicamente é diferente do nome da ação ativada pela mensagem). Uma *expressão de tempo* é uma expressão calculada como um valor de tempo absoluto ou relativo. Uma expressão de tempo também pode ser formada usando o nome de uma mensagem e um indicador de um estágio em seu processamento, por exemplo, tempos de envio ou de recepção de uma requisição, `request.sendTime` OU `request.receiveTime`. Uma *restrição de tempo* é uma declaração semântica sobre o valor de tempo relativo ou absoluto. Graficamente, uma restrição de tempo é representada como qualquer outra restrição – ou seja, uma sequência de caracteres entre colchetes e geralmente conectada a um elemento por um relacionamento de dependência. A *localização* é o local de um componente em um nó. A localização é definida pelo atributo `location` de um objeto.

## TEMPO

Os sistemas em tempo real são, pelo seu próprio nome, sistemas críticos em relação ao tempo. Eventos podem acontecer em momentos regulares ou irregulares; a resposta a um evento deve ocorrer em momentos absolutos

previsíveis ou em momentos relativos previsíveis em relação ao próprio evento.

- Os eventos, inclusive os eventos de tempo, são examinados no [Capítulo 21](#); as mensagens e interações são apresentadas no [Capítulo 16](#); as restrições são explicadas no [Capítulo 6](#).

A passagem de mensagens representa o aspecto dinâmico de qualquer sistema; portanto, quando você faz a modelagem da natureza crítica de tempo de um sistema com a UML, pode dar um nome a cada mensagem existente em uma interação, para ser utilizado como marca de tempo. As mensagens em uma interação costumam não ter nomes. São representadas principalmente pelo nome de um evento, como um sinal ou uma chamada. Como resultado, você não pode usar o nome do evento para escrever uma expressão, pois o mesmo evento poderá ativar mensagens diferentes. Se a mensagem designada for ambígua, use o nome explícito da mensagem em uma marca de tempo para designar a mensagem que você deseja mencionar em uma expressão de tempo. A marca de tempo nada mais é do que uma expressão formada a partir do nome de uma mensagem em uma interação. Dado um nome de mensagem, é possível fazer referência a uma de três funções relativas ao tempo dessa mensagem, seu tempo de envio, recepção e transmissão – ou seja, `sendTime`, `receiveTime` e `transmissionTime`. (Essas são nossas funções sugeridas, não oficiais da UML. Um sistema em tempo real pode ter mais funções.) Para chamadas síncronas, você também pode referenciar o tempo de ciclo completo de uma mensagem com `executionTime` (mais uma vez, nossa sugestão). Essas funções podem ser empregadas para especificar arbitrariamente expressões de tempo complexas, talvez até usar pesos ou medidas que são constantes ou variáveis (desde que essas variáveis possam ser vinculadas em tempo de execução). Por fim, conforme mostra a [Figura 24.2](#), essas expressões de tempo podem ser colocadas em uma restrição de tempo para especificar o comportamento de tempo do sistema. As restrições podem ser representadas, sendo colocadas adjacentes à mensagem apropriada

ou explicitamente anexas com a utilização de relacionamentos de dependência.

**Nota:** Especialmente no caso de sistemas complexos, é uma boa ideia escrever expressões com constantes nomeadas em vez de escrever tempos explícitos. Você pode definir essas restrições em uma parte de seu modelo e depois referenciar essas constantes em vários locais. Dessa maneira, é mais fácil atualizar o modelo, caso os requisitos de tempo do sistema sejam alterados.

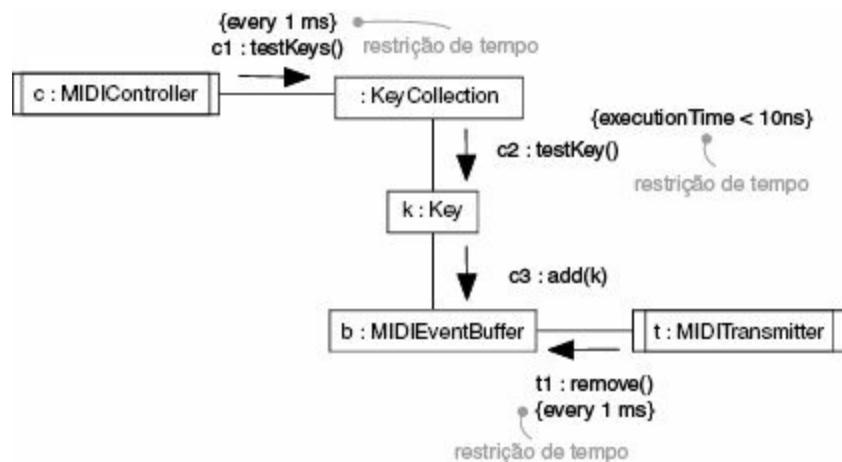


Figura 24.2:

Tempo

## LOCALIZAÇÃO

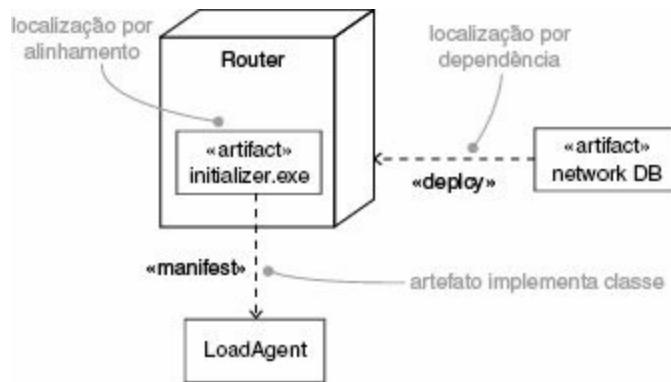
Os sistemas distribuídos, por sua própria natureza, abrangem componentes que estão fisicamente espalhados pelos nós de um sistema. No caso de muitos sistemas, os componentes ficam fixos no local quando são carregados no sistema; em outros sistemas, os componentes podem migrar de um nó para outro.

- ➡ Os componentes são examinados no [Capítulo 15](#); os nós são apresentados no [Capítulo 27](#); os diagramas de implantação são examinados no [Capítulo 31](#); a dicotomia classe/objeto é apresentada nos [Capítulos 2 e 13](#); as classes são examinadas nos [Capítulos 4 e 9](#).

Na UML, é possível fazer a modelagem da visão de implantação de um sistema, utilizando diagramas de implantação que representam a topologia de processadores e de dispositivos no qual o sistema é executado. Componentes tais como executáveis, bibliotecas e tabelas residem nesses nós. Cada instância de um nó terá instâncias de certos componentes e cada instância de um componente pertencerá a exatamente uma única instância de um nó (embora as instâncias do mesmo tipo de componente possam estar espalhadas por diferentes nós).

Componentes e classes podem ser manifestados como artefatos. Por exemplo, conforme mostra a [Figura 24.3](#), a classe LoadAgent é manifestada pelo artefato initializer.exe que se encontra no nó do tipo Router.

Conforme a figura ilustra, é possível fazer a modelagem da localização de um elemento de duas maneiras na UML. Primeiro, conforme é mostrado em relação a Router, você pode aninhar fisicamente o elemento (textual ou graficamente) em um compartimento extra no nó que o contém. Segundo, você pode usar uma dependência com a palavra-chave `deploy` do artefato para o nó que o contém.



[Figura 24.3:](#)  
Localização

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE RESTRIÇÕES DE TEMPO

A modelagem do tempo absoluto de um evento, a modelagem do tempo relativo entre eventos e a modelagem do tempo necessário para executar uma ação são as três propriedades primárias críticas em relação a tempo de sistemas em tempo real para os quais você utilizará restrições de tempo.

- » As restrições, um dos mecanismos de extensibilidade da UML, são examinadas no [Capítulo 6](#).

Para fazer a modelagem de restrições de tempo:

- » Para cada evento existente em uma interação, considere se ele deve ser iniciado em algum tempo absoluto. Faça a modelagem dessa propriedade de tempo real como uma restrição de tempo para a mensagem.
- » Para cada sequência de mensagens relevantes em uma interação, considere se existe um tempo relativo máximo associado a essa sequência. Faça a modelagem dessa propriedade de tempo real como uma restrição de tempo para a sequência.

Por exemplo, conforme mostra a [Figura 24.4](#), a restrição mais à esquerda especifica o tempo de repetição do início, restabelecido pelo evento refresh. De modo semelhante, no centro, a restrição de tempo especifica a duração máxima para as chamadas executarem a obtenção de uma imagem, getImage.

Com frequência, você escolherá nomes breves para as mensagens, de modo a não confundi-las com nomes de operações.

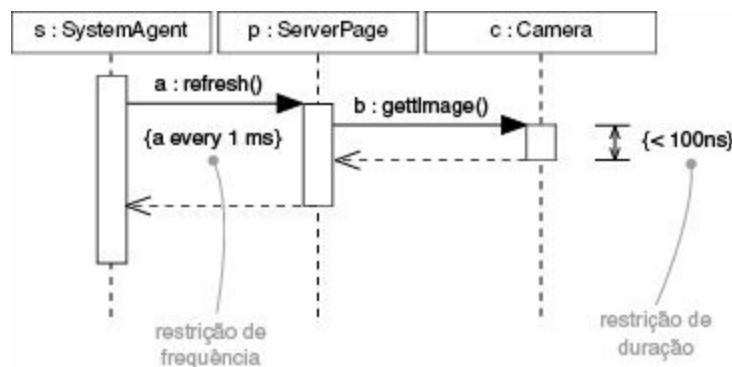


Figura 24.4:

## MODELAGEM DA DISTRIBUIÇÃO DE OBJETOS

Ao fazer a modelagem da topologia de um sistema distribuído, você desejará considerar a colocação física tanto de nós como de artefatos. Se o seu foco é o gerenciamento da configuração do sistema entregue, a modelagem da distribuição de nós é especialmente importante para visualizar, especificar, construir e documentar a posição física de itens como executáveis, bibliotecas e tabelas. Se o seu foco é a funcionalidade, escalabilidade e o tempo de resposta do sistema, a modelagem da distribuição de objetos é o que é importante.

- » *A modelagem da distribuição de componentes é examinada no [Capítulo 15](#).*

Decidir como distribuir os objetos em um sistema é um problema sério e não apenas porque os problemas de distribuição interagem com os problemas de concorrência. Soluções ingênuas tendem a gerar um desempenho bastante fraco e as soluções com excesso de engenharia não são muito melhores. De fato, provavelmente são piores, pois geralmente acabam sendo frágeis.

- » *A modelagem de processos e threads é examinada no [Capítulo 23](#).*

Para fazer a modelagem da distribuição de objetos:

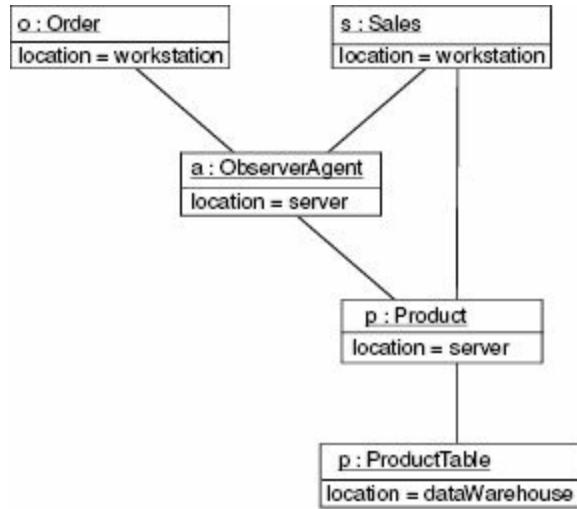
- » Para cada classe interessante de objetos existente em seu sistema, considere sua localização de referência. Em outras palavras, considere todos os seus vizinhos e respectivas localizações. Uma localização fortemente acoplada terá objetos vizinhos próximos; uma fracamente acoplada terá objetos distantes (e, portanto, haverá latência na comunicação entre eles). Experimente alocar os objetos o mais próximo possível dos atores que os manipulam.
- » A seguir, considere os padrões de interação entre conjuntos relacionados de objetos. Coloque próximos os conjuntos de objetos que mantêm alto

grau de interação, para reduzir os custos de comunicação. Divida conjuntos de objetos que tenham baixo grau de interação.

- » A seguir, considere a distribuição de responsabilidades no sistema. Redistribua seus objetos de forma a equilibrar a carga de cada nó.
- » Considere também questões de segurança, volatilidade e qualidade de serviço e redistribua seus objetos, conforme seja apropriado.
- » Atribua objetos a artefatos para que os objetos mais acoplados fiquem no mesmo artefato.
- » Atribua artefatos a nós para que as necessidades de computação de cada nó estejam dentro da capacidade. Acrescente nós adicionais se necessário.
- » Equilibre os custos de desempenho e comunicação atribuindo ao mesmo nó os artefatos mais acoplados.

A [Figura 24.5](#) mostra um diagrama de objeto que modela a distribuição de certos objetos de um sistema de venda a varejo. O valor desse diagrama está em permitir que você visualize a distribuição física de certos objetos importantes. Conforme o diagrama mostra, dois objetos residem em uma estação de trabalho, workstation (os objetos relativos a pedidos e vendas, respectivamente Order e Sales), dois objetos residem em um servidor, server (os objetos relativos ao agente de observação e ao próprio produto, ObserverAgent e Product) e um único objeto reside no banco de dados, dataWarehouse (o objeto relativo à tabela de produtos, ProductTable).

- ➡ Os diagramas de objetos são examinados no [Capítulo 14](#).



**Figura 24.5:**

A modelagem da distribuição de objetos

## MODELAGEM DE OBJETOS QUE MIGRAM

### DICAS E SUGESTÕES

Um modelo bem estruturado com propriedades de tempo e espaço:

- » Expõe somente as propriedades de tempo e espaço necessárias e suficientes para capturar o comportamento desejado do sistema.
- » Centraliza o uso dessas propriedades de modo que seja fácil encontrá-las e modificá-las.

Ao definir propriedades de tempo ou espaço na UML:

- » Dê às suas marcas de tempo (os nomes das mensagens) nomes significativos.
- » Diferencie com clareza expressões de tempo relativo e absoluto.
- » Mostre as propriedades de espaço somente quando isso for importante para visualizar a localização dos elementos em um sistema implantado.
- » Para obter necessidades mais avançadas, considere o *UML Profile for Schedulability, Performance, and Time*. Essa especificação de OMG atende às necessidades de sistemas reativos em tempo real e de alto

desempenho.

CAPÍTULO

---

25

---

# Diagramas de Estados

## Neste capítulo

- » Modelagem de objetos reativos
- » Engenharia direta e reversa

**O**s diagramas de estados são um dos cinco diagramas disponíveis na UML para a modelagem dos aspectos dinâmicos de sistemas. Um diagrama de estados mostra uma máquina de estados. Os diagramas de atividades são um caso especial de diagramas de estados, em que todos ou a maioria dos estados são estados de atividades e todas ou a maioria das transições são ativadas pela conclusão de atividades no estado de origem. Portanto, tanto os diagramas de atividades como os diagramas de estados são úteis para a modelagem do tempo de vida de um objeto. Entretanto, enquanto um diagrama de atividades mostra o fluxo de controle de uma atividade para outra, o diagrama de estados mostra o fluxo de controle de um estado para outro.

- ➡ Os diagramas de sequências, diagramas de comunicação, diagramas de atividades e diagramas de casos de uso também permitem fazer a modelagem dos aspectos dinâmicos de sistemas; os diagramas de sequências e os diagramas de comunicação são examinados no [Capítulo 19](#); os diagramas de atividades são apresentados no [Capítulo 20](#); os diagramas de casos de uso são explicados no [Capítulo 18](#).

Os diagramas de estados são empregados para a modelagem dos aspectos dinâmicos de um sistema. Na maior parte, isso envolve a modelagem do comportamento de objetos reativos. Um objeto reativo é aquele cujo comportamento é mais bem caracterizado por sua resposta a eventos ativados externamente ao seu contexto. Um objeto reativo tem um claro tempo de vida cujo comportamento atual é afetado pelo seu passado. Os diagramas de estados podem ser anexados a classes, a casos de uso ou a sistemas inteiros para visualizar, especificar, construir e documentar a dinâmica de um objeto individual.

Os diagramas de estados não são importantes somente para a modelagem dos aspectos dinâmicos de um sistema, mas também para a construção de sistemas executáveis por meio de engenharia direta e reversa.

## PRIMEIROS PASSOS

Considere o investidor que financia a construção de um novo prédio. Provavelmente, ele não está interessado em detalhes do processo de construção. A seleção de materiais, a programação de compras e as muitas reuniões sobre detalhes de engenharia são atividades importantes para o construtor, mas pouco interessam à pessoa que está financiando o projeto.

- ➡ As diferenças entre a construção de uma casa de cachorro e de um prédio alto são examinadas no [Capítulo 1](#).

O investidor está interessado em conseguir um bom retorno de seu investimento e isso também significa proteger o investimento contra riscos. Um investidor realmente confiante entregará ao construtor uma pilha de dinheiro, visita o local por alguns instantes e retorna somente quando o construtor está pronto para lhe entregar as chaves do prédio. Esse investidor realmente está interessado no estado final da construção.

Um investidor mais pragmático confiará no construtor, mas também desejará verificar se o projeto tem um andamento adequado antes de liberar

seu dinheiro. Assim, em vez de entregar ao construtor uma pilha de dinheiro, o investidor prudente definirá etapas claras para o projeto, cada uma vinculada à conclusão de determinadas atividades, e, somente depois da conclusão de cada etapa, liberará ao construtor o dinheiro para a fase seguinte do projeto. Por exemplo, uma quantia modesta poderá ser liberada na concepção do projeto, para financiar o trabalho dos arquitetos. Depois de a ideia dos arquitetos ter sido aprovada, então mais fundos poderão ser liberados para o pagamento do trabalho de engenharia. Quando esse trabalho for concluído para satisfação dos financiadores do projeto, uma pilha maior de dinheiro poderá ser liberada para que o construtor possa começar a trabalhar no terreno.

Ao longo do caminho, da fundação à emissão do certificado de habite-se, existirão outras etapas do andamento do trabalho. Cada uma dessas etapas de trabalho dá nome a um estado estável do projeto: a arquitetura completa, a engenharia concluída, trabalho no terreno, infraestrutura completa, o licenciamento de uso do prédio e assim por diante. Para o investidor, acompanhar o estado de mudança do prédio é mais importante do que acompanhar o fluxo de atividades, que é responsabilidade do construtor. O construtor, por sua vez, pode fazer o acompanhamento do fluxo de atividades com os gráficos Pert que refletem a evolução do projeto.

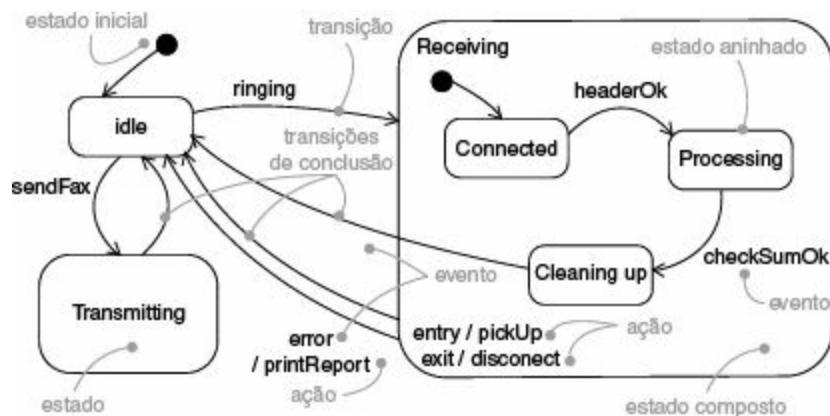
► *Os gráficos de Gantt e de Pert são examinados no Capítulo 20.*

Na modelagem de sistemas complexos de software, você também concluirá que a forma mais natural para visualizar, especificar, construir e documentar o comportamento de certos tipos de objetos consiste em focalizar o fluxo de controle de um estado para outro e não de uma atividade para outra. Você faria o último procedimento com um fluxograma (e em UML, com um diagrama de atividades). Imagine, por um momento, a modelagem do comportamento de sistema de segurança embutido doméstico. Esse sistema é executado continuamente, reagindo a eventos externos, como uma janela que se quebra. Além disso, a ordem dos eventos modifica a maneira

como o sistema se comporta. Por exemplo, a detecção de uma janela se quebrando somente ativará um alarme se o sistema antes tiver sido armado. O comportamento desse sistema é mais bem especificado pela modelagem de seus estados estáveis (por exemplo, Idle, Armed, Active, Checking e assim por diante), os eventos que iniciam a alteração de um estado para outro e as ações que ocorrem em cada alteração de estado.

- Os *diagramas de atividades* como fluxogramas são examinados no [Capítulo 20](#); as máquinas de estados são apresentadas no [Capítulo 22](#).

Na UML, a modelagem de comportamentos ordenados por eventos de um objeto é feita pela utilização de diagramas de estados. Conforme mostra a [Figura 25.1](#), um diagrama de estados é simplesmente a apresentação de uma máquina de estados, dando ênfase ao fluxo de controle de um estado para outro.



**Figura 25.1:**  
Diagrama de estados

## TERMOS E CONCEITOS

Um *diagrama de estados* mostra uma máquina de estados, dando ênfase ao fluxo de controle de um estado para outro. Uma *máquina de estados* é um comportamento que especifica as sequências de estados pelos quais um

objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos. Um *estado* é uma condição ou situação na vida de um objeto durante a qual ele satisfaz a alguma condição, realiza alguma atividade ou aguarda algum evento. Um *evento* é uma especificação de uma ocorrência significativa que tem uma localização no tempo e no espaço. No contexto de uma máquina de estados, um evento é uma ocorrência de um estímulo capaz de ativar uma transição de estado. Uma *transição* é um relacionamento entre dois estados, indicando que um objeto no primeiro estado realizará certas ações e entrará no segundo estado quando um evento especificado ocorrer e as condições especificadas estão satisfeitas. Uma *atividade* é uma execução não atômica em andamento em uma máquina de estados. Uma *ação* é uma computação atômica executável que resulta em uma alteração do estado do modelo ou no retorno de um valor. Graficamente, um diagrama de estados é uma coleção de vértices e arcos.

**Nota:** Os diagramas de estados usados na UML baseiam-se na notação de estado inventada por David Harel. Em particular, os conceitos de estados aninhados e estados disjuntos foram desenvolvidos por Harel em um sistema preciso e formal. Os conceitos na UML são um pouco menos formais que a notação de Harel e diferem em alguns detalhes, especialmente, por enfatizar sistemas orientados a objetos.

## PROPRIEDADES COMUNS

Um diagrama de estados é apenas um tipo especial de diagrama e compartilha as mesmas propriedades comuns de todos os demais diagramas – ou seja, um nome e um conteúdo gráfico que são uma projeção em um modelo. O que diferencia um diagrama de estados de todos os outros tipos de diagramas é o seu conteúdo.

- ➡ As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

## CONTEÚDO

Diagramas de estados costumam conter:

- » Estados simples e estados compostos
- » Transições, incluindo eventos e ações
  - ➡ Os estados simples, estados compostos, transições, eventos e ações são examinados no [Capítulo 22](#); os diagramas de atividades são apresentados no [Capítulo 20](#); as notas e as restrições são examinadas no [Capítulo 6](#).

**Nota:** Um diagrama de estados é basicamente uma projeção dos elementos encontrados em uma máquina de estados. Isso significa que os diagramas de estados poderão conter ramificações, bifurcações, uniões, estados de ações, estados de atividades, objetos, estados iniciais, estados finais, estados de históricos e assim por diante. Na verdade, um diagrama de estados poderá conter toda e qualquer característica de uma máquina de estados.

Como todos os demais diagramas, os diagramas de estados podem conter notas e restrições.

## USOS COMUNS

Os diagramas de estados podem ser empregados para fazer a modelagem dos aspectos dinâmicos de um sistema. Esses aspectos dinâmicos podem envolver o comportamento ordenado por eventos de qualquer tipo de objeto em qualquer visão da arquitetura do sistema, incluindo classes (o que inclui as classes ativas), interfaces, componentes e nós.

- ➡ As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#); as instâncias são apresentadas no [Capítulo 13](#); as classes são explicadas nos Capítulos [4](#) e [9](#).

Ao utilizar um diagrama de estados para a modelagem de algum aspecto dinâmico de um sistema, você o faz no contexto virtual de qualquer elemento da modelagem. Tipicamente, entretanto, os diagramas de estados serão empregados no contexto do sistema como um todo, de um subsistema ou de

uma classe. Você também poderá anexar diagramas de estados a casos de uso (para a modelagem de um cenário).

Ao fazer a modelagem dos aspectos dinâmicos de um sistema, classe ou caso de uso, você tipicamente usará diagramas de estados de uma única maneira.

- As classes ativas são examinadas no [Capítulo 23](#); as interfaces são apresentadas no [Capítulo 11](#); os componentes são discutidos no [Capítulo 15](#); os nós são examinados no [Capítulo 27](#); os casos de uso aparecem no [Capítulo 17](#); os sistemas são explicados no [Capítulo 32](#).

Um objeto reativo – ou orientado por eventos – é aquele cujo comportamento é mais bem caracterizado pela sua resposta a eventos ativados externamente ao seu contexto. Um objeto reativo tipicamente fica ocioso até receber um evento. Ao receber um evento, sua resposta geralmente depende de eventos anteriores. Após o objeto responder ao evento, ele fica ocioso novamente, aguardando o próximo evento. No caso desse tipo de objeto, você focalizará os estados estáveis desse objeto, os eventos que ativam a transição de um estado para outro e as ações que ocorrem em cada alteração de estado.

**Nota:** Em contraste, você usará os diagramas de atividades para fazer a modelagem de um fluxo de trabalho ou de uma operação. Os diagramas de atividades são mais adequados para a modelagem do fluxo de atividades ao longo do tempo, como você as representaria em um fluxograma.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE OBJETOS REATIVOS

O propósito mais comum para o qual você usará os diagramas de estados é a modelagem do comportamento de objetos reativos, especialmente instâncias de classes, de casos de uso e do sistema como um todo. Enquanto

as interações modelam o comportamento de uma sociedade de objetos trabalhando em conjunto, o diagrama de estados modela o comportamento de um único objeto ao longo de seu tempo de vida. Enquanto o diagrama de atividades modela o fluxo de controle de uma atividade para outra, o diagrama de estados modela o fluxo de controle de um evento para outro.

- As interações são examinadas no [Capítulo 16](#); os diagramas de atividades são apresentados no [Capítulo 20](#).

Ao fazer a modelagem do comportamento de um objeto reativo, você essencialmente especifica três coisas: os estados estáveis em que o objeto poderá viver, os eventos que ativam a transição de um estado para outro e as ações que ocorrem em cada mudança de estado. A modelagem do comportamento de um objeto reativo também envolve a modelagem do tempo de vida de um objeto, começando no momento em que o objeto é criado e continuando até sua destruição, destacando os estados estáveis em que o objeto poderá ser encontrado.

- A modelagem do tempo de vida de um objeto é examinada no [Capítulo 22](#).

Um estado estável representa a condição em que o objeto poderá existir por algum período identificável de tempo. Quando um evento ocorre, o objeto poderá realizar a transição de um estado para outro. Esses eventos também poderão ativar autotransições e transições internas, em que a origem e o destino da transição são o mesmo estado. Em reação a um evento ou a uma alteração de estado, o objeto poderá responder ativando uma ação.

- O tempo e o espaço são examinados no [Capítulo 24](#).

**Nota:** Ao fazer a modelagem do comportamento de um objeto reativo, você pode especificar sua ação, relacionando-a a uma transição ou a uma alteração de estado. Em termos técnicos, uma máquina de estados cujas ações estão todas anexadas a transições é chamada uma máquina de Mealy; uma máquina de estados cujas ações estão todas anexadas a estados é chamada uma máquina de Moore.

*Matematicamente, os dois estilos são equivalentes. Na prática, você tipicamente desenvolverá diagramas de estados que utilizam uma combinação das máquinas de Mealy e de Moore.*

Para fazer a modelagem de um objeto reativo:

- › Escolha o contexto para a máquina de estados, seja uma classe, um caso de uso ou o sistema como um todo.
- › Escolha os estados inicial e final para o objeto. Para orientar o restante de seu modelo, possivelmente estabeleça as pré e pós-condições dos estados inicial e final, respectivamente.
- › Decida sobre os estados estáveis do objeto, levando em consideração as condições em que o objeto poderá existir por algum período identificável de tempo. Comece com os estados de nível mais alto do objeto e somente depois considere seus possíveis subestados.
- › Decida sobre a ordenação parcial significativa dos estados estáveis ao longo do tempo de vida do objeto.
- › Decida sobre os eventos que poderão ativar a transição de um estado para outro. Faça a modelagem desses eventos como a ativação de transições que passam de uma ordenação legal de estados para outra ordenação.
- › Anexe ações a essas transições (como em uma máquina de Mealy) e/ou a esses estados (como em uma máquina de Moore).
- › Considere formas para simplificar sua máquina, utilizando subestados, ramificações, bifurcações, uniões e estados de histórico.
- › Verifique se todos os estados podem ser alcançados sob alguma combinação de eventos.
- › Verifique se nenhum estado é um “buraco negro” em que nenhuma combinação de eventos fará a transição do objeto para outro estado.

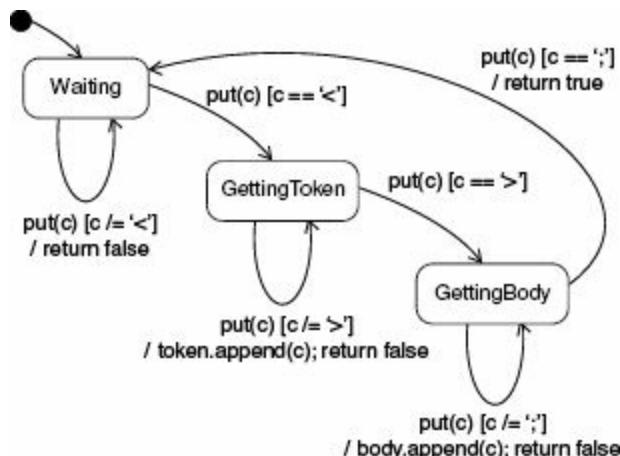
» Examine a máquina de estados, manualmente ou utilizando ferramentas para verificá-la em relação a sequências esperadas de eventos e respectivas respostas.

► *Pré e pós-condições são discutidas no Capítulo 10; interfaces são discutidas no Capítulo 11.*

Por exemplo, a [Figura 25.2](#) mostra o diagrama de estados para análise de uma linguagem simples livre de contexto, como você poderá encontrar em sistemas que fornecem e recebem mensagens para XML. Nesse caso, a máquina é designada a analisar uma sequência de caracteres que corresponde à sintaxe:

mensagem : '<' sequência de caracteres '>' sequência de caracteres ';'

A primeira sequência de caracteres representa um desvio; a segunda sequência de caracteres representa o corpo da mensagem. Dada uma sequência de caracteres, somente as mensagens bem formadas que obedecem a essa sintaxe poderão ser aceitas.



**Figura 25.2:**  
A modelagem de objetos reativos

Conforme mostra a figura, existem somente três estados estáveis para essa máquina de estados, representando os estados de espera, de obtenção do significado da mensagem e de obtenção do próprio corpo da mensagem:

Waiting, GettingToken e GettingBody. Esse gráfico de estados é designado como uma máquina de Mealy, com ações vinculadas às transições. De fato, existe somente um evento de interesse nessa máquina de estados: a invocação de `put` com o parâmetro real `c` (um caractere). Enquanto no estado Waiting, essa máquina despreza qualquer caractere que não seja capaz de designar o início de um token (conforme é especificado pela condição de guarda). Quando o início de um token é recebido, o estado do objeto muda para GettingToken. Enquanto se encontra nesse estado, a máquina salva qualquer caractere que não designa o final de um token (conforme é especificado pela condição de guarda). Quando o fim de um token é recebido, o estado do objeto se modifica para GettingBody. Enquanto se encontra nesse estado, a máquina salva qualquer caractere que não designa o fim de corpo de mensagem (conforme é especificado pela condição de guarda). Quando o fim da mensagem é recebido, o estado do objeto se modifica para Waiting e um valor é retornado, indicando que a mensagem foi analisada (e que a máquina está pronta para receber outra mensagem).

► Os eventos são examinados no [Capítulo 21](#).

Observe que esse estado especifica uma máquina que roda continuamente; não existe um estado final.

## ENGENHARIA DIRETA E REVERSA

A *engenharia direta* (a criação de código a partir de um modelo) é possível para diagramas de estados, especialmente se o contexto do diagrama é uma classe. Por exemplo, utilizando o diagrama de estados apresentado anteriormente, uma ferramenta de engenharia direta poderia gerar o seguinte código em Java para a classe MessageParser.

```
class MessageParser {  
    public  
    boolean put(char c) {  
        switch (state) {  
            case Waiting:  
                if (c == '<') {
```

```

state = GettingToken;
token = new StringBuffer( );
body = new StringBuffer( );
}
break;
case GettingToken :
if (c == '>')
state = GettingBody;
else
token.append(c);
break;
case GettingBody :
if (c == ' ; ') {
state = Waiting;
return true; }
else
body.append (c);
}
return false;
}
StringBuffer getToken( ) {
return token;
}
StringBuffergetBody( ) {
return body;
}
private
final static int Waiting = 0;
final static int GettingToken = 1;
final static int GettingBody = 2;
int state = Waiting;
StringBuffer token, body;
}

```

Isso requer um pouco de esperteza. A ferramenta de engenharia direta deve gerar os atributos private necessários e as constantes finais static.

*Engenharia reversa* (a criação de um modelo a partir de um código) é teoricamente possível, mas praticamente não é muito útil. A escolha do que constitui um estado significativo está nos olhos do projetista. As ferramentas da engenharia reversa não têm capacidade para abstração e, portanto, não são capazes de produzir automaticamente diagramas de estados significativos.

Mais interessante do que a engenharia reversa de um modelo a partir de um código é a animação de um modelo em relação à execução de um sistema. Por exemplo, dado o diagrama anterior, uma ferramenta poderia animar os estados mostrados no diagrama, conforme foram alcançados no sistema em execução. De modo semelhante, a ativação de transições poderia ser animada, mostrando o destinatário dos eventos e a consequente ativação de ações. Sob o controle de um depurador, você poderia controlar a velocidade de execução, definindo pontos de parada para interromper a ação em estados interessantes e examinar os valores de atributos de objetos individuais.

## DICAS E SUGESTÕES

Ao criar diagramas de estados na UML, lembre-se de que todo diagrama de estados é apenas uma projeção no mesmo modelo de aspectos dinâmicos de um sistema. Um único diagrama de estados é capaz de capturar a semântica de um único objeto reativo, mas nenhum diagrama de estados poderá capturar a semântica de todo um sistema não trivial.

Um diagrama de estados bem estruturado:

- » Tem como foco a comunicação de um aspecto da dinâmica de um sistema.
- » Contém somente os elementos essenciais à compreensão desse aspecto.
- » Fornece detalhes consistentes com seu nível de abstração (expõe somente as características essenciais à compreensão).
- » Utiliza um equilíbrio entre os estilos de máquinas de Mealy e de Moore.

Ao definir um diagrama de estados:

- » Dê-lhe um nome capaz de comunicar seu propósito.
- » Inicie com a modelagem dos estados estáveis do objeto e depois prossiga com a modelagem das transições legais de um estado para outro. Inclua ramificação, concorrência e fluxo de objetos como

considerações secundárias, possivelmente em diagramas separados.

- › Distribua seus elementos de forma a minimizar o cruzamento de linhas.
- › Em diagramas de estados grandes, considere as características avançadas, como as submáquinas que são incluídas na especificação completa da UML.

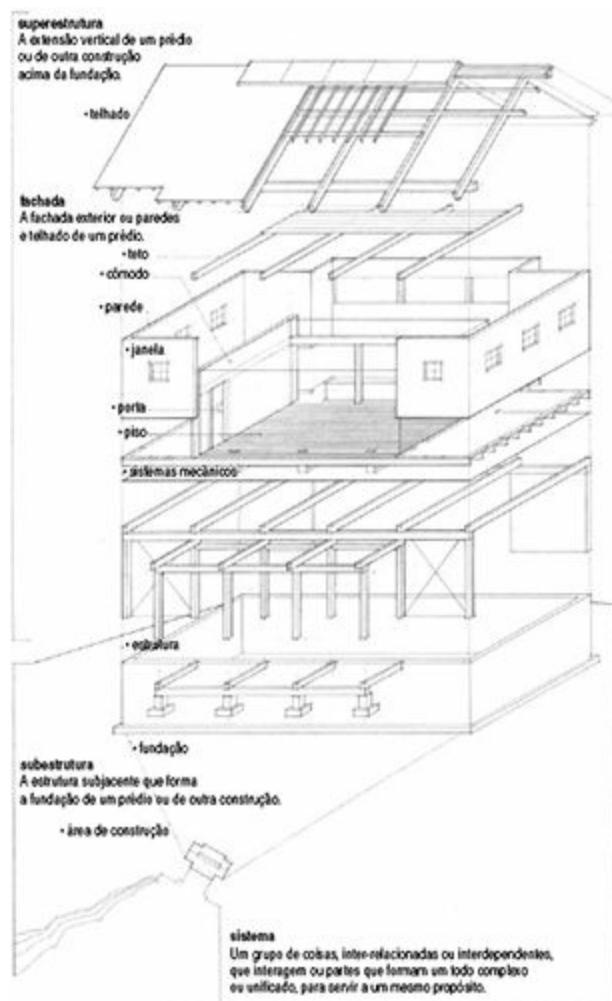
P A R T E

---

6

---

# MODELAGEM DA ARQUITETURA



CAPÍTULO

---

**26**

---

# Artefatos

## Neste capítulo

- » Artefatos, classes e manifestação
- » Modelagem de executáveis e bibliotecas
- » Modelagem de tabelas, arquivos e documentos
- » Modelagem de código-fonte

**O**s artefatos vivem no mundo material dos bits e, portanto, são um importante bloco de construção para a modelagem de aspectos físicos de um sistema. Um artefato é a parte física e substituível de um sistema.

Os artefatos são empregados para a modelagem de itens físicos que podem residir em um nó, como executáveis, bibliotecas, tabelas, arquivos e documentos. Um artefato tipicamente representa o pacote físico de elementos lógicos, como classes, interfaces e colaborações.

Bons componentes definem abstrações com interfaces bem definidas, tornando possível substituir facilmente componentes mais antigos por outros compatíveis mais novos.

## PRIMEIROS PASSOS

O produto final de uma construtora é o prédio físico que existe no mundo real. Você constrói modelos lógicos para visualizar, especificar e documentar suas decisões sobre o envelope da construção; a posição de paredes, portas e janelas; a distribuição dos sistemas elétrico e de encanamento; e o estilo geral da arquitetura. Quando você realmente constrói o prédio, essas paredes,

portas, janelas e outras coisas conceituais são transformadas em coisas físicas, reais.

Essas visões lógica e física são ambas necessárias. Se você estiver construindo algo cujo custo de desfazer e refazer seja essencialmente zero (por exemplo, se estiver construindo uma casa de cachorro), provavelmente poderá passar direto à construção física sem fazer qualquer modelagem lógica. Se, por outro lado, estiver construindo algo para durar, cujo custo para realizar alterações ou corrigir erros seja alto, então a construção de modelos lógico e físico será a ação pragmática a ser feita para gerenciar a possibilidade de riscos.

- As diferenças entre a construção de uma casa de cachorro e a construção de um prédio alto são examinadas no Capítulo 1.

O mesmo ocorre quando se constrói um sistema complexo de software. Você faz a modelagem lógica para visualizar, especificar e documentar suas decisões sobre o vocabulário do seu domínio e a forma estrutural e comportamental com que essas coisas colaboram. Você faz a modelagem física para construir o sistema executável. Enquanto essas coisas lógicas vivem no mundo conceitual, as coisas físicas vivem no mundo dos bits – ou seja, efetivamente residem nos nós físicos e podem ser executadas diretamente ou, de alguma maneira indireta, participar em um sistema em execução.

Na UML, todas essas coisas físicas são modeladas como artefatos. Um artefato é a coisa física no nível da plataforma de implementação.

Em relação ao software, muitos sistemas operacionais e linguagens de programação oferecem suporte direto para o conceito de um artefato. As bibliotecas de objetos, os executáveis, os componentes .NET e Enterprise Java Beans são todos exemplos de artefatos que poderão ser representados diretamente na UML. Não somente os artefatos podem ser utilizados para fazer a modelagem desses tipos de coisas, mas também podem ser

empregados para representar outras coisas que participam em um sistema em execução, como tabelas, arquivos e documentos.

A UML fornece uma representação gráfica de um artefato, conforme mostra a [Figura 26.1](#). Essa notação canônica permite visualizar um artefato independente de qualquer sistema operacional ou linguagem de programação. Utilizando estereótipos, um dos mecanismos de extensibilidade da UML, você poderá ajustar essa notação para a representação de tipos específicos de artefatos.

► Os estereótipos são examinados no [Capítulo 6](#).



**Figura 26.1:**

Artefatos

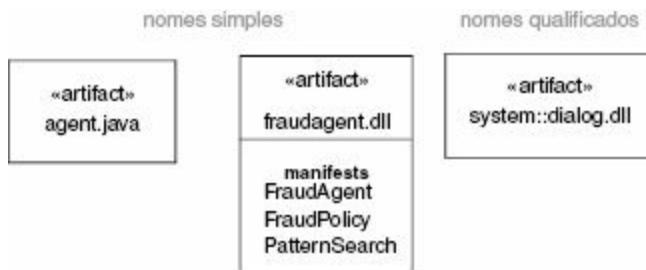
## TERMOS E CONCEITOS

Um *artefato* é uma parte física de um sistema que existe no nível da plataforma de implementação. Graficamente, um artefato é representado como um retângulo com a palavra-chave «artifact».

### NOMES

Todo artefato precisa ter um nome que o diferencie dos outros artefatos. Um *nome* é uma sequência de caracteres textual. Esse nome sozinho é conhecido como um *nome simples*; um *nome qualificado* é o nome de um artefato que tem, como prefixo, o nome do pacote em que esse artefato vive. Um artefato tipicamente é definido mostrando somente seu nome, como na [Figura 26.2](#). Assim como as classes, é possível definir artefatos que têm, como adornos, valores atribuídos ou compartimentos adicionais para expor seus detalhes, conforme se observa na figura.

- » *O nome de um artefato deve ser único no nó que o contém.*



**Figura 26.2:**

Nomes de artefatos simples e qualificados

**Nota:** *O nome de um artefato pode ser texto formado por qualquer quantidade de letras, números e certos sinais de pontuação (com exceção de sinais como dois-pontos, utilizados para separar o nome do artefato e o nome do pacote que o contém) e poderá continuar por várias linhas. Na prática, os nomes de artefatos são expressões nominais ou nomes curtos, definidos a partir do vocabulário da implementação e, dependendo do sistema operacional destino, incluem extensões (como java e dll).*

## ARTEFATOS E CLASSES

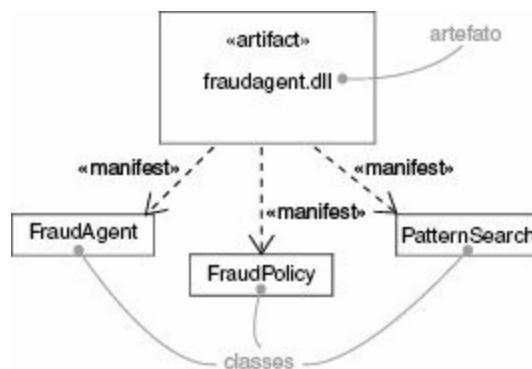
Artefatos e classes são classificadores. Entretanto, existem algumas diferenças significativas entre artefatos e classes.

- » As classes são examinadas nos Capítulos 4 e 9; as interações são apresentadas no Capítulo 16.
- » As classes representam abstrações lógicas; os artefatos representam coisas físicas que vivem no mundo dos bits. Em resumo, os artefatos podem viver em nós, as classes não.
- » Os artefatos representam o pacote físico de bits na plataforma de implementação.
- » As classes podem ter atributos e operações. Os artefatos podem implementar classes e métodos, mas eles mesmos não têm atributos ou operações.

A primeira diferença é a mais importante. Ao fazer a modelagem de um sistema, a decisão de utilizar uma classe ou um artefato envolve uma escolha simples – se o que é modelado vive diretamente em um nó, use um artefato; caso contrário, use uma classe. A segunda diferença também esclarece isso.

- Os nós são examinados no [Capítulo 27](#).

