Parte I Introdução

Capítulo

1

Análise e Projeto Orientados a Objetos

O tempo é um grande professor, mas infelizmente
ele mata todos os seus alunos.
– Hector Berlioz

Objetivos

- Descrever as metas e o escopo do livro.
- Definir análise e projeto orientados a objetos (A/POO).
- Ilustrar um breve exemplo de A/POO.
- Dar uma visão geral da UML e da modelagem visual ágil

1.1 O que você aprenderá? Será útil?

O que significa ter um bom projeto de objetos? Este livro é uma ferramenta para ajudar desenvolvedores e estudantes a aprenderem as habilidades básicas usadas na análise e no projeto orientados a objetos (A/POO). Essas habilidades são essenciais para a criação de um software bem-projetado, robusto e manutenível, usando tecnologias e linguagens orientadas a objetos, tais como Java ou C#.

O que vem a seguir?

Este capítulo introduz os objetivos do livro e A/POO. O próximo capítulo introduz o desenvolvimento iterativo e evolutivo que molda como a A/POO é apresentada neste livro. Os estudos de caso são evoluídos ao longo de três iterações.

O provérbio "possuir um martelo não torna alguém um arquiteto" é particularmente verdadeiro em relação à tecnologia de objetos. Conhecer uma linguagem orientada a objetos (como Java) é um primeiro passo necessário, mas insuficiente, para criar sistemas orientados a objetos. Saber "pensar em termos de objetos" é crucial.

Esta é uma introdução à A/POO aplicando a Linguagem de Modelagem Unificada (UML) e padrões. Também, ao desenvolvimento iterativo, usando uma abordagem ágil ao processo unificado como um exemplo de processo iterativo. Ele não pretende ser um texto avançado; antes, enfatiza o domínio dos fundamentos, como atribuir responsabilidades a objetos, notação UML freqüentemente usada e padrões de projeto comuns. Ao mesmo tempo, principalmente nos capítulos posteriores, o material avança para tópicos de nível intermediário, como projeto de um framework e análise arquitetural.

UML vs. Pensar em objetos

O livro não trata somente da UML. A **UML** é uma notação padrão de diagramação. Embora seja útil aprender a notação, há questões mais cruciais orientadas a objetos para aprender; especificamente, como pensar em objetos. A UML não é A/POO ou um método, é apenas uma notação de diagramação. Assim, não adianta aprender diagramação UML e, talvez, uma ferramenta CASE UML, e não ser capaz de criar um excelente projeto OO, ou avaliar e melhorar um existente. Esta é a habilidade mais difícil e de maior importância. Conseqüentemente, este livro é uma introdução ao projeto de objetos.

Ainda assim, é necessária uma linguagem para a A/POO e para as "plantas do software", tanto como uma ferramenta de raciocínio quanto uma forma de comunicação. Portanto, este livro mostra como aplicar a UML na execução de A/POO, e cobre a notação UML freqüentemente usada.

POO: princípios e padrões

Como as **responsabilidades** devem ser atribuídas a classes de objetos? Como os objetos devem interagir? Quais classes devem fazer o quê? Estas são questões importantes no projeto de um sistema e este livro ensina a clássica metáfora de projeto OO: **projeto guiado por responsabilidades.** Também, certas soluções consagradas para os problemas de projeto podem ser (e têm sido) expressas na forma de princípios de melhores práticas, heurísticas ou **padrões** – fórmulas do tipo problema-solução, devidamente nomeadas, que codificam princípios exemplares de projeto. Este livro, ao ensinar como aplicar padrões ou princípios, permite um aprendizado mais rápido e o uso eficiente desses idiomas fundamentais do projeto de objetos.

Estudos de caso

Esta introdução à A/POO é ilustrada por meio de alguns estudos de caso em desenvolvimento que são discutidos ao longo de todo o livro, em cuidadosa e profunda abordagem da análise e do projeto, de modo que detalhes difíceis do que deve ser considerado e solucionado em um problema real são tratados e resolvidos.

Casos de uso

POO (e todo projeto de software) está fortemente relacionado à atividade pré-requisito de **análise de requisitos**, a qual inclui escrever **casos de uso**. Portanto, o estudo de caso começa com uma introdução a esses tópicos, embora eles não sejam especificamente orientados a objetos.

