Capítulo 7

Diagramas de Pacotes

As classes representam a forma básica de estruturação de um sistema orientado a objetos. Embora elas sejam maravilhosamente úteis, você precisa de algo mais para estruturar sistemas grandes, os quais podem ter centenas de classes.

Um **pacote** é uma construção de agrupamento que permite a você pegar qualquer construção na UML e agrupar seus elementos em unidades de nível mais alto. Seu uso mais comum é o agrupamento de classes e é dessa maneira que o estou descrevendo aqui, mas lembre-se de que você também pode usar pacotes para todos os outros elementos da UML.

Em um modelo da UML, cada classe é membro de um único pacote. Os pacotes também podem ser membros de outros pacotes, de modo que você obtém uma estrutura hierárquica na qual os pacotes de nível superior são divididos em subpacotes que possuem seus próprios subpacotes e assim por diante, até que a hierarquia chegue nas classes. Um pacote pode conter subpacotes e classes.

Em termos de programação, os pacotes correspondem a contruções de agrupamento como pacotes (em Java) e espaços de nomes (em C++ e .NET).

Cada pacote representa um **espaço de nomes**, o que significa que toda classe deve ter um nome exclusivo dentro do pacote a que pertence. Se eu quiser criar uma classe chamada Date e já houver uma classe Date no pacote System, posso ter minha classe Date, desde que a coloque em um pacote separado. Para tornar claro qual é qual, posso usar um **nome totalmente qualificado**; isto é, um nome que mostra a estrutura de pacotes ao qual pertence. Para indicar nomes de pacote na UML, você usa dois pontos duplos; portanto, as datas poderiam ser `System::Date` e `MartinFowler::Util::Date`.

Nos diagramas, os pacotes são mostrados por uma pasta com guia, como se vê na Figura 7.1. Você pode mostrar simplesmente o nome do pacote ou mostrar também o conteúdo. Em qualquer ponto, você pode usar nomes totalmente qualificados ou apenas nomes normais. Exibir o conteúdo com ícones de classe permite a você mostrar todos os detalhes de uma classe, podendo apresentar até mesmo um diagrama de classes dentro do pacote. Listar simplesmente os nomes faz sentido quando tudo que você quer fazer é indicar quais classes estão em quais pacotes.

É muito comum ver uma classe rotulada com algo como Date (`de java.util`), em vez da forma totalmente qualificada. Esse estilo é uma convenção que foi muito utilizada pela Rational Rose; ele não faz parte do padrão.

A UML permite que as classes de um pacote sejam públicas ou privadas. Uma classe pública faz parte da interface do pacote e pode ser usada por classes de outros pacotes; uma classe privada fica oculta. Diferentes ambientes de programação têm regras distintas sobre visibilidade entre suas construções de empacotamento; você deve seguir a

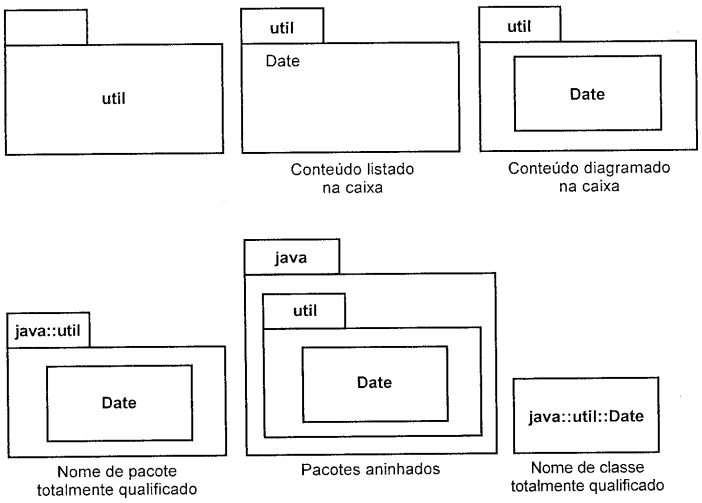

FIGURA 7.1 Maneiras de mostrar pacotes em diagramas.

convenção de seu ambiente de programação, mesmo que isso signifique violar as regras da UML.

Uma técnica útil, aqui, é reduzir a interface do pacote, exportando apenas um pequeno subconjunto das operações associadas às classes públicas do pacote. Você pode fazer isso fornecendo visibilidade privada a todas as classes, para que elas possam ser vistas apenas por outras classes do mesmo pacote, e adicionando classes públicas extras para o comportamento público. Então, essas classes extras, chamadas *Facades* [Gangue dos Quatro], delegam operações públicas às suas companheiras mais tímidas do pacote.

Como você escolhe que classes vai colocar em quais pacotes? Na verdade, essa é uma questão bastante complicada, que precisa de uma boa habilidade com projetos para ser respondida. Dois princípios úteis são o Princípio do Fechamento Comum e o Princípio da Reutilização Comum [Martin]. O Princípio do Fechamento Comum diz que as classes de um pacote devem precisar de alteração por motivos semelhantes. O Princípio da Reutilização Comum diz que todas as classes de um pacote devem ser reutilizadas juntas. Muitos dos motivos para o agrupamento de classes em pacotes estão relacionados às dependências entre os pacotes, que é o que veremos a seguir.