A terceira diferença sugere um relacionamento entre artefatos e classes. Em particular, um artefato é a implementação física de um conjunto de elementos lógicos, como classes e colaborações. Como mostra a [Figura 26.3](#), o relacionamento entre um artefato e as classes que ele implementa pode ser apresentado explicitamente usando um relacionamento de manifestação.



**Figura 26.3:**  
Artefatos e classes

## TIPOS DE ARTEFATOS

Três tipos de artefatos podem ser diferenciados.

Primeiro, existem *artefatos de implantação*. São os artefatos necessários e suficientes para formar um sistema executável, como as bibliotecas dinâmicas (DLLs) e os executáveis (EXEs). A definição da UML para artefato é suficientemente ampla para incluir modelos clássicos de objeto, como .NET, CORBA e Enterprise Java Beans, além de modelos alternativos de objeto, talvez envolvendo páginas da Web dinâmicas, tabelas de bancos de dados e executáveis utilizando mecanismos proprietários de comunicação.

Segundo, existem os *artefatos do produto do trabalho*. Esses artefatos são essencialmente o resíduo do processo de desenvolvimento, formados por coisas como arquivos de código-fonte e arquivos de dados a partir dos quais os artefatos de implantação são criados. Esses artefatos não participam diretamente em um sistema executável, mas são os produtos do trabalho de desenvolvimento, utilizados para a criação do sistema executável.

Terceiro, existem os *artefatos de execução*. Esses artefatos são criados como uma consequência de um sistema em execução, como um objeto .NET, que é instanciado a partir de uma DLL.

## ELEMENTOS-PADRÃO

Todos os mecanismos de extensibilidade da UML se aplicam aos artefatos. Com maior frequência, você utilizará valores atribuídos para estender as propriedades dos artefatos (como a especificação da versão de um artefato do desenvolvimento) e estereótipos para especificar novos tipos de artefatos (como os artefatos específicos de um sistema operacional).

- ➡ Os mecanismos de extensibilidade da UML são examinados no Capítulo 6.

A UML define estereótipos-padrão que se aplicam aos artefatos:

1. executable                          Especifica um programa executado em um nó.
2. library                              Especifica uma biblioteca de objetos estática ou dinâmica.
3. file                                 Especifica um arquivo que contém um documento, código-fonte ou outros dados.
4. document                         Especifica um artefato que representa um documento.

Outros podem ser definidos para plataformas e sistemas específicos.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE EXECUTÁVEIS E DE BIBLIOTECAS

O propósito mais comum para o qual os artefatos serão utilizados é a modelagem de artefatos de implantação que compõem a sua implementação. Se você estiver entregando um sistema trivial, cuja implementação consiste em exatamente um único arquivo executável, não precisará de qualquer modelagem de artefatos. Se, por outro lado, o sistema a ser entregue é composto por vários executáveis e está associado a bibliotecas de objetos, fazer a modelagem de artefatos auxiliará a visualizar, especificar, construir e documentar as decisões tomadas em relação ao sistema físico. A modelagem de artefatos é ainda mais importante se você quiser controlar as versões e o gerenciamento da configuração dessas partes, à medida que o sistema evolui.

Na maioria dos sistemas, esses artefatos de implantação são definidos a partir das decisões tomadas para segmentar a implementação física do sistema. Essas decisões serão afetadas por várias questões técnicas (como a escolha do sistema operacional com base nas facilidades de seus artefatos), questões referentes ao gerenciamento da configuração (como as decisões sobre as partes que provavelmente serão alteradas ao longo do tempo) e questões de reutilização (ou seja, decidir quais artefatos poderão ser reaproveitados em ou de outros sistemas).

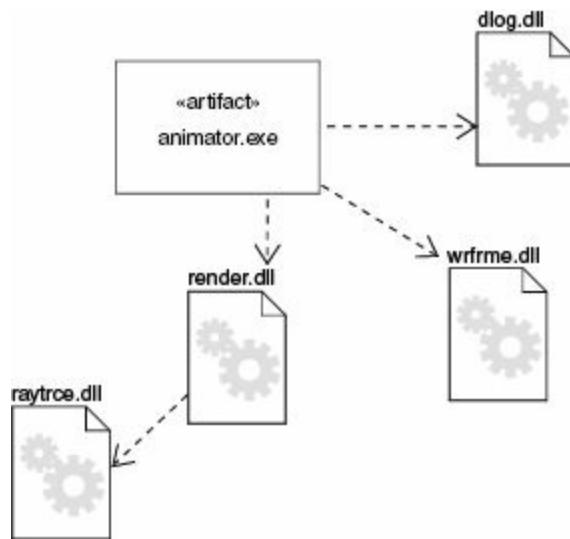
- » *Essas decisões também são afetadas pela topologia do sistema destino, conforme é examinado no [Capítulo 27](#).*

Para fazer a modelagem de executáveis e de bibliotecas:

- » Identifique o particionamento de seu sistema físico. Considere o impacto de questões técnicas, de gerenciamento da configuração e de reutilização.

- » Faça a modelagem de todos os executáveis e bibliotecas como artefatos, utilizando os elementos-padrão apropriados. Se a implementação introduzir novos tipos de artefatos, inclua um novo estereótipo adequado.
- » Se for importante gerenciar as costuras de seu sistema, faça a modelagem das interfaces significativas que alguns artefatos usam e que outros realizam.
- » Conforme seja necessário para comunicar sua intenção, faça a modelagem dos relacionamentos entre esses executáveis, bibliotecas e interfaces. Com muita frequência, você desejará fazer a modelagem das dependências entre essas partes com a finalidade de visualizar o impacto da alteração.

Por exemplo, a [Figura 26.4](#) mostra um conjunto de artefatos definidos a partir de uma ferramenta para a criação de animações gráficas, executada em um único computador pessoal. Essa figura inclui um executável (`animator.exe`, com um valor atribuído indicando o número de sua versão) e quatro bibliotecas auxiliares para o registro de modificações, desenho de contornos e para a geração de imagens tridimensionais (`dlog.dll`, `wrfrme.dll`, `render.dll` e `raytrce.dll`, respectivamente), todos utilizando os elementos-padrão da UML para executáveis e bibliotecas, respectivamente. Esse diagrama também apresenta as dependências entre os artefatos.



**Figura 26.4:**

A modelagem de executáveis e de bibliotecas

À medida que seus modelos se tornarem maiores, você descobrirá que muitos artefatos tendem a se reunir em grupos que estão relacionados conceitual e semanticamente. Na UML, os pacotes podem ser empregados para fazer a modelagem desses agrupamentos de artefatos.

► *Os pacotes são examinados no [Capítulo 12](#).*

No caso de sistemas maiores, a serem instalados em vários computadores, você desejará fazer a modelagem da forma como esses artefatos serão distribuídos, assinalando os nós em que eles estarão localizados.

► *A modelagem de implantação é examinada no [Capítulo 27](#).*

## MODELAGEM DE TABELAS, ARQUIVOS E DOCUMENTOS

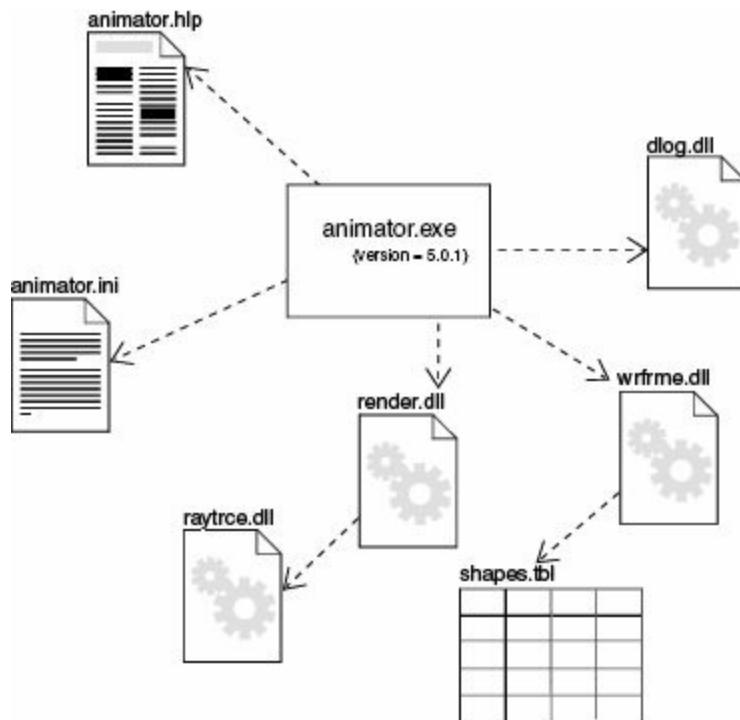
A modelagem dos executáveis e das bibliotecas que compõem a implementação física do sistema é útil, mas frequentemente você descobrirá a existência de alguns artefatos de implantação auxiliares que não são executáveis, nem bibliotecas, e ainda assim são críticos para a entrega física do sistema. Por exemplo, sua implementação poderá incluir arquivos de dados, documentos de ajuda, scripts, arquivos de registro, arquivos de

iniciação e arquivos de instalação/remoção. A modelagem desses artefatos é uma parte importante para controlar a configuração de seu sistema. Felizmente, você pode utilizar os artefatos da UML para fazer a modelagem de todos esses artefatos.

Para fazer a modelagem de tabelas, arquivos e documentos:

- » Identifique os artefatos que fazem parte da implementação física de seu sistema.
- » Faça a modelagem desses itens como artefatos. Se a sua implementação introduzir novos tipos de artefatos, inclua um novo estereótipo apropriado.
- » Conforme seja necessário para comunicar sua intenção, faça a modelagem dos relacionamentos existentes entre esses artefatos auxiliares e os demais executáveis, bibliotecas e interfaces do sistema. Com muita frequência, você desejará fazer a modelagem das dependências entre essas partes com a finalidade de visualizar o impacto da alteração.

Por exemplo, a [Figura 26.5](#) é elaborada a partir da figura anterior e mostra as tabelas, arquivos e documentos que fazem parte do sistema entregue ao redor do executável `animator.exe`. Essa figura inclui um documento de ajuda (`animator.hlp`), um arquivo de configuração simples (`animator.ini`) e uma tabela de banco de dados que contém formas geométricas (`shapes.tbl`). Esse exemplo ilustra alguns estereótipos definidos pelo usuário e ícones para artefatos.



**Figura 26.5:**

A modelagem de tabelas, arquivos e documentos

A modelagem de bancos de dados pode se tornar complicada quando você começar a lidar com várias tabelas, recursos de ativação e procedimentos de armazenamento. Para visualizar, especificar, construir e documentar essas características, será necessário fazer a modelagem do esquema lógico, assim como do banco de dados físico.

- As *modelagens lógica e física de bancos de dados* são examinadas nos Capítulos 8 e 30, respectivamente.

## MODELAGEM DE CÓDIGO-FONTE

O propósito mais comum para o qual os artefatos serão empregados é a modelagem das partes físicas que compõem sua implementação. Isso também inclui a modelagem de todas as partes auxiliares do sistema entregue, incluindo tabelas, arquivos, documentos e APIs. O segundo propósito mais comum para o uso dos artefatos é a modelagem da configuração de todos os arquivos de código-fonte que suas ferramentas de desenvolvimento usam

para criar esses artefatos. Esses representam os artefatos do produto do trabalho do processo de desenvolvimento.

A modelagem gráfica do código-fonte é particularmente útil para a visualização das dependências de compilação entre seus arquivos de código-fonte e para o gerenciamento da separação e reunião de grupos desses arquivos ao criar bifurcações e uniões de caminhos de desenvolvimento. Dessa maneira, os artefatos da UML podem ser interfaces gráficas para as ferramentas de gerenciamento da configuração e de controle de versão.

Na maioria dos sistemas, os arquivos de código-fonte são definidos a partir de decisões relacionadas ao modo como são segmentados os arquivos de que seu ambiente de desenvolvimento necessita. Esses arquivos são utilizados para armazenar os detalhes de suas classes, interfaces, colaborações e outros elementos lógicos como uma etapa intermediária para a criação dos artefatos binários, físicos, que são derivados desses elementos pelas suas ferramentas. Na maior parte do tempo, essas ferramentas deverão impor um estilo de organização (um ou dois arquivos por classe é comum), mas você ainda desejará visualizar os relacionamentos existentes entre esses arquivos. A maneira como você organiza grupos com esses arquivos utilizando pacotes e como gerencia as versões desses arquivos é orientada pelas suas decisões a respeito de como gerenciar as alterações.

Para fazer a modelagem de código-fonte:

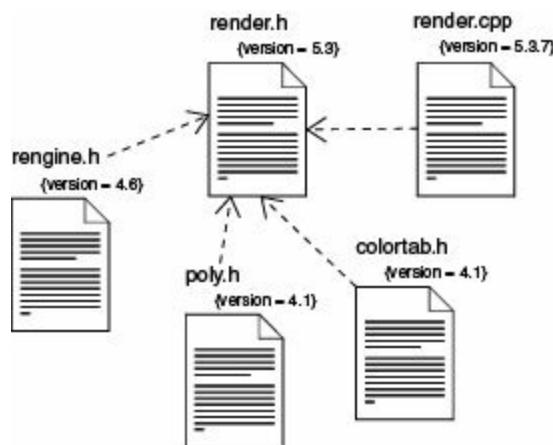
- » Dependendo das restrições impostas por suas ferramentas de desenvolvimento, modele os arquivos utilizados para o armazenamento de detalhes de todos os elementos lógicos, juntamente com as respectivas dependências de compilação.
- » Se é importante vincular esses modelos às suas ferramentas de gerenciamento de configuração e controle de versão, você desejará incluir valores atribuídos, como versão, autor e informações de verificação, para cada arquivo que se encontra sob o gerenciamento da configuração.

- › Tanto quanto possível, permita que suas ferramentas de desenvolvimento gerenciem os relacionamentos entre esses arquivos e use a UML somente para visualizar e documentar esses relacionamentos.

Por exemplo, a [Figura 26.6](#) mostra alguns arquivos de código-fonte que são utilizados para construir a biblioteca render.dll a partir dos exemplos anteriores. Essa figura inclui quatro arquivos de cabeçalho, dois relacionados à renderização da imagem, um aos polígonos que a compõem e outro às suas cores (render.h, rengine.h, poly.h e colortab.h), representando o código-fonte para a especificação de certas classes. Também existe um arquivo de implementação (render.cpp), representando a implementação de um desses cabeçalhos.

À medida que seus modelos forem ficando maiores, você descobrirá que muitos arquivos de código-fonte tendem a se reunir em grupos relacionados conceitual e semanticamente. Na maior parte do tempo, suas ferramentas de desenvolvimento colocarão esses grupos em diretórios separados. Na UML, os pacotes podem ser empregados para a modelagem desses agrupamentos de arquivos de código-fonte.

- ➡ Os pacotes são examinados no [Capítulo 12](#); os relacionamentos trace, um tipo de dependência, são examinados nos [Capítulos 5 e 10](#).



**Figura 26.6:**  
A modelagem de código-fonte

Na UML, é possível visualizar o relacionamento de uma classe com o respectivo arquivo de código-fonte e, por sua vez, o relacionamento entre um arquivo de código-fonte e seu executável ou biblioteca com a utilização de relacionamentos trace. Entretanto, raramente será necessário chegar a esses detalhes de modelagem.

## DICAS E SUGESTÕES

Ao fazer a modelagem de artefatos na UML, lembre-se de que você está modelando na dimensão física. Um artefato bem estruturado:

- » Implementa diretamente um conjunto de classes que trabalham juntas para a execução da semântica dessas interfaces com economia e elegância.
- » Está fracamente acoplado a outros artefatos.

CAPÍTULO

---

27

---

# Implantação

## Neste capítulo

- } Nós e conexões
- } Modelagem de processadores e dispositivos
- } Modelagem da distribuição de artefatos
- } Engenharia de sistemas

**A**ssim como os artefatos, os nós vivem no mundo material e são um bloco de construção importante para a modelagem dos aspectos físicos de um sistema. Um nó é um elemento físico que existe em tempo de execução e representa um recurso computacional, geralmente tendo pelo menos alguma memória e, frequentemente, capacidade de processamento.

Os nós são empregados para a modelagem da topologia do hardware em que o sistema é executado. Um nó tipicamente representa um processador ou um dispositivo em que os artefatos poderão ser instalados.

Bons nós representam o vocabulário do hardware disponível no domínio de sua solução.

## PRIMEIROS PASSOS

Os artefatos que você desenvolve ou reutiliza como parte de um sistema complexo de software devem ser instalados em algum conjunto de hardware para serem executados. Isso é, na verdade, o que compõe um *sistema* complexo de software – esse sistema abrange o software e o hardware.

- A modelagem de itens que não são software é examinada no Capítulo 4; as cinco visões de uma arquitetura são examinadas no [Capítulo 2](#).

Quando você define a arquitetura de um sistema complexo de software, você precisa considerar as dimensões lógica e física do sistema. Na parte lógica, você encontrará itens como classes, interfaces, colaborações, interações e máquinas de estados. Na parte física, encontrará os artefatos (que representam o pacote físico desses itens lógicos) e os nós (que representam o hardware em que esses artefatos são instalados e executados).

A UML fornece uma representação gráfica do nó, conforme mostra a [Figura 27.1](#). Essa notação canônica permite que você visualize um nó independente de qualquer hardware específico. Utilizando os estereótipos – um dos mecanismos de extensibilidade da UML –, você pode (e muitas vezes desejará) adequar essa notação para representar tipos específicos de processadores e dispositivos.

- Os estereótipos são examinados no [Capítulo 6](#).

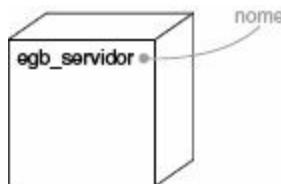


Figura 27.1:

Nós

**Nota:** A UML é principalmente destinada à modelagem de sistemas complexos de software, embora, em conjunto com as linguagens textuais de modelagem de hardware, como a VHDL, a UML pode ser bastante expressiva para a modelagem de sistemas de hardware. A UML também é suficientemente expressiva para a modelagem de topologias de sistemas isolados, embutidos, cliente/servidor e distribuídos.

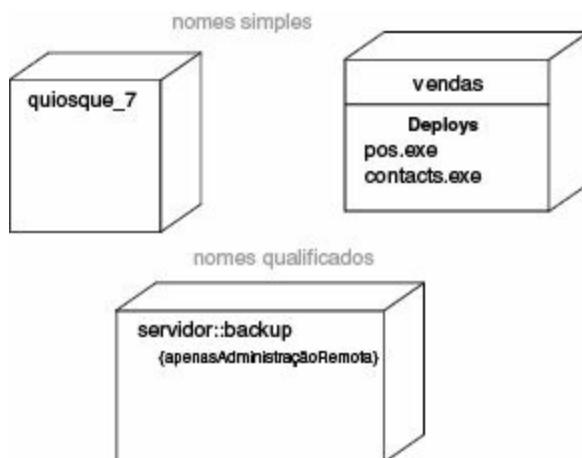
## TERMOS E CONCEITOS

Um *nó* é um elemento físico que existe em tempo de execução e representa um recurso computacional, geralmente tendo pelo menos alguma memória e, com frequência, capacidade de processamento. Graficamente, um nó é representado por um cubo.

## NOMES

Todo nó precisa ter um nome que o diferencie de outros nós. Um *nome* é uma sequência de caracteres textual. Esse nome sozinho é conhecido como um *nome simples*; um *nome qualificado* é o nome do nó prefixado pelo nome do pacote onde esse nó vive.

Um nó é tipicamente definido mostrando somente seu nome, como na [Figura 27.2](#). Assim como ocorre com as classes, você poderá definir os nós colocando, como adornos, valores atribuídos ou comportamentos adicionais para mostrar seus detalhes.



**Figura 27.2:**

Nós com nomes simples e qualificados

- *O nome de um nó deve ser único no pacote que o contém, conforme é examinado no [Capítulo 12](#).*

**Nota:** Um nome de um nó poderá ser um texto formado por qualquer quantidade de letras, números e certos sinais de pontuação (com exceção de sinais como dois-pontos, utilizados para separar o nome do nó e o nome do pacote que o contém) e

*poderá continuar por várias linhas. Na prática, os nomes de nós são expressões nominais ou nomes curtos, definidos a partir do vocabulário da implementação.*

## NÓS E ARTEFATOS

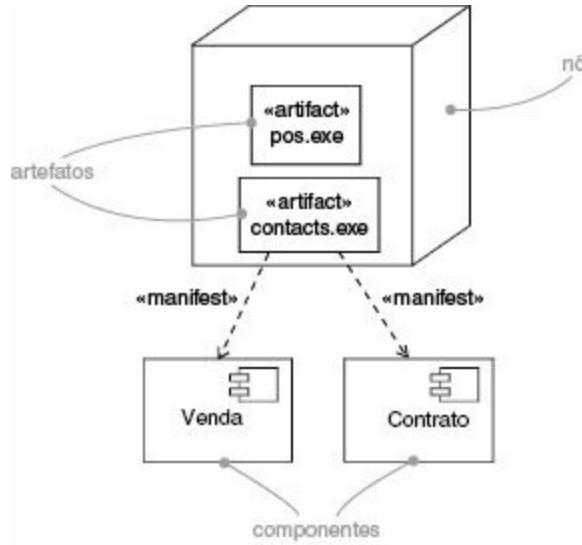
Sob vários aspectos, os nós são muito parecidos com os artefatos: ambos têm nomes; ambos podem participar de relacionamentos de dependência, generalização e associação; ambos podem ser aninhados; ambos podem ter instâncias; ambos podem ser participantes em interações. Entretanto, existem algumas diferenças significativas entre os nós e os artefatos.

» *Os artefatos são examinados no [Capítulo 26](#).*

- » Os artefatos são itens que participam da execução de um sistema; os nós são itens que executam os artefatos.
- » Os artefatos representam os pacotes físicos de elementos lógicos; os nós representam o funcionamento físico dos artefatos.

A primeira diferença é a mais importante. Dito de uma forma simples, os nós executam os artefatos; os artefatos são itens que são executados pelos nós.

A segunda diferença sugere um relacionamento entre classes, artefatos e nós. Em particular, o artefato é a materialização de um conjunto de outros elementos lógicos, como as classes e as colaborações, enquanto o nó é a localização em que os artefatos são instalados. Uma classe poderá ser implementada por um ou mais artefatos e, por sua vez, um artefato poderá ser instalado em um ou mais nós. Conforme mostra a [Figura 27.3](#), o relacionamento entre um nó e os artefatos instalados pode ser mostrado explicitamente pela utilização de um relacionamento de dependência. Na maior parte do tempo, você não precisará visualizar esses relacionamentos graficamente, mas os manterá como parte da especificação do nó, por exemplo, usando uma tabela.



**Figura 27.3:**

Os nós e os artefatos

- Os relacionamentos de dependência são examinados nos Capítulos 5 e 10.

Um conjunto de objetos ou artefatos que são alocados a um nó como um grupo é chamado de *unidade de distribuição*.

**Nota:** Os nós também são parecidos com as classes, pois é possível especificar atributos e operações para eles. Por exemplo, você poderá especificar que um nó fornecerá os atributos velocidadeDoProcessador e memória, além das operações ligar, desligar e suspender.

## ORGANIZAÇÃO DOS NÓS

Você pode organizar os nós agrupando-os em pacotes, da mesma maneira como organiza as classes e os artefatos.

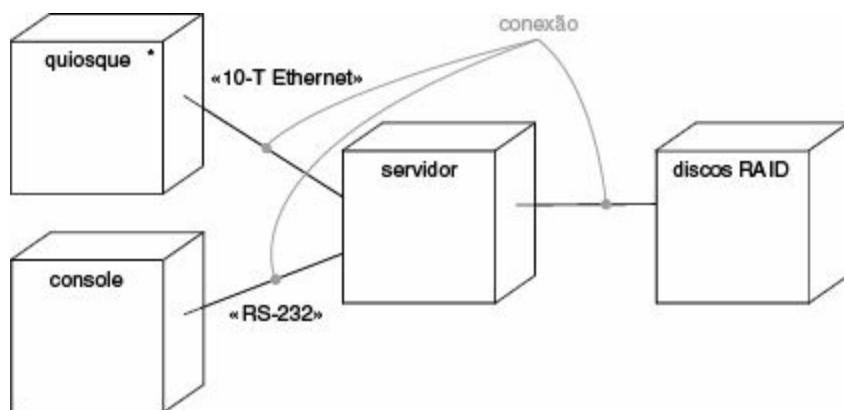
- Os pacotes são examinados no Capítulo 12.

Você também pode organizar os nós especificando relacionamentos de dependência, generalização e associação (incluindo agregação) entre eles.

- Os relacionamentos são examinados nos Capítulos 5 e 10.

## CONEXÕES

O tipo de relacionamento mais comum que você utilizará entre os nós é uma associação. Nesse contexto, uma associação representa uma conexão física entre os nós, assim como uma conexão de Ethernet, uma linha serial ou um barramento compartilhado, conforme mostra a [Figura 27.4](#). Você até pode usar as associações para fazer a modelagem de conexões indiretas, como uma ligação de satélite entre processadores distantes.



**Figura 27.4:**

As conexões

Como os nós são parecidos com as classes, você tem todo o poder das associações à sua disposição. Isso significa que pode incluir papéis, multiplicidade e restrições. Conforme mostra a figura anterior, você pode estereotipar essas associações, se quiser fazer a modelagem de novos tipos de conexões – por exemplo, para distinguir entre uma conexão Ethernet 10-T e uma conexão serial RS-232.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE PROCESSADORES E DISPOSITIVOS

A modelagem dos processadores e dispositivos que formam a topologia de sistemas isolados, embutidos, cliente/servidor ou distribuídos é a utilização mais comum dos nós.

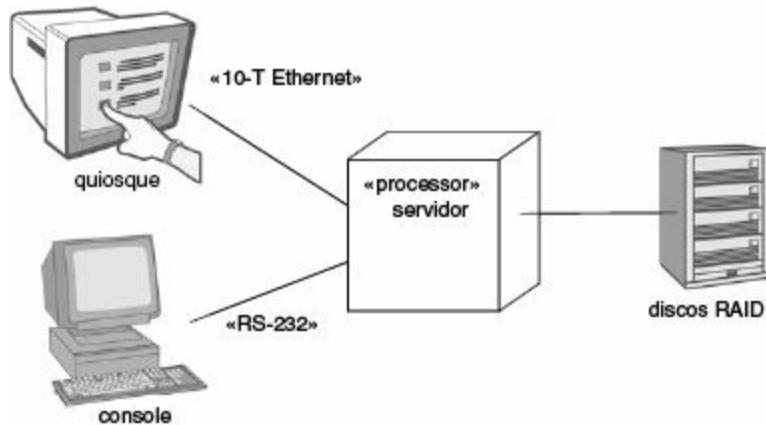
Como todos os mecanismos de extensibilidade da UML se aplicam aos nós, com frequência você desejará utilizar os estereótipos para especificar novos tipos de nós que podem ser utilizados para representar tipos específicos de processadores e dispositivos. Um *processador* é um nó, indicado pelo termo «processor», que tem capacidade de processamento, significando que ele pode executar um componente. Um *dispositivo* é um nó que não tem capacidade de processamento (pelo menos, nenhum dos que são modelados nesse nível de abstração) e, em geral, representa algo como interfaces para o mundo real.

- ➡ Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).

Para fazer a modelagem de processadores e dispositivos:

- › Identifique os elementos computacionais da visão de implantação de seu sistema e modele cada um como um nó.
- › Se esses elementos representarem processadores e dispositivos genéricos, então devem ser assim estereotipados. Se esses tipos de processadores e de dispositivos fazem parte do vocabulário de seu domínio, então especifique um estereótipo apropriado com um ícone para cada um deles.
- › Assim como na modelagem de classes, considere os atributos e operações que poderão ser aplicados a cada nó.

Por exemplo, a [Figura 27.5](#) pega o diagrama anterior e estereotipa cada nó. O servidor é um nó estereotipado como um processador genérico «processor»; o quiosque e o console são nós estereotipados como tipos especiais de processadores e o conjunto de discos RAID é um nó estereotipado como um tipo especial de dispositivo.



**Figura 27.5:**

Processadores e dispositivos

**Nota:** Os nós provavelmente são os blocos de construção mais estereotipados disponíveis na UML. Quando, como parte da engenharia de sistemas, você modela a visão de implantação de um sistema complexo de software, é de grande valor fornecer indicações visuais capazes de comunicar à audiência pretendida. Se você estiver fazendo a modelagem de um processador que seja um tipo comum de computador, represente-o com um ícone parecido com um computador. Se estiver fazendo a modelagem de um dispositivo comum, como um telefone celular, fax, modem ou câmera, represente-o com um ícone parecido com o respectivo dispositivo.

## MODELAGEM DA DISTRIBUIÇÃO DE ARTEFATOS

Quando você faz a modelagem da topologia de um sistema, costuma ser de grande ajuda visualizar ou especificar a distribuição física de seus artefatos pelos processadores e dispositivos que compõem o sistema.

► A semântica de localização é examinada no [Capítulo 24](#).

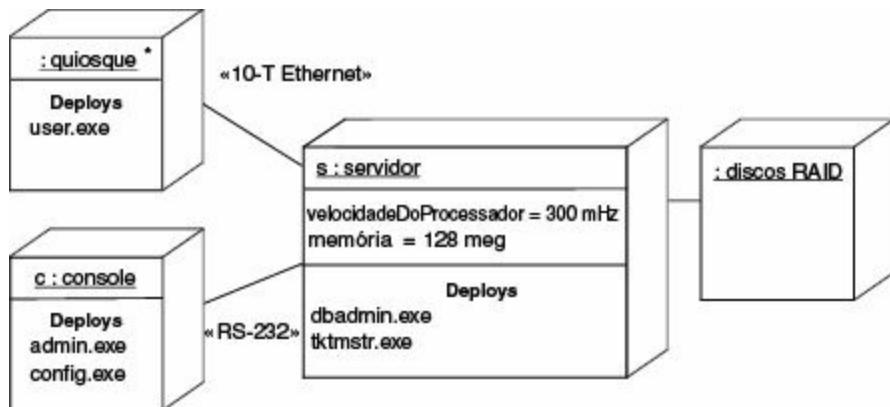
Para fazer a modelagem da distribuição dos artefatos:

- › Para cada artefato significativo existente em seu sistema, faça sua alocação para um determinado nó.
- › Considere as localizações duplicadas para os artefatos. Não é incomum para o mesmo tipo de artefato (como executáveis e bibliotecas específicas) residir em vários nós simultaneamente.

» Represente essa alocação em uma de três maneiras.

1. Não torne a alocação visível, mas deixe-a como parte da base de seu modelo – ou seja, na especificação de cada nó.
2. Utilizando relacionamentos de dependência, conecte cada nó com os artefatos nele instalados.
3. Liste os artefatos instalados em um nó em um compartimento adicional.

Utilizando a terceira solução, a [Figura 27.6](#) pega os diagramas anteriores e especifica os artefatos executáveis que residem em cada nó. Esse diagrama é um pouco diferente dos anteriores por ser um diagrama de objeto, em que são visualizadas instâncias específicas de cada nó. Nesse caso, as instâncias discos RAID e quiosque são anônimas e as outras duas instâncias são nomeadas (*c* para o console e *s* para o servidor). Cada processador mostrado nessa figura está representado com um compartimento adicional exibindo o artefato por ele instalado. O objeto servidor também é representado com seus atributos (*velocidadeDoProcessador* e *memória*) e seus valores visíveis. O compartimento de implantação pode mostrar uma lista textual de nomes de artefatos, ou pode mostrar símbolos de artefatos aninhados.



**Figura 27.6:**

A modelagem da distribuição de artefatos

- As instâncias são examinadas no [Capítulo 3](#); os diagramas de objetos são apresentados no [Capítulo 14](#).

## DICAS E SUGESTÕES

Um nó bem estruturado:

- › Fornece uma clara abstração de algum item definido a partir do vocabulário do hardware existente em sua solução de domínio.
- › É decomposto apenas no nível necessário para transmitir sua intenção ao leitor.
- › Expõe somente aqueles atributos e operações relevantes para o domínio que está sendo modelado.
- › Instala diretamente um conjunto de artefatos que residem no nó.
- › É conectado a outros nós, refletindo a topologia do sistema do mundo real.

Ao definir um nó na UML:

- › Para seu projeto ou organização como um todo, defina um conjunto de estereótipos com os ícones apropriados para fornecer indicações visuais significativas para seus leitores.
- › Mostre somente os atributos e operações (se houver) necessários para a compreensão do significado do nó em um determinado contexto.

CAPÍTULO

---

**28**

---

# Colaborações

## Neste capítulo

- » *Colaborações, realizações e interações*
- » *Modelagem da realização de um caso de uso*
- » *Modelagem da realização de uma operação*
- » *Modelagem de um mecanismo*
- » *Reificação de interações*

**N**o contexto da arquitetura de um sistema, uma colaboração permite nomear um agrupamento conceitual que abrange aspectos estáticos e dinâmicos. Uma colaboração nomeia uma sociedade de classes, interfaces e outros elementos que trabalham em conjunto para fornecer algum comportamento cooperativo maior do que a soma de todas as suas partes.

As colaborações são empregadas para especificar a realização de casos de uso e operações e para fazer a modelagem de mecanismos significativos da arquitetura do seu sistema.

## PRIMEIROS PASSOS

Pense sobre a construção mais bonita que você já viu – talvez o Taj Mahal ou a Catedral de Notre-Dame. Essas duas estruturas exibem uma qualidade que é difícil definir. De muitas maneiras, as duas estruturas são simples em termos de arquitetura, mas são também profundamente intensas. Em cada uma, pode-se reconhecer imediatamente uma simetria consistente. Observe atentamente e verá detalhes que são por si próprios maravilhosos e que

funcionam em conjunto para produzir uma beleza e funcionalidade maiores do que suas partes individuais.

Agora pense sobre a construção mais feia que você já viu – talvez uma lanchonete local. Nela você encontrará uma cacofonia visual de estilos de arquitetura – um toque de modernismo combinado com um teto georgiano, toda decorada de uma maneira chocante, com cores fortes que agredem os olhos. Normalmente, essas construções são pura manipulação, com pouca funcionalidade e sem uma forma bem definida.

Qual é a diferença entre esses dois tipos de arquitetura civil? Primeiro, nas construções de qualidade, você encontrará uma harmonia de projeto que está ausente em outras. A arquitetura de qualidade utiliza um pequeno conjunto de estilos aplicados de maneira consistente. Por exemplo, o Taj Mahal utiliza elementos geométricos complexos, simétricos e equilibrados em toda parte. Segundo, nas construções de qualidade, você encontrará padrões comuns que transcendem os elementos individuais da construção. Por exemplo, na Notre-Dame, algumas paredes são muito resistentes e servem para suportar a cúpula da catedral. Já algumas dessas mesmas paredes, juntamente com outros detalhes da arquitetura, servem como parte do sistema da construção para distribuir água e esgoto.

O mesmo ocorre com o software. Um sistema complexo de software de qualidade não somente parece funcional, mas também exibe uma harmonia e equilíbrio de projeto que o tornam flexível a modificações. Essa harmonia e esse equilíbrio muitas vezes provêm do fato de que todos os sistemas orientados a objeto bem estruturados são cheios de padrões. Observe alguns sistemas orientados a objeto de qualidade e você verá elementos que trabalham em conjunto para proporcionar algum comportamento cooperativo maior do que a soma de todas as suas partes. Nos sistemas bem estruturados, muitos dos elementos, em várias combinações, participarão de diferentes mecanismos.

► As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#).

**Nota:** Um padrão produz uma solução comum para um problema comum em algum contexto. Em qualquer sistema bem estruturado, você encontrará um espectro de padrões, incluindo idiomas (representando maneiras comuns de programação), mecanismos (padrões de projeto que representam partes conceituais da arquitetura de um sistema) e frameworks (padrões de arquitetura que fornecem templates extensíveis a aplicações dentro de um domínio).

► Padrões e frameworks são examinados no [Capítulo 29](#).

Na UML, os mecanismos são modelados, utilizando as colaborações. A colaboração fornece um nome para os blocos de construção conceituais do sistema, abrangendo tanto os elementos estruturais, como os comportamentais. Por exemplo, você poderá ter um sistema de informações de gerenciamento distribuído cujos bancos de dados estão espalhados em vários nós. A partir da perspectiva do usuário, a atualização das informações parece atômica; a partir da perspectiva interna, ela não é tão simples, pois, como uma ação, precisa alcançar várias máquinas. Para dar a ilusão de simplicidade, você planejará um mecanismo de transação com o qual o cliente poderá nomear o que parece uma transação atômica e simples, ainda que se estenda por vários bancos de dados. Esse mecanismo poderá ser dividido em múltiplas classes trabalhando em conjunto para a execução de uma transação. Muitas dessas classes também estariam envolvidas em outros mecanismos, tais como mecanismos para tornar as informações persistentes. Essa coleção de classes (a parte estrutural) juntamente com suas interações (a parte comportamental) formam um mecanismo que, na UML, pode ser representado como uma colaboração.

► A modelagem estrutural é examinada nas Partes 2 e 3; a modelagem comportamental é apresentada nas Partes 4 e 5; as interações são explicadas no [Capítulo 16](#).

As colaborações não apenas nomeiam os mecanismos do sistema, mas também servem como a realização de casos de uso e operações.

- ➡ Os casos de uso são examinados no [Capítulo 17](#); as operações são apresentadas nos [Capítulos 4 e 9](#).

A UML fornece uma representação gráfica para as colaborações, conforme mostra a [Figura 28.1](#), representando mensagens de segurança trocadas entre os nós. Essa notação permite que você visualize os blocos de construção estrutural e comportamental de um sistema, especialmente como eles podem sobrepor as classes, interfaces e outros elementos do sistema.



**Figura 28.1:**  
Colaborações

**Nota:** Essa notação permite que você visualize externamente uma colaboração como um grupo. O que costuma ser mais interessante é o que se encontra no interior dessa notação. Observe uma colaboração e você será conduzido a outros diagramas – principalmente, diagramas de classes (para a parte estrutural da colaboração) e diagramas de interação (para a parte comportamental da colaboração).

- ➡ Os diagramas de classes são examinados no [Capítulo 8](#); os diagramas de interação são apresentados no [Capítulo 19](#).

## TERMOS E CONCEITOS

Uma *colaboração* é uma sociedade de classes, interfaces e outros elementos que trabalham em conjunto para fornecer algum comportamento cooperativo maior do que a soma de todas as suas partes. Uma colaboração também é a especificação do modo como um elemento, tal como um classificador (incluindo uma classe, interface, componente, nó ou caso de uso) ou uma operação, é realizada por um conjunto de classificadores e associações desempenhando papéis específicos, utilizados de uma maneira específica. Graficamente, uma colaboração é representada como uma elipse com linhas tracejadas.

- ➡ A notação para as colaborações é intencionalmente semelhante aos casos de uso, conforme é examinado no [Capítulo 17](#).

## NOMES

Toda colaboração deve ter um nome que a diferencie de outras colaborações. Um *nome* é uma sequência de caracteres textual. Esse nome sozinho é conhecido como um *nome simples*; um *nome de caminho* é o nome da colaboração prefixado pelo nome do pacote em que a colaboração vive. Tipicamente, uma colaboração é definida somente mostrando seu nome, como na figura anterior.

- ➡ O nome de uma colaboração deve ser único dentro de seu pacote, conforme é examinado no [Capítulo 12](#).

**Nota:** Um nome de uma colaboração poderá ser um texto contendo qualquer número de letras, números e certos sinais de pontuação (com exceção de sinais como dois-pontos, utilizados para separar o nome da colaboração e o nome de seu pacote) e poderá continuar por várias linhas. Na prática, os nomes de colaborações são expressões nominais ou nomes curtos, definidos a partir do vocabulário do sistema que está sendo modelado. Tipicamente, a primeira letra do nome da colaboração é maiúscula, como em Transação ou Cadeia de responsabilidade.

## ESTRUTURA

As colaborações têm dois aspectos: uma parte estrutural que especifica as classes, interfaces e outros elementos que trabalham em conjunto para executar a colaboração nomeada e a parte comportamental que especifica a dinâmica de como esses elementos interagem.

- ➡ Os elementos estruturais são examinados nas Partes 2 e 3.

A parte estrutural de uma colaboração poderá incluir qualquer combinação de classificadores, como classes, interfaces, componentes e nós. Dentro da colaboração, esses classificadores poderão ser organizados usando todos os relacionamentos usuais da UML, incluindo associações,

generalizações e dependências. De fato, os aspectos estruturais de uma colaboração poderão usar toda a gama de facilidades da modelagem estrutural disponível na UML.

- ➡ Os classificadores são examinados no [Capítulo 9](#); os relacionamentos são apresentados nos [Capítulos 5 e 10](#); a estrutura interna é apresentada no [Capítulo 15](#); os pacotes são examinados no [Capítulo 12](#); os subsistemas são examinados no [Capítulo 32](#); os casos de uso são examinados no [Capítulo 17](#).

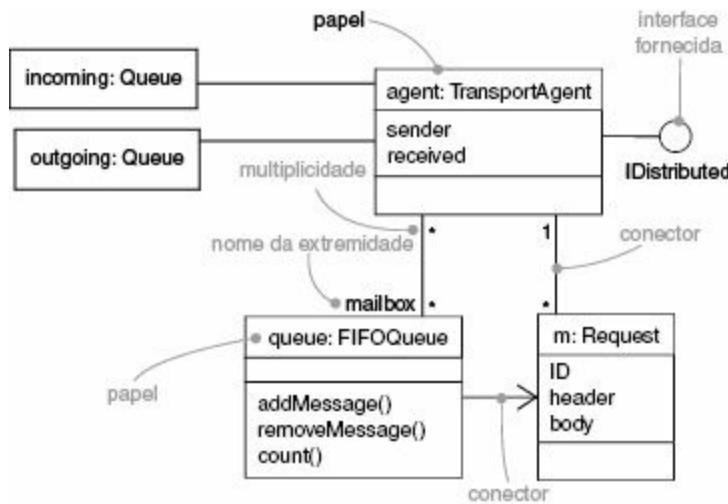
Entretanto, ao contrário dos pacotes ou subsistemas, uma colaboração não possui nenhum de seus elementos estruturais. Em vez disso, uma colaboração simplesmente referencia ou usa as classes, interfaces, componentes, nós e outros elementos estruturais que são declarados em outra parte. É por isso que a colaboração nomeia um grupo conceitual – e não um grupo físico da arquitetura do sistema. Portanto, as colaborações poderão reduzir muitos níveis em um sistema. Além disso, o mesmo elemento poderá aparecer em mais de uma colaboração (e alguns elementos não serão nomeados como parte de qualquer colaboração).

Por exemplo, considerando um sistema de vendas baseado na Web descrito por uma dúzia ou mais de casos de uso (como Adquirir Itens, Devolver Itens e Pesquisar Pedido), cada caso de uso será realizado por uma única colaboração. Além disso, cada uma dessas colaborações compartilhará alguns dos mesmos elementos estruturais (assim como as classes Cliente e Pedido), mas eles serão organizados de maneiras diferentes. Você também encontrará colaborações mais profundas no sistema, que representam mecanismos significativos em termos de arquitetura. Por exemplo, nesse mesmo sistema de vendas, poderá haver uma colaboração chamada Internode messaging, especificando os detalhes das mensagens de segurança trocadas entre os nós.

Em uma colaboração que nomeia um grupo conceitual de um sistema, você pode observar que a colaboração expõe os detalhes estruturais de suas partes. Por exemplo, a [Figura 28.2](#) ilustra como o detalhamento da

colaboração Internode messaging poderá revelar o seguinte conjunto de classes, representado em um diagrama de classes.

- Os diagramas de classe são examinados no [Capítulo 8](#).