Desenvolvimento iterativo, modelagem ágil e um PU ágil

Considerando as muitas atividades possíveis, desde a análise de requisitos até a implementação, como deve proceder um desenvolvedor ou uma equipe de desenvolvimento? A análise de requisitos e a A/POO precisam ser apresentadas e praticadas no contexto de algum processo de desenvolvimento. Nesse caso, uma **abordagem ágil** (leve, flexível) para o bem conhecido **Processo Unificado** (PU) é usada como exemplo de **processo de desenvolvimento iterativo**, na qual estes tópicos são introduzidos. Entretanto, os tópicos de análise e projeto cobertos são comuns a muitas abordagens e seu aprendizado no contexto de um PU ágil não invalida sua aplicação a outros métodos, como Scrum, Feature-Driven Development, Lean Development, Crystal Methods, etc.

Concluindo, este livro ajuda um estudante ou um desenvolvedor a:

- Aplicar princípios e padrões para criar melhores projetos de objetos.
- Seguir iterativamente um conjunto de atividades comuns de análise e projeto, baseando-se em uma abordagem ágil para o PU como exemplo.
- Criar diagramas freqüentemente usados na notação UML.

Isso é ilustrado no contexto de estudos de caso a longo prazo, que evoluem por meio de diversas iterações.

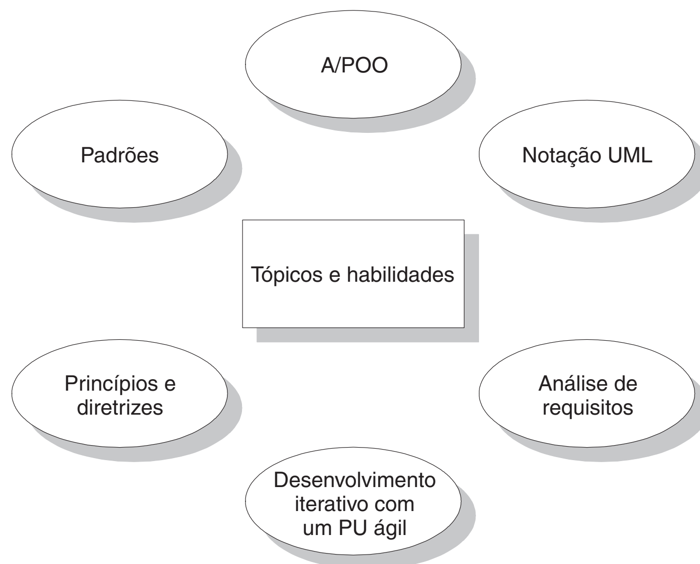


Figura 1.1 Tópicos e habilidades abordadas.

Muitas outras habilidades são importantes!

Este não é um *Livro Completo de Software*; é, principalmente, uma introdução à A/POO, UML e desenvolvimento iterativo, tocando em assuntos correlatos. Construir um software demanda muitas habilidades e passos, por exemplo, engenharia de usabilidade, projeto da interface com o usuário e projeto de base de dados são decisivas para seu sucesso.

1.2 Qual é o objetivo mais importante do aprendizado?

Existem muitas atividades e artefatos possíveis na A/POO introdutória, além de uma vasta gama de princípios e diretrizes. Suponha que devamos escolher uma única habilidade prática dentre todos os tópicos discutidos aqui – uma habilidade a ser empregada em uma "ilha deserta". Qual deveria ser ela?

> Uma habilidade crucial no desenvolvimento OO é atribuir, habilmente, responsabilidades aos objetos de software.

Por quê? Porque essa é uma atividade que precisa ser executada – seja ao desenhar um diagrama UML ou ao programar – e ela influencia drasticamente a robustez, a facilidade de manutenção e a reusabilidade de componentes de software.

Naturalmente, existem outras habilidades importantes na A/POO, mas a *atribuição de responsabilidades* é enfatizada nesta introdução porque ela tende a ser uma habilidade difícil de ser dominada (com muitos "graus de liberdade" ou alternativas), e no entanto, é de vital importância. Em um projeto real, um desenvolvedor pode não ter a oportunidade de executar quaisquer outras atividades de modelagem – um processo de desenvolvimento do tipo "corrida para codificar". Apesar disso, mesmo nessa situação, atribuir responsabilidades é inevitável.

Conseqüentemente, os passos de projeto neste livro enfatizam princípios de atribuição de responsabilidades.

> São apresentados e aplicados nove princípios fundamentais para projeto de objetos e atribuição de responsabilidades. Eles são organizados em uma forma que ajuda o aprendizado, chamada **GRASP**, com princípios que possuem nomes, tais como *Especialista em Informação* e *Criador.*

1.3 O que são análise e projeto?

A **análise** enfatiza uma *investigação* do problema e dos requisitos, em vez de uma solução, por exemplo, se desejamos um novo sistema online de comercialização, como ele será usado? Quais são as suas funções?

"Análise" é um termo de significado amplo, melhor qualificado como *análise de requisitos* (uma investigação dos requisitos) ou *análise orientada a objetos* (uma investigação dos objetos do domínio).