PACOTES E DEPENDÊNCIAS

Um **diagrama de pacotes** mostra pacotes e suas dependências. Eu apresentei a noção de dependência na página 47. Se você tem pacotes de apresentação e de domínio, então tem uma dependência do pacote de apresentação para o pacote de domínio, caso qualquer classe no pacote de apresentação tenha uma dependência de qualquer classe no pacote de

domínio. Desse modo, as dependências entre os pacotes resumem as dependências entre seus conteúdos.

A UML tem muitas variedades de dependências, cada uma com uma semântica e um estereótipo em particular. Acho mais fácil começar com a dependência não-estereotipada e usar as dependências mais particulares somente se for necessário, o que dificilmente acontece.

Em um sistema médio ou grande, representar um diagrama de pacotes pode ser uma das coisas mais valiosas que você pode fazer para controlar a estrutura de larga escala do sistema. De preferência, esse diagrama deve ser gerado a partir da própria base de código, para que você possa ver o que realmente há no sistema.

Uma boa estrutura de pacotes tem um fluxo claro das dependências, um conceito difícil de definir, mas freqüentemente mais fácil de reconhecer. A Figura 7.2 mostra um exemplo de diagrama de pacotes para uma aplicação comercial, o qual é bem estruturado e tem um fluxo claro.

Freqüentemente, você pode identificar um fluxo claro porque todas as dependências seguem uma única direção. Embora esse seja um bom indicador de um sistema bem-estruturado, os pacotes de mapeamento de dados da Figura 7.2 mostram uma exceção a essa regra geral. Os pacotes de mapeamento de dados atuam como uma camada de isolamento entre os pacotes de domínio e de banco de dados, um exemplo do padrão Mapper [Fowler, P de EAA].

Muitos autores dizem que não devem existir ciclos nas dependências (o Princípio da Dependência Acíclica [Martin]). Não trato isso como uma regra absoluta, mas acho que os ciclos devem ser localizados e, em particular, que você não deve ter ciclos que cruzem camadas.

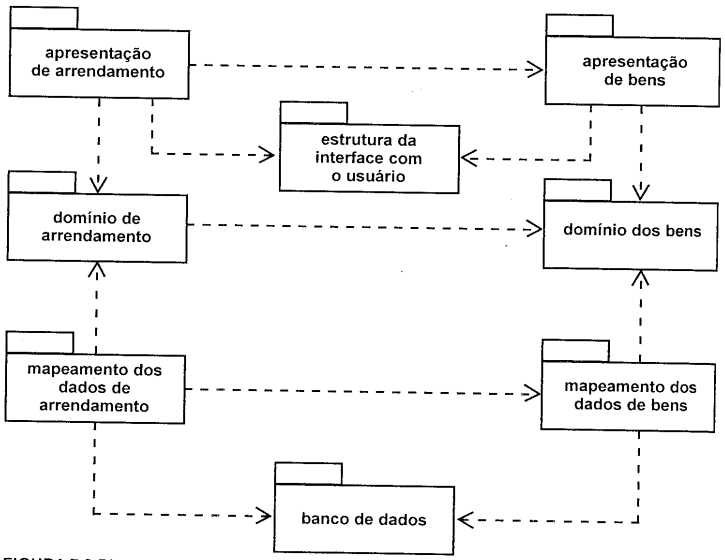

FIGURA 7.2 Diagrama de pacotes para uma aplicação comercial.

Quanto mais dependências entram em um pacote, mais estável a interface do pacote precisa ser, pois qualquer alteração em sua interface será propagada para todos os pacotes que são dependentes dela (o Princípio das Dependências Estáveis [Martin]). Assim, na Figura 7.2, o pacote do domínio dos bens precisa de uma interface mais estável do que o pacote de mapeamento de dados de arrendamento. Freqüentemente, você verá que os pacotes mais estáveis tendem a ter uma proporção mais alta de interfaces e classes abstratas (o Princípio das Abstrações Estáveis [Martin]).

Os relacionamentos de dependência não são transitivos (página 48). Para ver por que isso é importante para dependências, veja novamente a Figura 7.2. Se uma classe no pacote do domínio de bens muda, talvez tenhamos que alterar as classes dentro do pacote do domínio de arrendamento. Mas essa alteração não se propaga necessariamente para a apresentação de arrendamento. (Ela só se propaga se o domínio de arrendamento muda sua interface.)

Alguns pacotes são usados em tantos lugares, que seria uma confusão desenhar todas as linhas de dependência para eles. Nesse caso, uma convenção é usar uma palavra-chave, como «global», no pacote.

Os pacotes da UML também definem construções para permitir a importação e mesclagem de classes de um pacote para outro, usando dependências com palavras-chave para denotar isso. Entretanto, as regras para esse tipo de coisa variam muito com as linguagens de programação. Em geral, considero a noção de dependências bem mais útil na prática.