**Figura 28.2:**

Os aspectos estruturais de uma colaboração

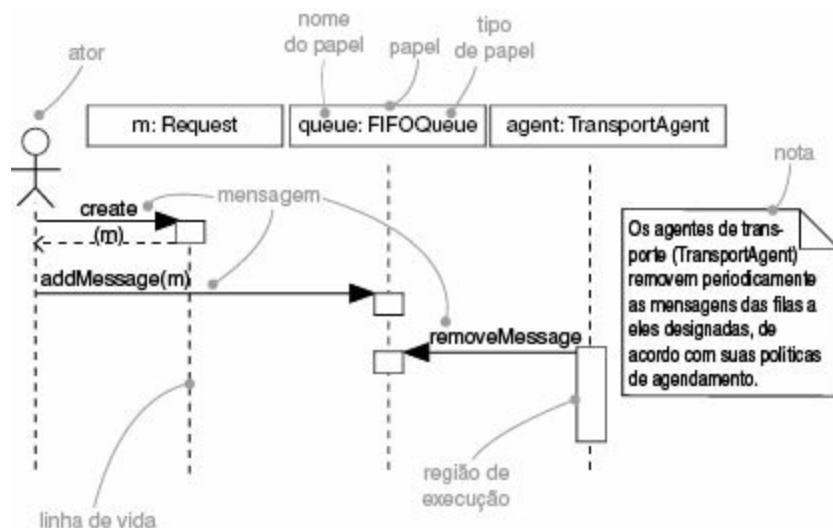
## COMPORTAMENTO

Enquanto a parte estrutural de uma colaboração é tipicamente representada pela utilização de um diagrama de classes, a parte comportamental da colaboração é tipicamente representada pela utilização de um diagrama de interação. O diagrama de interação especifica uma interação que representa um comportamento composto por um conjunto de mensagens que são trocadas em um conjunto de objetos dentro de um contexto para a realização de um propósito específico. O contexto da interação é fornecido pela colaboração que a contém, estabelecendo as classes, interfaces, componentes, nós e outros elementos estruturais cujas instâncias poderão participar da interação.

- Os diagramas de interação são examinados no [Capítulo 19](#); as instâncias são apresentadas no [Capítulo 13](#); a estrutura composta aparece no [Capítulo 15](#).

A parte comportamental de uma colaboração poderá ser especificada por um ou mais diagramas de interação. Se você quiser dar ênfase à ordenação temporal das mensagens, utilize o diagrama de sequências. Se preferir dar ênfase aos relacionamentos estruturais entre esses objetos ao realizarem colaborações, use o diagrama de colaboração. Cada diagrama é apropriado, porque, para a maioria dos propósitos, eles são semanticamente equivalentes.

Isso significa que, ao fazer a modelagem de uma sociedade de classes nomeando sua interação como uma colaboração, você pode ampliar essa colaboração para expor os detalhes de seu comportamento. Por exemplo, detalhando a colaboração nomeada Internode messaging, poderá revelar o diagrama de interação mostrado na [Figura 28.3](#).



**Figura 28.3:**

Os aspectos comportamentais de uma colaboração

**Nota:** As partes comportamentais de uma colaboração devem ser coerentes com suas partes estruturais. Isso significa que os objetos encontrados nas interações da colaboração devem ser instâncias de classes encontradas em sua parte estrutural. De maneira semelhante, as mensagens nomeadas na interação devem estar relacionadas a operações visíveis na parte estrutural da colaboração. Pode haver mais de uma interação associada à colaboração, cada qual podendo mostrar um aspecto diferente – mas consistente – de seu comportamento.

## ORGANIZAÇÃO DAS COLABORAÇÕES

O coração da arquitetura de um sistema se encontra em suas colaborações, porque os mecanismos que formam o sistema representam decisões de projeto significativas. Todos os sistemas orientados a objeto bem estruturados também são compostos por um conjunto, regular e de tamanho modesto, dessas colaborações; por isso, é importante que você organize bem suas colaborações. Existem dois tipos de relacionamentos referentes às colaborações que você precisará levar em consideração.

Primeiro, existe o relacionamento entre a colaboração e o item que a realiza. Uma colaboração poderá realizar um classificador ou uma operação, o que significa que a colaboração especifica as realizações estruturais e comportamentais desse classificador ou dessa operação. Por exemplo, um caso de uso (que nomeia um conjunto de sequências de ações executadas pelo sistema) poderá ser realizado por uma colaboração. Esse caso de uso, incluindo seus atores associados e casos de uso vizinhos, fornece um contexto para a colaboração. De modo semelhante, uma operação (que nomeia a implementação de um serviço) poderá ser realizada por uma colaboração. Essa operação, incluindo seus parâmetros e possível valor retornado, também fornece um contexto para a colaboração. O relacionamento entre um caso de uso ou uma operação e a colaboração que os realiza é modelado como um relacionamento de realização.

- *Os casos de usos são examinados no [Capítulo 17](#); as operações são apresentadas nos [Capítulos 4 e 9](#); as realizações dos relacionamentos são explicadas no [Capítulo 10](#).*

**Nota:** Uma colaboração poderá realizar qualquer tipo de classificador, incluindo classes, casos de uso, interfaces, componentes e nós. Uma colaboração que modela um mecanismo do sistema também poderá aparecer sozinha; portanto, seu contexto será o sistema como um todo.

- *Os classificadores são examinados no [Capítulo 9](#).*

Segundo, existe o relacionamento entre as colaborações. As colaborações podem refinar outras colaborações e você também poderá fazer a modelagem

desse relacionamento como um refinamento. Os relacionamentos aprimorados entre as colaborações tipicamente refletem o refinamento dos relacionamentos entre os casos de uso que eles representam.

A Figura 28.4 ilustra esses dois tipos de relacionamentos.

**Nota:** As colaborações, assim como qualquer outro elemento de modelagem da UML, poderão ser agrupadas em pacotes maiores. Tipicamente, você apenas precisará fazer isso no caso de sistemas muito grandes.

► Os pacotes são examinados no [Capítulo 12](#).



**Figura 28.4:**

A organização de colaborações

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE PAPÉIS

Os objetos representam indivíduos únicos em uma situação ou execução. Eles são úteis em exemplos concretos, mas, na maior parte do tempo, é desejável mostrar partes gerais em algum contexto. Uma parte em um contexto chama-se *papel*. Talvez a função mais importante para os papéis seja a modelagem de interações dinâmicas. Quando você modela tais interações, geralmente não modela instâncias concretas que existem no

mundo real. Em vez disso, modela papéis em um padrão reutilizável no qual os papéis são essencialmente agentes ou substitutos de objetos que aparecerão em instâncias individuais do padrão. Por exemplo, para modelar as maneiras como os objetos reagem a um evento do mouse em aplicativos com janelas, você desenharia um diagrama de interação contendo papéis cujos tipos seriam janelas, eventos e manipuladores.

Para fazer a modelagem de papéis:

- » Identifique um contexto no qual vários objetos interagem.
- » Identifique os papéis necessários e suficientes para visualizar, especificar, construir ou documentar o contexto que está sendo modelado.
- » Represente esses papéis na UML como papéis em um contexto estruturado. Quando possível, dê um nome a cada papel. Se não houver um nome significativo para o papel, represente-o como um papel anônimo.
- » Exponha as propriedades de cada papel necessário e suficiente para modelar o seu contexto.
- » Represente esses papéis e seus relacionamentos em um diagrama de interação ou de classes.

**Nota:** A diferença semântica entre objetos concretos e papéis é sutil, mas não é difícil. Para ser preciso, um papel da UML é uma parte predefinida de um classificador estruturado, como uma classe estruturada ou colaboração. Um papel não é um objeto, mas uma descrição; está ligado a um valor em cada instância de um classificador estruturado. Um papel, portanto, corresponde a muitos valores possíveis, exatamente como um atributo. Objetos concretos aparecem em exemplos específicos, tais como diagramas de objetos, diagramas de componentes e diagramas de implantação. Os papéis aparecem em descrições genéricas como diagramas de interação e diagramas de atividades.

A [Figura 28.5](#) mostra um diagrama de interação que ilustra um cenário parcial para iniciar uma chamada telefônica no contexto de um comutador. Há quatro papéis: a (um AgenteDeChamada), c (uma Conexão) e t1 e t2 (instâncias de

Terminal). Esses quatro papéis representam agentes conceituais de objetos concretos que podem existir no mundo real.

- Os diagramas de interação são apresentados no [Capítulo 19](#); os diagramas de atividades aparecem no [Capítulo 20](#).

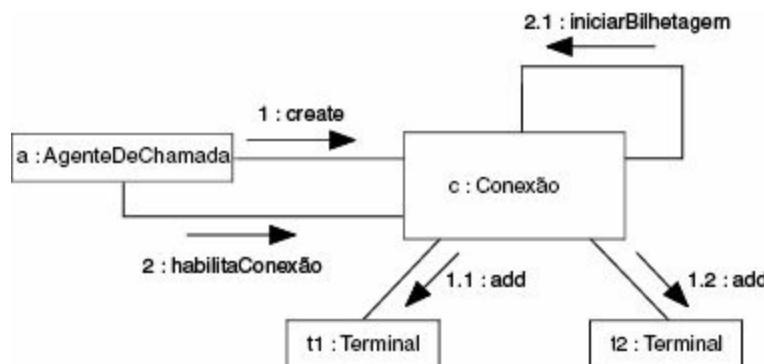


Figura 28.5:

Modelagem de papéis

**Nota:** Esse exemplo é uma colaboração, que representa uma sociedade de objetos e outros elementos que trabalham juntos para fornecer um comportamento cooperativo maior que a soma de todos os elementos. As colaborações têm dois aspectos – um estrutural (que representa os papéis do classificador e seus relacionamentos) e um dinâmico (que representa as interações entre as instâncias prototípicas).

## MODELAGEM DA REALIZAÇÃO DE UM CASO DE USO

Um dos propósitos para os quais as colaborações serão empregadas é a modelagem da realização de um caso de uso. Você tipicamente conduzirá a análise de seu sistema pela identificação dos casos de uso, mas quando finalmente retornar à implementação, será necessário realizar esses casos de uso com estruturas e comportamentos concretos. Em geral, todo caso de uso pode ser realizado por uma ou mais colaborações. Para o sistema como um todo, os classificadores envolvidos em uma determinada colaboração vinculada a um caso de uso também poderão participar de outras colaborações. Dessa maneira, os conteúdos estruturais das colaborações tendem a se sobrepor uns aos outros.

► Os casos de uso são examinados no [Capítulo 17](#).

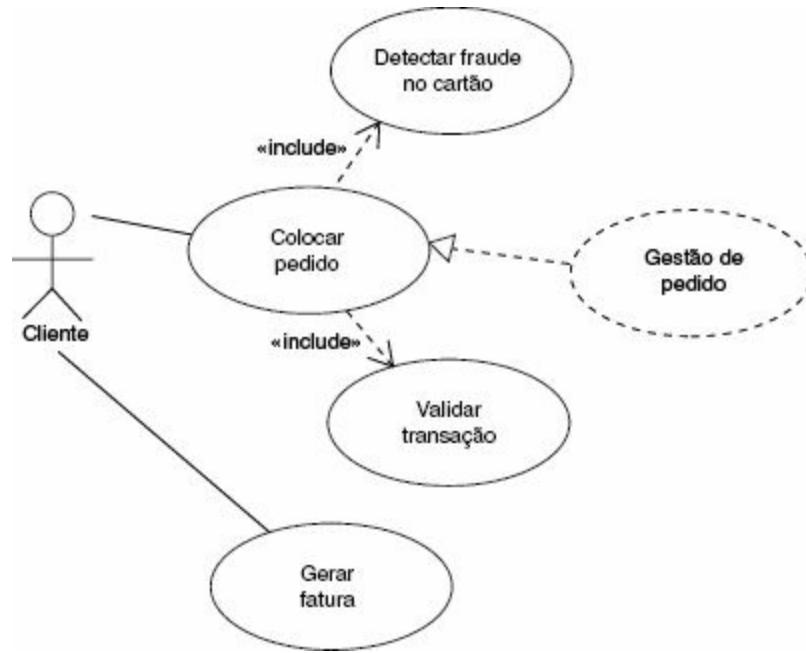
Para fazer a modelagem da realização de um caso de uso:

- » Identifique os elementos estruturais necessários e suficientes para a execução da semântica do caso de uso.
- » Capture a organização desses elementos estruturais em diagramas de classes.
- » Considere os cenários individuais que representam esse caso de uso. Cada cenário representa um caminho específico no caso de uso.
- » Capture a dinâmica desses cenários em diagramas de interação. Utilize os diagramas de sequências, caso queira dar ênfase à ordenação temporal das mensagens. Utilize os diagramas de colaboração, caso prefira dar ênfase aos relacionamentos estruturais entre os objetos à medida que colaboram.
- » Organize esses elementos estruturais e comportamentais como uma colaboração que você possa conectar ao caso de uso por intermédio da realização.

Por exemplo, a [Figura 28.6](#) mostra um conjunto de casos de uso definido a partir de um sistema de validação de cartão de crédito, incluindo os casos de uso primários Colocar pedido e Gerar pedido, juntamente com dois outros casos de uso subordinados, Detectar fraude no cartão e Validar transação. Embora na maior parte do tempo você não precise fazer a modelagem desse relacionamento explicitamente (mas deixará isso a cargo de suas ferramentas), essa figura modela claramente a realização de Colocar pedido pela colaboração Gestão de pedido. Por sua vez, essa colaboração pode ser expandida em seus aspectos estruturais e comportamentais, levando você a diagramas de classes e a diagramas de interação. É por meio da realização de relacionamentos que você conecta um caso de uso aos respectivos cenários.

Em muitos casos, você não precisará fazer a modelagem do relacionamento entre um caso de uso e a colaboração que o realiza

explicitamente. Em vez disso, você tenderá a deixar isso na base do seu modelo. Então, permita que determinadas ferramentas usem a conexão para ajudá-lo a navegar entre um caso de uso e sua realização.



**Figura 28.6:**

A modelagem da realização de um caso de uso

## MODELAGEM DA REALIZAÇÃO DE UMA OPERAÇÃO

Outro propósito para o qual as colaborações serão empregadas é a modelagem da realização de uma operação. Em muitos casos, você pode especificar a realização de uma operação indo direto ao código. Entretanto, para as operações que requerem a colaboração de vários objetos, é melhor fazer a modelagem de sua implementação por intermédio de colaborações, antes de se aprofundar em um código.

- As operações são examinadas nos Capítulos 4 e 9.

**Nota:** Você também pode modelar uma operação, usando os diagramas de atividades. Os diagramas de atividades são essencialmente fluxogramas. Assim, para as operações que fazem uso intensivo de algoritmos e cuja modelagem você deseja fazer explicitamente, os diagramas de atividades normalmente são a melhor escolha. Entretanto, caso sua operação requeira a participação de muitos objetos,

*você precisará utilizar as colaborações, porque elas permitirão modelar os aspectos estruturais, bem como os comportamentais de uma operação.*

- » *Os diagramas de atividades são examinados no [Capítulo 20](#).*

Os parâmetros, os valores retornados e os objetos locais de uma operação fornecem o contexto para sua realização. Portanto, esses elementos são visíveis para os aspectos estruturais da colaboração que realiza a operação, assim como os atores são visíveis para os aspectos estruturais da colaboração que realiza um caso de uso. Você pode modelar o relacionamento entre essas partes usando os diagramas de classes que especificam a parte estrutural da colaboração

Para fazer a modelagem da implementação de uma operação:

- » Identifique os parâmetros, valor retornado e outros objetos visíveis para a operação. Eles se tornam os papéis da colaboração.
- » Se a operação é trivial, represente sua implementação diretamente no código, que pode ser mantido na base de seu modelo ou visualizado explicitamente em uma nota.
- » Caso a operação utilize algoritmos intensivamente, o modelo de sua realização deve usar um diagrama de atividade.
- » Se a operação é complexa ou, por outro lado, exigir algum trabalho de projeto detalhado, represente sua implementação como uma colaboração. Você pode expandir mais as partes estruturais e comportamentais dessa colaboração, usando os diagramas de classes e de interação, respectivamente.

- » *As notas são examinadas no [Capítulo 6](#).*

Por exemplo, a [Figura 28.7](#) mostra a classe ativa RenderFrame, responsável pela renderização de quadros de imagens, com três de suas operações sendo exibidas. A função progress, que mostra o progresso da renderização, é simples o bastante para ser implementada diretamente no código, conforme está

especificado na nota anexa. Entretanto, a operação `render` é muito mais complicada, por isso sua implementação é realizada pela colaboração `Ray trace`, que tratará da montagem dos objetos tridimensionais.

- ➡ As classes ativas são examinadas no [Capítulo 23](#).

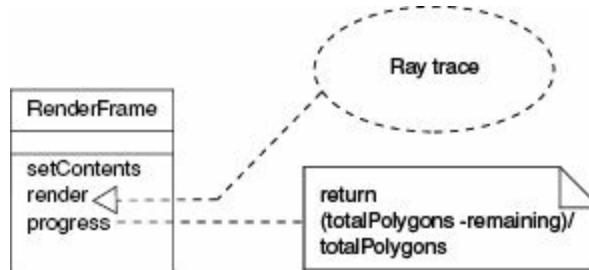


Figura 28.7

A modelagem da realização de uma operação

## MODELAGEM DE UM MECANISMO

Em todos os sistemas orientados a objeto bem estruturados, você encontrará um espectro de padrões. Em uma extremidade, encontrará os idiomas que representam os padrões de uso da linguagem de implementação. Na outra extremidade, encontrará os padrões de arquitetura e os frameworks que formam o sistema como um todo e impõem um estilo particular. No centro, encontrará os mecanismos que representam padrões de projeto comuns por meio dos quais os itens do sistema interagem uns com os outros de maneira comum. Você pode representar um mecanismo na UML como uma colaboração.

- ➡ Padrões e frameworks são examinados no [Capítulo 29](#); um exemplo da modelagem de um mecanismo é examinado no mesmo capítulo.

Os mecanismos são colaborações que aparecem sozinhas; seu contexto não é um único caso de uso ou operação, mas o sistema como um todo. Qualquer elemento visível nessa parte do sistema é um candidato a participar em um mecanismo.

Mecanismos como esses representam decisões de projeto significativas em termos de arquitetura e não poderão ser tratadas de modo superficial. Tipicamente, a arquitetura do seu sistema indicará seus mecanismos e você os incluirá em cada nova versão. No final, você considerará seu sistema simples (porque esses mecanismos reificam interações comuns), compreensível (porque você poderá ter acesso ao sistema a partir de seus mecanismos) e flexível (ajustando cada mecanismo, você ajustará o sistema como um todo).

Para fazer a modelagem de um mecanismo:

- › Identifique os principais mecanismos que formam a arquitetura de seu sistema. Esses mecanismos são orientados pelo estilo de arquitetura que você escolheu para impor à sua implementação, juntamente com o estilo apropriado para o domínio de seu problema.
- › Represente cada um desses mecanismos como uma colaboração.
- › Procure expandir a parte estrutural e comportamental de cada colaboração. Procure compartilhamentos, onde for possível.
- › Valide esses mecanismos no início do ciclo de vida do desenvolvimento (eles são de importância estratégica), mas evolua-os em cada nova versão, à medida que aprender mais sobre os detalhes de sua implementação.

## DICAS E SUGESTÕES

Ao fazer a modelagem de colaborações na UML, lembre-se de que toda colaboração poderá representar a realização de um caso de uso ou operação ou deverá aparecer sozinha como um mecanismo do sistema. Uma colaboração bem estruturada:

- › É composta por aspectos estruturais e comportamentais.
- › Fornece uma clara abstração de alguma interação identificável no sistema.

- › Raramente é independente por completo, mas poderá se sobrepor aos elementos estruturais de outras colaborações.
- › É compreensível e simples.

Ao definir uma colaboração na UML:

- › Represente uma colaboração explicitamente, apenas quando for necessário para a compreensão de seu relacionamento com outras colaborações, classificadores, operações ou o sistema como um todo. Caso contrário, utilize as colaborações, mas mantenha-as na base.
- › Organize as colaborações de acordo com o classificador ou a operação que elas representam ou em pacotes associados ao sistema como um todo.

CAPÍTULO

---

**29**

---

# Padrões e Frameworks

## Neste capítulo

- » *Padrões e frameworks*
- » *A modelagem de padrões de projeto*
- » *A modelagem de padrões de arquitetura*
- » *Criando padrões acessíveis*

Todos os sistemas bem estruturados são repletos de padrões. Um padrão fornece uma solução comum para um problema básico em um determinado contexto. Um mecanismo é um padrão de projeto aplicado à sociedade de classes; um framework é tipicamente um padrão de arquitetura que fornece um template extensível para aplicações em um domínio.

Os padrões são utilizados para especificar os mecanismos e frameworks que compõem a arquitetura do sistema. Um padrão acessível é criado pela clara identificação de conexões, guias, botões e indicadores que o usuário desse padrão poderá ajustar aplicando o padrão em um contexto em particular.

## PRIMEIROS PASSOS

É surpreendente pensar nas várias maneiras como você pode juntar uma pilha de madeira para construir uma casa. Nas mãos de um empreiteiro de São Francisco, você poderá ver essa pilha transformada em uma casa no estilo vitoriano, com um telhado ornamentado e alegremente colorido, como em um livro de histórias. Nas mãos de um empreiteiro no Maine, poderá ver

a mesma pilha transformada em uma casa rústica, com as paredes reforçadas com tábuas retangulares por toda parte.

Externamente, essas duas casas representam claramente estilos diferentes de arquitetura. Todo construtor, a partir de sua experiência, deve escolher o estilo que melhor atenda às necessidades de seu cliente e então adaptar esse estilo aos desejos do cliente e às restrições do terreno e local convenientes.

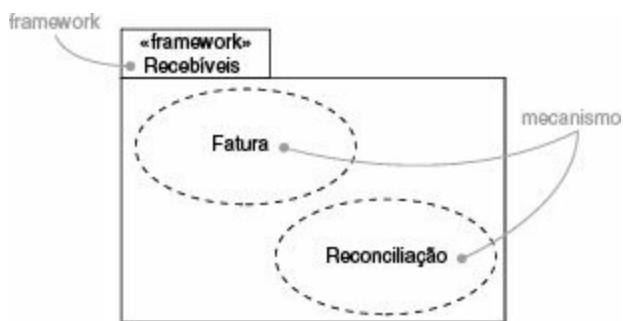
Internamente, cada construtor também deve projetar a casa para solucionar alguns problemas comuns. Existem inúmeras maneiras de erguer uma estrutura para sustentar um telhado; existem inúmeras maneiras de projetar paredes fortes que também devem conter aberturas para portas e janelas. Cada construtor deve selecionar os mecanismos apropriados que resolvam esses problemas comuns, adaptados a um estilo geral de arquitetura e aos códigos locais para edificações.

A construção de um sistema complexo de software é semelhante. Cada vez que você erguer os olhos acima das linhas de código individuais, encontrará mecanismos comuns que determinam a maneira como você organiza suas classes e outras abstrações. Por exemplo, em um sistema orientado a eventos, a utilização do padrão de projeto de cadeia de responsabilidades é um modo comum de organizar a manipulação de eventos. Levante seus olhos acima do nível desses mecanismos e encontrará os frameworks comuns que formam toda a arquitetura de seu sistema. Por exemplo, em sistemas de informações, a utilização da arquitetura em três níveis é uma maneira comum de conseguir uma clara separação de questões entre a interface para o usuário do sistema, suas informações persistentes e suas regras e objetos de negócio.

Na UML, você tipicamente modelará padrões de projeto – também chamados de mecanismos –, que podem ser representados como colaborações. De modo semelhante, você modelará padrões de arquitetura como frameworks, que podem ser representados como pacotes estereotipados.

- As colaborações são examinadas no [Capítulo 28](#); os pacotes são apresentados no [Capítulo 12](#).

A UML fornece uma representação gráfica para os dois tipos de padrões, conforme mostra a [Figura 29.1](#).



**Figura 29.1:**  
Mecanismos e frameworks

---

## TERMOS E CONCEITOS

Um *padrão* é uma solução comum para um problema básico em um determinado contexto. Um *mecanismo* é um padrão de projeto aplicado a uma sociedade de classes. Um *framework* é um padrão de arquitetura que fornece um template extensível a aplicações dentro de um domínio.

## PADRÕES DE ARQUITETURA

Se estiver projetando um novo sistema ou desenvolvendo um já existente, você nunca realmente estará começando do nada. Em vez disso, a experiência e as convenções o levarão a aplicar formas comuns para resolver problemas básicos. Por exemplo, se estiver construindo um sistema de uso intensivo, uma maneira comprovada para organizar suas abstrações é a utilização de um padrão modelo-visão-controlador (em inglês model-view-controller ou, simplesmente, MVC). Esse modelo permite separar com clareza os objetos (o modelo) de sua apresentação (a visão) e os agentes que mantêm os dois em sincronia (o controlador). De modo semelhante, se estiver construindo um

sistema para solucionar criptogramas, uma maneira comprovada de organizar seu sistema é utilizar uma arquitetura Blackboard (tratada mais adiante neste capítulo), que é bastante apropriada para abordar problemas intratáveis de formas oportunistas.

► *A arquitetura de software é examinada no [Capítulo 2](#).*

Esses são dois exemplos de padrões – soluções comuns para problemas básicos em um determinado contexto. Em todos os sistemas bem estruturados, você encontrará uma grande quantidade de padrões em vários níveis de abstração. Padrões de projeto especificam a estrutura e o comportamento de uma sociedade de classes; os padrões de arquitetura especificam a estrutura e o comportamento de um sistema inteiro.

Os padrões são parte da UML simplesmente porque são partes importantes do vocabulário dos desenvolvedores. Criando padrões explicitamente em seu sistema, você torna seu sistema mais compreensível e fácil de desenvolver e manter. Por exemplo, se você receber um novo corpo de código bruto para ampliar, você quebrará a cabeça por alguns instantes tentando compreender como tudo se encaixa nesse código. Por outro lado, se receber o mesmo corpo de código com a instrução “Essas classes colaboram usando um mecanismo publicar- e-inscrever,” você estará bem mais adiante no caminho para compreender como ele funciona. A mesma ideia se aplica ao sistema como um todo. Dizendo “Esse sistema é organizado como um conjunto de encadeamentos (pipes) e filtros”, explica-se uma grande parte da arquitetura do sistema que, ao contrário, seria difícil compreender apenas pela observação de classes individuais.

Os padrões ajudam você a visualizar, especificar, construir e documentar os artefatos de um sistema complexo de software. Você pode usar a engenharia direta para um sistema, selecionando um conjunto de padrões apropriados e aplicando-os às abstrações específicas do seu domínio. Também pode usar a engenharia reversa para um sistema, descobrindo os padrões nele existentes, embora dificilmente isso seja um processo perfeito.

Ainda melhor, ao entregar um sistema, você pode especificar os padrões nele existentes para que, mais tarde, quando alguém quiser reutilizar ou adaptar esse sistema, seus padrões sejam manifestados com clareza.

Na prática, existem dois tipos de padrões de interesse – os padrões de projeto e os frameworks –, e a UML fornece um meio para modelar cada um deles. Ao fazer a modelagem de outro padrão, você descobrirá que tipicamente ele aparece sozinho no contexto de algum pacote maior, com exceção dos relacionamentos de dependência ligando-os a outras partes do sistema.

## MECANISMOS

Um mecanismo é apenas outro nome para um padrão de projeto aplicado a uma sociedade de classes. Por exemplo, um problema comum de projeto que você encontrará em Java é a adaptação de uma classe que sabe como responder a um certo conjunto de eventos, de modo que ela passe a responder a um conjunto ligeiramente diferente, sem alterar a classe original. Uma solução comum para esse problema é o padrão adaptador, um padrão de projeto estrutural que converte uma interface em outra. Esse padrão é tão comum que faz sentido nomeá-lo e depois modelá-lo para poder utilizá-lo a qualquer momento que encontre um problema semelhante.

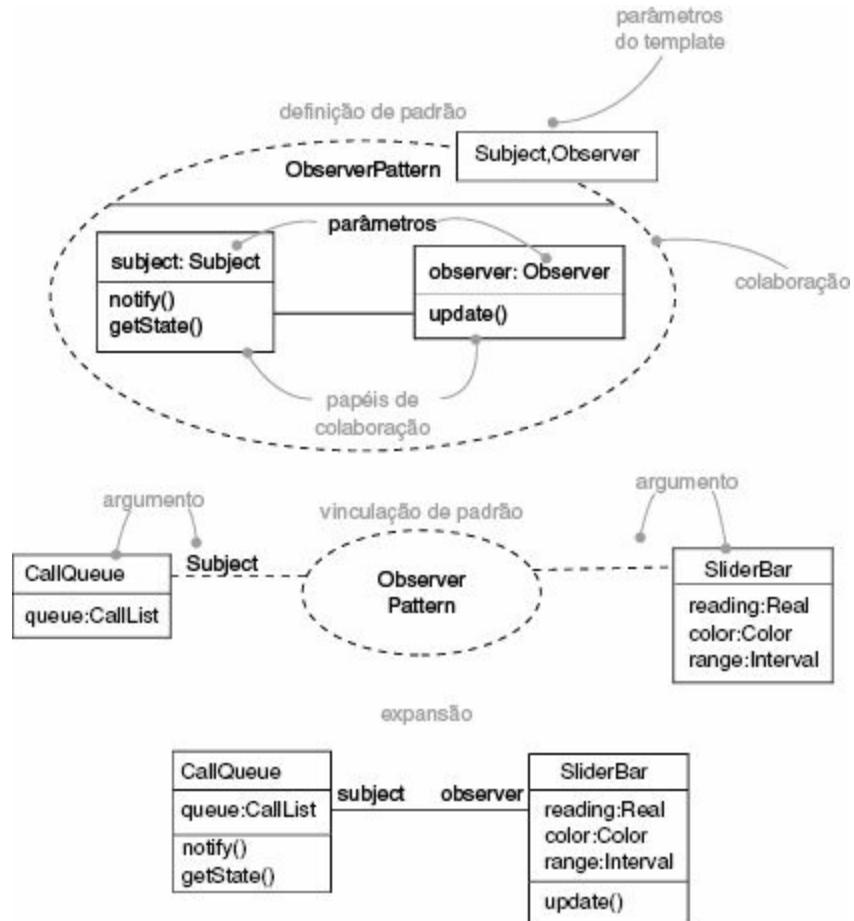
Na modelagem, esses mecanismos são mostrados de duas maneiras.

Primeiro, conforme é mostrado na figura anterior, um mecanismo simplesmente nomeia um conjunto de abstrações que trabalham juntas para alcançar algum comportamento comum e relevante. Esses mecanismos são modelados como colaborações plenas, porque apenas nomeiam uma sociedade de classes. Observe detalhadamente essa colaboração e verá seus aspectos estruturais (tipicamente representados como diagramas de classes), além de seus aspectos comportamentais (tipicamente representados como diagramas de interação). Colaborações como essas recortam abstrações individuais do sistema; uma determinada classe provavelmente será um membro de muitas colaborações.

► As colaborações são examinadas no [Capítulo 28](#).

Segundo, conforme mostra a [Figura 29.2](#), um mecanismo nomeia um template para um conjunto de abstrações que trabalham juntas para alcançar algum comportamento comum e relevante. Esses mecanismos são modelados como colaborações parametrizadas, representadas na UML de modo semelhante à representação utilizada para classes template. Observe detalhadamente essa colaboração e verá seus aspectos estruturais e comportamentais. Observe essa colaboração de uma maneira geral e verá como esse padrão se aplica ao sistema pela vinculação de partes template da colaboração às abstrações existentes em seu sistema. Ao fazer a modelagem de um mecanismo como uma colaboração parametrizada, você identifica os conectores, guias, botões e mostradores utilizados para adaptar esse padrão por meio de seus parâmetros template. Colaborações como essas poderão aparecer repetidamente em seu sistema, vinculadas a diferentes conjuntos de abstrações. Nesse exemplo, o Subject e o Observer (Sujeito e Observador) do padrão são vinculados às classes concretas que representam a fila de tarefas e a barra de rolagem para a sua seleção: CallQueue e SliderBar, respectivamente.

► As classes template são examinadas no [Capítulo 9](#).



**Figura 29.2:**

Os mecanismos

**Nota:** A decisão de fazer a modelagem de um mecanismo como uma colaboração plena em oposição a uma parametrizada é algo objetivo. Utilize uma colaboração plena, se tudo o que você estiver fazendo for nomear uma sociedade de classes específica que trabalha em conjunto em seu sistema; use uma colaboração template, se puder abstrair os aspectos estrutural e comportamental essenciais do mecanismo de uma maneira completamente independente do domínio, que você então poderá vincular às suas abstrações em um determinado contexto.

## FRAMEWORKS

Um framework é um padrão de arquitetura que fornece um template extensível para aplicações dentro de um domínio. Por exemplo, um padrão de arquitetura comum que você encontrará em sistemas em tempo real é uma execução cíclica, que divide o tempo em frames e subframes, durante os

quais o processamento ocorre de acordo com prazos restritos. A escolha desse padrão em oposição à sua alternativa (uma arquitetura orientada a eventos) colorirá o sistema inteiro. Como esse padrão (e sua alternativa) é comum, faz sentido nomeá-lo como um framework.

Um framework é maior do que um mecanismo. De fato, você pode pensar em um framework como um tipo de microarquitetura abrangendo um conjunto de mecanismos que trabalham juntos para resolver um problema básico de um domínio comum. Ao especificar um framework, você especifica o esqueleto da arquitetura, juntamente com os conectores, guias, botões e indicadores que você expõe aos usuários que desejam adaptar esse framework ao seu próprio contexto.

- As cinco visões de uma arquitetura são examinadas no [Capítulo 2](#).

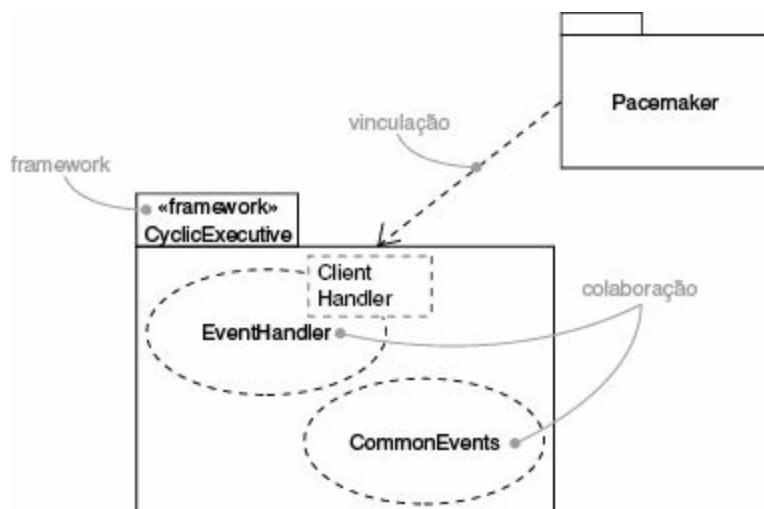
Na UML, a modelagem de um framework é realizada como um pacote estereotipado. Observe detalhadamente esse pacote e verá os mecanismos existentes em qualquer uma das várias visões da arquitetura do sistema. Por exemplo, você não somente poderá encontrar colaborações parametrizadas, como também casos de uso (que explicam como utilizar o framework), além de colaborações plenas (que fornecem conjuntos de abstrações que você pode usar para construir – por exemplo, por meio de subclasses).

- Os pacotes são examinados no [Capítulo 12](#); os estereótipos são apresentados no [Capítulo 6](#).

A [Figura 29.3](#) ilustra um desses frameworks, nomeado CyclicExecutive. Entre outras coisas, esse framework inclui uma colaboração (CommonEvents) abrangendo um conjunto de classes comuns de eventos, juntamente com um mecanismo (EventHandler) para o tratamento desses eventos em forma cíclica. Um cliente que utiliza esse framework (como Pacemaker, um marca-passo) poderá construir as abstrações em CommonEvents por meio de subclasses e também poderá aplicar uma instância do mecanismo EventHandler.

► Os eventos são examinados no [Capítulo 21](#).

**Nota:** Os frameworks podem ser diferenciados pelas bibliotecas de classes plenas. Uma biblioteca de classes contém abstrações que são instanciadas ou chamadas pelas suas abstrações; um framework contém abstrações que poderão instanciar ou chamar suas abstrações. Esses tipos de conexões que constituem o framework são os conectores, guias, botões e indicadores que você deve ajustar para adaptar o framework ao seu contexto.



**Figura 29.3:**  
Frameworks

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE PADRÕES DE PROJETO

Uma coisa para a qual você utilizará os padrões é a modelagem de um padrão de projeto. Ao fazer a modelagem de um mecanismo como esse, é preciso levar em consideração a perspectiva interna, como também a externa.

Quando visualizado a partir da perspectiva externa, o padrão de projeto é representado como uma colaboração parametrizada. Sendo uma colaboração, o padrão fornece um conjunto de abstrações cuja estrutura e comportamento trabalham em conjunto para conduzir a alguma função útil. Os parâmetros da colaboração nomeiam os elementos que um usuário desse padrão deve vincular. Isso faz do padrão de projeto um template a ser utilizado em um

contexto particular, fornecendo os elementos que atendem aos parâmetros do template.

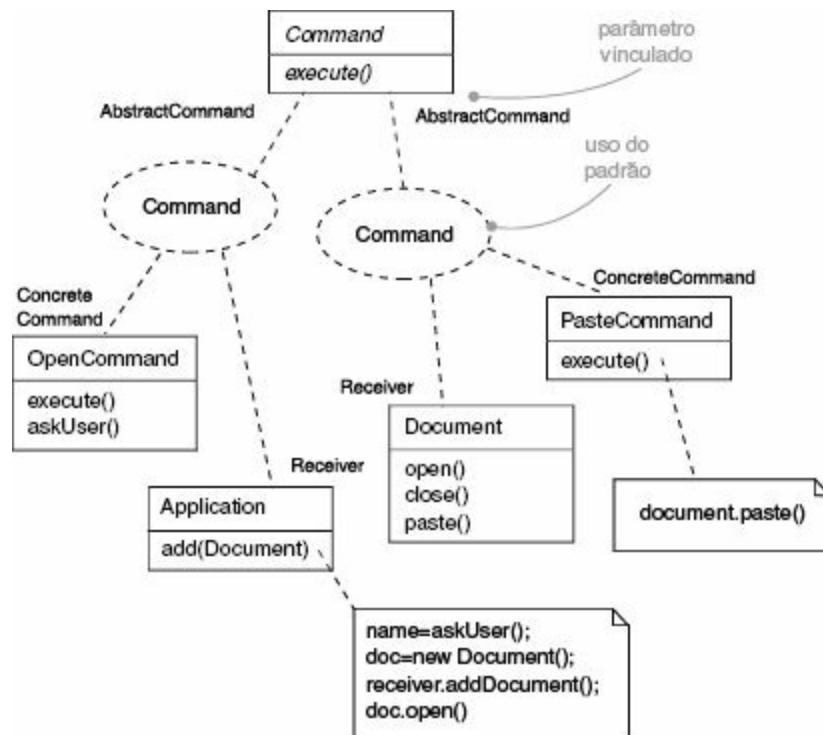
Quando visualizado a partir da perspectiva interna, o padrão de projeto é simplesmente uma colaboração representada com suas partes estruturais e comportamentais. Tipicamente, você pode fazer a modelagem interna dessa colaboração com um conjunto de diagramas de classes (do aspecto estrutural) e um conjunto de interações (do aspecto comportamental). Os parâmetros da colaboração nomeiam inúmeros desses elementos estruturais, que, se o padrão de projeto está vinculado a um contexto particular, são instanciados usando abstrações a partir desse contexto.

Para fazer a modelagem de um padrão de projeto:

- » Identifique a solução comum para o problema básico e reifique-a como um mecanismo.
  - » Faça a modelagem do mecanismo como uma colaboração, fornecendo seus aspectos estruturais, assim como os aspectos comportamentais.
  - » Identifique os elementos do padrão de projeto que devem ser vinculados aos elementos em um contexto específico e represente-os como parâmetros para a colaboração.
- ➡ A utilização de colaborações para a modelagem de um mecanismo é examinada no [Capítulo 28](#).

Por exemplo, a [Figura 29.4](#) mostra o uso do padrão de projeto Command (conforme é apresentado em Gamma et al., *Design Patterns*, Reading, Massachusetts: Addison-Wesley, 1995). Conforme sua documentação afirma, esse padrão “encapsula uma solicitação como um objeto, permitindo assim que sejam parametrizados clientes com diferentes solicitações, filas ou solicitações de registro, além de dispor de suporte para operações que não possam ser realizadas”. Conforme o modelo indica, esse padrão de projeto tem quatro parâmetros que, no momento em que você aplica o padrão, devem estar vinculados aos elementos em um determinado contexto. Esse modelo

mostra essa vinculação, em que PasteCommand e OpenCommand estão vinculados aos parâmetros do padrão de projeto.



**Figura 29.4:**

A modelagem de um padrão de projeto

Os parâmetros são o AbstractCommand, que deve estar vinculado a alguma superclasse abstrata em qualquer caso; o ConcreteCommand, que está vinculado às várias classes específicas em vários projetos; e o Receiver, que está vinculado à classe na qual o comando funciona. A classe Command pode ser criada pelo padrão, mas torná-la um parâmetro permite a criação de várias hierarquias de comando.

Observe que PasteCommand e OpenCommand são subclasses da classe Command. Muito provavelmente, seu sistema usará esse padrão várias vezes, talvez com vínculos diferentes. A habilidade para reutilizar um padrão de projeto semelhante a esse como um elemento de modelagem de primeira classe é o que faz o desenvolvimento com padrões tão poderoso.

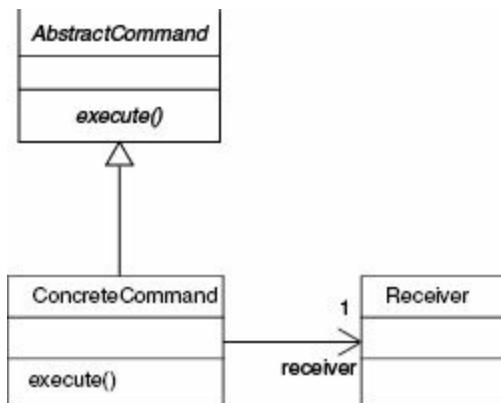
Para completar seu modelo de um padrão de projeto, você deve especificar suas partes estruturais, assim como as comportamentais, que

representam a parte interna da colaboração.

- As colaborações são examinadas no [Capítulo 28](#); os diagramas de classe são apresentados no [Capítulo 8](#); os diagramas de interação são explicados no [Capítulo 19](#).

Por exemplo, a [Figura 29.5](#) mostra um diagrama de classes que representa a estrutura desse padrão de projeto. Observe como esse diagrama usa classes nomeadas como parâmetros para o padrão.

A [Figura 29.6](#) mostra um diagrama de sequência que representa o comportamento desse padrão de projeto. Observe que o diagrama só sugere possibilidades; um padrão de projeto não é rígido.

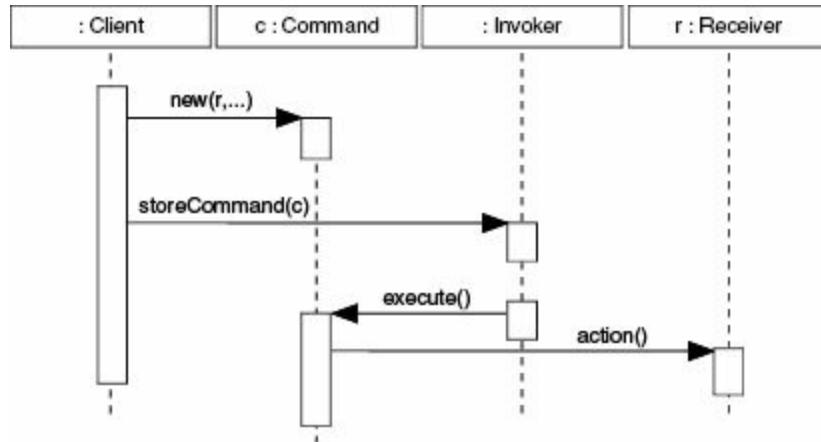


**Figura 29.5:**

A modelagem do aspecto estrutural de um padrão de projeto

## MODELAGEM DE PADRÕES DE ARQUITETURA

A outra coisa para a qual você utilizará os padrões é a modelagem dos padrões de arquitetura. Ao fazer a modelagem desse framework, você está, de fato, fazendo a modelagem da infraestrutura de uma arquitetura inteira que você planeja reutilizar e adaptar a algum contexto.



**Figura 29.6:**

A modelagem do aspecto comportamental de um padrão de projeto

Um framework é representado como um pacote estereotipado. Como um pacote, um framework fornece um conjunto de elementos, incluindo – mas certamente não limitado a – classes, interfaces, casos de uso, componentes, nós, colaborações e até outros frameworks. De fato, você poderá colocar em um framework todas as abstrações que trabalham em conjunto com a finalidade de fornecer um template extensível para aplicações dentro de um domínio. Alguns desses elementos poderão ser públicos e representarão recursos que os clientes podem construir. Esses são os “guias” do framework que você pode conectar às abstrações em seu contexto. Alguns desses elementos públicos poderão ser padrões de projeto e representarão recursos aos quais os clientes serão vinculados. Esses são os “conectores” do framework que você preenche quando vincula ao padrão de projeto. Por fim, alguns desses elementos poderão ser protegidos ou privados e representarão elementos encapsulados do framework que estarão ocultos a partir da perspectiva externa.