O **projeto** enfatiza uma *solução conceitual* (em software ou hardware) que satisfaça os requisitos e não sua implementação. Uma descrição de um esquema de banco de dados e objetos de software é um bom exemplo. Idéias de projeto excluem freqüentemente detalhes de baixo nível ou "óbvios" – óbvios para os consumidores visados. Em última instância, projetos podem ser implementados e a implementação (como por exemplo, o código) expressa o verdadeiro e completo projeto realizado.

Da mesma forma que na análise, o termo projeto é melhor qualificado como *projeto de objetos* ou *projeto de banco de dados*.

A análise e o projeto úteis foram resumidos na frase *faça a coisa certa (análise) e faça certo a coisa (projeto)*.

1.4 O que são análise e projeto orientados a objetos?

Durante a **análise orientada a objetos**, há uma ênfase em encontrar e descrever os objetos – ou conceitos – no domínio do problema. Por exemplo, no caso de um sistema de informação de vôo, alguns dos conceitos incluem *avião, vôo* e *piloto.*

Durante o **projeto orientado a objetos** (ou simplesmente projeto de objetos), há uma ênfase na definição dos objetos de software e como eles colaboram para a satisfação dos requisitos. Por exemplo, um objeto de software *avião* pode ter um atributo *numDaCauda* e um método *obterHistoricoDoVoo* (ver Figura 1.2).

Finalmente, durante a implementação ou programação orientada a objetos, os objetos de projeto são implementados, como, por exemplo, uma classe *Avião* em Java.

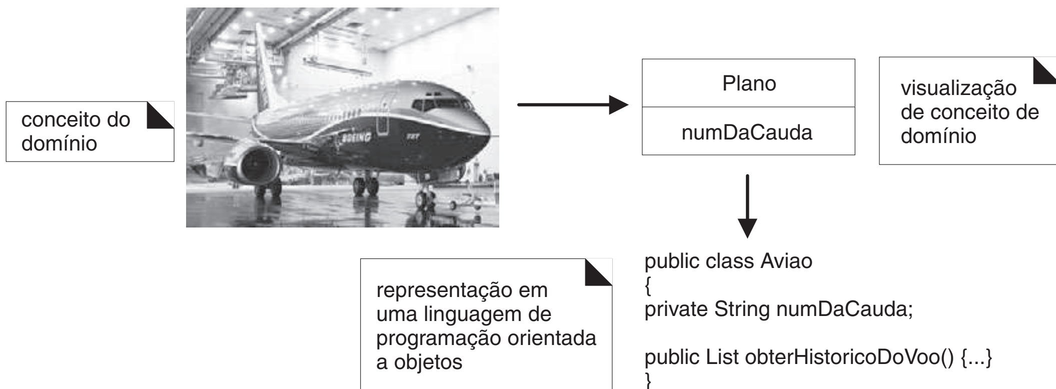
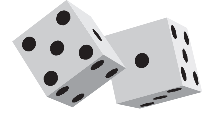


Figura 1.2 A orientação a objetos enfatiza a representação de objetos.

1.5 Um pequeno exemplo

Antes de aprofundar os detalhes do desenvolvimento iterativo UML, da análise de requisitos e da A/POO, esta seção apresenta uma visão geral de uns poucos passos-chave e diagramas, usando um exemplo simples – um jogo de dados no qual um jogador lança dois dados. Se o total for sete, ele vence; caso contrário, perde.



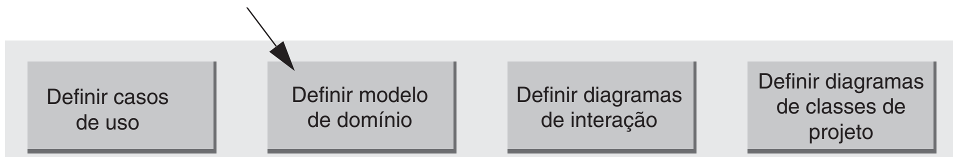
Definir casos de uso

A análise de requisitos pode incluir narrativas ou cenários sobre como as pessoas usam a aplicação; estes podem ser escritos como **casos de uso**.

Casos de uso não são artefatos orientados a objetos – eles são simplesmente narrativas escritas. Contudo, são uma ferramenta popular para a utilização na análise de requisitos. Por exemplo, temos uma versão simplificada do caso de uso *Jogar um Jogo de Dados*:

> **Jogar um Jogo de Dados**: um jogador pede que os dados sejam lançados. O sistema apresenta o resultado: se a soma do valor das faces dos dados totalizar sete, ele vence; caso contrário, perde.

Definir um modelo de domínio



A análise orientada a objetos se preocupa com a criação de uma descrição do domínio, a partir da perspectiva dos objetos. Há uma identificação dos conceitos, atributos e associações que são considerados de interesse.

O resultado pode ser expresso em um **modelo de domínio**, que mostra os conceitos ou objetos do domínio que são de interesse.

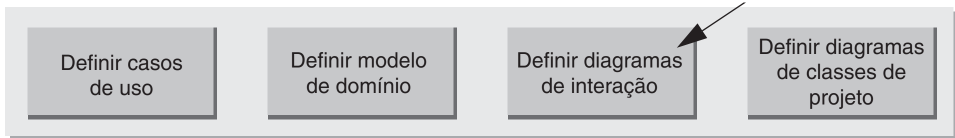
Por exemplo, um modelo parcial de domínio é mostrado na Figura 1.3.

Esse modelo ilustra os conceitos de interesse *Jogador*, *Dado* e *JogoDeDados*, com suas associações e atributos.

Note que um modelo de domínio não é uma descrição dos objetos de software; é uma visualização de conceitos ou modelos mentais do domínio do mundo real. Assim, tem também sido chamado de **modelo conceitual de objetos.**

Figura 1.3 Modelo parcial de domínio do jogo de dados.

Atribuir responsabilidade aos objetos e desenhar os diagramas de interação

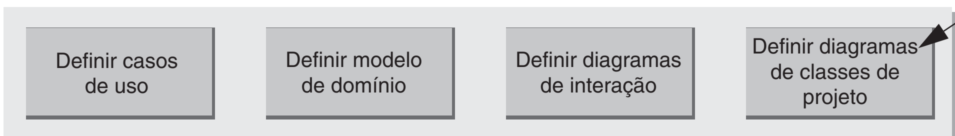


O projeto orientado a objetos se preocupa com a definição de objetos de software e suas responsabilidades e colaborações. Uma notação comum para ilustrar essas colaborações é o **diagrama de seqüência** (uma espécie do diagrama de interação da UML). Ele mostra o fluxo de mensagens entre os objetos de software e, assim, a invocação de métodos.

Por exemplo, o diagrama de seqüência da Figura 1.4 ilustra um projeto de software OO, em que são enviadas mensagens a instâncias das classes *JogoDeDados* e *Dado.* Note que isso ilustra o modo comum no mundo real pelo qual a UML é aplicada: rascunhando em um quadro branco

Observe que, embora no mundo real um jogador lance os dados, no projeto de software o objeto *JogoDeDados* "lança" os dados (isto é, envia mensagens para objetos *Dado*). Objetos do projeto de software e programas se inspiram, em parte, em domínios do mundo real, mas eles *não* são modelos diretos ou simulações do mundo real.

Definir diagramas de classes de projeto



Além de uma visão *dinâmica* de objetos colaborativos mostrada nos diagramas de interação, é útil criar uma visão *estática* das definições de classes com um **diagrama de classes de projeto**. Esse diagrama ilustra os atributos e métodos das classes.

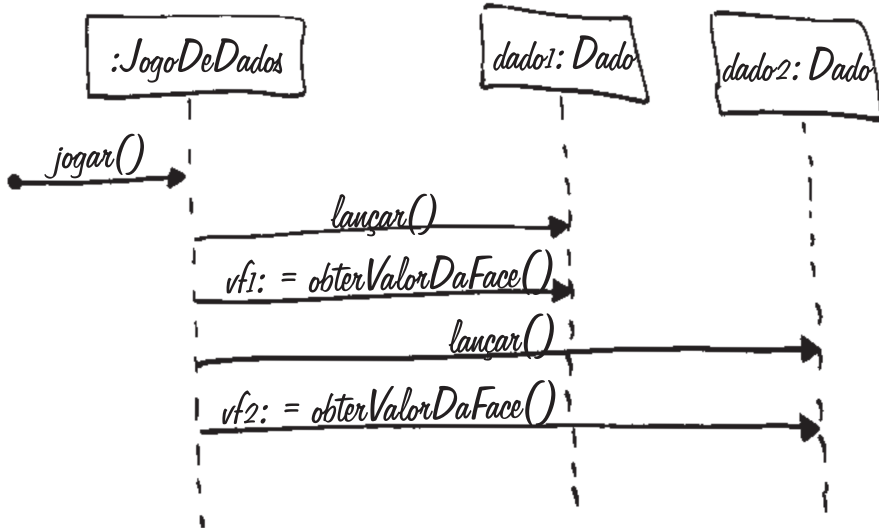


Figura 1.4 Diagrama de seqüência ilustrando mensagens entre os objetos de software.