► Os pacotes são examinados no [Capítulo 12](#).

Quando você fizer a modelagem de um padrão de arquitetura, lembre-se de que um framework é, de fato, uma descrição de uma arquitetura, embora incompleta e possivelmente parametrizada. Dessa maneira, tudo o que você conhece sobre a modelagem de uma arquitetura bem estruturada se aplica à

modelagem de frameworks bem estruturados. Os melhores frameworks não são projetados isoladamente; fazer isso seria uma garantia de falha. Portanto, os melhores frameworks são encontrados nas arquiteturas existentes que comprovadamente funcionam e os frameworks evoluem para encontrar conectores, guias, botões e indicadores necessários e suficientes para tornar esse framework adaptável a outros domínios.

» A arquitetura de software é examinada no [Capítulo 2](#).

Para fazer a modelagem de um padrão de arquitetura:

- » Encontre o framework a partir de uma arquitetura existente e comprovada.
- » Faça a modelagem do framework como um pacote estereotipado, contendo todos os elementos (e especialmente os padrões de projeto) necessários e suficientes para descrever as várias visões desse framework.
- » Exponha os conectores, guias, botões e indicadores necessários para adaptar o framework sob a forma de padrões de projeto e de colaborações. Na maior parte, isso significa deixar claro aos usuários do padrão que as classes devem ser estendidas, que as operações devem ser implementadas e que os sinais devem ser manipulados.

Por exemplo, a [Figura 29.7](#) mostra a especificação do padrão de arquitetura Blackboard (conforme é discutido em Buschmann, et al., *Pattern-Oriented Software Architecture*, Nova York, Nova York: Wiley, 1996). Conforme sua documentação afirma, esse padrão “aborda problemas que não têm uma possível solução para a transformação do dado bruto em estruturas de dados de alto nível”. O coração dessa arquitetura é o padrão de projeto Blackboard, que determina como KnowledgeSources, Blackboard e Controller colaboram. Esse framework também inclui o padrão de projeto Reasoning engine, que especifica um mecanismo geral a respeito de como cada KnowledgeSource é orientado. Por fim, conforme a figura mostra, esse framework expõe um caso

de uso, *Apply new knowledge sources*, que explica para um cliente como fazer para adaptar o próprio framework.

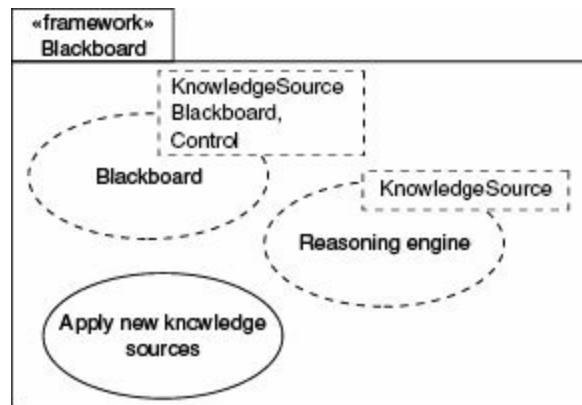


Figura 29.7:

A modelagem de um padrão de arquitetura

**Nota:** Na prática, a modelagem completa de um framework não é uma tarefa muito diferente da modelagem completa da arquitetura de um sistema. De alguma forma, a tarefa é ainda mais difícil, pois, para tornar os frameworks acessíveis, é necessário também expor os conectores, guias, botões e indicadores do framework e, talvez, produzir metacasos de uso (como *Apply new knowledge sources*) que explicam como fazer para adaptar o framework, além de casos de uso plenos, que explicam como o framework se comporta.

## DICAS E SUGESTÕES

Ao fazer a modelagem de padrões na UML, lembre-se de que eles trabalham em muitos níveis de abstração, desde classes individuais até a forma do sistema como um todo. Os tipos de padrões mais interessantes são os mecanismos e os frameworks. Um padrão bem estruturado:

- › Resolve um problema básico de uma maneira comum.
- › É formado por aspectos estruturais e comportamentais.
- › Expõe conectores, guias, botões e indicadores, aos quais você adapta esses aspectos para aplicá-los a algum contexto.

- » É atômico, significando que não é facilmente dividido em pequenos padrões.
- » Tem a tendência a recortar abstrações individuais no sistema.

Ao definir um padrão na UML:

- » Exiba os elementos do padrão que você deve adaptar para aplicá-lo a um contexto.
- » Torne o padrão acessível, fornecendo casos de uso a serem aplicados, além de adaptados, ao padrão.

CAPÍTULO

---

30

---

# Diagramas de Artefatos

## Neste capítulo

- » Modelagem de código-fonte
- » Modelagem de versões executáveis
- » Modelagem de bancos de dados físicos
- » Modelagem de sistemas adaptáveis
- » Engenharia direta e reversa

**O**s diagramas de artefatos são um dos dois tipos de diagramas disponíveis para a modelagem de aspectos físicos de sistemas orientados a objetos. O diagrama de artefatos mostra a organização e as dependências existentes entre um conjunto de artefatos.

► *Os diagramas de implantação, o segundo tipo de diagrama utilizado para a modelagem de aspectos físicos de um sistema orientado a objetos, são examinados no Capítulo 31.*

Os diagramas de artefatos são empregados para a modelagem da visão estática de implementação de um sistema. Isso envolve a modelagem de itens físicos que residem em um nó, como executáveis, bibliotecas, tabelas, arquivos e documentos. Os diagramas de artefatos são essencialmente diagramas de classes que focalizam os artefatos de um sistema.

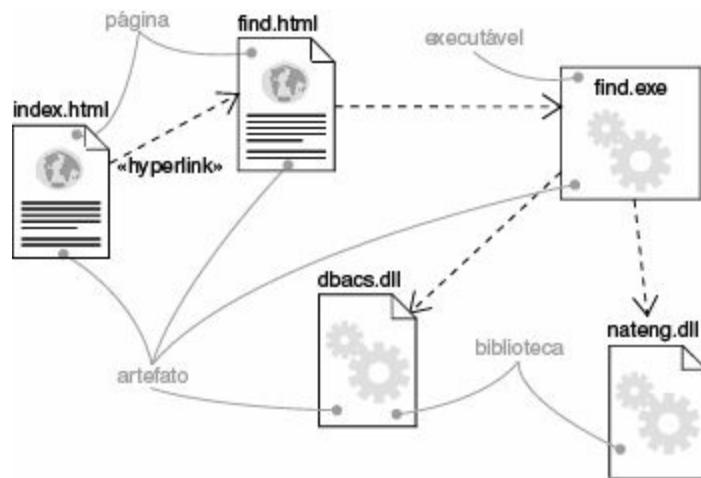
Os diagramas de artefatos não são importantes somente para visualizar, especificar e documentar sistemas baseados em artefatos, mas também para a elaboração de sistemas executáveis por meio de engenharia direta e reversa.

## PRIMEIROS PASSOS

Ao construir uma casa, você deve fazer mais do que criar esquemas de projetos. Os esquemas de projetos são importantes por ajudarem a visualizar, especificar e documentar o tipo de casa que você deseja construir, para que erga a casa certa no tempo correto com um preço adequado. Eventualmente, entretanto, você transformará suas plantas baixas e de elevação em paredes, pisos e tetos de verdade, compostos por madeira, pedras ou metal. Não somente você construirá sua casa a partir desse material bruto, mas também incorporará artefatos pré-fabricados, como armários, janelas, portas e respiradouros. Se estiver reformando uma casa, aproveitará artefatos ainda maiores, como cômodos inteiros e a estrutura.

O mesmo acontece com o software. Você cria diagramas de casos de uso para pensar sobre o comportamento desejado para o sistema. Especifica o vocabulário do domínio com diagramas de classes. Cria diagramas de sequências, diagramas de colaboração, diagramas de gráficos de estados e diagramas de atividades para especificar a forma como os itens em seu vocabulário trabalharão em conjunto para a execução desse comportamento. Eventualmente, você transformará esses projetos lógicos em itens que vivem no mundo dos bits, como executáveis, bibliotecas, tabelas, arquivos e documentos. Você concluirá que deve criar alguns desses artefatos a partir do zero, mas também acabará reutilizando artefatos antigos de maneiras novas.

Com a UML, os diagramas de artefatos são empregados para visualizar o aspecto estático desses artefatos físicos e seus relacionamentos e para especificar detalhes para a construção, conforme mostra a [Figura 30.1](#).



**Figura 30.1:**

Um diagrama de artefatos

## TERMOS E CONCEITOS

Um *diagrama de artefatos* mostra um conjunto de artefatos e seus relacionamentos. Graficamente, o diagrama de artefatos é uma coleção de vértices e arcos.

### PROPRIEDADES COMUNS

Um diagrama de artefatos é apenas um tipo especial de diagrama e compartilha as mesmas propriedades comuns a todos os outros diagramas – um nome e um conteúdo gráfico que são uma projeção em um modelo. O que diferencia um diagrama de artefatos de todos os demais tipos de diagramas é o seu conteúdo particular.

- As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

## CONTEÚDO

Os diagramas de artefatos costumam conter:

- » Artefatos

## » Relacionamentos de dependência, generalização, associação e realização

► Os artefatos são examinados no [Capítulo 26](#); as interfaces são apresentadas no [Capítulo 11](#); os relacionamentos são explicados nos Capítulos 5 e 10; os pacotes aparecem no [Capítulo 12](#); os subsistemas são discutidos no [Capítulo 32](#); as instâncias são examinadas no [Capítulo 13](#); os diagramas de classes são apresentados no [Capítulo 8](#); as visões de implementação, no contexto da arquitetura de software, aparecem no [Capítulo 2](#).

Como todos os outros diagramas, os diagramas de artefatos podem conter notas e restrições.

## USOS COMUNS

Os diagramas de artefatos são empregados para a modelagem da visão estática de implementação de um sistema. Essa perspectiva primariamente proporciona suporte ao gerenciamento da configuração das partes de um sistema, formadas pelos artefatos que podem ser reunidos de várias formas para produzir um sistema em execução.

Ao fazer a modelagem da visão estática de implementação de um sistema, você tipicamente usa os diagramas de artefatos em uma de quatro maneiras.

### 1. Para fazer a modelagem de código-fonte.

Com a maioria das linguagens de programação orientadas a objetos, você recortará o código, utilizando ambientes de desenvolvimento integrados que armazenam seu código-fonte em arquivos. Os diagramas de artefatos podem ser utilizados para fazer a modelagem do gerenciamento da configuração desses arquivos, que representam artefatos do produto do trabalho, e para preparar o sistema de gerenciamento de configuração.

### 2. Para fazer a modelagem de versões executáveis.

Uma versão é um conjunto, relativamente completo e consistente, de artefatos entregues a um usuário interno ou externo. No contexto dos artefatos, uma versão focaliza as partes necessárias para a entrega do sistema

para utilização. Ao fazer a modelagem de uma versão usando diagramas de artefatos, você está visualizando, especificando e documentando as decisões a respeito das partes físicas que constituem o software – ou seja, os artefatos a serem entregues.

### 3. Para fazer a modelagem de bancos de dados físicos.

Pense em um banco de dados físico como a realização concreta de um esquema, existente no mundo dos bits. Os esquemas, com efeito, oferecem uma API para informações persistentes; o modelo de um banco de dados físico representa o armazenamento dessas informações nas tabelas de um banco de dados relacional ou nas páginas de um banco de dados orientado a objetos. Os diagramas de artefatos são utilizados para representar esses e outros tipos de bancos de dados físicos.

- ➡ A persistência é examinada no [Capítulo 24](#); a modelagem de esquemas de bancos de dados lógicos é apresentada no [Capítulo 8](#).

### 4. Para fazer a modelagem de sistemas adaptáveis.

Alguns sistemas são quase estáticos; seus artefatos entram em cena, participam de uma execução e depois desaparecem. Outros sistemas são mais dinâmicos, envolvendo agentes móveis ou artefatos que migram com a finalidade de equilibrar a carga de trabalho e evitar erros. Os diagramas de artefatos são utilizados em conjunção com alguns dos diagramas da UML destinados à modelagem de comportamento, para representar esses tipos de sistemas.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE CÓDIGO-FONTE

Se você desenvolve software em Java, normalmente salva seu código-fonte em arquivos .java. Se desenvolve software utilizando o C++, tipicamente armazena seu código-fonte em arquivos de cabeçalho (arquivos .h) e de corpo (arquivos .cpp). Se utiliza a IDL para desenvolver aplicações em COM+ ou

CORBA, uma interface em sua visão de projeto geralmente será expandida em quatro arquivos de código-fonte: a própria interface, o proxy cliente, o stub servidor e a classe bridge. À medida que sua aplicação vai crescendo, qualquer que seja a linguagem utilizada, você se verá organizando esses arquivos em grupos cada vez maiores. Além disso, durante a fase de construção do desenvolvimento, provavelmente você acabará criando novas versões de alguns desses arquivos para cada novo incremento de versão que produzir e você desejará colocar essas versões sob o controle de um sistema de gerenciamento de configuração.

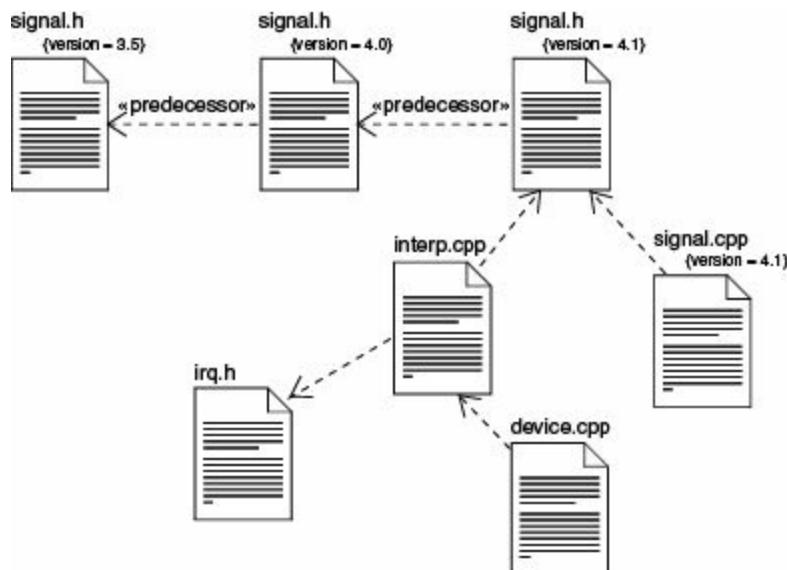
Em boa parte do tempo, não será necessário fazer a modelagem desse aspecto do sistema diretamente. Em vez disso, você permitirá que seu ambiente de desenvolvimento faça o acompanhamento desses arquivos e de seus respectivos relacionamentos. De vez em quando, entretanto, é de grande ajuda visualizar esses arquivos de código-fonte e seus relacionamentos, usando os diagramas de artefatos. Utilizados dessa maneira, os diagramas de artefatos tipicamente contêm somente os artefatos do produto do trabalho estereotipados como arquivos, juntamente com os relacionamentos de dependência. Por exemplo, você poderá fazer a engenharia reversa de um conjunto de arquivos de código-fonte para visualizar sua trama de dependências de compilação. Você pode seguir outra direção, especificando os relacionamentos entre seus arquivos de código-fonte e depois utilizando esses modelos como entrada para ferramentas de compilação, como `make` em Unix. De modo semelhante, você desejará usar os diagramas de artefatos para visualizar o histórico de um conjunto de arquivos de código-fonte que estejam sob o gerenciamento de configuração. Extraindo informações de seu sistema de gerenciamento da configuração, como a quantidade de vezes que um arquivo de código-fonte foi verificado em determinado período, você pode usar essas informações para colorir os diagramas de artefatos, mostrando “pontos de destaque” de alterações de seus arquivos de código-fonte e setores de modificação da arquitetura.

► *O estereótipo file para os artefatos é examinado no [Capítulo 26](#).*

Para fazer a modelagem do código-fonte do sistema:

- › Usando engenharia direta ou reversa, identifique o conjunto de arquivos de código-fonte de interesse e modele-o como artefatos estereotipados como arquivos.
- › No caso de sistemas maiores, use pacotes para mostrar grupos de arquivos de código-fonte.
- › Considere a exposição de um valor atribuído, indicando informações como o número da versão do arquivo de código-fonte, seu autor e a data da última alteração. Use ferramentas para gerenciar o valor dessa atribuição.
- › Faça a modelagem das dependências de compilação entre esses arquivos, utilizando dependências. Mais uma vez, use ferramentas para ajudá-lo a gerar e a gerenciar essas dependências.

Por exemplo, a [Figura 30.2](#) mostra cinco arquivos de código-fonte. signal.h é um arquivo de cabeçalho. Três de suas versões são mostradas, relacionando as novas versões e as anteriores. Cada variante desse arquivo de código-fonte é representada por um valor atribuído, indicando o número da versão.



**Figura 30.2:**  
A modelagem de código-fonte

► *O estereótipo de dependência trace é examinado no Capítulo 10.*

Esse arquivo de cabeçalho (`signal.h`) é utilizado por dois outros arquivos (`interp.cpp` e `signal.cpp`), ambos arquivos de corpo. Um desses arquivos (`interp.cpp`) tem uma dependência de compilação com outro arquivo de cabeçalho (`irq.h`); por sua vez, `device.cpp` tem uma dependência de compilação com `interp.cpp`. Dado esse diagrama de artefatos, é fácil acompanhar o impacto das modificações. Por exemplo, a modificação do arquivo de código-fonte `signal.h` exigirá a recompilação de três outros arquivos: `signal.cpp`, `interp.cpp` e, transitivamente, `device.cpp`. Conforme esse diagrama também mostra, o arquivo `irq.h` não é afetado.

Diagramas como esse podem ser gerados facilmente pela engenharia reversa a partir das informações mantidas pelas ferramentas do gerenciamento da configuração de seu ambiente de desenvolvimento.

## MODELAGEM DE UMA VERSÃO EXECUTÁVEL

É fácil liberar uma versão de uma aplicação simples: basta gravar os bits de um único arquivo executável em um disco e seus usuários apenas rodarem esse executável. Para esses tipos de aplicações, não serão necessários diagramas de artefatos, pois não existirá nenhuma dificuldade para visualizar, especificar, construir ou documentar.

Não é tão fácil liberar algo que não seja uma aplicação simples. Você precisa do executável principal (geralmente, um arquivo `.exe`), mas também precisa de todas as partes auxiliares, como as bibliotecas (geralmente arquivos `.dll`, caso esteja trabalhando no contexto de COM+, ou arquivos `.class` e `.jar`, caso esteja trabalhando no contexto de Java), bancos de dados, arquivos de ajuda e arquivos de recursos. No caso de sistemas distribuídos, provavelmente haverá vários executáveis e outras partes espalhadas por vários nós. Se você estiver trabalhando com um sistema de aplicações, descobrirá que alguns desses artefatos são únicos em relação a cada aplicação, mas muitos são compartilhados entre as aplicações. À medida que seu sistema evolui, o controle da configuração desses vários artefatos se

tornará uma atividade importante – e a mais difícil, pois as alterações dos artefatos associados com uma aplicação poderão afetar a operação de outras aplicações.

Por essa razão, os diagramas de artefatos são empregados para visualizar, especificar, construir e documentar a configuração de suas versões executáveis, abrangendo os artefatos entregues que formam cada versão e os relacionamentos entre esses artefatos. Os diagramas de artefatos podem ser empregados para a engenharia direta de um novo sistema e para a engenharia reversa de um já existente.

Ao criar diagramas de artefatos como esses, na verdade, você está apenas fazendo a modelagem de uma parte dos itens e relacionamentos que formam a visão de implementação de seu sistema. Por essa razão, cada diagrama de artefatos deverá focalizar um único conjunto de artefatos por vez.

Para fazer a modelagem de uma versão executável:

- › Identifique o conjunto de artefatos cuja modelagem você deseja fazer. Tipicamente, isso envolverá alguns ou todos os artefatos que vivem em um único nó ou a distribuição desses conjuntos de artefatos por todos os nós existentes no sistema.
- › Considere o estereótipo de cada artefato desse conjunto. Na maioria dos sistemas, será encontrado um pequeno número de tipos diferentes de artefatos (como executáveis, bibliotecas, tabelas, arquivos e documentos). Você poderá usar os mecanismos de extensibilidade da UML para fornecer indicações visuais referentes a esses estereótipos.
- › Para cada artefato existente no conjunto, considere seu relacionamento com os vizinhos. Com muita frequência, isso envolverá as interfaces que são exportadas (realizadas) por certos artefatos e depois são importadas (utilizadas) por outros artefatos. Se você quiser expor as costuras de seu sistema, faça a modelagem dessas interfaces explicitamente. Caso prefira um nível de abstração mais alto para o modelo, oculte esses relacionamentos, mostrando somente as dependências entre os artefatos.

- Os mecanismos de extensibilidade da UML são examinados no Capítulo 6; as interfaces são apresentadas no Capítulo 11.

Por exemplo, a Figura 30.3 faz a modelagem de parte da versão executável para um robô autônomo. Essa figura focaliza artefatos de implantação associados com as funções de cálculos e de direção do robô. Você encontrará um artefato (driver.dll), que exporta uma interface (IDrive), que, por sua vez, é importada por outro artefato (path.dll). driver.dll exporta uma outra interface (ISelfTest), que provavelmente é utilizada por outros artefatos do sistema, embora não sejam mostrados aqui. Existe um outro artefato apresentado nesse diagrama (collision.dll) e ele também exporta um conjunto de interfaces, embora esses detalhes estejam ocultos: path.dll é mostrado com uma dependência diretamente relacionada a collision.dll.

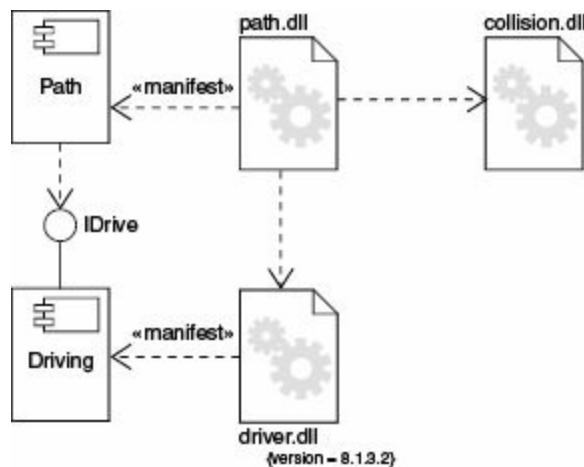


Figura 30.3:

A modelagem de uma versão executável

Existem muito mais artefatos envolvidos no sistema. Entretanto, esse diagrama somente focaliza os artefatos de implantação que estão diretamente envolvidos na movimentação do robô. Observe que, nessa arquitetura baseada em artefatos, você substituiria uma versão específica de driver.dll por uma outra que realizasse as mesmas (e talvez adicionais) interfaces e path.dll ainda funcionaria adequadamente.

## MODELAGEM DE UM BANCO DE DADOS FÍSICO

Um esquema de bancos de dados lógico captura o vocabulário de dados persistentes de um sistema, juntamente com a semântica de seus relacionamentos. Fisicamente, esses itens são armazenados em um banco de dados (relacional, orientado a objetos ou um híbrido objeto/relacional) para recuperação posterior. A UML é bastante adequada para fazer a modelagem de um banco de dados físico, como também esquemas de banco de dados lógico.

- A modelagem do esquema de bancos de dados lógicos é examinada no [Capítulo 8](#).

Mapear o esquema de um banco de dados lógico para um banco de dados orientado a objetos é algo simples, pois até as heranças complexas podem ser convertidas em persistentes diretamente. Entretanto, o mapeamento de um esquema de banco de dados lógico para um banco de dados relacional não é tão simples. Na presença da herança, é necessário decidir como mapear classes em tabelas. Tipicamente, você pode aplicar uma ou a combinação de três estratégias.

1. (Push down) Defina uma tabela separada para cada classe. Essa é uma solução simples, mas ingênuia, pois gera dores de cabeça em termos de manutenção sempre que for necessário adicionar novas classes-filha ou modificar as classes-mãe.
2. (Pull up) Resuma as heranças, de forma que todas as instâncias de qualquer classe em uma hierarquia tenham o mesmo estado. A desvantagem dessa solução é que você acaba armazenando informações supérfluas para muitas instâncias.
3. (Split tables) Separe estados de classes-mãe e filha em tabelas diferentes. Essa solução reflete melhor a herança, mas a desvantagem é que para o cruzamento dos seus dados serão necessárias várias uniões de tabelas de referência cruzada.

- » *O projeto de um banco de dados físico ultrapassa o escopo deste livro; o foco aqui é simplesmente mostrar como você pode fazer a modelagem de bancos de dados e de tabelas com a utilização da UML.*

Ao projetar um banco de dados físico, você também precisa tomar decisões a respeito de como fazer o mapeamento de operações definidas no esquema de seu banco de dados lógico. Os bancos de dados orientados a objetos tornam o mapeamento bastante transparente. Entretanto, com bancos de dados relacionais, é preciso tomar algumas decisões sobre o modo como essas operações lógicas são implementadas. Mais uma vez, você tem algumas opções.

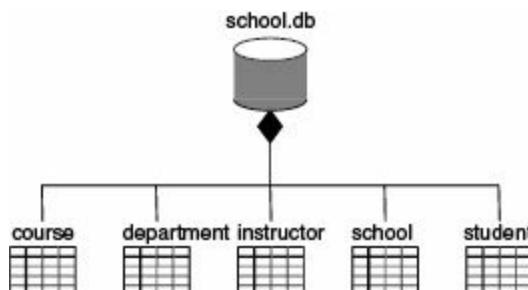
1. Para simples operações de criar, ler, atualizar e excluir, implemente-as com chamadas SQL ou ODBC padrão.
2. Para um comportamento mais complexo (como regras de negócios), mapeie-as para ativação ou para procedimentos de armazenamento.

Dadas essas diretrizes gerais, para fazer a modelagem de um banco de dados físico:

- » Identifique as classes existentes no modelo que representam o esquema de seu banco de dados lógico.
- » Selecione uma estratégia para o mapeamento dessas classes para tabelas. Você também desejará considerar a distribuição física de seus bancos de dados. Sua estratégia de mapeamento será afetada pela localização em que deseja que seus dados permaneçam no sistema instalado.
- » Para visualizar, especificar, construir e documentar o mapeamento, crie um diagrama de artefatos, incluindo os artefatos estereotipados como tabelas.
- » Onde for possível, use ferramentas para ajudá-lo a transformar seu projeto lógico em um projeto físico.

A [Figura 30.4](#) mostra um conjunto de tabelas de banco de dados, definidas a partir de um sistema de informações de uma escola. Existe um banco de dados (`school.db`, representado como um artefato estereotipado como `database`), composto por cinco tabelas: `student`, `class`, `instructor`, `department` e `course` (representado como um artefato estereotipado como `interface`, um dos elementos padrão da UML). No correspondente esquema de banco de dados lógico, não havia herança; portanto, o mapeamento para o projeto desse banco de dados físico foi direto.

Embora não seja mostrado nesse exemplo, é possível especificar o conteúdo de cada tabela. Como os artefatos admitem atributos, um idioma comum ao fazer a modelagem do banco de dados físico é a utilização desses atributos para especificar as colunas em cada tabela. De modo semelhante, os artefatos admitem operações e estas podem ser empregadas para denotar procedimentos de armazenamento.



**Figura 30.4:**

A modelagem de um banco de dados físico

## MODELAGEM DE SISTEMAS ADAPTÁVEIS

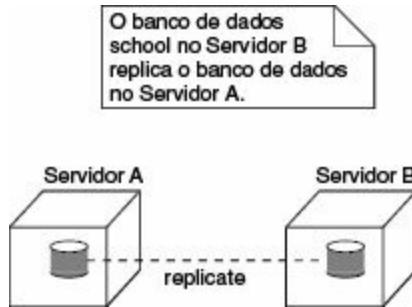
Todos os diagramas de artefatos mostrados até agora foram utilizados para fazer a modelagem de visões estáticas. Seus artefatos passam a vida inteira em um único nó. Essa é a situação mais comum a ser encontrada, mas especialmente no domínio de sistemas distribuídos complexos será necessário fazer a modelagem das visões dinâmicas. Por exemplo, você poderá ter um sistema que replica seu banco de dados por vários nós, alternando aquele que funciona como banco de dados primário quando ocorrem problemas em um

servidor. De modo semelhante, ao fazer a modelagem de uma operação 24x7 distribuída globalmente (ou seja, um sistema que funciona 24 horas por dia, sete dias na semana), você provavelmente encontrará agentes móveis, artefatos que migram de um nó para outro executando alguma transação. Para fazer a modelagem dessas visões dinâmicas, é necessário utilizar uma combinação de diagramas de artefatos, diagramas de objetos e diagramas de interação.

Para fazer a modelagem de um sistema adaptável:

- » Considere a distribuição física dos artefatos que poderão migrar de um nó para outro. Especifique a localização de uma instância do artefato, marcando-a com um atributo de localização, que pode ser representado em um diagrama de artefatos.
  - » Se quiser fazer a modelagem das ações que causam a migração de um artefato, crie um diagrama de interação correspondente, contendo instâncias do artefato. Ilustre uma alteração da localização, desenhando a mesma instância mais de uma vez, mas com valores diferentes para seu estado, que inclui a localização.
- *O atributo location é examinado no Capítulo 24; os diagramas de objetos são apresentados no Capítulo 14.*

Por exemplo, a [Figura 30.5](#) apresenta a modelagem da replicação do banco de dados da figura anterior. São mostradas duas instâncias do artefato school.db. As duas instâncias são anônimas e ambas têm um valor diferente para seu valor atribuído location. Também existe uma nota, especificando explicitamente que instância replica a outra.



**Figura 30.5:**

A modelagem de sistemas adaptáveis

Se quiser mostrar os detalhes de cada banco de dados, você pode representá-los em sua forma canônica – um artefato estereotipado como database. Embora não seja mostrado aqui, um diagrama de interação poderia ser empregado para fazer a modelagem da dinâmica da mudança de um banco de dados primário para outro.

► *Os diagramas de interação são examinados no [Capítulo 19](#).*

## ENGENHARIA DIRETA E REVERSA

Os artefatos da engenharia direta e da engenharia reversa são bastante diretos, pois eles próprios são coisas físicas (executáveis, bibliotecas, tabelas, arquivos e documentos) que estão, portanto, próximos ao sistema em execução. Ao fazer a engenharia direta de uma classe ou de uma colaboração, você realmente está fazendo a engenharia direta para artefato que representa o código-fonte, biblioteca binária ou executável dessa classe ou colaboração. De modo semelhante, ao fazer a engenharia reversa do código-fonte, bibliotecas binárias ou executáveis, você realmente está fazendo a engenharia reversa de um artefato ou conjunto de artefatos que, por sua vez, levam a classes ou colaborações.

A escolha da engenharia direta (a criação de código a partir de um modelo) de uma classe ou colaboração para o código-fonte, uma biblioteca binária ou um executável é uma decisão de mapeamento que precisa ser tomada. Você desejará transformar seus modelos lógicos em código-fonte, caso esteja interessado em controlar o gerenciamento da configuração de

arquivos que estão sendo manipulados pelo ambiente de desenvolvimento. Você desejará transformar seus modelos lógicos diretamente em bibliotecas binárias ou em executáveis, caso esteja interessado no gerenciamento dos artefatos que realmente serão entregues no sistema em execução. Em alguns casos, você desejará as duas coisas. Uma classe ou colaboração poderá ser denotada pelo código-fonte, como também por uma biblioteca binária ou por um executável.

Para fazer a engenharia direta de um diagrama de artefatos:

- › Para cada artefato, identifique as classes ou colaborações que o artefato implementa. Mostre isso com um relacionamento explícito.
- › Escolha a forma para cada artefato. Basicamente sua escolha está entre o código-fonte (uma forma que pode ser manipulada por ferramentas de desenvolvimento) ou uma biblioteca binária ou um executável (uma forma que pode ser incluída em um sistema em execução).
- › Use as ferramentas para fazer a engenharia direta dos seus modelos.

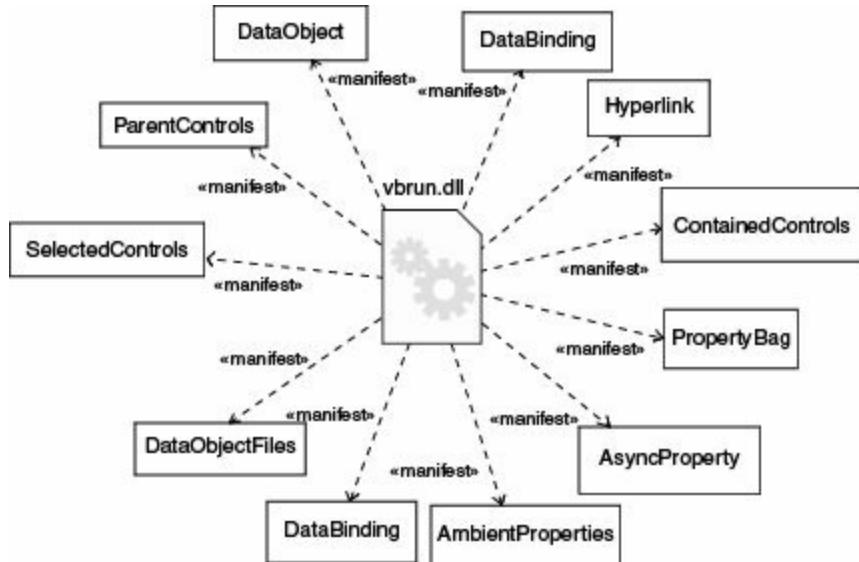
A engenharia reversa (a criação de um modelo a partir de código) de um diagrama de artefatos não é um processo perfeito, pois sempre ocorre alguma perda de informação. A partir do código-fonte, é possível fazer a engenharia reversa e retornar às classes; essa é a coisa mais comum que você fará. A engenharia reversa de código- fonte a artefatos revelará dependências de compilação entre esses arquivos. No caso das bibliotecas binárias, o melhor resultado a ser esperado é denotar a biblioteca como um artefato e depois descobrir suas interfaces por meio da engenharia reversa. Essa é a segunda coisa mais comum que você fará com os diagramas de artefatos. De fato, essa é uma forma útil de abordar um conjunto de novas bibliotecas que, caso contrário, poderão ter uma documentação pobre. No caso de executáveis, o melhor resultado a ser esperado é denotar o executável como um artefato e então decompor seu código – algo que raramente será necessário, a menos que você trabalhe com linguagem assembly.

- ➡ A engenharia reversa de diagramas de classes é examinada no [Capítulo 8](#).

Para fazer a engenharia reversa de um diagrama de artefatos:

- › Escolha o destino desejado para a engenharia reversa. O código-fonte pode ser submetido à engenharia reversa para obter artefatos e depois classes. As bibliotecas binárias podem ser submetidas à engenharia reversa para revelar suas interfaces. Por fim, os executáveis podem ser submetidos à engenharia reversa.
- › Utilizando uma ferramenta, aponte o código a ser submetido à engenharia reversa. Use sua ferramenta para gerar um novo modelo ou para modificar um modelo já existente que tenha sido submetido à engenharia direta anteriormente.
- › Utilizando sua ferramenta, crie um diagrama de artefatos, realizando consultas ao modelo. Por exemplo, você poderá iniciar com um ou mais artefatos e depois expandir o diagrama pelos relacionamentos seguintes ou artefatos vizinhos. Exiba ou oculte os detalhes do conteúdo desse diagrama de artefatos, conforme seja necessário para comunicar sua intenção.

Por exemplo, a [Figura 30.6](#) apresenta um diagrama de artefatos, representando a engenharia reversa do artefato ActiveX vbrun.dll. Conforme mostra a figura, o artefato manifesta 11 interfaces. Considerando esse diagrama, você pode começar a compreender a semântica do artefato pela exploração dos detalhes de suas classes de interfaces.



**Figura 30.6:**

A engenharia reversa

Especialmente quando você aplicar a engenharia reversa a código-fonte e, às vezes, a bibliotecas binárias e a executáveis, você o estará fazendo no contexto de um sistema de gerenciamento da configuração. Isso significa que você frequentemente estará trabalhando com versões específicas de arquivos ou bibliotecas, com todas as versões de uma configuração compatível com outra. Nesses casos, você desejará incluir um valor atribuído representando a versão do artefato, que pode ser derivada de seu sistema de gerenciamento da configuração. Dessa maneira, a UML pode ser utilizada para visualizar o histórico de um artefato em suas várias versões.

## DICAS E SUGESTÕES

Ao criar diagramas de artefatos na UML, lembre-se de que todo diagrama de artefatos é apenas uma apresentação gráfica da visão estática de implementação de um sistema. Isso significa que um único diagrama de artefatos não precisará capturar tudo sobre a visão de implementação do sistema. Coletivamente, todos os diagramas de artefatos representam a visão estática de implementação completa; individualmente, cada diagrama representa apenas um único aspecto.

Um diagrama de artefatos bem estruturado:

- › Tem como foco comunicar um aspecto da visão estática de implementação do sistema.
- › Contém somente os elementos essenciais à compreensão desse aspecto.
- › Fornece detalhes consistentes com seu nível de abstração, mostrando somente os adornos essenciais à compreensão do que é apresentado.
- › Não é tão minimalista que informe mal o leitor sobre a semântica importante.

Ao definir um diagrama de artefatos:

- › Dê-lhe um nome capaz de comunicar seu propósito.
- › Distribua seus elementos para minimizar o cruzamento de linhas.
- › Organize seus elementos espacialmente, de modo que os itens semanticamente afins fiquem próximos.
- › Use notas e cores como indicações visuais com a finalidade de chamar a atenção para características importantes de seu diagrama.
- › Use elementos estereotipados com cuidado. Escolha um pequeno conjunto de ícones comuns para seu projeto ou organização e utilize-os de maneira consistente.

CAPÍTULO

---

31

---

# Diagramas de Implantação

## Neste capítulo

- » *Modelagem de um sistema embarcado*
- » *Modelagem de um sistema cliente/servidor*
- » *Modelagem de um sistema totalmente distribuído*
- » *Engenharia direta e reversa*

**O**s diagramas de implantação são um dos dois tipos de diagramas empregados para a modelagem dos aspectos físicos de um sistema orientado a objetos. O diagrama de implantação mostra a configuração dos nós de processamento em tempo de execução e os artefatos que nele existem.

- *Os diagramas de artefatos, o segundo tipo de diagrama utilizado para a modelagem de aspectos físicos de um sistema orientado a objetos, são examinados no Capítulo 30.*

Os diagramas de implantação são empregados para a modelagem da visão estática da implantação de um sistema. Na maior parte, isso envolve a modelagem da topologia do hardware em que o sistema é executado. Os diagramas de implantação são essencialmente diagramas de classes que focalizam os nós do sistema.

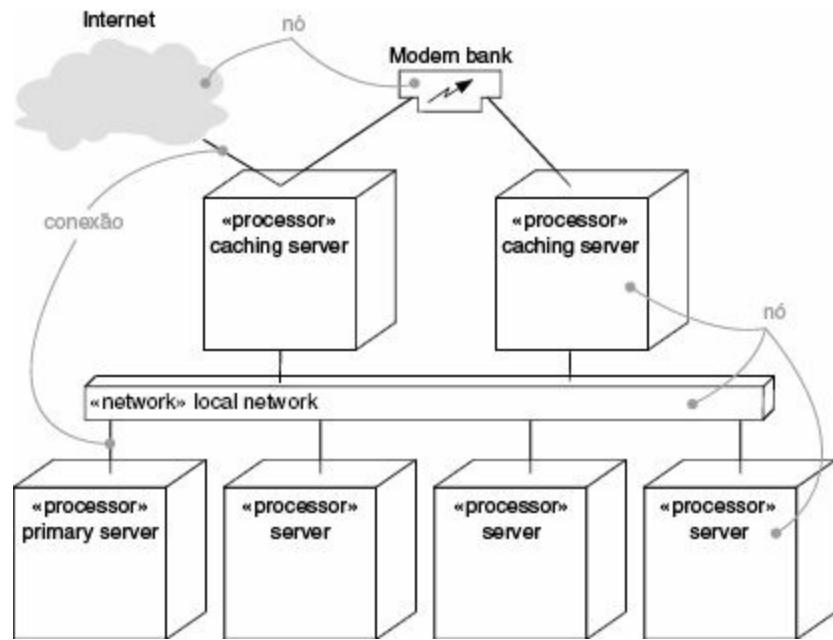
Os diagramas de implantação não são importantes somente para visualizar, especificar e documentar sistemas embarcados, cliente/servidor e distribuídos, mas também para o gerenciamento de sistemas executáveis por meio de engenharia direta e reversa.

## PRIMEIROS PASSOS

Ao criar um sistema complexo de software, seu foco principal como desenvolvedor de software está na arquitetura e implantação do software. Entretanto, como um engenheiro de sistemas, seu foco principal está no hardware *e* no software do sistema e no gerenciamento da compatibilidade dos dois. Enquanto os desenvolvedores de software trabalham com artefatos de alguma forma intangíveis, como modelos e código, os desenvolvedores de sistemas trabalham com hardware quase tangível.

A UML focaliza primariamente as facilidades para visualização, especificação, construção e documentação de artefatos de software, mas também se destina a abranger artefatos de hardware. Isso não significa que a UML seja uma linguagem de propósito geral para descrição de hardware, como a VHDL. Em vez disso, a UML se destina a fazer a modelagem de muitos aspectos de hardware para um sistema, suficientes para que o engenheiro de software especifique a plataforma em que o software do sistema é executado e para que o engenheiro de sistemas gerencie a fronteira entre o hardware e o software do sistema. Na UML, os diagramas de classes e os diagramas de artefatos são empregados para analisar a estrutura do software. Os diagramas de sequências, diagramas de colaboração, diagramas de estados e diagramas de atividades são utilizados para especificar o comportamento do software. No limite do software e do hardware de seu sistema, os diagramas de implantação são utilizados para analisar a topologia dos processadores e dispositivos nos quais o software é executado.

Com a UML, os diagramas de implantação podem ser utilizados para visualizar o aspecto estático desses nós físicos e seus relacionamentos e para especificar seus detalhes referentes à construção, conforme mostra a [Figura 31.1](#).



**Figura 31.1:**

Um diagrama de implantação

## TERMOS E CONCEITOS

O *diagrama de implantação* é um diagrama que mostra a configuração de nós de processamento em tempo de execução e os artefatos que neles existem. Graficamente, o diagrama de implantação é uma coleção de vértices e arcos.

## PROPRIEDADES COMUNS

Um diagrama de implantação é apenas um tipo especial de diagrama e compartilha as mesmas propriedades comuns a todos os outros diagramas – um nome e um conteúdo gráfico que são a projeção em um modelo. O que diferencia o diagrama de implantação de todos os outros tipos de diagramas é o seu conteúdo particular.

- ➡ As propriedades gerais dos diagramas são examinadas no [Capítulo 7](#).

## CONTEÚDO

Os diagramas de implantação costumam conter:

- » Nós
- » Relacionamentos de dependência e associação

► Os nós são examinados no [Capítulo 27](#); os relacionamentos são apresentados nos [Capítulos 5 e 10](#); os artefatos são explicados no [Capítulo 26](#); os pacotes aparecem no [Capítulo 12](#); os subsistemas são discutidos no [Capítulo 32](#); as instâncias são examinadas no [Capítulo 13](#); os diagramas de classes são apresentados no [Capítulo 8](#).

Como todos os outros diagramas, os diagramas de implantação podem conter notas e restrições.

Os diagramas de implantação também podem conter artefatos, cada um dos quais deve residir em algum nó. Os diagramas de implantação também podem conter pacotes ou subsistemas, os dois utilizados para agrupar elementos do modelo em conjuntos maiores. De vez em quando, você também desejará incluir instâncias em seus diagramas de implantação, especialmente quando quiser visualizar uma instância das topologias de uma família de hardware.

**Nota:** Em muitos aspectos, o diagrama de implantação é apenas um tipo especial de diagrama de classes, que focaliza os nós do sistema.

## USOS COMUNS

Os diagramas de implantação são utilizados para a modelagem da visão estática de funcionamento de um sistema. Essa visão direciona primariamente a distribuição, entrega e instalação das partes que formam o sistema físico.

► As visões de implantação no contexto da arquitetura de software são examinadas no [Capítulo 2](#).