Por exemplo, no jogo de dados, uma inspeção do diagrama de seqüência leva ao diagrama de classes de projeto parcial mostrado na Figura 1.5. Uma vez que uma mensagem *jogar* é enviada para um objeto *JogoDeDados*, a classe *JogoDeDados* requer um método *jogar* e a classe *Dado* requer um método *lançar* e um método *obterValorDaFace*.

Ao contrário do modelo de domínio, que mostra as classes do mundo real, esse diagrama mostra as classes de software.

Note que, apesar do diagrama de classes de projeto não ser o mesmo que o modelo de domínio, alguns nomes de classe e conteúdos são semelhantes. Assim, os projetos e linguagens OO podem favorecer um **baixo hiato representacional** entre os componentes de software e nossos modelos mentais de um domínio. Isso melhora a compreensão.

Resumo

O jogo de dados é um problema simples, apresentado com a intenção de focalizar alguns dos passos e artefatos na análise e projeto orientados a objetos. Visando manter

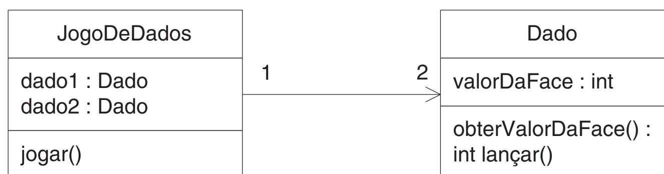


Figura 1.5 Diagrama de classes de projeto parcial.

a introdução simples, não foi explicada toda a notação UML ilustrada. Os próximos capítulos explorarão a análise, o projeto e esses artefatos em mais detalhes.

1.6 O que é UML?

Para citar:

> A Linguagem de Modelagem Unificada (UML) é uma linguagem visual para especificar, construir e documentar os artefatos dos sistemas [OMG03a].

A palavra *visual* na definição é um ponto chave – a UML é a *notação diagramática* padrão, de fato, para desenhar ou apresentar figuras (com algum texto) relacionadas a software – principalmente software OO.

Este livro não cobre minuciosamente os aspectos da UML, que é uma notação volumosa. Ele enfoca os diagramas mais usados, os recursos mais comuns nesses diagramas e a notação básica, que é a que tem menor possibilidade de mudar nas futuras versões da UML.

A UML define vários **perfis UML** que especializam subconjuntos da notação para áreas de assunto comum, tais como diagramação de Entreprise JavaBeans (com o *perfil UML EJB*).

Em um nível mais baixo – de interesse principalmente para vendedores de ferramentas CASE para a **arquitetura guiada por modelos** (Model Driven Architecture – MDA) – subjacente à notação UML, está o meta-modelo da **UML**, que descreve a semântica dos elementos de modelagem. Não se trata de algo que o desenvolvedor precisa saber.

Três modos de aplicar UML

Em [Fowler03] três modos pelos quais as pessoas aplicam UML são apresentados:

- **UML como rascunho –** diagramas incompletos e informais (freqüentemente rascunhados à mão em quadros brancos) criados para explorar partes difíceis do problema ou espaço de soluções, explorando o poder das linguagens visuais.
- **UML como planta de software –** diagramas de projeto relativamente detalhados usados seja para: 1) engenharia reversa para visualizar e melhor entender o código existente em diagramas UML; seja para 2) geração de código (engenharia avante).
    - Em engenharia reversa uma ferramenta UML lê o código fonte ou o código binário e gera (tipicamente) diagramas UML de pacotes, de classes e de seqüência. Essas “plantas de software” podem ajudar o leitor a entender os elementos, estrutura e colaborações globais.
    - Antes da programação, alguns diagramas detalhados podem fornecer diretrizes para a geração de código (po r exemplo, em Java), quer manualmente quer automaticamente, com uma ferramenta. É comum que os diagramas sejam usados para uma parte do código e outra parte seja preenchida por um desenvolvedor que esteja codificando (talvez também aplicando rascunhos UML).

UML e a idéia da "bala de prata"

Há um artigo bem conhecido de 1986, intitulado "Não é uma Bala de Prata", escrito pelo Dr. Frederick Brooks, também publicado no seu livro clássico *Mythical Man Month* (edição de 20^0 aniversário). Leitura recomendada! Um ponto essencial é que é um erro fundamental (até agora repetido continuamente) acreditar que existe alguma ferramenta ou técnica especial em software que vá fazer uma dramática diferença em ordem de magnitude em produtividade, redução de defeitos, confiabilidade ou simplicidade. *E ferramentas não compensam a ignorância em projeto.*

No entanto, você ouvirá alegações – usualmente de vendedores de ferramentas – de que o desenho de diagramas UML vai tornar as coisas muito melhores, ou de que as ferramentas de arquitetura guiada por modelos (MDA) baseadas na UML vão ser a bala de prata que vai superar os limites.

Hora de cair na real. A UML é simplesmente uma notação padrão de diagramação – caixas, linhas, etc. A modelagem visual como uma notação comum pode ser uma grande ajuda, mas dificilmente é tão importante quanto saber como projetar e pensar em termos de objetos. Esse conhecimento de projeto é uma habilidade muito mais importante e diferente, e não é conseguida pelo aprendizado da notação UML ou pelo uso de uma ferramenta CASE ou MDA. Uma pessoa que não tenha boas habilidades de projeto e programação OO que desenha UML, está somente desenhando maus projetos. Eu sugiro o artigo *Death by UML Fever [Bell04]* (endossado pelo criador da UML Grady Booch) para mais informações neste assunto e também *What UML Is and Isn´t* [Larman04].

Assim, este livro é uma introdução à A/POO e à aplicação da UML para apoiar um projeto orientado a objetos competente.

- **UML como linguagem de programação –** especificação executável completa de um sistema de software em UML. Código executável será automaticamente gerado, mas não é normalmente visto ou modificado por desenvolvedores; trabalha-se apenas na "linguagem de programação" UML. Esse uso da UML requer um modo prático de diagramar todo o comportamento ou a lógica (provavelmente usando diagramas de interação ou estado) e está ainda em desenvolvimento em termos de teoria, ferramentas robustas e usabilidade.

Modelagem ágil (págs. 57-58)

Modelagem ágil enfatiza a *UML como rascunho*; trata-se de um modo comum de aplicar a UML, freqüentemente com alto retorno no investimento de tempo (que é tipicamente curto). As ferramentas UML podem ser úteis, mas eu incentivo as pessoas a também considerar uma abordagem ágil de modelagem para aplicar a UML.

Três perspectivas para aplicar a UML

A UML descreve tipos de esboço de diagramas, tais como diagramas de classe e diagramas de seqüência. Ela não superpõe a eles uma perspectiva de modelagem. Por exemplo, a mesma notação UML de diagrama de classes pode ser usada para desenhar imagens de conceitos do mundo real ou de classes de software em Java.

Essa colocação foi enfatizada no método orientado a objetos Syntropy [CD94]. Isto é, a mesma notação pode ser usada para três perspectivas e tipos de modelos (Figura 1.6):

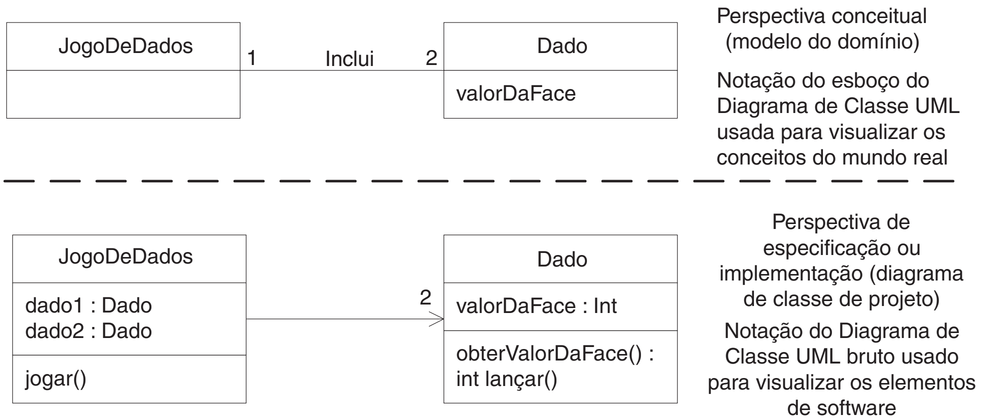


Figura 1.6 Diferentes perspectivas em UML.

1. **Perspectiva Conceitual –** os diagramas são interpretados como descrevendo coisas em uma situação do mundo real ou domínio de interesse.
2. **Perspectiva de Especificação (software) –** os diagramas (usando a mesma notação da perspectiva conceitual) descrevem abstrações de software ou componentes com especificações e interfaces, mas nenhum comprometimento com uma implementação particular (por exemplo, não especificamente uma classe em C# ou Java).
3. **Perspectiva de Implementação (software) –** os diagramas descrevem implementações de software em uma tecnologia particular (tal como Java).

Já vimos um exemplo disso nas Figuras 1.3 e 1.5, nas quais a mesma notação de diagrama de classes UML é usada para visualizar um modelo de domínio e um modelo de projeto.