Existem alguns tipos de sistemas para os quais os diagramas de implantação são desnecessários. Se você estiver desenvolvendo um trecho de software que reside em uma única máquina e interage somente com dispositivos-padrão dessa máquina, que já são gerenciados pelo sistema operacional desta (por exemplo, o teclado, o monitor e o modem de um computador pessoal), você pode ignorar os diagramas de implantação. Por outro lado, se estiver desenvolvendo um trecho de software que interage com dispositivos que o sistema operacional da máquina não gerencia diretamente, ou que é distribuído fisicamente por vários processadores, então o uso de diagramas de implantação ajudará a analisar o mapeamento de software-para-hardware de seu sistema.

Ao fazer a modelagem da visão estática de implantação de um sistema, tipicamente você usará os diagramas de implantação em uma de três maneiras.

### 1. Para fazer a modelagem de sistemas embarcados.

Um sistema embarcado é uma coleção complexa de software para o hardware que interage com o mundo físico. Os sistemas embarcados envolvem o software que controla dispositivos como motores, atuadores e monitores e que, por sua vez, é controlado por estímulos externos como entrada de sensor, movimentação e mudança de temperatura. Os diagramas de implantação podem fazer a modelagem dos dispositivos e processadores que formam um sistema embarcado.

### 2. Para fazer a modelagem de sistemas cliente/servidor.

O sistema cliente/servidor é uma arquitetura comum, cujo foco é a criação de uma clara separação de questões entre a interface para o usuário (que reside no cliente) e os dados persistentes do sistema (que residem no servidor). Os sistemas cliente/servidor são uma das extremidades da continuação de sistemas distribuídos e exigem que você tome decisões sobre a conectividade da rede de clientes para servidores e sobre a distribuição física dos artefatos de software de seu sistema pelos nós. A modelagem da topologia desses sistemas pode ser feita com a utilização de diagramas de implantação.

### 3. Para fazer a modelagem de sistemas totalmente distribuídos.

Na outra extremidade da continuação de sistemas distribuídos estão aqueles que são ampla, se não globalmente, distribuídos, tipicamente abrangendo vários níveis de servidores. Esses sistemas costumam ser hospedeiros para várias versões de artefatos de software, algum dos quais poderão até migrar de um nó para outro. Ajustar esses sistemas exige tomar decisões que permitam a alteração contínua da topologia do sistema. Os diagramas de implantação podem ser empregados para visualizar a topologia atual do sistema e a distribuição de artefatos para analisar o impacto das modificações sobre essa topologia.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DE UM SISTEMA EMBARCADO

O desenvolvimento de um sistema embarcado é bem mais do que um problema de software. É preciso gerenciar o mundo físico onde se encontram as partes móveis em que ele se divide, os sinais têm ruídos e o comportamento não é linear. Ao fazer a modelagem de um desses sistemas, é necessário levar em consideração sua interface com o mundo real e isso significa analisar dispositivos pouco usuais, além dos nós.

► *Os nós e os dispositivos são examinados no [Capítulo 27](#).*

Os diagramas de implantação são úteis por facilitarem a comunicação entre os engenheiros de hardware e os desenvolvedores de software do projeto. Com a utilização de nós estereotipados para se parecerem com dispositivos familiares, você pode criar diagramas comprehensíveis pelos dois grupos. Os diagramas de implantação também são úteis para a análise de compatibilidade de hardware e software. Você usará os diagramas de implantação para visualizar, especificar, construir e documentar suas decisões de engenharia de sistema.

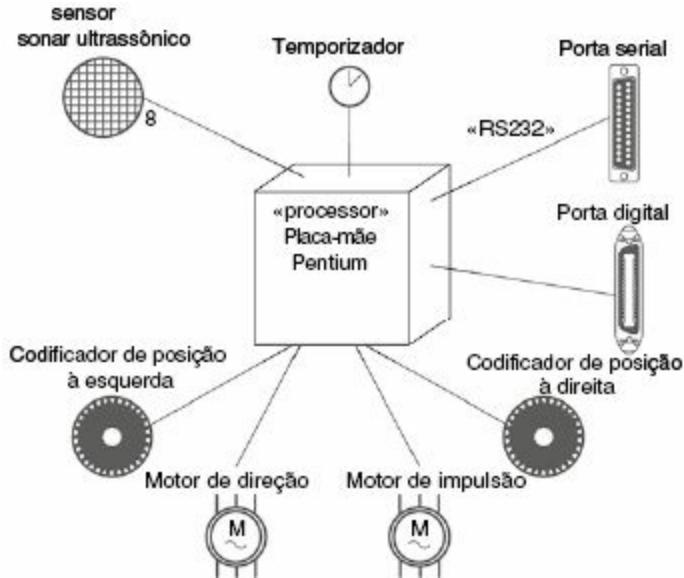
- ➡ Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).

Para fazer a modelagem de um sistema embarcado:

- › Identifique os dispositivos e os nós que são únicos para o sistema.
- › Forneça indicações visuais, especialmente para dispositivos pouco usuais, utilizando os mecanismos de extensibilidade da UML para definir estereótipos específicos do sistema com os ícones apropriados. No final, você desejará diferenciar os processadores (que contêm os artefatos de software) e dispositivos (que, nesse nível de abstração, não contêm software diretamente).
- › Faça a modelagem dos relacionamentos entre esses processadores e dispositivos em um diagrama de implantação. De modo semelhante, especifique o relacionamento entre os artefatos da visão de implementação e os nós na visão de implantação do sistema.
- › Conforme seja necessário, procure expandir quaisquer dispositivos inteligentes pela modelagem de sua estrutura com um diagrama de implantação mais detalhado.

Por exemplo, a [Figura 31.2](#) mostra o hardware para um robô autônomo simples. Existe um nó (Placa-mãe Pentium) estereotipado como um processador.

Ao redor desse nó existem oito dispositivos, cada um estereotipado como um dispositivo e representado por um ícone que oferece uma clara indicação visual ao seu equivalente do mundo real.



**Figura 31.2:**

A modelagem de um sistema embarcado

## MODELAGEM DE UM SISTEMA CLIENTE/SERVIDOR

No momento em que você começa o desenvolvimento de um sistema cujo software não mais reside em um único processador, você se depara com várias decisões: qual a melhor maneira de distribuir os artefatos de software pelos nós? Como eles se comunicam? Como lidar com falhas e ruído? Em uma extremidade do espectro de sistemas distribuídos, você encontrará sistemas cliente/servidor, em que existe uma clara separação de questões entre a interface para o usuário (tipicamente gerenciada pelo cliente) e seus dados (tipicamente gerenciados pelo servidor).

Existem muitas variações sobre esse tema. Por exemplo, você poderá escolher um cliente fino, significando que ele terá uma quantidade limitada de capacidade computacional e pouco faz além de gerenciar a interface para o usuário e a visualização de informações. Clientes finos até poderão não hospedar muitos artefatos, mas, em vez disso, poderão ser destinados a carregar artefatos do servidor, conforme seja necessário, como acontece com Enterprise Java Beans. Por outro lado, você poderá escolher um cliente completo, significando que ele terá uma boa quantidade de capacidade computacional e faz mais do que apenas uma visualização. Um cliente

completo tipicamente executa algumas das regras lógicas e de negócios do sistema. A escolha entre clientes finos e completos é uma decisão de arquitetura, influenciada por vários fatores técnicos, econômicos e políticos.

Em qualquer uma dessas maneiras, a partição de um sistema em suas partes cliente e servidor envolve a tomada de algumas decisões difíceis a respeito do local em que serão colocados fisicamente seus artefatos de software e como impor uma distribuição equilibrada de responsabilidades entre esses artefatos. Por exemplo, a maioria dos sistemas de informações de gerenciamento são essencialmente arquiteturas tripartidas, significando que a GUI, a lógica do negócio e o banco de dados do sistema são distribuídos fisicamente. Costuma ser bastante óbvio decidir onde colocar a GUI e o banco de dados; portanto, a parte mais difícil é a decisão de onde a lógica do negócio ficará.

Os diagramas de implantação da UML podem ser utilizados para visualizar, especificar e documentar suas decisões sobre a topologia do sistema cliente/servidor e como seus artefatos de software estão distribuídos no cliente e no servidor. Tipicamente, você desejará criar um diagrama de implantação para o sistema como um todo, juntamente com outros diagramas, mais detalhados, apresentando segmentos individuais do sistema.

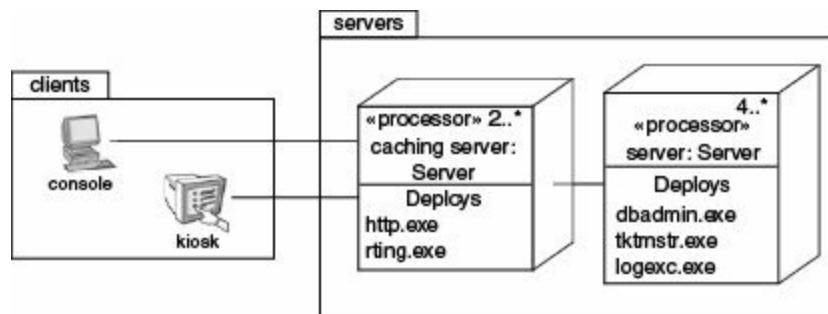
Para fazer a modelagem de um sistema cliente/servidor:

- » Identifique os nós que representam os processadores do cliente e do servidor do sistema.
- » Destaque os dispositivos que são relevantes para o comportamento do sistema. Por exemplo, você desejará fazer a modelagem de dispositivos especiais, como leitores de cartão de crédito, leitores de crachá e dispositivos de vídeo que não sejam monitores, pois sua posição na topologia de hardware do sistema provavelmente é significativa em termos de arquitetura.
- » Forneça indicações visuais para esses processadores e dispositivos por meio de estereótipos.

» Faça a modelagem da topologia desses nós em um diagrama de implantação. De modo semelhante, especifique o relacionamento existente entre os artefatos na visão de implementação e os nós na visão de implantação do sistema.

Por exemplo, a [Figura 31.3](#) mostra a topologia de um sistema de recursos humanos, que segue uma arquitetura cliente/servidor clássica. Essa figura ilustra a divisão cliente/servidor explicitamente, pela utilização dos pacotes nomeados como client e server. O pacote cliente contém dois nós (console e kiosk), ambos estereotipados e visualmente diferenciados. O pacote servidor contém dois tipos de nós (caching server e server) e ambos têm, como adornos, alguns dos artefatos que residem em cada um deles. Observe também que caching server e Server estão marcados com multiplicidades explícitas, especificando quantas instâncias de cada um são esperadas em uma determinada configuração de instalação. Por exemplo, esse diagrama indica que poderá haver dois ou mais caching servers em qualquer instância de instalação do sistema.

- » Os pacotes são examinados no [Capítulo 12](#); a multiplicidade é apresentada no [Capítulo 10](#).



**Figura 31.3:**

A modelagem de um sistema cliente/servidor

## MODELAGEM DE UM SISTEMA TOTALMENTE DISTRIBUÍDO

Os sistemas distribuídos aparecem de muitas formas, desde sistemas simples com dois processadores até os que estão em muitos nós dispersos geograficamente. Os últimos tipicamente nunca são estáticos. Os nós são

adicionados e removidos, à medida que o tráfego da rede se modifica e os processadores falham; novos e mais rápidos caminhos de comunicação podem ser estabelecidos em paralelo com os canais anteriores e mais lentos, que eventualmente são desativados. Não apenas a topologia desses sistemas poderá mudar, mas a distribuição de seus artefatos de software também poderá ser alterada. Por exemplo, as tabelas do banco de dados poderão ser replicadas pelos servidores, somente para serem movidas, conforme as necessidades de tráfego. No caso de alguns sistemas globais, os artefatos poderão acompanhar o movimento do Sol, migrando de um servidor para outro, à medida que o dia comercial começa em uma parte do mundo e termina em outra parte.

Visualizar, especificar e documentar a topologia de sistemas totalmente distribuídos como esses são atividades valiosas para o administrador de sistemas, que deve controlar os recursos de computação da empresa. Os diagramas de implantação da UML podem ser utilizados para analisar a topologia desses sistemas. Ao documentar sistemas totalmente distribuídos utilizando diagramas de implantação, você desejará expandir os detalhes de dispositivos de rede do sistema, cada um dos quais poderá ser representado como um nó estereotipado.

Para fazer a modelagem de um sistema totalmente distribuído:

- › Identifique os dispositivos e processadores para sistemas cliente/servidor mais simples.
- › Se for necessário analisar o desempenho da rede do sistema ou o impacto de alterações da rede, certifique-se de fazer a modelagem desses dispositivos de comunicação em um nível de detalhe suficiente para a realização dessas avaliações.
- › Dedique maior atenção aos agrupamentos lógicos de nós, que você pode especificar utilizando pacotes.
- › Faça a modelagem desses dispositivos e processadores, utilizando diagramas de implantação. Onde for possível, use ferramentas para

descobrir a topologia de seu sistema, percorrendo a rede do sistema.

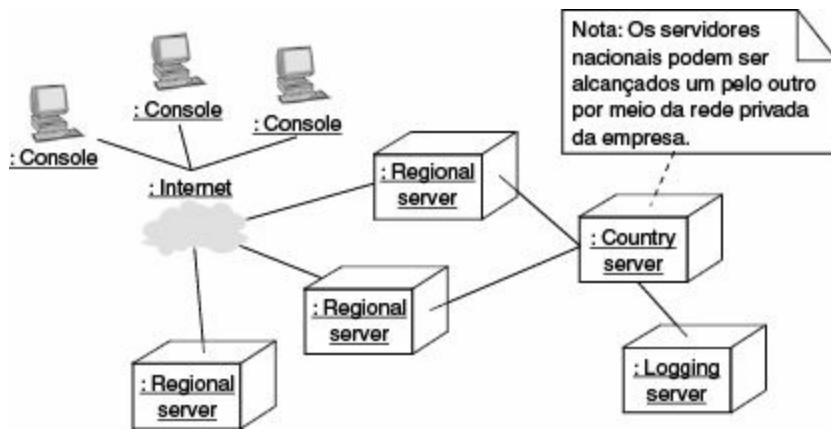
» Se for necessário focalizar a dinâmica do sistema, introduza diagramas de casos de uso para especificar os tipos de comportamento em que você está interessado e procure expandir esses casos de uso com diagramas de interação.

► *Os pacotes são examinados no [Capítulo 12](#).*

► *Os casos de uso são examinados no [Capítulo 17](#); os diagramas de interação são descritos no [Capítulo 19](#); as instâncias são discutidas no [Capítulo 13](#).*

**Nota:** Quando se faz a modelagem de um sistema totalmente distribuído, é comum reificar a própria rede como um nó. Por exemplo, a Internet poderá ser representada como um nó (conforme, na Figura 31.1, é mostrado um nó estereotipado). Também é possível reificar uma rede local (LAN) ou uma rede remota (WAN) da mesma maneira (como na [Figura 31.1](#)). Em cada um desses casos, os atributos e as operações dos nós podem ser utilizados para capturar propriedades da rede.

A [Figura 31.4](#) mostra a topologia de um sistema totalmente distribuído. Esse particular diagrama de implantação também é um diagrama de objetos, por conter somente instâncias. Existem três consoles (instâncias anônimas do nó estereotipado console), que estão vinculados à Internet (claramente um nó de uma única função). Por sua vez, existem três instâncias de servidores regionais, que servem como front- ends de servidores nacionais, somente um dos quais é mostrado. Conforme a nota indica, os servidores nacionais estão conectados a um outro, mas seus relacionamentos não são exibidos nesse diagrama.



**Figura 31.4:**

A modelagem de um sistema totalmente distribuído

Nesse diagrama, a Internet é reificada como um nó estereotipado.

## ENGENHARIA DIRETA E REVERSA

Existe somente uma modesta quantidade de engenharia direta (a criação de código a partir de modelos) que pode ser realizada com os diagramas de implantação. Por exemplo, após especificar a distribuição física de artefatos pelos nós em um diagrama de implantação, é possível usar ferramentas que então projetarão esses artefatos para o mundo real. No caso de administradores de sistemas, a utilização da UML dessa maneira ajuda você a visualizar o que pode ser uma tarefa muito complicada.

A engenharia reversa (a criação de modelos a partir de código) do mundo real de volta para os diagramas de implantação é de grande valor, especialmente para sistemas totalmente distribuídos que sofrem mudanças constantes. Você desejará fornecer um conjunto de nós estereotipados, capazes de falar a linguagem dos administradores de rede do sistema, com a finalidade de ajustar a UML ao seu domínio. A vantagem de utilização da UML está em oferecer uma linguagem-padrão capaz de abranger não somente suas necessidades, mas também as necessidades dos desenvolvedores de software do projeto.

Para fazer a engenharia reversa de um diagrama de implantação:

- › Escolha o destino desejado para a engenharia reversa. Em alguns casos, você desejará vasculhar toda a rede; em outros, poderá limitar sua pesquisa.
- › Escolha também a fidelidade de sua engenharia reversa. Em alguns casos, é suficiente que a engenharia seja aplicada apenas no nível de todos os processadores do sistema; em outros, desejará que a engenharia reversa também se estenda aos periféricos da rede do sistema.
- › Use uma ferramenta capaz de percorrer o sistema, descobrindo a sua topologia de hardware. Registre essa topologia em um modelo de implantação.
- › Ao longo do processo, ferramentas semelhantes podem ser utilizadas para descobrir os artefatos existentes em cada nó, que também podem ser registrados em um modelo de implantação. Você desejará realizar pesquisas inteligentes, pois até um computador pessoal básico é capaz de conter gigabytes de artefatos, muitos dos quais poderão não ser relevantes ao seu sistema.
- › Utilizando suas ferramentas de modelagem, crie um diagrama de implantação pela realização de consultas ao modelo. Por exemplo, você poderá começar com a visualização da topologia cliente/servidor básica e depois expandir o diagrama preenchendo certos nós com artefatos de interesse neles existentes. Exiba ou oculte os detalhes do conteúdo desse diagrama de implantação, conforme seja necessário para comunicar suas intenções.

## DICAS E SUGESTÕES

Ao criar diagramas de implantação na UML, lembre-se de que todo diagrama de implantação é apenas uma apresentação gráfica da visão estática de funcionamento de um sistema. Isso significa que nenhum único diagrama de implantação deve capturar tudo sobre a visão de implantação do sistema.

Coletivamente, todos os diagramas de implantação de um sistema representam a visão estática de implantação completa; individualmente, cada um representa apenas um aspecto.

Um diagrama de implantação bem estruturado:

- › Tem como foco a comunicação de um aspecto da visão estática de implantação do sistema.
- › Contém somente os elementos essenciais à compreensão desse aspecto.
- › Fornece detalhes consistentes com seu nível de abstração; exibe somente os adornos essenciais à compreensão.
- › Não é tão minimalista que acabe informando mal o leitor sobre a semântica importante.

Ao definir um diagrama de implantação:

- › Dê-lhe um nome capaz de comunicar seu propósito.
- › Distribua os elementos de forma a minimizar o cruzamento de linhas.
- › Organize seus elementos espacialmente, de modo que os itens que são semanticamente afins fiquem próximos.
- › Use notas e cores como indicações visuais com a finalidade de chamar atenção para características importantes de seu diagrama.
- › Use elementos estereotipados com cuidado. Escolha um pequeno conjunto de ícones comuns para seu projeto ou organização e use-os de forma consistente.

CAPÍTULO

---

32

---

# Sistemas e Modelos

## Neste capítulo

- » *Sistemas, subsistemas, modelos e visões*
- » *Modelagem da arquitetura de um sistema*
- » *Modelagem de sistemas de sistemas*
- » *Organização de artefatos de desenvolvimento*

A UML é uma linguagem gráfica para visualizar, especificar, construir e documentar os artefatos de um sistema complexo de software. Você usa a UML para fazer a modelagem de sistemas. Um modelo é uma simplificação da realidade – uma abstração de um sistema – criada com a finalidade de uma compreensão melhor do sistema. Um sistema, possivelmente decomposto em uma coleção de subsistemas, é um conjunto de elementos organizados para a realização de um propósito e é descrito como um conjunto de modelos, possivelmente sob pontos de vista diferentes. Itens como classes, interfaces, componentes e nós são partes importantes do modelo de um sistema. Na UML, os modelos são utilizados para organizar essas e todas as outras abstrações do sistema. À medida que você passa a domínios mais complexos, observará que o sistema em um nível de abstração parece um subsistema em outro nível mais alto. Na UML, é possível fazer a modelagem de sistemas e subsistemas como um todo, permitindo lidar facilmente com problemas de escala.

Modelos bem estruturados ajudam a visualizar, especificar, construir e documentar sistemas complexos sob aspectos diferentes, apesar de inter-relacionados. Sistemas bem estruturados são funcional, lógica e fisicamente coesos, formados por subsistemas fracamente acoplados.

## PRIMEIROS PASSOS

A construção de uma casa de cachorro não requer muito planejamento. As necessidades de um cão são simples e, portanto, para satisfazer a todos, com exceção de um cachorro muito exigente, você mesmo poderá fazer a casa.

A construção de uma casa ou de um prédio alto exigirá um pouco mais. As necessidades de uma família ou dos ocupantes de um prédio não são tão simples e, portanto, para satisfazer até ao cliente menos exigente, você não conseguirá fazê-lo. Em vez disso, é necessário fazer alguma modelagem. Diferentes interessados observarão o problema sob ângulos distintos e com questões diversas. É por isso que, para edificações complexas, você acabará criando plantas baixas, plantas de elevação, plantas de aquecimento/refrigeração, plantas de eletricidade, plantas de encanamentos e talvez até plantas de redes. Não existe um modelo capaz de capturar adequadamente todos os aspectos de interesse de uma construção complexa.

- As diferenças entre a construção de uma casa de cachorro e a construção de um prédio alto são apresentadas no Capítulo 1.

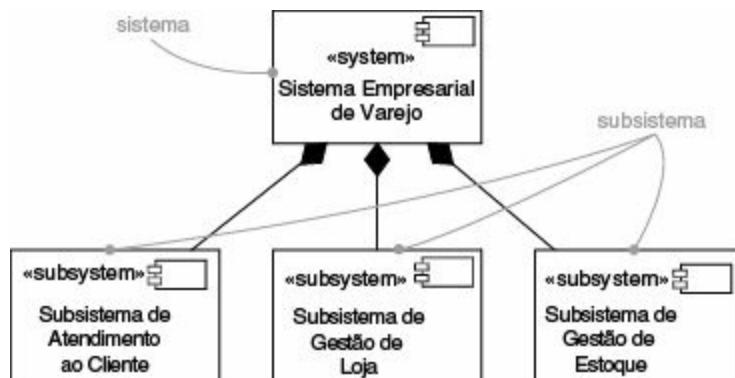
Na UML, todas as abstrações de um sistema complexo de software são organizadas em modelos, cada um representando algum aspecto relativamente independente, mas importante, do sistema em desenvolvimento. Você então utiliza diagramas para visualizar coleções interessantes dessas abstrações. Observar as cinco visões de uma arquitetura é uma maneira particularmente útil para chamar a atenção dos diferentes interessados em um sistema de software. Coletivamente, esses modelos funcionam em conjunto para fornecer uma visão completa da estrutura e comportamento de um sistema.

- Os diagramas são examinados no [Capítulo 7](#); as cinco visões da arquitetura de um software são apresentadas no [Capítulo 2](#).

No caso de sistemas maiores, você concluirá que os elementos desses sistemas podem ser decompostos de modo significativo em subsistemas separados, cada um dos quais parece um sistema menor, quando visualizado em um nível mais baixo de abstração.

A UML fornece uma representação gráfica de sistemas e subsistemas, conforme mostra a [Figura 32.1](#). Essa notação permite visualizar a decomposição do sistema em subsistemas menores. Graficamente, o sistema e o subsistema são representados como um ícone estereotipado de pacote. Os modelos e as visões não têm uma representação gráfica especial (além de serem representados como pacotes estereotipados), pois são primariamente itens manipulados por ferramentas destinadas a organizar os diferentes aspectos do sistema.

- *Os mecanismos de extensibilidade da UML são examinados no Capítulo 6; os pacotes são apresentados no Capítulo 12.*



**Figura 32.1:**  
Sistemas e subsistemas

## TERMOS E CONCEITOS

Um *sistema*, possivelmente decomposto em uma coleção de subsistemas, é um conjunto de elementos organizados para a realização de um propósito e é descrito por um conjunto de modelos, possivelmente sob pontos de vista diferentes. Um *subsistema* é um agrupamento de elementos, alguns dos quais

constituem a especificação do comportamento oferecido por outros elementos nele contidos. Graficamente, um sistema e um subsistema são representados como o ícone estereotipado de um pacote. Um *modelo* é uma simplificação da realidade, uma abstração de um sistema, criada com a finalidade de permitir uma melhor compreensão do sistema. Uma *visão* é uma projeção de um modelo, visto sob uma perspectiva ou ponto de vantagem e omite entidades que não sejam relevantes nessa perspectiva.

## SISTEMAS E SUBSISTEMAS

O sistema é o próprio item que está sendo desenvolvido e para qual os modelos são construídos. O sistema abrange todos os artefatos que constituem esse item, incluindo todos os seus modelos e elementos de modelagem, como classes, interfaces, componentes, nós e seus relacionamentos. Tudo que você precisa visualizar, especificar, construir e documentar faz parte do sistema e tudo que você não precisa visualizar, especificar, construir e documentar se encontra fora do sistema.

Na UML, um sistema é representado como um pacote estereotipado, conforme é mostrado na figura anterior. Como um pacote estereotipado, um sistema possui elementos. Se você observar detalhes do sistema, verá todos os seus modelos e elementos individuais da modelagem (incluindo os diagramas) talvez decompostos em subsistemas. Como um classificador, o sistema poderá ter instâncias (o sistema poderá ser desmembrado em várias instâncias no mundo real), atributos e operações (atores externos ao sistema poderão agir sobre o sistema como um todo), casos de uso, máquinas de estados e colaborações, todos podendo especificar o comportamento do sistema. O sistema até poderá realizar interfaces, o que é importante quando você estiver construindo sistemas de sistemas.

- Os estereótipos são examinados no [Capítulo 6](#); os pacotes são apresentados no [Capítulo 12](#); as classes são explicadas nos Capítulos [4](#) e [9](#); os casos de uso são descritos no [Capítulo 17](#); as máquinas de estados aparecem no [Capítulo 22](#); as colaborações são apresentadas

*no Capítulo 28.*

Um subsistema é simplesmente uma parte de um sistema, utilizado para decompor um sistema complexo em partes quase independentes. Um sistema em um nível de abstração poderá ser um subsistema de um sistema em um nível mais alto de abstração.

O relacionamento primário entre sistemas e subsistemas é a agregação. O sistema (o todo) poderá conter zero ou mais subsistemas (as partes). Também poderá haver relacionamentos de generalização entre sistemas e subsistemas. Utilizando a generalização, você pode fazer a modelagem de famílias de sistemas e subsistemas, alguns dos quais representam tipos gerais de sistemas e outros representam ajustes específicos desses sistemas. Os subsistemas têm várias conexões entre eles.

- ➡ A agregação e a generalização são apresentadas nos Capítulos 5 e 10.

**Nota:** *Um sistema representa um item de nível mais alto em um determinado contexto; os subsistemas que formam o sistema fornecem uma partição completa e sem sobreposições do sistema como um todo. Um sistema é um subsistema de nível mais alto.*

## MODELOS E VISÕES

Um modelo é uma simplificação da realidade, em que a realidade é definida no contexto do sistema cuja modelagem está sendo feita. Em resumo, um modelo é uma abstração de um sistema. Um subsistema representa uma partição dos elementos de um sistema maior em partes independentes; um modelo é uma partição das abstrações que visualizam, especificam, constroem e documentam o sistema. A diferença é sutil, mas importante. Você decompõe o sistema em subsistemas para poder desenvolver e desmembrar essas partes de alguma forma independente; você divide as abstrações de um sistema ou subsistema em modelos para compreender melhor o item que está sendo desenvolvido e implementado.

Assim como um sistema complexo como uma aeronave pode ter muitas partes (por exemplo, subsistemas de aerodinâmica, propulsão, aeroviários e passageiros), esses subsistemas e o sistema como um todo poderão ser modelados sob diferentes pontos de vista (como a visão de modelos estruturais, dinâmicos, de eletricidade e aquecimento/refrigeração, por exemplo).

O modelo é um tipo especial de pacote. Como raramente será necessário fazer a modelagem de modelos explicitamente, não existe uma representação gráfica especial definida para modelos na UML. Entretanto, as ferramentas necessitam manipular os modelos; portanto, tipicamente uma ferramenta usa a notação de pacote para representar o modelo conforme é percebido por essa ferramenta.

- *Os pacotes são examinados no [Capítulo 12](#).*

Sendo um pacote, um modelo possui elementos. Os modelos associados a um sistema ou subsistema dividem completamente os elementos desse sistema ou subsistema, significando que todos os elementos pertencem exatamente a um único pacote. Tipicamente, você organizará os artefatos de um sistema ou subsistema em um conjunto de modelos que não se sobreponham, cobertos pelas cinco visões da arquitetura de software, descritas em outra seção.

- *As cinco visões da arquitetura de um software são examinadas no [Capítulo 2](#).*

Um modelo (por exemplo, um modelo de um processo) poderá conter tantos artefatos (como classes ativas, relacionamentos e interações) quantos existirem em sistemas de escala; portanto, você simplesmente não consegue abranger todos esses artefatos de uma só vez. Pense em uma visão como uma projeção em um modelo. Para cada modelo, haverá uma quantidade de diagramas destinados a fornecer uma visão dos itens que pertencem ao modelo. Uma visão abrange um subconjunto de itens que pertencem ao

modelo; tipicamente uma visão não poderá ultrapassar as fronteiras do modelo. Conforme é descrito na próxima seção, não existem relacionamentos diretos entre modelos, embora seja possível traçar relacionamentos entre os elementos contidos em modelos diferentes.

- *Os diagramas são examinados no [Capítulo 7](#).*

**Nota:** A UML não determina quais modelos deverão ser utilizados para visualizar, especificar, construir e documentar um sistema, embora a Rational Unified Process sugira um conjunto provado de modelos.

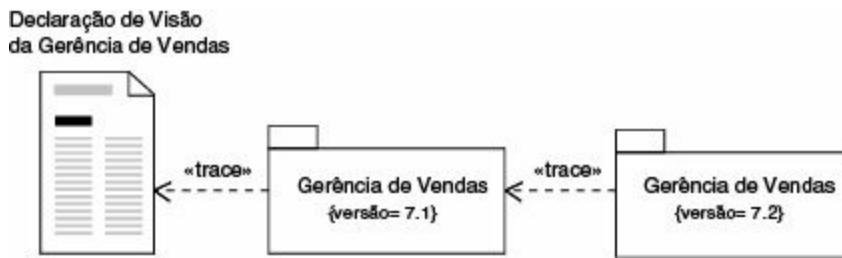
## RASTREAMENTO

A especificação de relacionamentos entre elementos como classes, interfaces, componentes e nós é uma parte estrutural importante de qualquer modelo. A especificação de relacionamentos entre elementos como documentos, diagramas e pacotes que existem em diferentes modelos é uma parte importante do gerenciamento dos artefatos de desenvolvimento de sistemas complexos, muitos dos quais poderão existir em várias versões.

- *Os relacionamentos são apresentados nos [Capítulos 5 e 10](#).*

Na UML, é possível fazer a modelagem do relacionamento conceitual entre elementos que vivem em modelos diferentes, utilizando um relacionamento de rastreamento; o rastreamento não poderá ser aplicado entre elementos de um mesmo modelo. O rastreamento é representado como uma dependência estereotipada. Com frequência, é possível ignorar a direção dessa dependência, embora tipicamente ela seja voltada na direção de um elemento mais antigo ou mais específico, como na [Figura 32.2](#). Os dois usos mais comuns para o relacionamento de rastreamento são o acompanhamento dos requisitos para a implementação (e de todos os artefatos existentes entre eles) e o acompanhamento de uma versão para outra.

- *As dependências são examinadas no [Capítulo 5](#); os estereótipos são apresentados no [Capítulo 6](#).*



**Figura 32.2:**

Relacionamentos de rastreamento

**Nota:** Na maior parte do tempo, você não desejará representar relacionamentos de rastreamento («trace») explicitamente, mas, em vez disso, deverá tratá-los como hyperlinks.

## TÉCNICAS BÁSICAS DE MODELAGEM

### MODELAGEM DA ARQUITETURA DE UM SISTEMA

O uso mais comum para os sistemas e os modelos será a organização dos elementos que você costuma usar para visualizar, especificar, construir e documentar a arquitetura de um sistema. Em definitivo, isso inclui virtualmente todos os artefatos encontrados em um projeto de desenvolvimento de um software. Ao fazer a modelagem da arquitetura do sistema, você captura decisões sobre os requisitos do sistema, seus elementos lógicos e seus elementos físicos. Você também fará a modelagem tanto de aspectos estruturais como comportamentais dos sistemas e dos padrões que dão forma a essas visões. Por fim, você desejará focalizar as costuras existentes entre os subsistemas e fazer o acompanhamento desde os requisitos até a entrega.

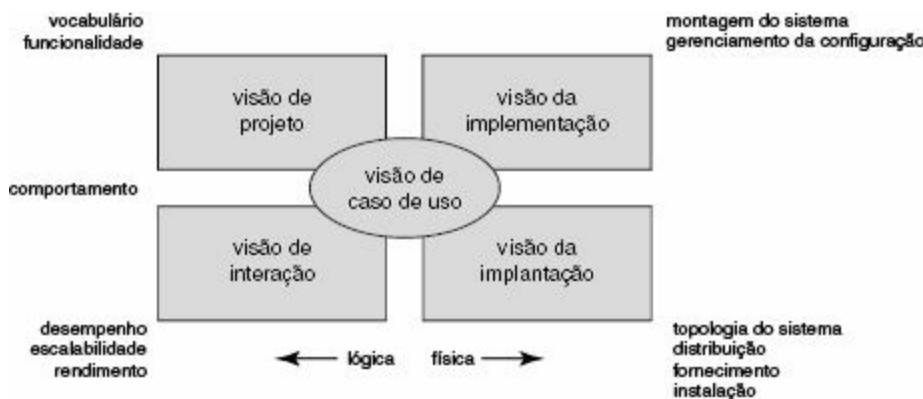
► A arquitetura e a modelagem são examinadas no [Capítulo 1](#).

Para fazer a modelagem da arquitetura de um sistema:

- » Identifique as visões que serão utilizadas para representar sua arquitetura. Com mais frequência, você desejará incluir uma visão de caso de uso, uma visão de projeto, uma visão de processo, uma visão de

implementação e uma visão de implantação, conforme mostra a [Figura 32.3](#).

- » Especifique o contexto para esse sistema, incluindo os atores que se encontram ao seu redor.



**Figura 32.3:**

A modelagem da arquitetura de um sistema

Conforme seja necessário, decomponha o sistema em seus subsistemas elementares.

- » As cinco visões da arquitetura de um software são examinadas no [Capítulo 2](#); os diagramas são apresentados no [Capítulo 7](#).

As seguintes atividades se aplicam ao sistema, como também aos seus subsistemas.

- » Especifique uma visão de caso de uso do sistema, abrangendo os casos que descrevem o comportamento do sistema, conforme são vistos pelos usuários finais, analistas e pessoal de teste. Aplique diagramas de casos de uso, para fazer a modelagem dos aspectos estáticos, e diagramas de interação, de estados e de atividades para fazer a modelagem dos aspectos dinâmicos.
- » Especifique uma visão de projeto do sistema, abrangendo classes, interfaces e colaborações que formam o vocabulário do problema e de

sua solução. Aplique diagramas de classes e diagramas de objetos para a modelagem dos aspectos estáticos e diagramas de interação, de estados e de atividades para a modelagem dos aspectos dinâmicos.

- › Especifique uma visão de interação do sistema, abrangendo threads e processos que formam os mecanismos de concorrência e sincronização do sistema. Aplique os mesmos diagramas da visão de projeto, mas focalizando as classes ativas e objetos que representam threads e processos, bem como as mensagens e o fluxo de controle.
- › Especifique uma visão de implementação do sistema, abrangendo os componentes utilizados para montar e liberar o sistema físico. Aplique diagramas de componentes para fazer a modelagem dos aspectos estáticos e diagramas de interação, de estados e de atividades para a modelagem dos aspectos dinâmicos.
- › Especifique uma visão de implantação do sistema, abrangendo os nós que formam a topologia de hardware em que o sistema é executado. Aplique diagramas de implantação para fazer a modelagem dos aspectos estáticos e diagramas de interação, de estados e de atividades para a modelagem dos aspectos dinâmicos do sistema em seu ambiente de execução.
- › Faça a modelagem dos padrões de arquitetura e padrões de projeto que formam cada um desses modelos utilizando as colaborações.

Compreenda que nunca é possível criar uma arquitetura de sistema em um único grande evento. Em vez disso, um processo bem estruturado para a UML envolve o aperfeiçoamento sucessivo da arquitetura do sistema de uma maneira orientada por casos de uso, centrada na arquitetura, iterativa e incremental.

- ➡ *O Rational Unified Process é examinado no Apêndice B.*

Para qualquer sistema, com exceção dos mais triviais, é necessário gerenciar as versões dos artefatos do sistema. Você pode usar os mecanismos de extensibilidade e valores atribuídos da UML, em particular para capturar suas decisões a respeito da versão de cada elemento.

- » *Os mecanismos de extensibilidade da UML são examinados no [Capítulo 6](#).*

## MODELAGEM DE SISTEMAS

Um sistema em um nível de abstração será parecido com um subsistema de um nível mais alto de abstração. De modo semelhante, um subsistema em um nível de abstração será parecido com um sistema completo sob a perspectiva da equipe responsável pela sua criação.

Todos os sistemas complexos exibem esse tipo de hierarquia. À medida que você passa para sistemas de complexidade cada vez maior, concluirá ser necessário decompor seus esforços em subsistemas, cada um dos quais poderá ser desenvolvido de certa forma separadamente, e crescer iterativa e incrementalmente para todo o sistema. O desenvolvimento de um subsistema se parece com o desenvolvimento de um sistema.

Para fazer a modelagem de um sistema ou subsistema:

- » Identifique as principais partes funcionais do sistema que poderão ser desenvolvidas, entregues e implementadas de certa forma independentemente. Com frequência, questões técnicas, políticas, de herança e legais determinam como você definirá as linhas ao redor de cada subsistema.
- » Para cada subsistema, especifique o respectivo contexto, da mesma maneira como seria feito para o sistema como um todo; os atores que se encontram ao redor de um subsistema abrangem todos os subsistemas vizinhos e, portanto, todos devem ser projetados para colaborar.
- » Para cada subsistema, faça a modelagem de sua arquitetura, da mesma forma como faria para o sistema como um todo.

## DICAS E SUGESTÕES

É importante escolher o conjunto adequado de modelos para visualizar, especificar, construir e documentar o sistema. Um modelo bem estruturado:

- » Proporciona uma simplificação da realidade sob um ponto de vista distinto e relativamente independente.
- » É autossuficiente por não requerer nenhum outro conteúdo para a compreensão de sua semântica.
- » Está ligeiramente relacionado a outros modelos por intermédio de relacionamentos de rastreabilidade.
- » Coletivamente (com outros modelos vizinhos) proporciona uma visão completa dos artefatos do sistema.

De modo semelhante, é importante decompor sistemas complexos em subsistemas bem estruturados. Um sistema bem estruturado:

- » É coeso, funcional, lógica e fisicamente.
- » Pode ser decomposto em subsistemas quase independentes, que por si só sejam sistemas em um nível mais baixo de abstração.
- » Pode ser visualizado, especificado, construído e documentado por meio de um conjunto de modelos inter-relacionados e que não se sobreponham.

A UML tem um símbolo gráfico para um modelo, mas é melhor evitá-lo. Faça a modelagem do sistema e não do modelo propriamente dito. As ferramentas de edição fornecerão os recursos para procurar, organizar e gerenciar conjuntos de modelos.

Ao definir um sistema ou subsistema na UML:

- » Use cada um como ponto de partida para todos os artefatos associados com esse sistema ou subsistema.
- » Mostre somente a agregação básica entre o sistema e seus subsistemas; tipicamente, você deixará os detalhes das conexões para diagramas de nível mais baixo.

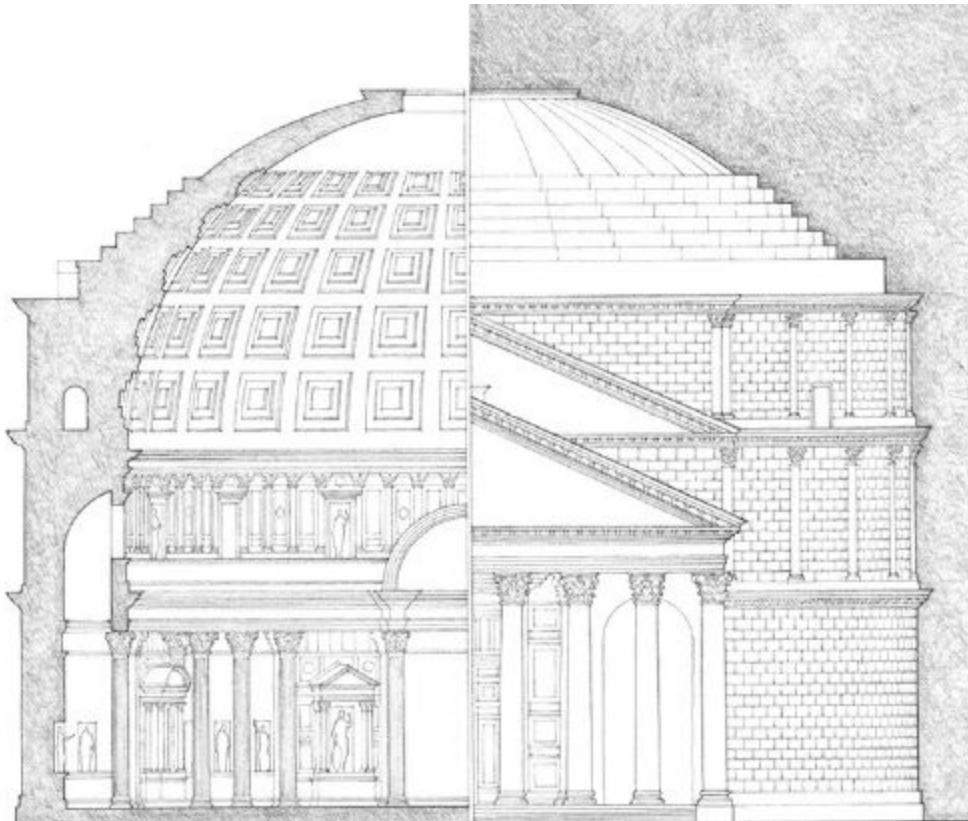
P A R T E

---

7

---

# CONCLUSÃO



CAPÍTULO

---

33

---

# Aplicando a UML

## Neste capítulo

- » *Fazendo a transição para a UML*
- » *Para onde ir a seguir*

**E**é fácil usar a UML para fazer a modelagem de problemas simples. Também é fácil fazer a modelagem de problemas difíceis, principalmente depois de conseguir fluência na linguagem.

Ler sobre a utilização da UML é uma coisa, mas somente pela *utilização* da linguagem você conseguirá dominá-la. Dependendo de sua formação, existem formas diferentes para abordar o uso da UML pela primeira vez. À medida que ganhar experiência, começará a compreender e a apreciar suas partes mais sutis.

Se você é capaz de pensar em algo, a UML pode fazer a modelagem.

## FAZENDO A TRANSIÇÃO PARA A UML

Você faz a modelagem de 80% da maioria dos problemas, utilizando cerca de 20% da UML. Coisas estruturais básicas, como classes, atributos, operações, casos de uso, componentes e pacotes, juntamente com relacionamentos estruturais básicos, como dependência, generalização e associação, são suficientes para a criação de modelos estáticos referentes a modelos para muitos tipos de domínios de problemas. Adicione a essa lista básica itens comportamentais, como máquinas de estados simples e interações e é possível fazer a modelagem de muitos aspectos úteis da dinâmica de um sistema. Será necessário utilizar as características mais

avançadas da UML somente quando você começar a modelagem de itens encontrados em situações mais complexas, como a modelagem de concorrência e de distribuição.

Um bom ponto de partida para utilizar a UML é fazer a modelagem de algumas das abstrações ou comportamentos básicos que já existem em algum dos seus sistemas. Desenvolva um modelo conceitual em UML, para obter uma estrutura básica a partir da qual poderá ampliar sua compreensão a respeito da linguagem. Posteriormente, você compreenderá melhor como as partes mais avançadas da UML trabalham em conjunto. À medida que você ataque problemas mais complexos, familiarize-se com características específicas da UML, estudando as técnicas básicas de modelagem apresentadas neste livro.

► *Um modelo conceitual para a UML é apresentado no [Capítulo 2](#).*

Se você for um iniciante na orientação a objetos:

- » Comece se familiarizando com a ideia de abstração. Exercícios em grupo com cartões CRC e análises de casos de uso são formas excelentes de desenvolver suas habilidades para identificar abstrações com clareza.
- » Faça a modelagem de uma parte estática simples de seu problema, utilizando classes, dependência, generalização e associação para se familiarizar com a visualização de sociedades de abstrações.
- » Use diagramas simples de sequências ou de colaboração para fazer a modelagem da parte dinâmica de seu problema. Construir um modelo da interação do usuário com o sistema é um bom ponto de partida e proporcionará um retorno imediato para ajudá-lo a analisar por meio de alguns dos casos de uso mais importantes do sistema.

Se você for um iniciante em modelagem:

- » Comece com uma parte de algum sistema que você já tenha construído – de preferência, implementado em alguma linguagem de programação

orientada a objetos, como Java ou C++ – e elabore um modelo da UML para essas classes e seus relacionamentos.

- › Utilizando a UML, tente capturar alguns detalhes dos idiomas ou mecanismos de programação utilizados nesse sistema, que estão na sua mente, mas você não pode incluir diretamente no código.
- › Especialmente se você tiver uma aplicação que não seja trivial, tente reconstruir um modelo de sua arquitetura, usando componentes (incluindo subsistemas) para representar seus principais elementos estruturais. Use pacotes para organizar o próprio modelo.
- › Depois de se sentir à vontade com o vocabulário da UML e antes de iniciar a escrita do código em seu próximo projeto, primeiro construa um modelo da UML para essa parte do sistema. Pense sobre a estrutura ou comportamento que foram especificados e somente então, quando estiver satisfeito com seu tamanho, forma e semântica, use o modelo como base para sua implementação.

Se você já tem experiência com outro método orientado a objetos:

- › Observe sua linguagem de modelagem atual e faça um mapeamento de seus elementos para os elementos da UML. Na maioria dos casos, você encontrará um mapeamento de um para um e observará que as modificações são cosméticas.
- › Considere algum problema de modelagem que você achou confuso ou impossível de solucionar em sua linguagem de modelagem atual. Examine algumas das características avançadas da UML que possam abordar o problema com maior clareza ou simplicidade.

Se você for um usuário avançado:

- › Certifique-se de primeiro desenvolver um modelo conceitual da UML. Você poderá perder sua harmonia de conceitos, se passar direto às partes mais sofisticadas da linguagem sem antes compreender seu vocabulário básico.

- › Dedique uma atenção particular às características da UML para a modelagem de componentes, concorrência, distribuição e padrões – questões que costumam envolver uma semântica complexa e sutil.
- › Examine também os mecanismos de extensibilidade da UML e veja como poderá ajustar a UML para falar diretamente o vocabulário de seu domínio. Tome o cuidado de resistir à tentação de chegar aos extremos permitidos por um modelo da UML, que ninguém mais, a não ser outro usuário avançado, seja capaz de reconhecer.

## PARA ONDE IR AGORA

Este guia do usuário é parte de um grupo maior de livros que, em conjunto, poderão ajudá-lo a aprender como aplicar a UML. Além do guia do usuário, existe:

- › RUMBAUGH, J., JACOBSON, I., BOOCHE, G. *The Unified Modeling Language Reference Manual, Second Edition*, Addison-Wesley, 2005. Este livro oferece uma referência abrangente à sintaxe e à semântica da UML.
- › RUMBAUGH, J., JACOBSON, I., BOOCHE, G. *The Unified Software Development Process*, Addison-Wesley, 1999. Este livro apresenta um processo de desenvolvimento recomendado para a utilização com a UML.

Para aprender mais sobre modelagem com os principais autores da UML, consulte as seguintes referências:

- › RUMBAUGH, J., BLAHA, M. *Object-Oriented Modeling and Design with UML, Second Edition*. Prentice Hall, 2005.
- › BOOCHE, G. *Object-Oriented Analysis and Design with Applications, Second Edition*. Addison-Wesley, 1993.

- › JACOBSON, I., CHRISTERSON, M., JONSSON, P. e OVERGAARD, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison- Wesley, 1992.

Informações sobre o Rational Unified Process podem ser encontradas em:

- › KRUCHTEN, P. *The Rational Unified Process: An Introduction, Third Edition*. Addison-Wesley, 2004.

As informações mais recentes sobre a UML podem ser encontradas na Web em [www.omg.org](http://www.omg.org), onde você encontrará a mais recente versão da UML padrão.

Há muitos outros livros que descrevem a UML e vários métodos de desenvolvimento, além de muitos livros que descrevem a prática da engenharia de software em geral.

## APÊNDICE

---

A

# Notação da UML

A UML é uma linguagem para visualizar, especificar, construir e documentar os artefatos de um sistema complexo de software. Sendo uma linguagem, a UML tem uma sintaxe e uma semântica bem definidas. A parte mais visível da sintaxe da UML é sua notação gráfica.

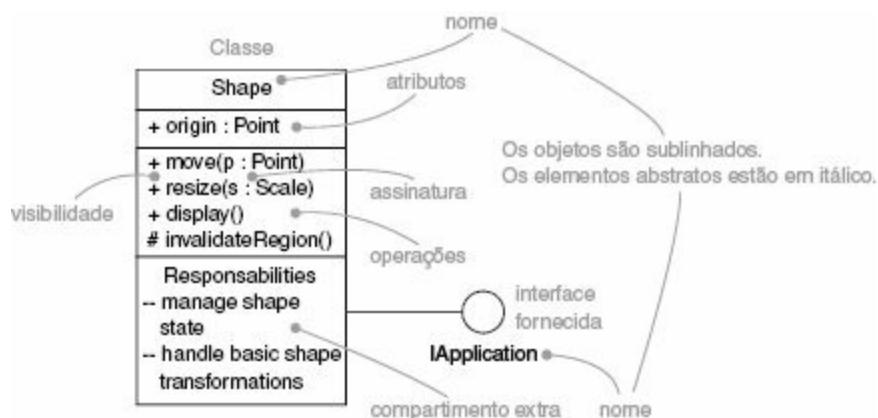
► *Uma visão geral da UML é apresentada no [Capítulo 2](#).*

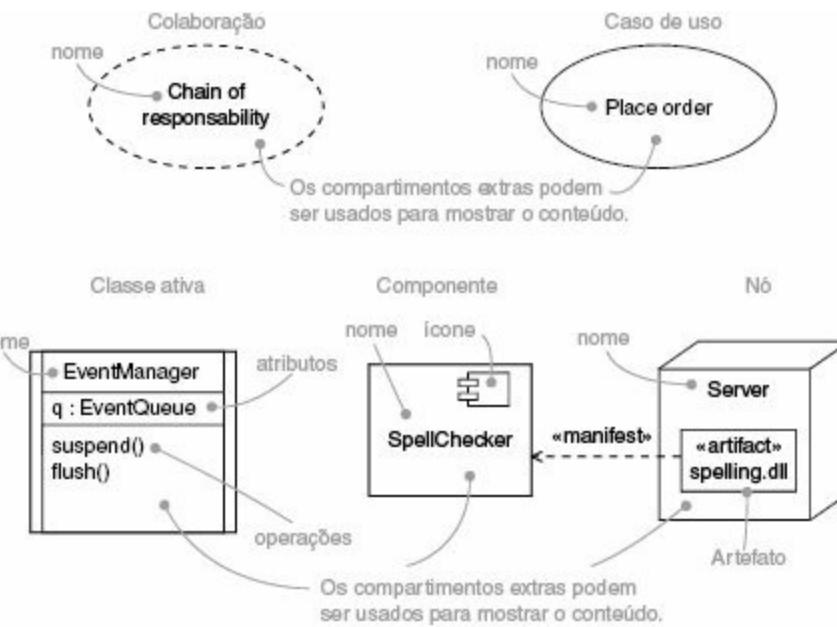
Este apêndice resume os elementos da notação da UML.

## ITENS

### ITENS ESTRUTURAIS

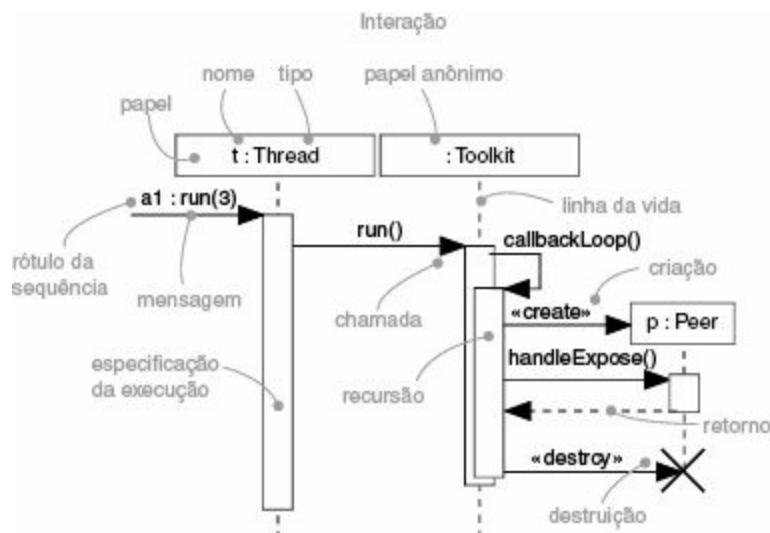
Os itens estruturais são os substantivos de modelos da UML, que incluem classes, interfaces, colaborações, casos de uso, classes ativas, componentes e nós.

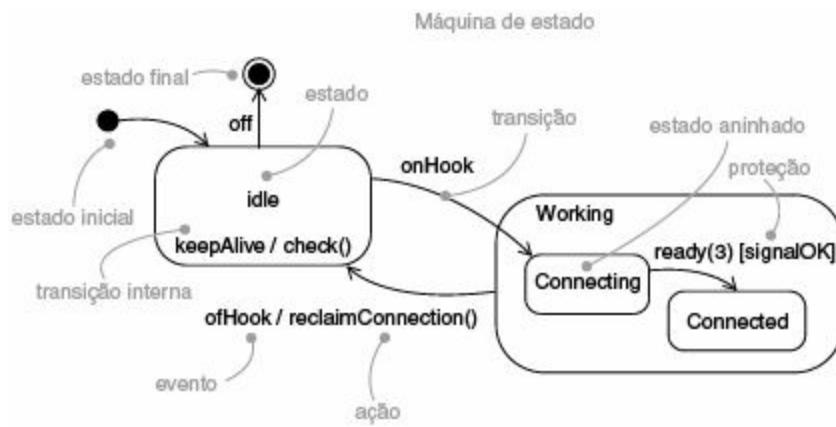




## ITENS COMPORTAMENTAIS

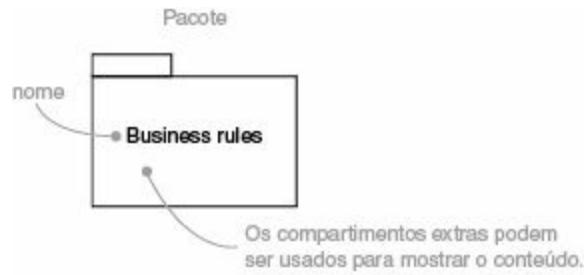
Os itens comportamentais são as partes dinâmicas dos modelos da UML, incluindo interações e máquinas de estados.





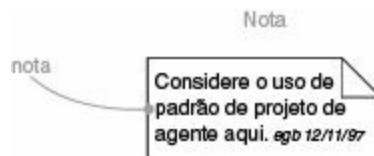
## ITENS DE AGRUPAMENTO

Os itens de agrupamento são as partes organizacionais dos modelos da UML. Isso inclui os pacotes.



## ITENS ANOTACIONAIS

Os itens anotacionais são as partes explanatórias dos modelos da UML. Isso inclui as notas.



## RELACIONAMENTOS DEPENDÊNCIA

A dependência é um relacionamento semântico entre dois itens, em que uma modificação em um item (um item independente) poderá afetar a semântica de outro item (o item dependente).



## ASSOCIAÇÃO

A associação é um relacionamento estrutural que descreve um conjunto de vínculos; o vínculo é uma conexão entre objetos.



## GENERALIZAÇÃO

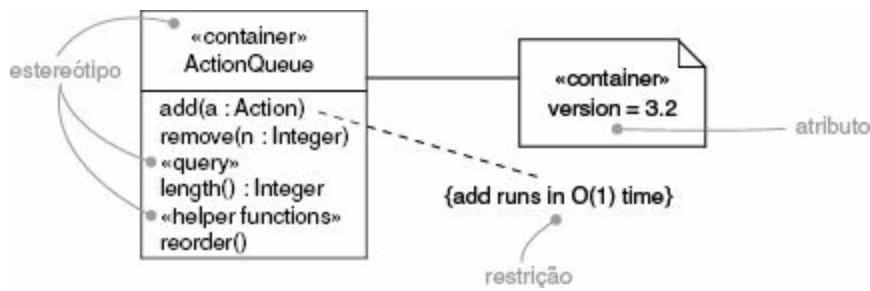
A generalização é um relacionamento de especialização/generalização, em que os objetos do elemento especializado (filho) são substituídos por objetos do elemento generalizado (pai).



## EXTENSÃO

A UML oferece três mecanismos para a extensão da sintaxe e da semântica da linguagem: os estereótipos (que representam novos elementos de modelagem), os valores atribuídos (que representam novos atributos de

modelagem) e as restrições (que representam uma nova semântica de modelagem).



## DIAGRAMAS

Um diagrama é uma apresentação gráfica de um conjunto de elementos, geralmente representada como um gráfico conectado de vértices (itens) e arcos (relacionamentos). Um diagrama é uma projeção em um sistema. A UML inclui 13 desses diagramas.

1. Diagrama de Um diagrama estrutural que mostra um conjunto de classes classes, interfaces, colaborações e seus relacionamentos.
2. Diagrama de Um diagrama estrutural que mostra um conjunto de objetos objetos e seus relacionamentos.
3. Diagrama de Um diagrama estrutural que mostra as interfaces componentes externas, incluindo portas e a composição interna de um componente.
4. Diagrama de Um diagrama estrutural que mostra as interfaces estrutura compostas externas e a composição interna de uma classe estruturada. Neste livro, combinamos o tratamento do diagrama de estrutura composta com o diagrama de componentes.
5. Diagrama de Um diagrama comportamental que mostra um conjunto casos de uso de casos de uso e atores e seus relacionamentos.

6. Diagrama de Um diagrama comportamental que mostra uma sequências interação, dando ênfase à ordenação temporal das mensagens.
7. Diagrama de Um diagrama comportamental que mostra uma comunicação interação, dando ênfase à organização estrutural de objetos que enviam e recebem mensagens.
8. Diagramas Um diagrama comportamental que mostra uma máquina de estados de estados, dando ênfase ao comportamento ordenado por eventos de um objeto.
9. Diagrama de Um diagrama comportamental que mostra um processo atividades computacional, dando ênfase ao fluxo de uma atividade para outra.
10. Diagrama Um diagrama estrutural que mostra os relacionamentos de entre um conjunto de nós, artefatos e classes implantação manifestadas e componentes. Neste livro, também especializamos a modelagem de artefatos como um diagrama de artefatos.
11. Diagrama Um diagrama estrutural que mostra a organização do de pacotes modelo em pacotes.
12. Diagrama Um diagrama comportamental que mostra uma de interação com mensagens em momentos específicos. temporização Este diagrama não é abordado neste livro.
13. Diagrama Um diagrama comportamental que combina aspectos de visão dos diagramas de atividades e dos diagramas de geral da sequências. Este diagrama não é abordado neste livro. interação

Os tipos de diagramas híbridos são permitidos; não há separação estrita entre elementos de modelo.

APÊNDICE

---

**B**

---

# Rational Unified Process

**U**m processo é um conjunto de passos parcialmente ordenados com a intenção de atingir uma meta. Na engenharia de software, sua meta é entregar, de maneira eficiente e previsível, um produto de software capaz de atender às necessidades de seu negócio.

A UML é amplamente independente de processo, significando que é possível utilizá-la com vários processos de engenharia de software. O Rational Unified Process, Processo Racional Unificado, consiste em uma abordagem desse ciclo de vida, especialmente adequada à UML. A meta do Rational Unified Process é permitir a produção de software da mais alta qualidade que atenda às necessidades do usuário final de acordo com planejamento e orçamentos previsíveis. O Rational Unified Process captura algumas das melhores práticas atuais de desenvolvimento de software de forma que pode ser adaptado a uma ampla variedade de projetos e empresas. O Rational Unified Process proporciona um procedimento disciplinado a respeito de como atribuir tarefas e responsabilidades em uma empresa de desenvolvimento de software, ao mesmo tempo em que permite que a equipe se adapte às necessidades mutáveis de um projeto.

Este apêndice apresenta um resumo dos elementos do Rational Unified Process.

## CARACTERÍSTICAS DO PROCESSO

O Rational Unified Process é um processo *iterativo*. No caso de sistemas simples, poderá ser perfeitamente viável definir sequencialmente todo o problema, estruturar a solução inteira, elaborar o software e então testar o

produto final. Entretanto, devido à complexidade e sofisticação exigidas pelos sistemas atuais, esse procedimento linear para o desenvolvimento de sistemas não é realista. Uma solução iterativa requer uma compreensão crescente do problema por meio de aperfeiçoamentos sucessivos e do desenvolvimento incremental de uma solução efetiva em vários ciclos. Inerente à solução iterativa é a flexibilidade para a acomodação de novos requisitos ou de mudanças táticas de objetivos de negócio. Também permite que o projeto identifique e solucione riscos de início, em vez de posteriormente.

As atividades do Rational Unified Process dão ênfase à criação e manutenção de *modelos* no lugar de documentos impressos. Os modelos – principalmente aqueles especificados com a utilização da UML – proporcionam representações semanticamente ricas do sistema de software em desenvolvimento. Essas representações podem ser visualizadas de várias maneiras e as informações representadas podem ser capturadas instantaneamente e controladas eletronicamente. O caráter racional subjacente ao foco em modelos no lugar de documentos impressos, empregado pelo Rational Unified Process, consiste em minimizar a sobrecarga associada à geração e manutenção de documentos e maximizar o conteúdo das informações relevantes.

Sob a orientação do Rational Unified Process, o desenvolvimento é centrado na arquitetura. O processo focaliza o desenvolvimento inicial e a linha de base da arquitetura de um software. Dispor de uma arquitetura robusta facilita o desenvolvimento paralelo, minimiza a necessidade de refazer o trabalho e aumenta a probabilidade de reutilização de componentes e a capacidade de manutenção eventual do sistema. Esse projeto de arquitetura serve como uma base sólida para o planejamento e o desenvolvimento de software a partir de componentes.

Sob a orientação do Rational Unified Process, as atividades de desenvolvimento são *orientadas por casos de uso*. O Rational Unified Process atribui uma forte ênfase à elaboração de sistemas com base em uma compreensão completa a respeito de como o sistema entregue será utilizado.

As noções de casos de uso e de cenários são empregadas para alinhar o fluxo do processo a partir da captura dos requisitos por meio de testes e para proporcionar threads que poderão ser acompanhados ao longo do desenvolvimento até o sistema entregue.

O Rational Unified Process dispõe de suporte para *técnicas orientadas a objetos*. Cada modelo é orientado a objetos. Os modelos do Rational Unified Process são baseados nos conceitos de objetos e classes e nos relacionamentos existentes entre eles e utilizam a UML como sua notação comum.

O Rational Unified Process é um processo *configurável*. Embora não exista um único processo adequado para todas as empresas de desenvolvimento de software, o Rational Unified Process pode ser ajustado e redimensionado para atender às necessidades de projetos que variam desde pequenas equipes até grandes empresas de desenvolvimento de software. O Rational Unified Process se fundamenta em uma arquitetura de processo simples e clara que proporciona uma base comum em uma família de processos e ainda pode ser modificada para acomodar várias situações. O Rational Unified Process contém uma orientação para a fim de configurar o processo atender às necessidades de uma empresa.

O Rational Unified Process encoraja o *controle de qualidade* e o *gerenciamento de riscos*, contínuos e objetivos. A avaliação da qualidade é inserida no processo, em todas as atividades e envolvendo todos os participantes, com a utilização de medidas e critérios objetivos. Não é tratada como algo pensado tarde ou como uma atividade separada. O gerenciamento de riscos é inserido no processo, de forma que os riscos para o sucesso do projeto são identificados e atacados no início do processo de desenvolvimento, quando há tempo suficiente para uma reação adequada.

## FASES E ITERAÇÕES

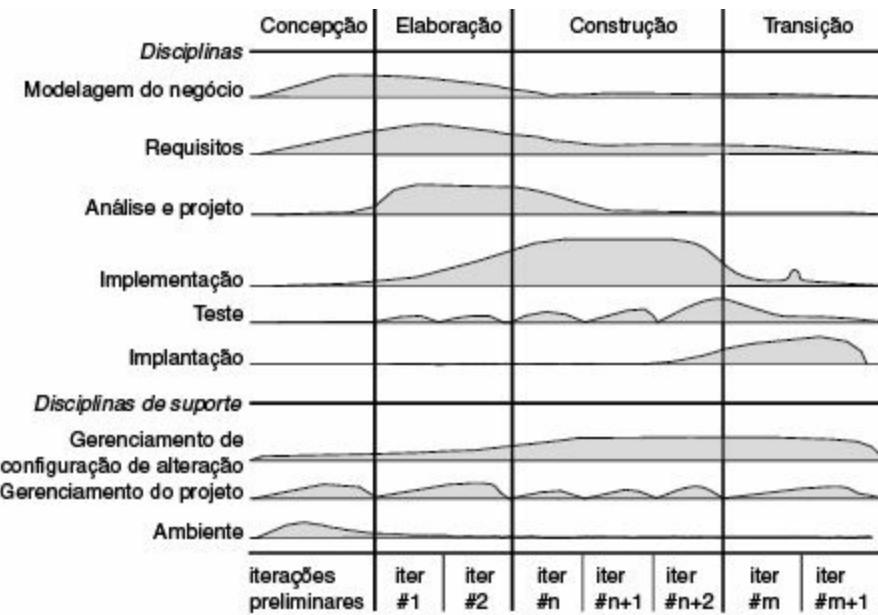
Uma fase é o período de tempo entre dois importantes marcos de progresso do processo em que um conjunto bem definido de objetivos é

alcançado, artefatos são concluídos e decisões são tomadas em relação à passagem para a fase seguinte. Conforme mostra a Figura B.1, o Rational Unified Process é composto pelas seguintes quatro fases:

- |               |   |
|---------------|---|
| 1. Concepção  | Estabelece a visão, o escopo e o plano inicial para o projeto.                    |
| 2. Elaboração | Projeta, implementa e testa uma arquitetura sólida e completa o plano de projeto. |
| 3. Construção | Cria a primeira versão do sistema operacional.                                    |
| 4. Transição  | Fornece o sistema a seus usuários finais.   |

A concepção e a elaboração abrangem as atividades criativas e de engenharia do ciclo de vida do desenvolvimento; a construção e a transição constituem sua produção.

Em cada fase, ocorrem várias iterações. Uma *iteração* representa um ciclo completo de desenvolvimento, desde a captação de requisitos na análise até a implementação e a realização de testes, resultando na versão de um projeto executável. A versão não precisa incluir um conjunto completo de características da versão comercial. Sua finalidade fornece uma base sólida para avaliar e testar, assim como uma linha de base uniforme para o próximo ciclo de desenvolvimento.



**Figura B.1:**

O ciclo de vida de desenvolvimento de um software

Cada fase e iteração têm algum foco de redução de riscos e concluem um marco de progresso bem definido. A revisão do marco de progresso proporciona um ponto no tempo para avaliar como as metas foram alcançadas e se o projeto necessitará ser reestruturado de alguma maneira para prosseguir.

## FASES

**Concepção** Durante a fase de concepção, você estabelece a visão para o sistema e delimita o escopo do projeto. Isso inclui caso de negócios, requisitos de alto nível e o plano de projeto inicial. O plano de projeto inclui critérios de sucesso, a avaliação de riscos, a estimativa de recursos necessários e um plano para a fase, mostrando a programação dos principais marcos de progresso. Durante a concepção, é comum a criação de um protótipo executável, servindo como teste para a concepção.

Essa fase normalmente envolve poucas pessoas.

No final da fase de concepção, você examina os objetivos do ciclo de vida do projeto e decide se deve prosseguir com o desenvolvimento em plena

escala.

**Elaboração** As metas da fase de elaboração são a análise do domínio do problema, o estabelecimento da fundação de uma arquitetura sólida, o desenvolvimento do plano do projeto e a eliminação dos elementos de mais alto risco do projeto. As decisões de arquitetura devem ser feitas com uma compreensão de todo o sistema. Isso implica uma descrição da maioria dos requisitos do sistema. Para verificar a arquitetura, você implementa um sistema que demonstre as escolhas de arquitetura e execute casos de uso significativos.

Esta fase envolve o arquiteto do sistema e o gerente de projeto como atores principais, bem como analistas, desenvolvedores, testadores e outros. Normalmente, a elaboração envolve uma equipe maior que a concepção e requer mais tempo.

No final da fase de elaboração, você examina o escopo e os objetivos detalhados do sistema, a escolha de arquitetura e a solução para os principais riscos, além de decidir se deve prosseguir com a construção.

**Construção** Durante a fase de construção, você desenvolve, de maneira iterativa e incremental, um produto completo, pronto para a transição à sua comunidade de usuários. Isso implica uma descrição dos requisitos restantes e de critérios de aceitação, dando corpo ao projeto e concluindo a implementação e o teste do software.

Esta fase envolve o arquiteto do sistema, o gerente de projeto e os líderes da equipe de construção, bem como a equipe completa de desenvolvimento e teste.

No final da fase de construção, você decide se o software, ambientes e usuários estão todos prontos para se tornarem operacionais.

**Transição** Durante a fase de transição, você torna o software disponível à comunidade de usuários. Observe que você se envolveu com os usuários ao longo do projeto por meio de demonstrações, workshops e versões alfa e beta. Após o sistema ser colocado nas mãos de seus usuários finais, sempre surgem questões que requerem algum desenvolvimento adicional, com a finalidade de ajustar o sistema, corrigir alguns problemas identificados ou

concluir algumas características que foram adiadas. Essa fase tipicamente é iniciada com uma versão beta do sistema, que depois é substituída pelo sistema de produção.

Os principais membros da equipe para esta fase incluem o gerente de projeto, os testadores, os especialistas em lançamento, os marqueteiros e o pessoal de vendas. Observe que esse trabalho de preparação do lançamento externo, de marketing e vendas começou muito mais cedo no projeto.

No final da fase de transição, você decide se foram alcançados os objetivos do ciclo de vida do projeto e determina se deverá iniciar outro ciclo de desenvolvimento. Esse também é um ponto em que as lições aprendidas no projeto deverão ser assimiladas para aprimorar o processo de desenvolvimento e ser aplicadas no próximo projeto.

## **ITERAÇÕES**

Cada fase do Rational Unified Process ainda pode ser dividida em iterações. Uma iteração é um ciclo completo de desenvolvimento, resultando em uma versão (interna ou externa) de um produto executável que constitui um subconjunto do produto final em desenvolvimento e cresce de modo incremental de uma iteração para outra para se tornar o sistema final. Cada iteração passa pelos vários fluxos de trabalho do processo, embora com uma ênfase diferente em cada um deles, dependendo da fase. Durante a concepção, o foco está na captação de requisitos. Durante a elaboração, o foco passa a ser a análise, o projeto e a implementação da arquitetura. Na construção, o projeto detalhado, a implementação e o teste são as atividades centrais, e a atividade central da transição é a entrega. O teste é importante ao longo do percurso todo.

## **CICLOS DE DESENVOLVIMENTO**

A passagem pelas quatro principais fases é chamada um ciclo de desenvolvimento e resulta na geração de um software. O primeiro passo das quatro fases é chamado o ciclo inicial de desenvolvimento. A menos que a

vida do produto seja interrompida, um produto existente evoluirá para sua próxima geração pela repetição da mesma sequência de fases de concepção, elaboração, construção e transição. Isso é a evolução do sistema, de modo que os ciclos de desenvolvimento posteriores aos ciclos iniciais são seus ciclos de evolução.

## DISCIPLINAS

O Rational Unified Process é composto por nove disciplinas.

1. Modelagem de Descreve a estrutura e a dinâmica da empresa.  
negócio
2. Requisitos Descreve os requisitos usando várias abordagens.
3. Análise e Descreve as várias visões da arquitetura.  
projeto
4. Implementação Leva em consideração o desenvolvimento do software, o teste da unidade e a integração.
5. Teste Descreve casos de teste, procedimentos e medidas para acompanhamento de erros.
6. Entrega Abrange listas, notas de versão, treinamento e outros aspectos da entrega de um aplicativo.
7. Gerenciamento Controla as modificações e mantém a integridade dos artefatos do projeto e das atividades de configuração.
8. Gerenciamento Descreve várias estratégias para o trabalho com um processo iterativo.
9. Ambiente Abrange a infraestrutura necessária para o desenvolvimento do sistema.

Capturado em cada disciplina está um conjunto de atividades e artefatos correlacionados. Um *artefato* é algum documento, relatório ou programa executável, que é produzido, manipulado ou consumido. Uma *atividade* descreve as tarefas – criando, realizando e verificando etapas – executadas pelos trabalhadores para criar ou modificar artefatos, juntamente com as técnicas e diretrizes para a realização de tarefas, possivelmente incluindo a utilização de ferramentas para ajudar a automação de algumas das tarefas.

Conexões importantes entre os artefatos estão associadas a alguns desses fluxos de trabalho de processo. Por exemplo, o modelo de caso de uso gerado durante a captação de requisitos é *realizado pelo* modelo de projeto a partir do processo de análise e estruturação, *implementado pelo* modelo de implementação a partir do processo de implementação e *verificado pelo* modelo de teste a partir da disciplina de teste.

## ARTEFATOS

Cada atividade do Rational Unified Process tem artefatos associados, ou exigidos como uma entrada ou gerados como uma saída. Alguns artefatos são utilizados com a finalidade de direcionar a entrada para atividades subsequentes, mantidos como recursos de referência sobre o projeto ou gerados em um formato como entregas contratuais.

## MODELOS

Os modelos são o tipo mais importante de artefato do Rational Unified Process. Um modelo é uma simplificação da realidade, criado para proporcionar uma melhor compreensão do sistema que está sendo criado. No Rational Unified Process, existem alguns modelos que, em conjunto, abrangem todas as decisões importantes para a visualização, especificação, construção e documentação de um sistema complexo de software.

► A modelagem é examinada no [Capítulo 1](#).

1. Modelo de caso de uso de negócio	Estabelece uma abstração da empresa.
2. Modelo de análise do negócio	Estabelece o contexto do sistema.
3. Modelo de caso de uso	Estabelece os requisitos funcionais do sistema.
4. Modelo de análise (opcional)	Estabelece um projeto conceitual.
5. Modelo de projeto	Estabelece o vocabulário do problema e de sua solução.
6. Modelo de dados (opcional)	Estabelece a representação de dados para bancos de dados e outros repositórios.
7. Modelo de implantação	Estabelece a topologia do hardware em que o sistema é executado, bem como a concorrência do sistema e os mecanismos de sincronização.
8. Modelo de implementação	Estabelece as partes utilizadas para montar e liberar o sistema físico.

Uma visão é uma projeção em um modelo. No Rational Unified Process, a arquitetura de um sistema é capturada em cinco visões interligadas: a visão de projeto, a visão de processo, a visão de implantação, a visão de implementação e a visão de caso de uso.

► A arquitetura é examinada no [Capítulo 2](#).

## OUTROS ARTEFATOS

Os artefatos do Rational Unified Process são categorizados como artefatos de gerenciamento ou artefatos técnicos. Os artefatos técnicos do Rational Unified Process podem ser divididos em cinco conjuntos principais.

1. Conjunto de requisitos Descreve o que o sistema deve fazer.
2. Conjunto de análise Descreve como o sistema é construído. e projeto
3. Conjunto de teste Descreve a abordagem pela qual o sistema é validado e verificado.
4. Conjunto de implementação Descreve a montagem dos componentes do software desenvolvido.
5. Conjunto de implantação Fornece todos os dados para a configuração de implantação.

**Conjunto de requisitos** Agrupa todas as informações descrevendo o que o sistema deve fazer. Poderá incluir um modelo de caso de uso, um modelo de requisitos não funcionais, um modelo de domínio, um modelo de análise e outras formas de expressão das necessidades do usuário, incluindo, mas sem estar limitado a, simulações, protótipos de interface, restrições regulatórias e assim por diante.

**Conjunto de projeto** Agrupa informações descrevendo como o sistema é construído e captura as decisões referentes ao modo como o sistema é construído, levando em consideração todas as restrições de tempo, orçamento, herança, reutilização, objetivos de qualidade e assim por diante. Isso pode abranger um modelo de projeto, um modelo de teste e outras formas de expressão da natureza do sistema, incluindo, mas sem estar limitado a, protótipos e arquiteturas executáveis.

**Conjunto de teste** Agrupa as informações sobre o teste do sistema, incluindo scripts, casos de teste, medições de acompanhamento de erro e critérios de aceitação.

**Conjunto de implementação** Agrupa todas as informações sobre os elementos de software que compõem o sistema, incluindo, mas sem estar limitado a, código- fonte em várias linguagens de programação, arquivos de

configuração, arquivos de dados, componentes de software e assim por diante, juntamente com as informações que descrevem como montar o sistema.

**Conjunto de implantação** Agrupa todas as informações sobre a forma como o software é atualmente empacotado, entregue, instalado e executado no ambiente de destino.

# Glossário

**abstração** A característica essencial de uma entidade que a diferencia de todos os outros tipos de entidades. Uma abstração define uma fronteira relativa à perspectiva do observador.

**ação** Uma computação executável que resulta em alteração do estado do sistema ou no retorno de um valor.

**ação assíncrona** Uma solicitação em que o objeto emissor não faz uma pausa para aguardar os resultados.

**ação síncrona** Uma solicitação em que o objeto emissor faz uma pausa para aguardar os resultados.

**adorno** Detalhe da especificação de um elemento, acrescentada à sua notação gráfica básica.

**agregação** Uma forma especial de associação, que especifica o relacionamento parte-todo entre o agregado (o todo) e o componente (a parte).

**agregada** Uma classe que representa o “todo” em um relacionamento de agregação.

**argumento** Um valor específico correspondente a um parâmetro.

**arquitetura** O conjunto de decisões significativas sobre a organização de um sistema de software, a seleção de elementos estruturais e suas interfaces que compõem o sistema, juntamente com seu comportamento, conforme é especificado nas colaborações entre esses elementos, a composição desses elementos estruturais e comportamentais em subsistemas progressivamente

maiores e o estilo de arquitetura que orienta essa organização – esses elementos e suas interfaces, suas colaborações e sua composição. A arquitetura de software não está relacionada somente com a estrutura e o comportamento, mas também com a utilização, funcionalidade, desempenho, flexibilidade, reutilização, abrangência, restrições e ajustes econômicos e tecnológicos e questões estéticas.

**artefato** Um conjunto de informações utilizado ou produzido por um processo de desenvolvimento de software ou sistema existente.

**assinatura** O nome e os parâmetros de uma operação.

**associação** Um relacionamento estrutural que descreve um conjunto de vínculos, em que o vínculo é uma conexão entre objetos; o relacionamento semântico entre dois ou mais classificadores que envolve as conexões entre suas instâncias.

**associação binária** Uma associação entre duas classes.

**associação enésima** A associação entre três ou mais classes.

**ativação** A execução de uma operação.

**ativar** Executar a transição de um estado.

**atividade** Comportamento expresso como um conjunto de ações conectadas por fluxos de controle e dados.

**ator** Um conjunto coerente de papéis que os usuários de casos de uso desempenham ao interagir com os casos de uso.

**atributo** Uma propriedade nomeada de um classificador, descrevendo uma faixa de valores que as instâncias da propriedade poderão manter.

**booleano** Uma enumeração cujos valores são verdadeiros ou falsos.

**característica** Uma propriedade, como uma operação ou um atributo, que é encapsulada em outra entidade, como uma interface, uma classe ou um tipo de dados.

**característica comportamental** Uma característica dinâmica de um elemento, como uma operação.

**característica estrutural** Uma característica estática de um elemento.

**cardinalidade** O número de elementos existentes em um conjunto.

**cartões CRC** Os cartões CRC (classe, responsabilidade e colaboração) são

uma metodologia criada por Kent Beck e Ward Cunningham para a modelagem de software orientada a objetos.

**caso de uso** A descrição de um conjunto de sequências de ações, incluindo variantes, que um sistema realiza, fornecendo o resultado observável do valor de um ator.

**cenário** Uma sequência específica de ações que ilustram o comportamento.

**centrado na arquitetura** No contexto do ciclo de vida de desenvolvimento do software, um processo que focaliza o desenvolvimento inicial e a linha de base da arquitetura de um software e então utiliza a arquitetura do sistema como um artefato primário para conceitualizar, construir, gerenciar e evoluir o sistema em desenvolvimento.

**chamada síncrona** Uma solicitação na qual o objeto emissor faz uma pausa para aguardar uma resposta.

**classe** A descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.

**classe abstrata** Uma classe que não pode ser instanciada diretamente.

**classe ativa** Uma classe cujas instâncias são objetos ativos.

**classe concreta** Uma classe que pode ser instanciada diretamente.

**classe de associação** Um elemento de modelagem que tem propriedades de classe e de associação. Uma classe de associação pode ser vista como uma associação que também tem propriedades de classe ou como uma classe que também tem propriedades de associação.

**classificação dinâmica** Uma variação semântica da generalização em que um objeto poderá mudar de tipo ou de papel.

**classificação estática** Uma variação semântica de uma generalização, em que um objeto poderá não alterar seu tipo, mas poderá mudar de papel.

**classificação múltipla** Uma variação semântica da generalização, em que um objeto pode pertencer diretamente a mais de uma classe.

**classificador** O mecanismo que descreve características estruturais e comportamentais. Os classificadores incluem classes, interfaces, tipos de dados, sinais, componentes, nós, casos de uso e subsistemas.

**cliente** Um classificador que solicita serviços de outro classificador.

**colaboração** Uma sociedade de papéis e outros elementos que trabalham em conjunto para proporcionar algum comportamento cooperativo maior do que a soma de todas as suas partes; a especificação de como um elemento, como casos de uso ou operações, é realizado por um conjunto de classificadores e associações desempenhando papéis específicos e utilizados de uma determinada maneira.

**comentário** Uma anotação anexada a um elemento ou a uma coleção de elementos.

**componente** Uma parte física e substituível de um sistema com o qual está em conformidade e proporciona a realização de um conjunto de interfaces.

**comportamento** A especificação de uma computação executável.

**composição** Uma forma de agregação com propriedade bem definida e tempo de vida coincidente das partes pelo todo; as partes com multiplicidade não fixada poderão ser criadas após a própria composição, mas, uma vez criadas, vivem e morrem com ela; essas partes também podem ser removidas explicitamente antes da morte do elemento composto.

**composta** Uma classe que é relacionada a uma ou mais classes por um relacionamento de composição.

**concepção** A primeira fase do ciclo de vida do desenvolvimento de um software, em que a ideia básica para o desenvolvimento é conduzida ao ponto de ser suficientemente bem fundamentada para garantir a passagem à fase de elaboração.

**concorrência** A ocorrência de duas ou mais atividades durante o mesmo intervalo de tempo. A concorrência pode ser realizada com intercalação ou executada simultaneamente por duas ou mais threads.

**condição de guarda** Uma condição que precisa ser satisfeita para permitir que uma transição associada seja ativada.

**construção** A terceira fase do ciclo de vida de desenvolvimento de um software, em que o software é levado da linha de base executável de uma arquitetura até o ponto em que está pronto para a transição para uma comunidade de usuários.

**container** Um objeto que existe para conter outros objetos e que proporciona

operações para acessar ou iterar seu conteúdo.

**contexto** Um conjunto de elementos relacionados para um determinado propósito, como especificar uma operação.

**delegação** A habilidade de um objeto em enviar uma mensagem a outro objeto como resposta a uma mensagem.

**dependência** Um relacionamento semântico entre dois itens, em que a alteração de um item (o item independente) poderá afetar a semântica do outro item (um item dependente).

**destinatário** O objeto ao qual uma mensagem é enviada.

**diagrama** A apresentação gráfica de um conjunto de elementos, em geral representada como um gráfico conectado de vértices (itens) e arcos (relacionamentos).

**diagrama de atividades** Um diagrama que mostra o fluxo de controle e dados de uma atividade para outra; os diagramas de atividades abrangem a visão dinâmica do sistema.

**diagrama de caso de uso** Um diagrama que mostra um conjunto de casos de uso e atores e seus relacionamentos; o diagrama de caso de uso abrange a visão estática de caso de uso de um sistema.

**diagrama de classes** O diagrama que mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos. Os diagramas de classes abrangem a visão estática de projeto de um sistema; um diagrama que mostra a coleção de elementos declarativos (estáticos).

**diagrama de componentes** Um diagrama que mostra a organização e as dependências existentes em um conjunto de componentes; os diagramas de componentes abrangem a visão estática de implementação de um sistema.

**diagrama de comunicação** Um diagrama de interação que dá ênfase à organização estrutural de objetos que enviam e recebem mensagens; um diagrama que mostra as interações organizadas ao redor de instâncias e os vínculos entre elas.

**diagrama de estados** Um diagrama que mostra uma máquina de estados; os diagramas de estados abrangem a visão dinâmica de um sistema.

**diagrama de implantação** Um diagrama que mostra a configuração dos nós

de processamento em tempo de execução e os componentes que nele existem; um diagrama de implantação abrange a visão estática de implantação de um sistema.

**diagrama de interação** Um diagrama que mostra uma interação, composta por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que podem ser trocadas entre eles; os diagramas de interação abrangem a visão dinâmica de um sistema; um termo genérico aplicado a vários tipos de diagramas que dão ênfase às interações de objetos, incluindo diagramas de comunicação e diagramas de sequências. Os diagramas de atividades estão relacionados, mas são semanticamente distintos.

**diagrama de objetos** Um diagrama que mostra um conjunto de objetos e seus relacionamentos em um ponto no tempo; os diagramas de objetos abrangem a visão estática de projeto ou a visão estática de processo de um sistema.

**diagrama de sequência** Um diagrama de interação que dá ênfase à ordenação temporal de mensagens.

**domínio** Uma área de conhecimento ou de atividade, caracterizada por um conjunto de conceitos e terminologia compreendidos pelos participantes dessa área.

**elaboração** A segunda fase do ciclo de vida de desenvolvimento de um software, em que a visão do produto e sua arquitetura são definidos.

**elemento** Um constituinte atômico de um modelo.

**elemento derivado** Um elemento do modelo que pode ser computado a partir de um elemento, mas que é mostrado por questão de clareza ou é incluído com o propósito de projeto, ainda que não acrescente informações semânticas.

**elemento parametrizado** O descritor de um elemento com um ou mais parâmetros não vinculados.

**elisão** Modelagem de um elemento com certas partes ocultas para simplificar a visão.

**emissor** O objeto a partir do qual uma mensagem é enviada.

**engenharia direta** O processo de transformar um modelo em código pelo

mapeamento para uma linguagem de implementação específica.

**engenharia reversa** O processo de transformação de um código em um modelo pelo mapeamento a partir de uma linguagem de implementação específica.

**enumeração** Uma lista de valores nomeados, utilizada como a faixa de um determinado tipo de atributo.

**envio** A passagem da instância de uma mensagem do objeto emissor para um objeto destinatário.

**escopo** O contexto que dá significado a um nome.

**espaço de nome** Um escopo em que os nomes podem ser definidos e utilizados; em um espaço de nome, cada nome denota um único elemento.

**especificação** Uma declaração textual da sintaxe e da semântica de um bloco de construção específico; uma descrição declarativa do que alguma coisa é ou faz.

**estado** Uma condição ou situação ao longo da vida de um objeto, durante a qual ele satisfaz alguma condição, realiza uma atividade ou aguarda algum evento.

**estado composto** Um estado formado por subestados concorrentes ou subestados em disjunção.

**estado de ação** Um estado que representa a execução de uma ação atômica, tipicamente a chamada de uma operação.

**estereótipo** Uma extensão do vocabulário da UML, que permite a criação de novos tipos de blocos de construção derivados dos já existentes, mas que são específicos ao seu problema.