Na prática, a perspectiva de especificação (adiando a definição da tecnologia alvo, tal como Java versus .NET) é raramente usada para projeto; a maior parte da diagramação em UML orientada a software considera uma perspectiva de implementação.

O significado de "classe" em diferentes perspectivas

Na UML pura, as caixas retangulares mostradas na Figura 1.6. são chamadas **classes**, mas esse termo engloba uma variedade de fenômenos – coisas físicas, conceitos abstratos, coisas de software, eventos, etc.[1]

Um método sobrepõe a terminologia alternativa à UML pura. Por exemplo, no PU, quando as caixas UML são desenhadas no modelo de domínio elas são chamadas **conceitos de domínio** ou **classes conceituais**; o modelo de domínio mostra uma perspectiva conceitual. No PU, quando as caixas UML são desenhadas no modelo de

[1] Uma classe UML é um caso especial do elemento geral de modelo UML **classificador** – algo com características e/ou comportamento estrutural, inclusive classes, atores, interfaces e casos de uso.

projeto, elas são chamadas **classes de projeto**; o Modelo de Projeto mostra uma perspectiva de especificação ou implementação, conforme desejado pelo modelador.

Para manter as coisas claras, este livro irá usar termos relacionados à classe consistentes com a UML e o PU, como segue:

- **Classe Conceitual –** coisa ou conceito do mundo real. Uma perspectiva conceitual ou essencial. O Modelo de Domínio no PU contém classes conceituais.
- **Classe de Software –** classe que representa uma perspectiva de especificação ou implementação de um elemento de software, independente do processo ou método.
- **Classe de Implementação –** classe implementada em uma linguagem OO específica, como a Java.

UML 1 e UML 2

No final de 2004 emergiu uma importante nova versão da UML, a UML 2. Este texto é baseado na UML 2; a notação aqui usada foi cuidadosamente revista com membros-chave da equipe de especificação da UML 2.

Por que durante alguns capítulos não veremos muito da UML?

Este não é essencialmente um livro sobre a notação UML, mas sim um livro que explora o panorama da aplicação da UML, padrões e um processo iterativo no contexto da A/POO e análise de requisitos relacionada. A/POO é normalmente precedida pela análise de requisitos. Portanto, os capítulos iniciais apresentam uma introdução aos tópicos de casos de uso e análise de requisitos, seguidos por capítulos sobre a A/POO, além de mais detalhes da UML.

1.7 Modelagem visual é uma boa coisa

Correndo o risco de afirmar o óbvio, o desenho ou a leitura de UML implica que estamos trabalhando mais visualmente, explorando a nossa capacidade cerebral de rapidamente abarcar símbolos, unidades e relacionamentos em (predominantemente) notações de caixas e linhas em 2D.

Essa idéia simples e antiga é freqüentemente perdida entre tantos detalhes e ferramentas UML. Não deveria ser! Diagramas nos ajudam a ver ou explorar mais do panorama e relacionamentos entre elementos de análise ou software, ao mesmo tempo em que nos permitem ignorar ou ocultar detalhes desinteressantes. Esse é o valor simples e essencial da UML ou de qualquer linguagem de diagramação.

1.8 Histórico

O histórico da A/POO tem muitas ramificações e este breve resumo não pode fazer justiça a todos os contribuintes. As décadas de 1960 e 1970 viram o surgimento de linguagens de programação OO, como Simula ou Smalltalk, com contribuintes-chave, tais como Kristen Nygaard e, especialmente, Alan Kay, o cientista de computação de

grande visão que criou o Smalltalk. Kay cunhou os termos *programação orientada a objetos* e *computação pessoal*, e ajudou a reunir as idéias do PC moderno enquanto estava na Xerox PARC.[2]

Mas a A/POO era informal ao longo daquele período e foi só em 1982 que o desenvolvimento OO emergiu como um tópico propriamente dito. Esse marco chegou quando Grady Booch (também um fundador da UML) escreveu o seu primeiro artigo intitulado *Object-Oriented Design*, provavelmente cunhando o termo [Booch82]. Muitos outros pioneiros bem conhecidos de A/POO desenvolveram suas idéias durante a década de 1980: Kent Beck, Peter Coad, Don Firesmith, Ivar Jacobson (um fundador da UML), Steve Mellor, Bertrand Meyer, Jim Rumbaugh (um fundador da UML) e Rebecca Wirfs-Brock, entre outros. Meyer publicou um dos primeiros livros influentes, *Object-Oriented Software Construction* em 1988. E Mellor e Schlaer publicaram *Object-Oriented Systems Analysis* cunhando o termo *análise orientada a objetos*, no mesmo ano. Peter Coad criou um método completo de A/POO no final da década de 1980 e publicou, em 1990 e 1991, os volumes *Object-Oriented Analysis* e *Object-Oriented Design*. Também em 1990 Wirfs-Brock e outros descreveram a abordagem de projeto guiada por responsabilidades para POO, no seu popular livro *Designing Object-Oriented Software*. Em 1991, dois livros muito populares de A/POO foram publicados. Um descrevia o método OMT, *Object Oriented Modeling and Design*, por Rumbaugh et al. O outro descrevia o método Booch, *Object Oriented Design with Applications.* Em 1992 Jacobson publicou o popular livro *Object Oriented Software Engineering* que promoveu não apenas a A/POO, mas casos de uso para requisitos.