**estímulo** Uma operação ou um sinal.

**evento** A especificação de uma ocorrência significativa, que tem uma localização no tempo e no espaço; no contexto de uma máquina de estados, a ocorrência de um evento é capaz de ativar a transição de um estado.

**evento de tempo** Um evento que denota o tempo decorrido desde que se entrou no estado atual.

**execução** A execução de um modelo dinâmico.

**exportar** No contexto dos pacotes, tornar visível um elemento fora do espaço

do nome que o contém.

**expressão** Uma sequência de caracteres avaliada como um valor de um determinado tipo.

**expressão booleana** Uma expressão avaliada como um valor booleano.

**expressão de ação** Uma expressão que é avaliada como uma coleção de ações.

**expressão de tempo** Uma expressão calculada com um valor de tempo absoluto ou relativo.

**expressão de tipo** Uma expressão avaliada como uma referência a um ou mais classificadores.

**extremidade da associação** O ponto final de uma associação, que conecta a associação a um classificador.

**extremidade do vínculo** Uma instância de uma extremidade de uma associação.

**fase** O intervalo de tempo entre dois marcos de progresso importantes do processo de desenvolvimento, durante o qual um conjunto bem definido de objetivos é atingido, artefatos são concluídos e decisões são tomadas em relação à passagem para a fase seguinte.

**filha** Uma subclasse.

**filho** Uma subclasse ou outro elemento especializado.

**foco de controle** Um símbolo em um diagrama de sequência, mostrando o período de tempo durante o qual um objeto realiza uma ação diretamente ou por meio de uma operação subordinada.

**fornecedor** Um tipo, classe ou componente que fornece serviços que podem ser chamados por outros.

**framework** Um padrão de arquitetura que fornece um template extensível para aplicações em um domínio.

**generalização** Um relacionamento de especialização/generalização, em que objetos do elemento especializado (o filho) podem ser substituídos por objetos do elemento generalizado (o pai).

**herança** O mecanismo pelo qual elementos mais específicos incorporam a estrutura e o comportamento de elementos mais gerais.

**herança de implementação** A herança da implementação de um elemento mais geral; também inclui a herança da interface.

**herança da interface** A herança da interface de um elemento mais específico; não inclui a herança da implementação.

**herança múltipla** Uma variação semântica da generalização, em que um filho pode ter mais de um pai.

**herança única** Uma variação semântica de uma generalização, em que um filho pode ter somente um pai.

**hierarquia de conteúdos** A hierarquia de um espaço de nome, formada por elementos e os relacionamentos de agregação existentes entre eles.

**implementação** Uma realização concreta do contrato declarado por uma interface; a definição de como algo é construído ou computado.

**import** No contexto dos pacotes, uma dependência que mostra o pacote cujas classes poderão ser referenciadas em um determinado pacote (incluindo pacotes recursivamente nele incorporados) sem fornecer um nome qualificado.

**incompleto** A modelagem de um elemento, em que faltam certas partes.

**inconsistente** A modelagem de um elemento em que a integridade do modelo não é garantida.

**incremental** No contexto do ciclo de vida do desenvolvimento de um software, é um processo que envolve a integração contínua da arquitetura do sistema para a produção de versões, cada nova versão incorporando aperfeiçoamentos incrementais em relação à anterior.

**instância** Uma manifestação concreta de uma abstração; uma entidade à qual um conjunto de operações pode ser aplicada e que tem um estado para armazenar os efeitos das operações.

**integridade** Como as coisas se relacionam umas com as outras, de maneira apropriada e consistente.

**interação** Um comportamento que abrange um conjunto de mensagens trocadas entre um conjunto de objetos em um determinado contexto para a realização de um propósito.

**interface** Uma coleção de operações utilizadas para especificar o serviço de

uma classe ou de um componente.

**iteração** Um conjunto distinto de atividades com um plano de linha de base e um critério de avaliação que resulta em uma versão, interna ou externa.

**iterativo** No contexto do ciclo de vida de desenvolvimento do software, um processo que envolve o gerenciamento de uma sequência de versões executáveis.

**linha de vida do objeto** Uma linha em um diagrama de sequências, que representa a existência de um objeto em um período de tempo.

**localização** A posição de um artefato em um nó.

**mãe** Uma superclasse.

**máquina de estados** Um comportamento que especifica as sequências de estados pelas quais um objeto passa durante seu tempo de vida como resposta a eventos, juntamente com sua resposta a esses eventos.

**marca de tempo** Uma denotação do tempo em que um evento ocorre.

**mecanismo** Um projeto-padrão que é aplicado a uma sociedade de classes.

**mecanismo de extensibilidade** Um dos três mecanismos (estereótipos, valores atribuídos e restrições) que permitem estender a UML de maneiras controladas.

**mensagem** A especificação de uma comunicação entre objetos que contêm informações à espera de que a atividade acontecerá; o destinatário da instância de uma mensagem costuma ser considerado a instância de um evento.

**metaclasse** Uma classe cujas instâncias são classes.

**método** A implementação de uma operação.

**modelo** Uma simplificação da realidade, criada com a finalidade de proporcionar uma melhor compreensão do sistema que está sendo gerado; uma abstração semanticamente próxima de um sistema.

**multiplicidade** A especificação de uma faixa de números cardinais, que um conjunto pode assumir.

**nível de abstração** Um lugar na hierarquia de abstrações, abrangendo desde os níveis mais altos de abstração (muito abstratos) até os mais baixos (muito concretos).

**nó** Um elemento físico existente em tempo de execução que representa um recurso computacional, geralmente dispondo de pelo menos alguma memória e, na maioria das vezes, capacidade de processamento.

**nome** O que você pode usar para denominar um item, relacionamento ou diagrama; uma sequência de caracteres utilizada para identificar um elemento.

**nota** Um símbolo gráfico para a representação de restrições ou de comentários anexados a um elemento ou a uma coleção de elementos.

**Object Constraint Language (OCL)** Uma linguagem formal utilizada para expressar restrições secundárias de efeito livre.

**objeto** Uma manifestação concreta de uma abstração; uma entidade com uma fronteira bem definida e uma identidade que encapsula estado e comportamento; a instância de uma classe.

**objeto ativo** Um objeto a que pertencem um processo ou thread e que é capaz de iniciar uma atividade de controle.

**objeto persistente** Um objeto que existe depois que o processo ou a thread que o criou deixa de existir.

**objeto transiente** Um objeto que existe somente durante a execução da thread ou do processo que o criou.

**ocorrência** Uma instância de um evento, incluindo uma localização no espaço e no tempo e um contexto. Uma ocorrência pode acionar uma transição da máquina de estados.

**ocultar** A modelagem de um elemento com determinadas partes ocultas para simplificar a exibição.

**operação** A implementação de um serviço que pode ser solicitado por qualquer objeto da classe com a finalidade de afetar um comportamento.

**orientado a caso de uso** No contexto do ciclo de vida do desenvolvimento de um software, um processo em que os casos de uso são utilizados como artefatos primários para o estabelecimento do comportamento desejado do sistema, para verificar e validar a arquitetura do sistema, para testar e para fazer a comunicação entre os participantes do projeto.

**orientado a riscos** No contexto do ciclo de vida de desenvolvimento de

software, um processo em que cada nova versão é dedicada a atacar e reduzir os riscos mais significativos para o sucesso do projeto.

**pacote** Um mecanismo de propósito geral para a organização de elementos em grupos.

**padrão** Uma solução comum para um problema comum em um determinado contexto.

**pai** Uma superclasse ou outro elemento mais geral.

**papel** Um participante estrutural em um determinado contexto.

**parâmetro** A especificação de uma variável que pode ser alterada, passada ou retornada.

**parâmetro formal** Um parâmetro.

**parâmetro real** Uma função ou argumento de procedimento.

**pós-condição** Uma restrição que precisa ser verdadeira na conclusão de uma operação.

**precondição** Uma restrição que precisa ser verdadeira quando uma operação é chamada.

**processo** Um fluxo de controle pesado, que pode ser executado concorrentemente com outros processos.

**produto** Os artefatos de desenvolvimento, como modelos, código, documentação e planos de trabalho.

**projeção** Um mapeamento a partir de um conjunto para um subconjunto dele.

**propriedade** Um valor nomeado, denotando uma característica de um elemento.

**pseudoeestado** Um nó em uma máquina de estados que tem a forma de um estado, mas não se comporta como um estado; os pseudoeestados incluem nós inicial, final e de histórico.

**qualificador** O atributo de uma associação cujos valores dividem o conjunto de objetos relacionados a um objeto em uma associação.

**raia de natação** A partição de um diagrama de sequência para a organização de responsabilidades para ações.

**realização** Os relacionamentos semânticos entre os classificadores, em que

**um classificador** especifica um contrato cuja execução é garantida por outro classificador.

**receber** A manipulação da instância de uma mensagem passada pelo objeto emissor.

**refinamento** Um relacionamento que representa a especificação completa de algo já especificado em um determinado nível de detalhe.

**reificação** Tratamento de elemento abstrato como se fosse um objeto real.

**relacionamento** Uma conexão semântica entre elementos.

**requisito** Uma característica, propriedade ou comportamento desejado de um sistema.

**responsabilidade** Um contrato ou obrigação em um tipo ou de uma classe.

**restrição** Uma extensão da semântica de um elemento da UML, permitindo acrescentar novas regras ou modificar as existentes.

**restrição de tempo** Uma declaração semântica sobre o valor de tempo relativo ou absoluto ou a duração.

**sequência de caracteres** Uma sequência de caracteres de texto.

**sinal** A especificação de um estímulo assíncrono comunicado entre instâncias.

**sistema** Um conjunto de elementos organizados para a realização de um propósito específico e descrito por um conjunto de modelos, provavelmente sob diferentes pontos de vista. Frequentemente, um sistema é decomposto em um conjunto de subsistemas.

**solicitação** A especificação de um estímulo enviado a um objeto.

**subclasse** Em um relacionamento de generalização, o filho, que é a especialização de uma outra classe.

**subestado** Um estado que é parte de um estado composto.

**subestado concorrente** Um subestado ortogonal que pode ser mantido simultaneamente com outros subestados contidos no mesmo estado composto.

**subestado disjunto** Um subestado que não pode ser mantido simultaneamente com outros subestados contidos no mesmo estado composto.

**subsistema** Um agrupamento de elementos, em que alguns constituem uma especificação do comportamento oferecido pelos outros elementos contidos nesse agrupamento.

**superclasse** Em um relacionamento de generalização, o pai, que é a generalização de uma outra classe.

**tarefa** Um caminho único para execução de um programa, um modelo dinâmico ou alguma outra representação do fluxo de controle; uma thread ou um processo.

**template** Um elemento parametrizado.

**tempo** Um valor representando um momento absoluto ou relativo.

**thread** Um fluxo leve de controle que pode ser executado concurrentemente com outras threads no mesmo processo.

**tipo** O relacionamento entre um elemento e sua classificação.

**tipo de dados** Um tipo cujos valores não têm identidade. Os tipos de dados incluem tipos primitivos inerentes (como números e sequências de caracteres), além de tipos enumerados (como booleano).

**tipo primitivo** Um tipo básico, como um inteiro ou uma sequência de caracteres.

**trace** Uma dependência que indica um relacionamento de processo ou de histórico entre dois elementos que representam o mesmo conceito, sem regras para derivar um a partir do outro.

**transição** A quarta fase do ciclo de vida de desenvolvimento de um software, em que o software é colocado nas mãos da comunidade de usuários; um relacionamento entre dois estados indicando que um objeto no primeiro estado realizará determinadas ações e passará ao segundo estado quando um evento especificado ocorrer e as condições forem satisfeitas.

**UML** (Unified Modeling Language) Linguagem de Modelagem Unificada, uma linguagem para a visualização, especificação, construção e documentação de artefatos de um sistema complexo de software.

**unidade de distribuição** Um conjunto de objetos ou componentes que são alocados para um nó como um grupo.

**utilização** A dependência em que um elemento (o cliente) requer a presença

de outro elemento (o fornecedor) para seu correto funcionamento ou implementação.

**valor** Um elemento de um domínio de tipo.

**valor atribuído** Uma extensão das propriedades de um estereótipo da UML, que permite a criação de novas informações na especificação de um elemento que suporta esse estereótipo.

**versão** Um conjunto de artefatos, relativamente completo e consistente, disponível para um usuário interno ou externo; a própria entrega desse conjunto.

**vinculação** A criação de um elemento a partir de um template, fornecendo os argumentos para os parâmetros do template.

**vínculo** Uma conexão semântica entre objetos; uma instância de uma associação.

**visão** A projeção em um modelo, vista a partir de uma determinada perspectiva ou ponto de vantagem e omite as entidades que não são relevantes para essa visão.

**visão de caso de uso** A visão da arquitetura de um sistema, abrangendo os casos de uso que descrevem o comportamento do sistema, conforme é visto pelos seus usuários finais, analistas e pessoal de teste.

**visão de implantação** A visão da arquitetura de um sistema, abrangendo os nós que formam a topologia de hardware, em que o sistema é executado; a visão de implantação abrange a distribuição, entrega e instalação das partes que constituem o sistema físico.

**visão de implementação** A visão da arquitetura de um sistema, abrangendo os artefatos utilizados para montar e liberar o sistema físico; a visão de implementação inclui o gerenciamento da configuração das versões do sistema, compostas por artefatos de alguma forma independentes que podem ser reunidos de vários modos para produzir um sistema em execução.

**visão de interação** A visão da arquitetura de um sistema, abrangendo os objetos, as threads e os processos que formam a concorrência do sistema e os mecanismos de sincronização, o conjunto de atividades e o fluxo de

mensagens, controle e dados entre eles. A visão de interação abrange o desempenho, a escalabilidade e o tempo de resposta do sistema.

**visão de projeto** A visão da arquitetura de um sistema, abrangendo as classes, interfaces e colaborações que formam o vocabulário do problema e sua solução; a visão de projeto abrange os requisitos funcionais do sistema.

**visão dinâmica** Um aspecto de um sistema que dá ênfase ao seu comportamento.

**visão estática** Um aspecto de um sistema que dá ênfase à sua estrutura.

**visibilidade** Como um nome pode ser visto e utilizado pelos outros.

# Índice

## A

---

abstração

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [53](#), [54](#), [55](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#)

nível de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [61](#),  
[62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#),  
[95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#)

ação

assíncrona [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#),  
[85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#),  
[101](#), [102](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#),  
[162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [225](#), [226](#), [227](#), [228](#), [297](#), [298](#),  
[299](#), [300](#), [363](#), [364](#), [433](#), [434](#), [435](#), [436](#), [629](#), [630](#), [1577](#), [1578](#),  
[1579](#), [1580](#), [1581](#), [1582](#), [1583](#), [1584](#), [1585](#), [1586](#), [1587](#), [1588](#),  
[1589](#)

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [231](#), [232](#), [233](#), [234](#), [835](#),  
[836](#), [1445](#), [1446](#), [1447](#), [1448](#), [1449](#), [1450](#)

ação de entrada [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

ação de saída [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

acesso [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#)

ActiveX [1](#)

Ada [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

alteração de estado [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

ambiente [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#)

análise [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#)

analista [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#)

anônimo [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

aperfeiçoamento sucessivo [1](#)

API [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

aplicação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#)

applet [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#)

arco [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#)

## B

---

base [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#)

biblioteca [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#),  
[41](#), [42](#), [43](#)

biblioteca de classe [1](#)

bifurcação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

bind [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#)

# C

---

C [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#), [622](#)

C++ [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#)

caixa postal [1](#)

caminho [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#)

ciclo [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#)

ciclo de desenvolvimento [1](#), [2](#), [3](#), [4](#)

ciclo de vida [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#)

classe [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#)

abstrata [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#)

associação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [613](#)

ativa [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [613](#)

atributo [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#)

base [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [613](#)

bem estruturada [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#), [532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#)

características avançadas de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#)

concreta [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [475](#), [476](#), [477](#), [478](#), [479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#)

[531](#), [532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#), [545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#), [558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [613](#)

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#)

definir [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [555](#), [556](#), [557](#), [558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#)

cliente completo [1](#)

cliente fino [1](#)

código [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#)

código-fonte [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#)

colaboração [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#)

COM+ [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

comportamento extra [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

compilação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

complete [1](#), [2](#), [3](#), [4](#)

conector [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#)

conector de delegação [1](#), [2](#)

conexão [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#)

cor [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#)

CORBA [1](#), [2](#), [3](#), [4](#), [5](#)

corrotina [1](#)

costura [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#)

criação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#)

## D

---

derive [1](#), [2](#), [3](#)

desempenho [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

desenvolvimento incremental [1](#)

destruição [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

destruir [1](#), [2](#), [3](#)

diagrama

bem estruturado [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#)

criando [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#)

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#)

definir [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#)

[117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#)

estrutural [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),

[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),  
[233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#),  
[246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#),  
[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),  
[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#),  
[337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#),  
[350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#),  
[363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#),  
[376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#),  
[389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),  
[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#),  
[415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#),  
[428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#),  
[441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#),  
[454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#),  
[467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#),  
[480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#),  
[493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#),  
[506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#),  
[519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#)

diagrama de estrutura composta [1](#)

diagrama estrutural [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

dicotomia classe  
objeto [1](#)

dinâmico [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#),  
[41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#),  
[60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#)

direção [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#)

disciplina [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

disjoint [1](#), [2](#)

dispositivo [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#),  
[21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#)

distribuição [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#),  
[21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#),  
[40](#), [41](#), [42](#)

distribuição de responsabilidades [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

DLL [1](#), [2](#)

documento [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#),  
[21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#)

## E

---

enredo [1](#), [2](#), [3](#)

entregas [1](#)

equilíbrio de projeto [1](#)

escalabilidade [1](#), [2](#), [3](#)

escritor [1](#), [2](#), [3](#)

esquema [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#), [24](#), [25](#)

essência [1](#), [2](#), [3](#), [4](#)

estática [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#)

estrutura [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#)

estrutura composta [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

estrutura de dados [1](#), [2](#)

estrutura interna [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

Ethernet [1](#), [2](#)

evento

adiado [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#),

71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,  
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,  
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,  
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,  
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,  
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155

de tempo 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,  
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,  
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,  
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,  
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,  
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,  
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,  
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,  
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155

definição de 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,  
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,  
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,  
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,  
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,  
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,  
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,  
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,  
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155

definir 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,  
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,  
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,  
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,  
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,  
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,  
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,  
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,

[143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#)

modelagem [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#)

tipo de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#)

executável [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#)

executionTime [1](#)

extend [1](#), [2](#), [3](#), [4](#), [5](#)

extensão [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#)

## F

---

física [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#),

[42](#), [43](#), [44](#), [45](#)

fluxo [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#)

fluxo de objeto [1](#), [2](#), [3](#), [4](#)

fluxograma [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

## G

---

generalização

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#)

nome [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#)

símbolo para [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#)

gerenciamento de configuração [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#)

global [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#)

## H

---

hardware [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#)

harmonia [1](#), [2](#), [3](#), [4](#)

Hello, World! [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#)

herança

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#)

estrutura [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#)

implementação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#)

interface [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#)

múltipla [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#)

simples [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#)

hyperlink [1](#), [2](#)

## I

---

ícone [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#)

---

# I

---

IDL [1](#), [2](#)

in [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#),

[422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#),  
[436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#),  
[450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#),  
[464](#), [465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#),  
[478](#), [479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#),  
[492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#),  
[506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#),  
[520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#), [532](#), [533](#),  
[534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#), [545](#), [546](#), [547](#),  
[548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#), [558](#), [559](#), [560](#), [561](#),  
[562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#),  
[576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#),  
[590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#),  
[604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#),  
[618](#), [619](#), [620](#), [621](#), [622](#), [623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#),  
[632](#), [633](#), [634](#), [635](#), [636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#),  
[646](#), [647](#), [648](#), [649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#),  
[660](#), [661](#), [662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#),  
[674](#), [675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#),  
[688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#), [701](#),  
[702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#), [714](#), [715](#),  
[716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#), [728](#), [729](#),  
[730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#), [741](#), [742](#), [743](#),  
[744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#), [754](#), [755](#), [756](#), [757](#),  
[758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#), [766](#), [767](#), [768](#), [769](#), [770](#), [771](#),  
[772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#), [779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#),  
[786](#), [787](#), [788](#), [789](#), [790](#), [791](#), [792](#), [793](#), [794](#), [795](#), [796](#), [797](#), [798](#), [799](#),  
[800](#), [801](#), [802](#), [803](#), [804](#), [805](#), [806](#), [807](#), [808](#), [809](#), [810](#), [811](#), [812](#), [813](#),  
[814](#), [815](#), [816](#), [817](#), [818](#), [819](#), [820](#), [821](#), [822](#), [823](#), [824](#), [825](#), [826](#), [827](#),  
[828](#), [829](#), [830](#), [831](#), [832](#), [833](#), [834](#), [835](#), [836](#), [837](#), [838](#), [839](#), [840](#), [841](#),  
[842](#), [843](#), [844](#), [845](#), [846](#), [847](#), [848](#), [849](#), [850](#), [851](#), [852](#), [853](#), [854](#), [855](#),  
[856](#), [857](#), [858](#), [859](#), [860](#), [861](#), [862](#), [863](#), [864](#), [865](#), [866](#), [867](#), [868](#), [869](#),  
[870](#), [871](#), [872](#), [873](#), [874](#), [875](#), [876](#), [877](#), [878](#), [879](#), [880](#), [881](#), [882](#), [883](#),  
[884](#), [885](#), [886](#), [887](#), [888](#), [889](#), [890](#), [891](#), [892](#), [893](#), [894](#), [895](#), [896](#), [897](#),  
[898](#), [899](#), [900](#), [901](#), [902](#), [903](#), [904](#), [905](#), [906](#), [907](#), [908](#), [909](#), [910](#), [911](#),  
[912](#), [913](#), [914](#), [915](#), [916](#), [917](#), [918](#), [919](#), [920](#), [921](#), [922](#), [923](#), [924](#), [925](#),  
[926](#), [927](#), [928](#), [929](#), [930](#), [931](#), [932](#), [933](#), [934](#), [935](#), [936](#), [937](#), [938](#), [939](#),

[940](#), [941](#), [942](#), [943](#), [944](#), [945](#), [946](#), [947](#), [948](#), [949](#), [950](#), [951](#), [952](#), [953](#),  
[954](#), [955](#), [956](#), [957](#), [958](#), [959](#), [960](#), [961](#), [962](#), [963](#), [964](#), [965](#), [966](#), [967](#),  
[968](#), [969](#), [970](#), [971](#), [972](#), [973](#), [974](#), [975](#), [976](#), [977](#), [978](#), [979](#), [980](#), [981](#),  
[982](#), [983](#), [984](#), [985](#), [986](#), [987](#), [988](#), [989](#), [990](#), [991](#), [992](#), [993](#), [994](#), [995](#),  
[996](#), [997](#), [998](#), [999](#), [1000](#), [1001](#), [1002](#), [1003](#), [1004](#), [1005](#), [1006](#), [1007](#),  
[1008](#), [1009](#), [1010](#), [1011](#), [1012](#), [1013](#), [1014](#), [1015](#), [1016](#), [1017](#), [1018](#),  
[1019](#), [1020](#), [1021](#), [1022](#), [1023](#), [1024](#), [1025](#), [1026](#), [1027](#), [1028](#), [1029](#),  
[1030](#), [1031](#), [1032](#), [1033](#), [1034](#), [1035](#), [1036](#), [1037](#), [1038](#), [1039](#), [1040](#),  
[1041](#), [1042](#), [1043](#), [1044](#), [1045](#), [1046](#), [1047](#), [1048](#), [1049](#), [1050](#), [1051](#),  
[1052](#), [1053](#), [1054](#), [1055](#), [1056](#), [1057](#), [1058](#), [1059](#), [1060](#), [1061](#), [1062](#),  
[1063](#), [1064](#), [1065](#), [1066](#), [1067](#), [1068](#), [1069](#), [1070](#), [1071](#), [1072](#), [1073](#),  
[1074](#), [1075](#), [1076](#), [1077](#), [1078](#), [1079](#), [1080](#), [1081](#), [1082](#), [1083](#), [1084](#),  
[1085](#), [1086](#), [1087](#), [1088](#), [1089](#), [1090](#), [1091](#), [1092](#), [1093](#), [1094](#), [1095](#),  
[1096](#), [1097](#), [1098](#), [1099](#), [1100](#), [1101](#), [1102](#), [1103](#), [1104](#), [1105](#), [1106](#),  
[1107](#), [1108](#), [1109](#), [1110](#), [1111](#), [1112](#), [1113](#), [1114](#), [1115](#), [1116](#), [1117](#),  
[1118](#), [1119](#), [1120](#), [1121](#), [1122](#), [1123](#), [1124](#), [1125](#), [1126](#), [1127](#), [1128](#),  
[1129](#), [1130](#), [1131](#), [1132](#), [1133](#), [1134](#), [1135](#), [1136](#), [1137](#), [1138](#), [1139](#),  
[1140](#), [1141](#), [1142](#), [1143](#), [1144](#), [1145](#), [1146](#), [1147](#), [1148](#), [1149](#), [1150](#),  
[1151](#), [1152](#), [1153](#), [1154](#), [1155](#), [1156](#), [1157](#), [1158](#), [1159](#), [1160](#), [1161](#),  
[1162](#), [1163](#), [1164](#), [1165](#), [1166](#), [1167](#), [1168](#), [1169](#), [1170](#), [1171](#), [1172](#),  
[1173](#), [1174](#), [1175](#), [1176](#), [1177](#), [1178](#), [1179](#), [1180](#), [1181](#), [1182](#), [1183](#),  
[1184](#), [1185](#), [1186](#), [1187](#), [1188](#), [1189](#), [1190](#), [1191](#), [1192](#), [1193](#), [1194](#),  
[1195](#), [1196](#), [1197](#), [1198](#), [1199](#), [1200](#), [1201](#), [1202](#), [1203](#), [1204](#), [1205](#),  
[1206](#), [1207](#), [1208](#), [1209](#), [1210](#), [1211](#), [1212](#), [1213](#), [1214](#), [1215](#), [1216](#),  
[1217](#), [1218](#), [1219](#), [1220](#), [1221](#), [1222](#), [1223](#), [1224](#), [1225](#), [1226](#), [1227](#),  
[1228](#), [1229](#), [1230](#), [1231](#), [1232](#), [1233](#), [1234](#), [1235](#), [1236](#), [1237](#), [1238](#),  
[1239](#), [1240](#), [1241](#), [1242](#), [1243](#), [1244](#), [1245](#), [1246](#), [1247](#), [1248](#), [1249](#),  
[1250](#), [1251](#), [1252](#), [1253](#), [1254](#), [1255](#), [1256](#), [1257](#), [1258](#), [1259](#), [1260](#),  
[1261](#), [1262](#), [1263](#), [1264](#), [1265](#), [1266](#), [1267](#), [1268](#), [1269](#), [1270](#), [1271](#),  
[1272](#), [1273](#), [1274](#), [1275](#), [1276](#), [1277](#), [1278](#), [1279](#), [1280](#), [1281](#), [1282](#),  
[1283](#), [1284](#), [1285](#), [1286](#), [1287](#), [1288](#), [1289](#), [1290](#), [1291](#), [1292](#), [1293](#),  
[1294](#), [1295](#), [1296](#), [1297](#), [1298](#), [1299](#), [1300](#), [1301](#), [1302](#), [1303](#), [1304](#),  
[1305](#), [1306](#), [1307](#), [1308](#), [1309](#), [1310](#), [1311](#), [1312](#), [1313](#), [1314](#), [1315](#),  
[1316](#), [1317](#), [1318](#), [1319](#), [1320](#), [1321](#), [1322](#), [1323](#), [1324](#), [1325](#), [1326](#),  
[1327](#), [1328](#), [1329](#), [1330](#), [1331](#), [1332](#), [1333](#), [1334](#), [1335](#), [1336](#), [1337](#),  
[1338](#), [1339](#), [1340](#), [1341](#), [1342](#), [1343](#), [1344](#), [1345](#), [1346](#), [1347](#), [1348](#),  
[1349](#), [1350](#), [1351](#), [1352](#), [1353](#), [1354](#), [1355](#), [1356](#), [1357](#), [1358](#), [1359](#),

[1360](#), [1361](#), [1362](#), [1363](#), [1364](#), [1365](#), [1366](#), [1367](#), [1368](#), [1369](#), [1370](#),  
[1371](#), [1372](#), [1373](#), [1374](#), [1375](#), [1376](#), [1377](#), [1378](#), [1379](#), [1380](#), [1381](#),  
[1382](#), [1383](#), [1384](#), [1385](#), [1386](#), [1387](#), [1388](#), [1389](#), [1390](#), [1391](#), [1392](#),  
[1393](#), [1394](#), [1395](#), [1396](#), [1397](#), [1398](#), [1399](#), [1400](#), [1401](#), [1402](#), [1403](#),  
[1404](#), [1405](#), [1406](#), [1407](#), [1408](#), [1409](#), [1410](#), [1411](#), [1412](#), [1413](#), [1414](#),  
[1415](#), [1416](#), [1417](#), [1418](#), [1419](#), [1420](#), [1421](#), [1422](#), [1423](#), [1424](#), [1425](#),  
[1426](#), [1427](#), [1428](#), [1429](#), [1430](#), [1431](#), [1432](#), [1433](#), [1434](#), [1435](#), [1436](#),  
[1437](#), [1438](#), [1439](#), [1440](#), [1441](#), [1442](#), [1443](#), [1444](#), [1445](#), [1446](#), [1447](#),  
[1448](#), [1449](#), [1450](#), [1451](#), [1452](#), [1453](#), [1454](#), [1455](#), [1456](#), [1457](#), [1458](#),  
[1459](#), [1460](#), [1461](#), [1462](#), [1463](#), [1464](#), [1465](#), [1466](#), [1467](#), [1468](#), [1469](#),  
[1470](#), [1471](#), [1472](#), [1473](#), [1474](#), [1475](#), [1476](#), [1477](#), [1478](#), [1479](#), [1480](#),  
[1481](#), [1482](#), [1483](#), [1484](#), [1485](#), [1486](#), [1487](#), [1488](#), [1489](#), [1490](#), [1491](#),  
[1492](#), [1493](#), [1494](#), [1495](#), [1496](#), [1497](#), [1498](#), [1499](#), [1500](#), [1501](#), [1502](#),  
[1503](#), [1504](#), [1505](#), [1506](#), [1507](#), [1508](#), [1509](#), [1510](#), [1511](#), [1512](#), [1513](#),  
[1514](#), [1515](#), [1516](#), [1517](#), [1518](#), [1519](#), [1520](#), [1521](#), [1522](#), [1523](#), [1524](#),  
[1525](#), [1526](#), [1527](#), [1528](#), [1529](#), [1530](#), [1531](#), [1532](#), [1533](#), [1534](#), [1535](#),  
[1536](#), [1537](#), [1538](#), [1539](#), [1540](#), [1541](#), [1542](#), [1543](#), [1544](#), [1545](#), [1546](#),  
[1547](#), [1548](#), [1549](#), [1550](#), [1551](#), [1552](#), [1553](#), [1554](#), [1555](#), [1556](#), [1557](#),  
[1558](#), [1559](#), [1560](#), [1561](#), [1562](#), [1563](#), [1564](#), [1565](#), [1566](#), [1567](#), [1568](#),  
[1569](#), [1570](#), [1571](#), [1572](#), [1573](#), [1574](#), [1575](#), [1576](#), [1577](#), [1578](#), [1579](#),  
[1580](#), [1581](#), [1582](#), [1583](#), [1584](#), [1585](#), [1586](#), [1587](#), [1588](#), [1589](#), [1590](#),  
[1591](#), [1592](#), [1593](#), [1594](#), [1595](#), [1596](#), [1597](#), [1598](#), [1599](#), [1600](#), [1601](#),  
[1602](#), [1603](#), [1604](#), [1605](#), [1606](#), [1607](#), [1608](#), [1609](#), [1610](#), [1611](#), [1612](#),  
[1613](#), [1614](#), [1615](#), [1616](#), [1617](#), [1618](#), [1619](#), [1620](#), [1621](#), [1622](#), [1623](#),  
[1624](#), [1625](#), [1626](#), [1627](#), [1628](#), [1629](#), [1630](#), [1631](#), [1632](#), [1633](#), [1634](#),  
[1635](#), [1636](#), [1637](#), [1638](#), [1639](#), [1640](#), [1641](#), [1642](#), [1643](#), [1644](#), [1645](#),  
[1646](#), [1647](#), [1648](#), [1649](#), [1650](#), [1651](#), [1652](#), [1653](#), [1654](#), [1655](#), [1656](#),  
[1657](#), [1658](#), [1659](#), [1660](#), [1661](#), [1662](#), [1663](#), [1664](#), [1665](#), [1666](#), [1667](#),  
[1668](#), [1669](#), [1670](#), [1671](#), [1672](#), [1673](#), [1674](#), [1675](#), [1676](#), [1677](#), [1678](#),  
[1679](#), [1680](#), [1681](#), [1682](#), [1683](#), [1684](#), [1685](#), [1686](#), [1687](#), [1688](#), [1689](#),  
[1690](#), [1691](#), [1692](#), [1693](#), [1694](#), [1695](#), [1696](#), [1697](#), [1698](#), [1699](#), [1700](#),  
[1701](#), [1702](#), [1703](#), [1704](#), [1705](#), [1706](#), [1707](#), [1708](#), [1709](#), [1710](#), [1711](#),  
[1712](#), [1713](#), [1714](#), [1715](#), [1716](#), [1717](#), [1718](#), [1719](#), [1720](#), [1721](#), [1722](#),  
[1723](#), [1724](#), [1725](#), [1726](#), [1727](#), [1728](#), [1729](#), [1730](#), [1731](#), [1732](#), [1733](#),  
[1734](#), [1735](#), [1736](#), [1737](#), [1738](#), [1739](#), [1740](#), [1741](#), [1742](#), [1743](#), [1744](#),  
[1745](#), [1746](#), [1747](#), [1748](#), [1749](#), [1750](#), [1751](#), [1752](#), [1753](#), [1754](#), [1755](#),  
[1756](#), [1757](#), [1758](#), [1759](#), [1760](#), [1761](#), [1762](#), [1763](#), [1764](#), [1765](#), [1766](#),

[1767](#), [1768](#), [1769](#), [1770](#), [1771](#), [1772](#), [1773](#), [1774](#), [1775](#), [1776](#), [1777](#),  
[1778](#), [1779](#), [1780](#), [1781](#), [1782](#), [1783](#), [1784](#), [1785](#), [1786](#), [1787](#), [1788](#),  
[1789](#), [1790](#), [1791](#), [1792](#), [1793](#), [1794](#), [1795](#), [1796](#), [1797](#), [1798](#), [1799](#),  
[1800](#), [1801](#), [1802](#), [1803](#), [1804](#), [1805](#), [1806](#), [1807](#), [1808](#), [1809](#), [1810](#),  
[1811](#), [1812](#), [1813](#), [1814](#), [1815](#), [1816](#), [1817](#), [1818](#), [1819](#), [1820](#), [1821](#),  
[1822](#), [1823](#), [1824](#), [1825](#), [1826](#), [1827](#), [1828](#), [1829](#), [1830](#), [1831](#), [1832](#),  
[1833](#), [1834](#), [1835](#), [1836](#), [1837](#), [1838](#), [1839](#), [1840](#), [1841](#), [1842](#), [1843](#),  
[1844](#), [1845](#), [1846](#), [1847](#), [1848](#), [1849](#), [1850](#), [1851](#), [1852](#), [1853](#), [1854](#),  
[1855](#), [1856](#), [1857](#), [1858](#), [1859](#), [1860](#), [1861](#), [1862](#), [1863](#), [1864](#), [1865](#),  
[1866](#), [1867](#), [1868](#), [1869](#), [1870](#), [1871](#), [1872](#), [1873](#), [1874](#), [1875](#), [1876](#),  
[1877](#), [1878](#), [1879](#), [1880](#), [1881](#), [1882](#), [1883](#), [1884](#), [1885](#), [1886](#), [1887](#),  
[1888](#), [1889](#), [1890](#), [1891](#), [1892](#), [1893](#), [1894](#), [1895](#), [1896](#), [1897](#), [1898](#),  
[1899](#), [1900](#), [1901](#), [1902](#), [1903](#), [1904](#), [1905](#), [1906](#), [1907](#), [1908](#), [1909](#),  
[1910](#), [1911](#), [1912](#), [1913](#), [1914](#), [1915](#), [1916](#), [1917](#), [1918](#), [1919](#), [1920](#),  
[1921](#), [1922](#), [1923](#), [1924](#), [1925](#), [1926](#), [1927](#), [1928](#), [1929](#), [1930](#), [1931](#),  
[1932](#), [1933](#), [1934](#), [1935](#), [1936](#), [1937](#), [1938](#), [1939](#), [1940](#), [1941](#), [1942](#),  
[1943](#), [1944](#), [1945](#), [1946](#), [1947](#), [1948](#), [1949](#), [1950](#), [1951](#), [1952](#), [1953](#),  
[1954](#), [1955](#), [1956](#), [1957](#), [1958](#), [1959](#), [1960](#), [1961](#), [1962](#), [1963](#), [1964](#),  
[1965](#), [1966](#), [1967](#), [1968](#), [1969](#), [1970](#), [1971](#), [1972](#), [1973](#), [1974](#), [1975](#),  
[1976](#), [1977](#), [1978](#), [1979](#), [1980](#), [1981](#), [1982](#), [1983](#), [1984](#), [1985](#), [1986](#),  
[1987](#), [1988](#), [1989](#), [1990](#), [1991](#), [1992](#), [1993](#), [1994](#), [1995](#), [1996](#), [1997](#),  
[1998](#), [1999](#), [2000](#), [2001](#), [2002](#), [2003](#), [2004](#), [2005](#), [2006](#), [2007](#), [2008](#),  
[2009](#), [2010](#), [2011](#), [2012](#), [2013](#), [2014](#), [2015](#), [2016](#), [2017](#), [2018](#), [2019](#),  
[2020](#), [2021](#), [2022](#), [2023](#), [2024](#), [2025](#), [2026](#), [2027](#), [2028](#), [2029](#), [2030](#),  
[2031](#), [2032](#), [2033](#), [2034](#), [2035](#), [2036](#), [2037](#), [2038](#), [2039](#), [2040](#), [2041](#),  
[2042](#), [2043](#), [2044](#), [2045](#), [2046](#), [2047](#), [2048](#), [2049](#), [2050](#), [2051](#), [2052](#),  
[2053](#), [2054](#), [2055](#), [2056](#), [2057](#)

include [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

incomplete

definição de [1](#), [2](#)

inconsistent

definição de [1](#), [2](#)

incremental

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#)

inout [1](#), [2](#), [3](#)

instalação [1](#), [2](#), [3](#), [4](#), [5](#)

instanceOf [1](#), [2](#), [3](#)

instância [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#),  
[41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#),  
[60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#),  
[79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#),  
[98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#),  
[113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#),  
[127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#),  
[141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#),  
[169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#)

instância concreta [1](#)

instância direta [1](#), [2](#), [3](#), [4](#)

instantiate [1](#), [2](#), [3](#)

integridade

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

interface requerida [1](#), [2](#), [3](#), [4](#), [5](#)

Internet [1](#), [2](#), [3](#)

introspecção [1](#)

invariant [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

## J

---

J2EE [1](#)

Java [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#)

# L

---

LAN [1](#)

linha da vida [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

local [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#),  
[42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#),  
[61](#), [62](#), [63](#)

local da execução [1](#)

lógica [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#)

loop [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#)

# M

---

Maine [1](#)

manifestação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#)

máquina

de Mealy [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#)

de Moore [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#)

marca de tempo

definição de [1](#), [2](#), [3](#), [4](#), [5](#)

símbolo para [1](#), [2](#), [3](#), [4](#), [5](#)

melhores práticas [1](#)

método para desenvolvimento de software [1](#)

migração [1](#), [2](#)

modelo

bem estruturado [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#),  
[18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#),  
[35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#),  
[52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#),  
[69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#),  
[86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#),  
[102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#),  
[115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#),  
[128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#),  
[141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#),  
[154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#),  
[167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#),  
[180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#),  
[193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#),  
[206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#),  
[219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#),  
[232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#),  
[245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#),  
[258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#),  
[271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#),  
[284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#),  
[297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#),  
[310](#), [311](#), [312](#), [313](#), [314](#)

bem formado [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#),  
[18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#),  
[35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#),  
[52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#),

69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,  
86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101,  
102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114,  
115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127,  
128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,  
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153,  
154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166,  
167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179,  
180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192,  
193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205,  
206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218,  
219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,  
232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244,  
245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257,  
258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,  
271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283,  
284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296,  
297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309,  
310, 311, 312, 313, 314

definição de 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,  
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,  
53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,  
70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,  
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102,  
103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,  
116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,  
129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141,  
142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154,  
155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167,  
168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180,  
181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193,  
194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206,  
207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219,  
220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232,  
233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245,  
246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258,

[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),  
[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#)

execução do [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),  
[233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#),  
[246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#),  
[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),  
[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#)

simulação do [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#),  
[18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#),  
[35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#),  
[52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#),  
[69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#),  
[86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#),  
[102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#),  
[115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#),

[128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#),  
[141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#),  
[154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#),  
[167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#),  
[180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#),  
[193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#),  
[206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#),  
[219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#),  
[232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#),  
[245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#),  
[258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#),  
[271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#),  
[284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#),  
[297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#),  
[310](#), [311](#), [312](#), [313](#), [314](#)

## N

---

navegação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

nó de atividade [1](#), [2](#)

nome da extremidade [1](#)

nome qualificado [1](#), [2](#)

nome simples [1](#)

## O

---

o todo [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#)

opt [1](#), [2](#)

ORB [1](#), [2](#), [3](#), [4](#), [5](#)

ordenação temporal [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#)

ordered [1](#), [2](#), [3](#), [4](#)

## Ó

---

órfão [1](#)

## O

---

organização [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#)

orientado a objeto [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#)

## P

---

padrão de projeto [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#)

página [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

papéis [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#)

par [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#)

[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#),  
[156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#),  
[170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#),  
[184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#),  
[198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#),  
[212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#),  
[226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#),  
[240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#),  
[254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#),  
[268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#),  
[282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#),  
[296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#),  
[310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#),  
[338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#),  
[352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#),  
[366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#),  
[380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#),  
[394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#),  
[408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#),  
[422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#),  
[436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#),  
[450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#),  
[464](#), [465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#),  
[478](#), [479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#),  
[492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#),  
[506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#),  
[520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#), [532](#), [533](#),  
[534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#), [545](#), [546](#), [547](#),  
[548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#), [558](#), [559](#), [560](#), [561](#),  
[562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#),  
[576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#),  
[590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#),  
[604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#),  
[618](#), [619](#), [620](#), [621](#), [622](#), [623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#),  
[632](#), [633](#), [634](#), [635](#), [636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#),  
[646](#), [647](#), [648](#), [649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#),