A UML começou como um esforço de Booch e Rumbaugh, em 1994, não apenas com o intuito de criar uma notação comum, mas de combinar seus dois métodos – os métodos de Booch e OMT. Assim, o primeiro rascunho público do que hoje é a UML foi apresentado como o Método Unificado (Unified Method). Ivar Jacobson, o criador do método Objectory, se juntou a eles na Rational Corporation e, já formando um grupo, vieram a ser conhecidos como os *três amigos*. Foi nesse ponto que eles decidiram reduzir o escopo do seu esforço e enfocar em uma notação comum de diagramação – a UML – em vez de um método comum. Esse não foi apenas um esforço de diminuição do escopo; o Object Management Group (OMG, um grupo de padrões da indústria para tratar de padrões relacionados a OO) foi convencido por vários vendedores de ferramenta que um padrão aberto era necessário. Assim, o processo abriu-se e uma força tarefa da OMG chefiada por Mary Loomis and Jim Odell organizaram o esforço inicial que levou à UML 1.0 em 1997. Muitos outros contribuíram para a UML, e talvez a contribuição mais notável tenha sido a de Cris Kobryn, um dos líderes do seu contínuo refinamento.

A UML emergiu como notação de diagramação padrão de fato e de direito para a modelagem orientada a objetos e tem continuado a ser refinada em novas versões UML OMG disponíveis em: www.omg.org ou www.uml.org.

[2] Kay começou a trabalhar em OO e PC na década de 1960, quando era estudante de graduação. Em dezembro de 1979 – incitado pelo grande Jef Raskin da Apple (o líder da criação do Mac) – Steve Jobs, co-fundador e principal executivo da Apple, visitou Alan Kay e equipes de pesquisa (inclusive Dan Ingalls, o implementador da visão de Kay) na Xerox PARC para ver uma demonstração do computador pessoal Smalltalk. Impressionado pelo que viu – uma interface gráfica do usuário com janelas superpostas de mapas de bits, programação OO e PCs em rede – ele voltou à Apple com uma nova visão (aquela que Raskin desejava) e os computadores Lisa e Macintosh da Apple nasceram.

1.9 Leituras recomendadas

Vários textos sobre A/POO são recomendados nos capítulos posteriores para assuntos específicos, tal como projeto OO. Os livros mencionados na seção histórico são todos valiosos para estudo – e ainda aplicáveis no que diz respeito ao seu assunto principal.

Um resumo legível e popular da notação UML essencial, *UML Distilled,* por Martin Fowler, é altamente recomendado: Fowler tem escrito muitos livros úteis com uma atitude prática e "ágil".

Para uma discussão detalhada da notação UML, o *The Unified Modeling Language Reference Manual,* por Rumbaugh, vale a pena. Note que esse texto não tem por objetivo o aprendizado de como fazer a modelagem de objetos ou a A/POO – ele é uma referência para a notação de diagramas da UML.

Para a descrição definitiva da versão em uso da UML, veja as página online UML Infrastructure Specification e UML Superstructure Specification em www.uml.org ou www.omg.org.

A modelagem visual UML, em um espírito de modelagem ágil, é descrita em *Agile Modeling* por Scott Ambler. Veja também www.agilemodeling.com.

Há uma grande coleção de links para métodos de A/POO em www.cetus-links.org e www.iturls.com (veja a grande subseção de *Software Engineering* em inglês em vez da seção chinesa).

Existem muitos livros sobre padrões de software, mas o clássico que inspirou a todos é *Design Patterns*[‡] de Gamma, Helm, Johnson e Vlissides. Este livro é uma leitura obrigatória para aqueles que estudam o projeto de objetos. Contudo, ele não é um texto introdutório e é melhor lê-lo após sentir-se confortável com os fundamentos do projeto e programação de objetos. Veja também www.hillside.net e www.iturls.com (a subseção *Software Engineering* em inglês) para links a muitos outros sites.

[‡] N. de R.T.: Publicado no Brasil pela Bookman Editora com o título *Padrões de Projeto.*