[660](#), [661](#), [662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#),  
[674](#), [675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#),  
[688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#), [701](#),  
[702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#), [714](#), [715](#),  
[716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#), [728](#), [729](#),  
[730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#), [741](#), [742](#), [743](#),  
[744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#), [754](#), [755](#), [756](#), [757](#),  
[758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#), [766](#), [767](#), [768](#), [769](#), [770](#), [771](#),  
[772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#), [779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#),  
[786](#), [787](#), [788](#), [789](#), [790](#), [791](#), [792](#), [793](#), [794](#), [795](#), [796](#), [797](#), [798](#), [799](#),  
[800](#), [801](#), [802](#), [803](#), [804](#), [805](#), [806](#), [807](#), [808](#), [809](#), [810](#), [811](#), [812](#), [813](#),  
[814](#), [815](#), [816](#), [817](#), [818](#), [819](#), [820](#), [821](#), [822](#), [823](#), [824](#), [825](#), [826](#), [827](#),  
[828](#), [829](#), [830](#), [831](#), [832](#), [833](#), [834](#), [835](#), [836](#), [837](#), [838](#), [839](#), [840](#), [841](#),  
[842](#), [843](#), [844](#), [845](#), [846](#), [847](#), [848](#), [849](#), [850](#), [851](#), [852](#), [853](#), [854](#), [855](#),  
[856](#), [857](#), [858](#), [859](#), [860](#), [861](#), [862](#), [863](#), [864](#), [865](#), [866](#), [867](#), [868](#), [869](#),  
[870](#), [871](#), [872](#), [873](#), [874](#), [875](#), [876](#), [877](#), [878](#), [879](#), [880](#), [881](#), [882](#), [883](#),  
[884](#), [885](#), [886](#), [887](#), [888](#), [889](#), [890](#), [891](#), [892](#), [893](#), [894](#), [895](#), [896](#), [897](#),  
[898](#), [899](#), [900](#), [901](#), [902](#), [903](#), [904](#), [905](#), [906](#), [907](#), [908](#), [909](#), [910](#), [911](#),  
[912](#), [913](#), [914](#), [915](#), [916](#), [917](#), [918](#), [919](#), [920](#), [921](#), [922](#), [923](#), [924](#), [925](#),  
[926](#), [927](#), [928](#), [929](#), [930](#), [931](#), [932](#), [933](#), [934](#), [935](#), [936](#), [937](#), [938](#), [939](#),  
[940](#), [941](#), [942](#), [943](#), [944](#), [945](#), [946](#), [947](#), [948](#), [949](#), [950](#), [951](#), [952](#), [953](#),  
[954](#), [955](#), [956](#), [957](#), [958](#), [959](#), [960](#), [961](#), [962](#), [963](#), [964](#), [965](#), [966](#), [967](#),  
[968](#), [969](#), [970](#), [971](#), [972](#), [973](#), [974](#), [975](#), [976](#), [977](#), [978](#), [979](#), [980](#), [981](#),  
[982](#), [983](#), [984](#), [985](#), [986](#), [987](#), [988](#), [989](#), [990](#), [991](#), [992](#), [993](#), [994](#), [995](#),  
[996](#), [997](#), [998](#), [999](#), [1000](#), [1001](#), [1002](#), [1003](#), [1004](#), [1005](#), [1006](#), [1007](#),  
[1008](#), [1009](#), [1010](#), [1011](#), [1012](#), [1013](#), [1014](#), [1015](#), [1016](#), [1017](#), [1018](#),  
[1019](#), [1020](#), [1021](#), [1022](#), [1023](#), [1024](#), [1025](#), [1026](#), [1027](#), [1028](#), [1029](#),  
[1030](#), [1031](#), [1032](#), [1033](#), [1034](#), [1035](#), [1036](#), [1037](#), [1038](#), [1039](#), [1040](#),  
[1041](#), [1042](#), [1043](#), [1044](#), [1045](#), [1046](#), [1047](#), [1048](#), [1049](#), [1050](#), [1051](#),  
[1052](#), [1053](#), [1054](#), [1055](#), [1056](#), [1057](#), [1058](#), [1059](#), [1060](#), [1061](#), [1062](#),  
[1063](#), [1064](#), [1065](#), [1066](#), [1067](#), [1068](#), [1069](#), [1070](#), [1071](#), [1072](#), [1073](#),  
[1074](#), [1075](#), [1076](#), [1077](#), [1078](#), [1079](#), [1080](#), [1081](#), [1082](#), [1083](#), [1084](#),  
[1085](#), [1086](#), [1087](#), [1088](#), [1089](#), [1090](#), [1091](#), [1092](#), [1093](#), [1094](#), [1095](#),  
[1096](#), [1097](#), [1098](#), [1099](#), [1100](#), [1101](#), [1102](#), [1103](#), [1104](#), [1105](#), [1106](#),  
[1107](#), [1108](#), [1109](#), [1110](#), [1111](#), [1112](#), [1113](#), [1114](#), [1115](#), [1116](#), [1117](#),  
[1118](#), [1119](#), [1120](#), [1121](#), [1122](#), [1123](#), [1124](#), [1125](#), [1126](#), [1127](#), [1128](#),  
[1129](#), [1130](#), [1131](#), [1132](#), [1133](#), [1134](#), [1135](#), [1136](#), [1137](#), [1138](#), [1139](#),

[1140](#), [1141](#), [1142](#), [1143](#), [1144](#), [1145](#), [1146](#), [1147](#), [1148](#), [1149](#), [1150](#),  
[1151](#), [1152](#), [1153](#), [1154](#), [1155](#), [1156](#), [1157](#), [1158](#), [1159](#), [1160](#), [1161](#),  
[1162](#), [1163](#), [1164](#), [1165](#), [1166](#), [1167](#), [1168](#), [1169](#), [1170](#), [1171](#), [1172](#),  
[1173](#), [1174](#), [1175](#), [1176](#), [1177](#), [1178](#), [1179](#), [1180](#), [1181](#), [1182](#), [1183](#),  
[1184](#), [1185](#), [1186](#), [1187](#), [1188](#), [1189](#), [1190](#), [1191](#), [1192](#), [1193](#), [1194](#),  
[1195](#), [1196](#), [1197](#), [1198](#), [1199](#), [1200](#), [1201](#), [1202](#), [1203](#), [1204](#), [1205](#),  
[1206](#), [1207](#), [1208](#), [1209](#), [1210](#), [1211](#), [1212](#), [1213](#), [1214](#), [1215](#), [1216](#),  
[1217](#), [1218](#), [1219](#), [1220](#), [1221](#), [1222](#), [1223](#), [1224](#), [1225](#), [1226](#), [1227](#),  
[1228](#), [1229](#), [1230](#), [1231](#), [1232](#), [1233](#), [1234](#), [1235](#), [1236](#), [1237](#), [1238](#),  
[1239](#), [1240](#), [1241](#), [1242](#), [1243](#), [1244](#), [1245](#), [1246](#), [1247](#), [1248](#), [1249](#),  
[1250](#), [1251](#), [1252](#), [1253](#), [1254](#), [1255](#), [1256](#), [1257](#), [1258](#), [1259](#), [1260](#),  
[1261](#), [1262](#), [1263](#), [1264](#), [1265](#), [1266](#), [1267](#), [1268](#), [1269](#), [1270](#), [1271](#),  
[1272](#), [1273](#), [1274](#), [1275](#), [1276](#), [1277](#), [1278](#), [1279](#), [1280](#), [1281](#), [1282](#),  
[1283](#), [1284](#), [1285](#), [1286](#), [1287](#), [1288](#), [1289](#), [1290](#), [1291](#), [1292](#), [1293](#),  
[1294](#), [1295](#), [1296](#), [1297](#), [1298](#), [1299](#), [1300](#), [1301](#), [1302](#), [1303](#), [1304](#),  
[1305](#), [1306](#), [1307](#), [1308](#), [1309](#), [1310](#), [1311](#), [1312](#), [1313](#), [1314](#), [1315](#),  
[1316](#), [1317](#), [1318](#), [1319](#), [1320](#), [1321](#), [1322](#), [1323](#), [1324](#), [1325](#), [1326](#),  
[1327](#), [1328](#), [1329](#), [1330](#), [1331](#), [1332](#), [1333](#), [1334](#), [1335](#), [1336](#), [1337](#),  
[1338](#), [1339](#), [1340](#), [1341](#), [1342](#), [1343](#), [1344](#), [1345](#), [1346](#), [1347](#), [1348](#),  
[1349](#), [1350](#), [1351](#), [1352](#), [1353](#), [1354](#), [1355](#), [1356](#), [1357](#), [1358](#), [1359](#),  
[1360](#), [1361](#), [1362](#), [1363](#), [1364](#), [1365](#), [1366](#), [1367](#), [1368](#), [1369](#), [1370](#),  
[1371](#), [1372](#), [1373](#), [1374](#), [1375](#), [1376](#), [1377](#), [1378](#), [1379](#), [1380](#), [1381](#),  
[1382](#), [1383](#), [1384](#), [1385](#), [1386](#), [1387](#), [1388](#), [1389](#), [1390](#), [1391](#), [1392](#),  
[1393](#), [1394](#), [1395](#), [1396](#), [1397](#), [1398](#), [1399](#), [1400](#), [1401](#), [1402](#), [1403](#),  
[1404](#), [1405](#), [1406](#), [1407](#), [1408](#), [1409](#), [1410](#), [1411](#), [1412](#), [1413](#), [1414](#),  
[1415](#), [1416](#), [1417](#), [1418](#), [1419](#), [1420](#), [1421](#), [1422](#), [1423](#), [1424](#), [1425](#),  
[1426](#), [1427](#), [1428](#), [1429](#), [1430](#), [1431](#), [1432](#), [1433](#), [1434](#), [1435](#), [1436](#),  
[1437](#), [1438](#), [1439](#), [1440](#), [1441](#), [1442](#), [1443](#), [1444](#), [1445](#), [1446](#), [1447](#),  
[1448](#), [1449](#), [1450](#), [1451](#), [1452](#), [1453](#), [1454](#), [1455](#), [1456](#), [1457](#), [1458](#),  
[1459](#), [1460](#), [1461](#), [1462](#), [1463](#), [1464](#), [1465](#), [1466](#), [1467](#), [1468](#), [1469](#),  
[1470](#), [1471](#), [1472](#), [1473](#), [1474](#), [1475](#), [1476](#), [1477](#), [1478](#), [1479](#), [1480](#),  
[1481](#), [1482](#), [1483](#), [1484](#), [1485](#), [1486](#), [1487](#), [1488](#), [1489](#), [1490](#), [1491](#),  
[1492](#), [1493](#), [1494](#), [1495](#), [1496](#), [1497](#), [1498](#)

parte [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#),  
[42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#),  
[61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#),

[80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#)

participantes [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#)

permit [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#)

pessoal de teste [1](#), [2](#), [3](#)

plano de projeto [1](#), [2](#)

polimórfica [1](#), [2](#), [3](#)

polimorfismo [1](#), [2](#), [3](#), [4](#)

porta [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#)

[128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#),  
[156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#),  
[170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#),  
[184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#),  
[198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#),  
[212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#),  
[226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#),  
[240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#),  
[254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#),  
[268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#),  
[282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#),  
[296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#),  
[310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#),  
[338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#),  
[352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#),  
[366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#),  
[380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#),  
[394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#),  
[408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#),  
[422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#),  
[436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#),  
[450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#),  
[464](#), [465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#),  
[478](#), [479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#),  
[492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#),  
[506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#),  
[520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#), [532](#), [533](#),  
[534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#), [545](#), [546](#), [547](#),  
[548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#), [558](#)

powertype [1](#), [2](#), [3](#), [4](#)

protegido [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)

prototípico [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

protótipo [1](#), [2](#), [3](#), [4](#), [5](#)

proxy [1](#)

## **Q**

---

quadro-negro [1](#)

qualificação [1, 2, 3](#)

qualificado [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14](#)

qualificador [1, 2](#)

## **R**

---

Rational Unified Process [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20](#)

realidade [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12](#)

reativo [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18](#)

recursão [1, 2, 3](#)

rede [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73](#)

refine [1, 2](#)

região de expansão [1, 2, 3](#)

relacionamentos “é um tipo de” [1](#)

relacionamentos de generalização [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11](#)

rendez-vous [1](#)

replicação [1](#)

retorno [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20](#)

risco [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#)

## S

---

São Francisco [1](#)

self [1](#)

semânticas [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

separação de conceitos [1](#), [2](#), [3](#)

sequencial [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#)

servidor [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#)

sincronização [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#)

síncrono [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

sintaxe [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#)

sistema

adaptável [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),

233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245,  
246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258,  
259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271,  
272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284,  
285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297,  
298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310,  
311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323,  
324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336,  
337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,  
350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362,  
363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375,  
376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388,  
389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401,  
402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414,  
415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427,  
428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440,  
441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453,  
454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466,  
467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479,  
480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492,  
493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505,  
506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518,  
519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531,  
532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544,  
545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557,  
558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570,  
571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583,  
584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596,  
597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609,  
610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622,  
623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635,  
636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648,  
649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661,  
662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674,  
675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687,  
688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700,  
701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713,

[714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#), [728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#), [741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#), [754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#), [766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#), [779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

bem estruturado [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#)

[427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#),  
[440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#),  
[453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#),  
[466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#),  
[479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#),  
[492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#),  
[505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#),  
[518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#),  
[531](#), [532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#),  
[544](#), [545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#),  
[557](#), [558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#),  
[570](#), [571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#),  
[583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#),  
[596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#),  
[609](#), [610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#),  
[622](#), [623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#),  
[635](#), [636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#),  
[648](#), [649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#),  
[661](#), [662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#),  
[674](#), [675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#),  
[687](#), [688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#),  
[700](#), [701](#), [702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#),  
[713](#), [714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#),  
[726](#), [727](#), [728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#),  
[739](#), [740](#), [741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#),  
[752](#), [753](#), [754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#),  
[765](#), [766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#),  
[778](#), [779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

cliente

servidor [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#),  
[17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#),  
[32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#),  
[47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#),  
[62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#),  
[77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#),  
[92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#),  
[105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),

[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#),  
[127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#),  
[138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#),  
[149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#),  
[160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#),  
[171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#),  
[182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#),  
[193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#),  
[204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#),  
[215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#),  
[226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#),  
[237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#),  
[248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#),  
[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#),  
[270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#),  
[281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#),  
[292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#),  
[303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#),  
[314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#),  
[325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#),  
[336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#),  
[347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#),  
[358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#),  
[369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#),  
[380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#),  
[391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),  
[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#),  
[413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#),  
[424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#),  
[435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#),  
[446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#),  
[457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#),  
[468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#),  
[479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#),  
[490](#), [491](#), [492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#),  
[501](#), [502](#), [503](#), [504](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#),  
[512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#),

[523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#), [532](#), [533](#),  
[534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#),  
[545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#),  
[556](#), [557](#), [558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#),  
[567](#), [568](#), [569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#),  
[578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#),  
[589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#),  
[600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#),  
[611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#),  
[622](#), [623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#),  
[633](#), [634](#), [635](#), [636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#),  
[644](#), [645](#), [646](#), [647](#), [648](#), [649](#), [650](#), [651](#), [652](#), [653](#), [654](#),  
[655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [661](#), [662](#), [663](#), [664](#), [665](#),  
[666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#), [674](#), [675](#), [676](#),  
[677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#),  
[688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#),  
[699](#), [700](#), [701](#), [702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#),  
[710](#), [711](#), [712](#), [713](#), [714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#),  
[721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#), [728](#), [729](#), [730](#), [731](#),  
[732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#), [741](#), [742](#),  
[743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#),  
[754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#),  
[765](#), [766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#),  
[776](#), [777](#), [778](#), [779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

complexo de software [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#),  
[16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#),  
[33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#),  
[50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#),  
[67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#),  
[84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#),  
[101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#),  
[114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#),  
[127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#),  
[140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#),  
[153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#),  
[166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#),  
[179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#)

[192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#),  
[205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#),  
[218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#),  
[231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#),  
[244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#),  
[257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#),  
[270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#),  
[283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#),  
[296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#),  
[309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#),  
[322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#),  
[335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#),  
[348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#),  
[361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#),  
[374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#),  
[387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#),  
[400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#),  
[413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#),  
[426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#),  
[439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#),  
[452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#),  
[465](#), [466](#), [467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#),  
[478](#), [479](#), [480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#),  
[491](#), [492](#), [493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#),  
[504](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#),  
[517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#),  
[530](#), [531](#), [532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#),  
[543](#), [544](#), [545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#),  
[556](#), [557](#), [558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#),  
[569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#),  
[582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#),  
[595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#),  
[608](#), [609](#), [610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#),  
[621](#), [622](#), [623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#),  
[634](#), [635](#), [636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#),  
[647](#), [648](#), [649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#),  
[660](#), [661](#), [662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#),

[673](#), [674](#), [675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#), [688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#), [701](#), [702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#), [714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#), [728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#), [741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#), [754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#), [766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#), [779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

contexto do [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#)

[389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),  
[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#),  
[415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#),  
[428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#),  
[441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#),  
[454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#),  
[467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#),  
[480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#),  
[493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#),  
[506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#),  
[519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#),  
[532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#),  
[545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#),  
[558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#),  
[571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#),  
[584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#),  
[597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#),  
[610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#), [622](#),  
[623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#), [635](#),  
[636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#), [648](#),  
[649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [661](#),  
[662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#), [674](#),  
[675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#),  
[688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#),  
[701](#), [702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#),  
[714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#),  
[727](#), [728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#),  
[740](#), [741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#),  
[753](#), [754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#),  
[766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#),  
[779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

costurar [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#),  
[20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#),  
[37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#),  
[54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#),  
[71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#),  
[88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#),

104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,  
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,  
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,  
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,  
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,  
169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,  
182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,  
195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,  
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,  
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,  
234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,  
247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,  
260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,  
273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,  
286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,  
299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311,  
312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,  
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337,  
338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,  
351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,  
364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,  
377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,  
390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,  
403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,  
416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,  
429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,  
442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,  
455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,  
468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,  
481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,  
494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506,  
507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,  
520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532,  
533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545,  
546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,  
559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571,  
572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584,

[585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#),  
[598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#),  
[611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#), [622](#), [623](#),  
[624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#), [635](#), [636](#),  
[637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#), [648](#), [649](#),  
[650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [661](#), [662](#),  
[663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#), [674](#), [675](#),  
[676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#), [688](#),  
[689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#), [701](#),  
[702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#), [714](#),  
[715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#),  
[728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#),  
[741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#),  
[754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#), [766](#),  
[767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#), [779](#),  
[780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

de sistemas [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),  
[233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#),  
[246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#),  
[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),

[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#),  
[337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#),  
[350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#),  
[363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#),  
[376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#),  
[389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),  
[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#),  
[415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#),  
[428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#),  
[441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#),  
[454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#),  
[467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#),  
[480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#),  
[493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#),  
[506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#),  
[519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#),  
[532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#),  
[545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#),  
[558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#),  
[571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#),  
[584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#),  
[597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#),  
[610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#), [622](#),  
[623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#), [635](#),  
[636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#), [648](#),  
[649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [661](#),  
[662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#), [674](#),  
[675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#),  
[688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#),  
[701](#), [702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#),  
[714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#),  
[727](#), [728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#),  
[740](#), [741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#),  
[753](#), [754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#),  
[766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#),

[779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

definição de [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),  
[233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#),  
[246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#),  
[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),  
[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#),  
[337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#),  
[350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#),  
[363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#),  
[376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#),  
[389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),  
[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#),  
[415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#),  
[428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#),  
[441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#),  
[454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#),  
[467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#),  
[480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#),

[493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#), [506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#), [532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#), [545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#), [558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#), [622](#), [623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#), [635](#), [636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#), [648](#), [649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [661](#), [662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#), [674](#), [675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#), [688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#), [701](#), [702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#), [714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#), [728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#), [741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#), [754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#), [766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#), [779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

definir [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#)

[208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#),  
[221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#),  
[234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#),  
[247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#),  
[260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#),  
[273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#),  
[286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#),  
[299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#),  
[312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#),  
[325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#),  
[338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#),  
[351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#),  
[364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#),  
[377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#),  
[390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#),  
[403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#),  
[416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#),  
[429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#),  
[442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#),  
[455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#),  
[468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#), [480](#),  
[481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#), [493](#),  
[494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#), [506](#),  
[507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#), [519](#),  
[520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#), [532](#),  
[533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#), [545](#),  
[546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#), [558](#),  
[559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#), [571](#),  
[572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#), [584](#),  
[585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#), [597](#),  
[598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#), [610](#),  
[611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#), [622](#), [623](#),  
[624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#), [635](#), [636](#),  
[637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#), [648](#), [649](#),  
[650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [661](#), [662](#),  
[663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#), [674](#), [675](#),  
[676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#), [688](#),

[689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#), [701](#),  
[702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#), [714](#),  
[715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#), [727](#),  
[728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#), [740](#),  
[741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#), [753](#),  
[754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#), [766](#),  
[767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#), [779](#),  
[780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

distribuído [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),  
[233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#),  
[246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#),  
[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),  
[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#),  
[337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#),  
[350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#),  
[363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#),  
[376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#),  
[389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),

[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#),  
[415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#),  
[428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#),  
[441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#),  
[454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#),  
[467](#), [468](#), [469](#), [470](#), [471](#), [472](#), [473](#), [474](#), [475](#), [476](#), [477](#), [478](#), [479](#),  
[480](#), [481](#), [482](#), [483](#), [484](#), [485](#), [486](#), [487](#), [488](#), [489](#), [490](#), [491](#), [492](#),  
[493](#), [494](#), [495](#), [496](#), [497](#), [498](#), [499](#), [500](#), [501](#), [502](#), [503](#), [504](#), [505](#),  
[506](#), [507](#), [508](#), [509](#), [510](#), [511](#), [512](#), [513](#), [514](#), [515](#), [516](#), [517](#), [518](#),  
[519](#), [520](#), [521](#), [522](#), [523](#), [524](#), [525](#), [526](#), [527](#), [528](#), [529](#), [530](#), [531](#),  
[532](#), [533](#), [534](#), [535](#), [536](#), [537](#), [538](#), [539](#), [540](#), [541](#), [542](#), [543](#), [544](#),  
[545](#), [546](#), [547](#), [548](#), [549](#), [550](#), [551](#), [552](#), [553](#), [554](#), [555](#), [556](#), [557](#),  
[558](#), [559](#), [560](#), [561](#), [562](#), [563](#), [564](#), [565](#), [566](#), [567](#), [568](#), [569](#), [570](#),  
[571](#), [572](#), [573](#), [574](#), [575](#), [576](#), [577](#), [578](#), [579](#), [580](#), [581](#), [582](#), [583](#),  
[584](#), [585](#), [586](#), [587](#), [588](#), [589](#), [590](#), [591](#), [592](#), [593](#), [594](#), [595](#), [596](#),  
[597](#), [598](#), [599](#), [600](#), [601](#), [602](#), [603](#), [604](#), [605](#), [606](#), [607](#), [608](#), [609](#),  
[610](#), [611](#), [612](#), [613](#), [614](#), [615](#), [616](#), [617](#), [618](#), [619](#), [620](#), [621](#), [622](#),  
[623](#), [624](#), [625](#), [626](#), [627](#), [628](#), [629](#), [630](#), [631](#), [632](#), [633](#), [634](#), [635](#),  
[636](#), [637](#), [638](#), [639](#), [640](#), [641](#), [642](#), [643](#), [644](#), [645](#), [646](#), [647](#), [648](#),  
[649](#), [650](#), [651](#), [652](#), [653](#), [654](#), [655](#), [656](#), [657](#), [658](#), [659](#), [660](#), [661](#),  
[662](#), [663](#), [664](#), [665](#), [666](#), [667](#), [668](#), [669](#), [670](#), [671](#), [672](#), [673](#), [674](#),  
[675](#), [676](#), [677](#), [678](#), [679](#), [680](#), [681](#), [682](#), [683](#), [684](#), [685](#), [686](#), [687](#),  
[688](#), [689](#), [690](#), [691](#), [692](#), [693](#), [694](#), [695](#), [696](#), [697](#), [698](#), [699](#), [700](#),  
[701](#), [702](#), [703](#), [704](#), [705](#), [706](#), [707](#), [708](#), [709](#), [710](#), [711](#), [712](#), [713](#),  
[714](#), [715](#), [716](#), [717](#), [718](#), [719](#), [720](#), [721](#), [722](#), [723](#), [724](#), [725](#), [726](#),  
[727](#), [728](#), [729](#), [730](#), [731](#), [732](#), [733](#), [734](#), [735](#), [736](#), [737](#), [738](#), [739](#),  
[740](#), [741](#), [742](#), [743](#), [744](#), [745](#), [746](#), [747](#), [748](#), [749](#), [750](#), [751](#), [752](#),  
[753](#), [754](#), [755](#), [756](#), [757](#), [758](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [765](#),  
[766](#), [767](#), [768](#), [769](#), [770](#), [771](#), [772](#), [773](#), [774](#), [775](#), [776](#), [777](#), [778](#),  
[779](#), [780](#), [781](#), [782](#), [783](#), [784](#), [785](#)

sistema cliente

servidor [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

sistema distribuído [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

Smalltalk [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)

sociedade [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),

[22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#)

software [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#),  
[41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#),  
[60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#),  
[79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#),  
[98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#),  
[113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#),  
[127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#),  
[141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#)

subestado sequencial [1](#), [2](#)

sub-região [1](#), [2](#)

## T

---

tabela [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#)

Taj Mahal [1](#), [2](#)

tempo de vida [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#),  
[21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#)

tempo de vida do objeto [1](#), [2](#)

tempo esgotado [1](#)

teste [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#)

testes [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#)

## U

---

UML

blocos de construção da [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#)

definição da [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),

[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),  
[233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#),  
[246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#),  
[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),  
[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#),  
[337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#),  
[350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#),  
[363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#),  
[376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#),  
[389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),  
[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#),  
[415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#),  
[428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#),  
[441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#),  
[454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#),  
[467](#), [468](#), [469](#)

diagramas na [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#),  
[18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#),  
[35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#),  
[52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#),

69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,  
86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101,  
102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114,  
115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127,  
128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,  
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153,  
154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166,  
167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179,  
180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192,  
193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205,  
206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218,  
219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,  
232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244,  
245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257,  
258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,  
271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283,  
284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296,  
297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309,  
310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322,  
323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335,  
336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348,  
349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361,  
362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374,  
375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387,  
388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400,  
401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413,  
414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426,  
427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439,  
440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452,  
453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465,  
466, 467, 468, 469

e hardware 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,  
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,  
53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,  
70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,  
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102,

103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,  
116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,  
129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141,  
142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154,  
155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167,  
168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180,  
181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193,  
194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206,  
207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219,  
220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232,  
233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245,  
246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258,  
259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271,  
272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284,  
285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297,  
298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310,  
311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323,  
324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336,  
337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,  
350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362,  
363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375,  
376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388,  
389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401,  
402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414,  
415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427,  
428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440,  
441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453,  
454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466,  
467, 468, 469

estender 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,  
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,  
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,  
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,  
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,  
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,  
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,

[130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#),  
[143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#),  
[156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#),  
[169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#),  
[182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#),  
[195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#),  
[208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#),  
[221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#),  
[234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#),  
[247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#),  
[260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#),  
[273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#),  
[286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#),  
[299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#),  
[312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#),  
[325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#),  
[338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#),  
[351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#),  
[364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#),  
[377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#),  
[390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#),  
[403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#),  
[416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#),  
[429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#),  
[442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#),  
[455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#),  
[468](#), [469](#)

fazendo a transição para a [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#),  
[15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#),  
[32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#),  
[49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#),  
[66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#),  
[83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#),  
[100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#),  
[113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#),  
[126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#),  
[139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#),

152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,  
165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,  
178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,  
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203,  
204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216,  
217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229,  
230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242,  
243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255,  
256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268,  
269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281,  
282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294,  
295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307,  
308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320,  
321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333,  
334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346,  
347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359,  
360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372,  
373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385,  
386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398,  
399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411,  
412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424,  
425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437,  
438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450,  
451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463,  
464, 465, 466, 467, 468, 469

história da, x-xiii 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,  
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,  
51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,  
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,  
85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,  
101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,  
114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126,  
127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,  
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,  
153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,  
166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178

[179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#),  
[192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#),  
[205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#),  
[218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#),  
[231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#),  
[244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#),  
[257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#),  
[270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#),  
[283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#),  
[296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#),  
[309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#),  
[322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#),  
[335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#),  
[348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#),  
[361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#),  
[374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#),  
[387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#),  
[400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#),  
[413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#),  
[426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#),  
[439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#),  
[452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#),  
[465](#), [466](#), [467](#), [468](#), [469](#)

itens na [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#),  
[20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#),  
[37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#),  
[54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#),  
[71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#),  
[88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#),  
[104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#),  
[117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#),  
[130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#),  
[143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#),  
[156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#),  
[169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#),  
[182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#),  
[195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#)

[208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#),  
[221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#),  
[234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#),  
[247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#),  
[260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#),  
[273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#),  
[286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#),  
[299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#),  
[312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#),  
[325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#),  
[338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#),  
[351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#),  
[364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#),  
[377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#),  
[390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#),  
[403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#),  
[416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#),  
[429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#),  
[442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#),  
[455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#),  
[468](#), [469](#)

mecanismos de extensibilidade da [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#),  
[13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#),  
[30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#),  
[47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#),  
[64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#),  
[81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#),  
[98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#),  
[111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#),  
[124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#),  
[137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#),  
[150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#),  
[163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#),  
[176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#),  
[189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#),  
[202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#),  
[215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#)

[228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#),  
[241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#),  
[254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#),  
[267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#),  
[280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#),  
[293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#),  
[306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#),  
[319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#),  
[332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#),  
[345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#),  
[358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#),  
[371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#),  
[384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#),  
[397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#),  
[410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#),  
[423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#),  
[436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#),  
[449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#),  
[462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#)

metamodelo [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#),  
[19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#),  
[36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#),  
[53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#),  
[70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#),  
[87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#),  
[103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#),  
[116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#),  
[129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#),  
[142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#),  
[155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#),  
[168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#),  
[181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#),  
[194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#),  
[207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#),  
[220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#),  
[233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#),  
[246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#)

[259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#),  
[272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#),  
[285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#),  
[298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#),  
[311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#),  
[324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#),  
[337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#),  
[350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#),  
[363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#),  
[376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#),  
[389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#),  
[402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#),  
[415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#),  
[428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#),  
[441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#),  
[454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#),  
[467](#), [468](#), [469](#)

modelo conceitual da [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#),  
[16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#),  
[33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#),  
[50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#),  
[67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#),  
[84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#),  
[101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#),  
[114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#),  
[127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#),  
[140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#),  
[153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#),  
[166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#),  
[179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#),  
[192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#),  
[205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#),  
[218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#),  
[231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#),  
[244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#),  
[257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#),  
[270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#)

[283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#),  
[296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#),  
[309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#),  
[322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#),  
[335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#),  
[348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#),  
[361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#),  
[374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#),  
[387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#),  
[400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#),  
[413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#),  
[426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#),  
[439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#),  
[452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#),  
[465](#), [466](#), [467](#), [468](#), [469](#)

notação [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#),  
[20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#),  
[37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#),  
[54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#),  
[71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#),  
[88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#),  
[104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#),  
[117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#),  
[130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#),  
[143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#),  
[156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#),  
[169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#),  
[182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#),  
[195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#),  
[208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#),  
[221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#),  
[234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#),  
[247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#),  
[260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#),  
[273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#),  
[286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#),  
[299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#),

[312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#)

objetivo da [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#)

[337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#)

regras da [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#)

[364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#), [387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#), [400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#), [413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#), [426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#), [439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#), [452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#), [465](#), [466](#), [467](#), [468](#), [469](#)

relacionamentos na [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#), [23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [61](#), [62](#), [63](#), [64](#), [65](#), [66](#), [67](#), [68](#), [69](#), [70](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [81](#), [82](#), [83](#), [84](#), [85](#), [86](#), [87](#), [88](#), [89](#), [90](#), [91](#), [92](#), [93](#), [94](#), [95](#), [96](#), [97](#), [98](#), [99](#), [100](#), [101](#), [102](#), [103](#), [104](#), [105](#), [106](#), [107](#), [108](#), [109](#), [110](#), [111](#), [112](#), [113](#), [114](#), [115](#), [116](#), [117](#), [118](#), [119](#), [120](#), [121](#), [122](#), [123](#), [124](#), [125](#), [126](#), [127](#), [128](#), [129](#), [130](#), [131](#), [132](#), [133](#), [134](#), [135](#), [136](#), [137](#), [138](#), [139](#), [140](#), [141](#), [142](#), [143](#), [144](#), [145](#), [146](#), [147](#), [148](#), [149](#), [150](#), [151](#), [152](#), [153](#), [154](#), [155](#), [156](#), [157](#), [158](#), [159](#), [160](#), [161](#), [162](#), [163](#), [164](#), [165](#), [166](#), [167](#), [168](#), [169](#), [170](#), [171](#), [172](#), [173](#), [174](#), [175](#), [176](#), [177](#), [178](#), [179](#), [180](#), [181](#), [182](#), [183](#), [184](#), [185](#), [186](#), [187](#), [188](#), [189](#), [190](#), [191](#), [192](#), [193](#), [194](#), [195](#), [196](#), [197](#), [198](#), [199](#), [200](#), [201](#), [202](#), [203](#), [204](#), [205](#), [206](#), [207](#), [208](#), [209](#), [210](#), [211](#), [212](#), [213](#), [214](#), [215](#), [216](#), [217](#), [218](#), [219](#), [220](#), [221](#), [222](#), [223](#), [224](#), [225](#), [226](#), [227](#), [228](#), [229](#), [230](#), [231](#), [232](#), [233](#), [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [240](#), [241](#), [242](#), [243](#), [244](#), [245](#), [246](#), [247](#), [248](#), [249](#), [250](#), [251](#), [252](#), [253](#), [254](#), [255](#), [256](#), [257](#), [258](#), [259](#), [260](#), [261](#), [262](#), [263](#), [264](#), [265](#), [266](#), [267](#), [268](#), [269](#), [270](#), [271](#), [272](#), [273](#), [274](#), [275](#), [276](#), [277](#), [278](#), [279](#), [280](#), [281](#), [282](#), [283](#), [284](#), [285](#), [286](#), [287](#), [288](#), [289](#), [290](#), [291](#), [292](#), [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [304](#), [305](#), [306](#), [307](#), [308](#), [309](#), [310](#), [311](#), [312](#), [313](#), [314](#), [315](#), [316](#), [317](#), [318](#), [319](#), [320](#), [321](#), [322](#), [323](#), [324](#), [325](#), [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#), [333](#), [334](#), [335](#), [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#), [343](#), [344](#), [345](#), [346](#), [347](#), [348](#), [349](#), [350](#), [351](#), [352](#), [353](#), [354](#), [355](#), [356](#), [357](#), [358](#), [359](#), [360](#), [361](#), [362](#), [363](#), [364](#), [365](#), [366](#), [367](#), [368](#), [369](#), [370](#), [371](#), [372](#), [373](#), [374](#), [375](#), [376](#), [377](#), [378](#), [379](#), [380](#), [381](#), [382](#), [383](#), [384](#), [385](#), [386](#)

[387](#), [388](#), [389](#), [390](#), [391](#), [392](#), [393](#), [394](#), [395](#), [396](#), [397](#), [398](#), [399](#),  
[400](#), [401](#), [402](#), [403](#), [404](#), [405](#), [406](#), [407](#), [408](#), [409](#), [410](#), [411](#), [412](#),  
[413](#), [414](#), [415](#), [416](#), [417](#), [418](#), [419](#), [420](#), [421](#), [422](#), [423](#), [424](#), [425](#),  
[426](#), [427](#), [428](#), [429](#), [430](#), [431](#), [432](#), [433](#), [434](#), [435](#), [436](#), [437](#), [438](#),  
[439](#), [440](#), [441](#), [442](#), [443](#), [444](#), [445](#), [446](#), [447](#), [448](#), [449](#), [450](#), [451](#),  
[452](#), [453](#), [454](#), [455](#), [456](#), [457](#), [458](#), [459](#), [460](#), [461](#), [462](#), [463](#), [464](#),  
[465](#), [466](#), [467](#), [468](#), [469](#)

união [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#)

unidade de distribuição [1](#)

unitária [1](#), [2](#)

Unix [1](#)

URL [1](#)

usuário final [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)

utilitário [1](#)

## V

---

variante [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#),  
[22](#), [23](#)

versão [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [21](#), [22](#),  
[23](#), [24](#), [25](#), [26](#), [27](#), [28](#), [29](#), [30](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [41](#),  
[42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [48](#), [49](#), [50](#), [51](#), [52](#)

vértice [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#)

VHDL [1](#), [2](#), [3](#)

visão de interação [1](#), [2](#), [3](#), [4](#)

visão física [1](#)

visão lógica [1](#)

Visual Basic [1](#), [2](#)

**W**

---

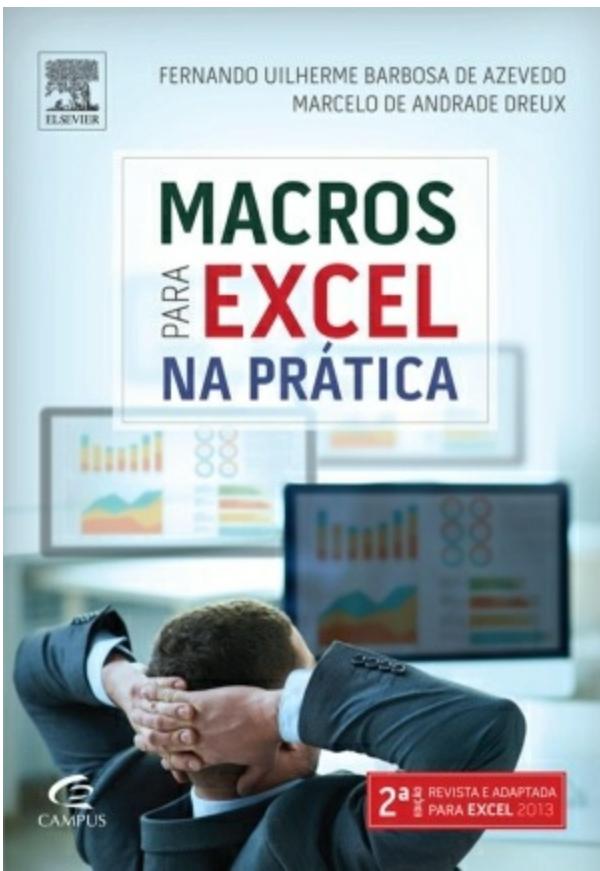
WAN [1](#)

Web [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#)

**X**

---

XML [1](#)



# Macros para excel na prática

Azevedo, Fernando

9788535272659

240 páginas

[Compre agora e leia](#)

O objetivo desta obra é ensinar ao leitor como desenvolver Macros para Excel. A cada capítulo devem ser apresentados os conceitos da linguagem VBA enquanto é desenvolvido um Sistema de Controle de Vendas de uma concessionária de automóveis fictícia. A apresentação dos comandos e funções para a criação do sistema proposto deve permitir ao leitor explorar as demais possibilidades de se trabalhar com Macros.

[Compre agora e leia](#)

*Andrews*

# ATLAS CLÍNICO DE DOENÇAS DA PELE

William D. James • Dirk M. Elston • Patrick J. McMahon



ELSEVIER

# Andrews atlas clínico de doenças da pele

James, William D.

9788535290295

624 páginas

[Compre agora e leia](#)

Este livro apresenta uma coleção de mais de 3.000 imagens de alta qualidade, incluindo novas doenças e condições raras, além de cabelo, unha e descobertas na membrana mucosa para atender as necessidades dos dermatologistas no momento do diagnóstico. Apresenta ainda um texto introdutório conciso para cada capítulo com uma visão geral e compreensiva para auxiliar no diagnóstico. Veja mais de 3.000 fotografias coloridas de alta qualidade que retratam o aspecto completo de doenças da pele em todos os tipos de pele em adultos, crianças e recém-nascidos; Destaca uma grande variedade de subtipos de condições comuns, tais como lichen planus, granuloma annulare e psoriase; Novas descobertas de doenças em cabelo, unha e membrana mucosa são apresentados no livro; Inclui representações de importantes condições sistêmicas como sarcoidose, lúpus eritematoso e doenças infecciosas; Apresenta imagens nunca antes publicadas contribuídas por 54 líderes globais em dermatologia; Texto introdutório conciso em cada capítulo fornece aos leitores uma visão geral rápida e compreensiva da doença abordada.

[Compre agora e leia](#)

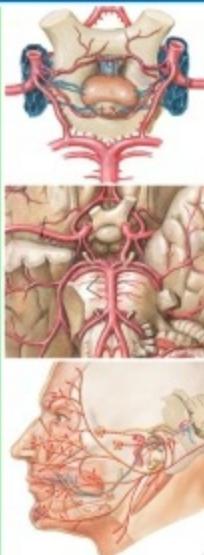


David L. Felten · Anil N. Shetty

**Netter**  
**ATLAS**  
**DE NEUROCIÊNCIA**



TRADUÇÃO DA 2<sup>ª</sup> EDIÇÃO



# Netter atlas de neurociência

Felten, David L.

9788535246261

464 páginas

[Compre agora e leia](#)

A nova edição do Netter Atlas de Neurociência oferece rica orientação visual, combinada a um texto conciso para ajudar você a dominar os princípios complexos, porém importantes, da neurociência. Uma cobertura de fácil entendimento dividida em três partes — uma visão geral do sistema nervoso, neurociência regional e neurociência sistêmica — permite o estudo das estruturas e sistemas neurais em múltiplos contextos. No conteúdo, você encontrará:

- Informações atualizadas e novas figuras que refletem os atuais conhecimentos dos componentes neurais e de tecido conjuntivo, regiões e sistemas do cérebro, medula espinal e periferia para garantir que você conheça os avanços mais recentes.
- Novas imagens coloridas em 3D de vias comissurais, de associação e de projeção do cérebro.
- Quase 400 ilustrações com a excelência e o estilo Netter que destacam os conceitos-chave da neurociência e as correlações clínicas, proporcionando uma visão geral rápida e fácil de memorizar da anatomia, da função e da relevância clínica.
- Imagens de alta qualidade — Imagens de Ressonância Magnética (RM) de alta resolução nos planos coronal e axial (horizontal), além de cortes transversais do tronco encefálico — bem como angiografia e venografia por RM e arteriografia clássica — o que oferece uma melhor perspectiva da complexidade do sistema nervoso.
- Anatomia esquemática transversa do

tronco encefálico e anatomia cerebral axial e coronal — com RM — para melhor ilustrar a correlação entre neuroanatomia e neurologia. • Uma organização regional do sistema nervoso periférico, da medula espinal, do tronco encefálico, cerebelo e prosencéfalo — e uma organização sistêmica dos sistemas sensitivos, sistemas motores (incluindo o cerebelo e os núcleos da base) e dos sistemas límbicos/hipotalâmicos/autonômicos — que torna as referências mais fáceis e mais eficientes. • Novos quadros de correlação clínica que enfatizam a aplicação clínica das neurociências fundamentais. A compra deste livro permite acesso gratuito ao site [studentconsult.com](http://studentconsult.com), um site com ilustrações para download para uso pessoal, links para material de referência adicional e conteúdo original do livro em inglês.

[Compre agora e leia](#)

*A maneira inteligente de estudar*

**Student Consult**

**Robbins**

# **PATOLOGIA BÁSICA**

TRADUÇÃO DA  
10<sup>a</sup> EDIÇÃO



KUMAR  
ABBAS  
ASTER

ELSEVIER

# Robbins Patologia Básica

Kumar, Vinay

9788535288551

952 páginas

[Compre agora e leia](#)

Parte da confiável família Robbins e Cotran, Robbins Patologia Básica 10<sup>a</sup> edição proporciona uma visão geral bem ilustrada, concisa e de leitura agradável dos princípios de patologia humana, ideal para os atarefados estudantes de hoje em dia. Esta edição cuidadosamente atualizada continua a ter forte ênfase na patogênese e nas características clínicas da doença, acrescentando novas ilustrações e diagramas mais esquemáticos para ajudar ainda mais no resumo dos principais processos patológicos e expandir o já impressionante programa de ilustrações.

[Compre agora e leia](#)



FUNDAMENTOS DE DIAGNÓSTICO POR

# Imagen em Pediatria

Lane F. Donnelly

Tradução da  
2<sup>a</sup> EDIÇÃO

ELSEVIER

FUNDAMENTALS OF  
RADIOLOGY SERIES

# Fundamentos de Diagnóstico por Imagem em Pediatria

Donnelly, Lane F.

9788535289121

376 páginas

[Compre agora e leia](#)

Realize com segurança e interprete com precisão os estudos de imagem pediátrica com este recurso conciso e altamente ilustrado! Fundamentos de Diagnóstico por Imagem em Pediatria, 2<sup>a</sup> edição, abrange os conceitos essenciais que residentes e profissionais precisam dominar, estabelecendo uma base sólida para a compreensão do básico e a realização de diagnósticos radiológicos precisos. Este título, fácil de usar na série Fundamentos de Radiologia, enfatiza técnicas avançadas de imagem, incluindo aplicações neurológicas, ao mesmo tempo em que destaca a anatomia básica necessária para entender essa complexa especialidade. • Novas informações revisadas sobre temas de qualidade e de segurança, neuroimagem, ultrassonografia em imagens pediátricas e muito mais. • Pela primeira vez especialistas adicionais fornecem atualizações em suas áreas: imagens neurológicas, musculoesqueléticas, cardíacas, torácicas e genitourinárias. • Cerca de 650 imagens digitais clinicamente relevantes e de alta qualidade demonstram, claramente, conceitos, técnicas e habilidades de interpretação essenciais. • Temas avançados de RM, como a enterografia por RM, a urografia por RM, a TC e a RM cardíacas, são discutidos minuciosamente. • Textos, tabelas e imagens acessíveis ao leitor facilitam e simplificam consultas a referências. •

Editado por Lane F. Donnelly, MD, agraciado com o Prêmio 2009 Singleton-Taybi da Sociedade de Radiologia Pediátrica pela dedicação profissional à educação médica.

[Compre agora e leia](#)