ASPECTOS DOS PACOTES

Se você pensar a respeito da Figura 7.2, perceberá que o diagrama tem dois tipos de estruturas. Uma delas é a estrutura de camadas na aplicação: apresentação, domínio, mapeamento de dados e banco de dados. A outra é uma estrutura de áreas de assunto: arrendamento e bens.

Você pode tornar isso mais aparente, separando os dois aspectos, como na Figura 7.3. Com esse diagrama, você pode ver claramente cada aspecto. Entretanto, esses dois aspectos não são pacotes verdadeiros, pois você não pode atribuir classes a um único pacote. (Você teria que escolher uma de cada aspecto.) Esse problema espelha o problema dos espaços de nomes hierárquicos nas linguagens de programação. Embora diagramas como o da Figura 7.3 não sejam UML padrão, freqüentemente eles são muito úteis na explicação da estrutura de uma aplicação complexa.

COMO IMPLEMENTAR PACOTES

Freqüentemente, você verá um caso em que um pacote define uma interface, que pode ser implementada por vários outros pacotes, como mostra a Figura 7.4. Nesse caso, o relacionamento de realização indica que o *gateway* do banco de dados define uma interface e que as outras classes de *gateway* fornecem uma implementação. Na prática, isso significaria que o pacote de *gateway* de banco de dados contém interfaces e classes abstratas que são totalmente implementadas pelos outros pacotes.

É muito comum que uma interface e sua implementação estejam em pacotes separados. Na verdade, um pacote de cliente freqüentemente contém uma interface para

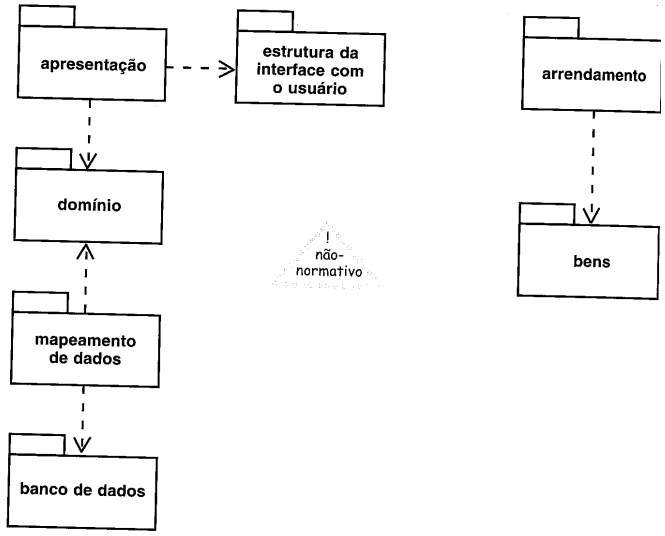

FIGURA 7.3 Separando a Figura 7.3 em dois aspectos.

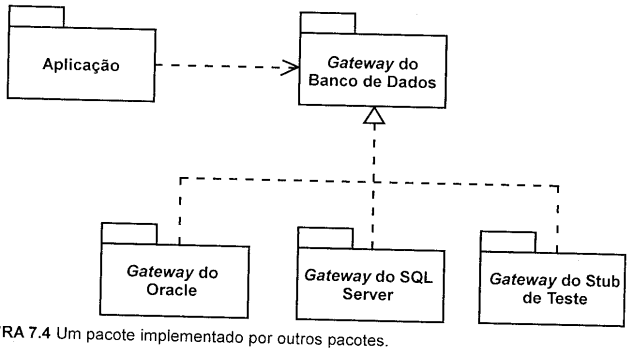

FIGURA 7.4 Um pacote implementado por outros pacotes.

um outro pacote implementar: a mesma noção de interface obrigatória que discuti na página 70.

Imagine que queiramos fornecer alguns controles de interface com o usuário (IU) para ligar e desligar coisas. Queremos que isso funcione com muitas coisas diferentes, como aquecedores e luzes. Os controles da interface com o usuário precisam executar métodos no aquecedor, mas não queremos que os controles tenham uma dependência do aquecedor. Podemos evitar essa dependência definindo, no pacote de controles, uma

interface que seja, então, implementada por qualquer classe que queira trabalhar com esses controles, como se vê na Figura 7.5. Esse é um exemplo do padrão Separated Interface [Fowler, P de EAA].

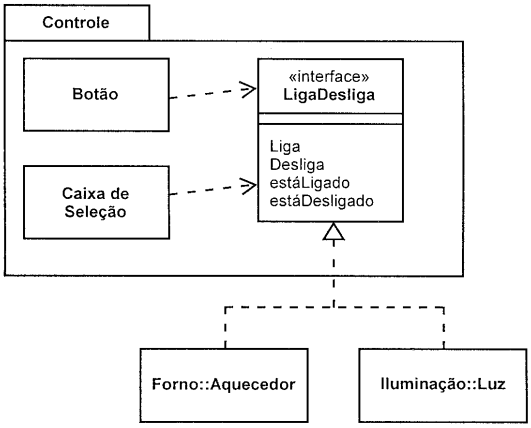

FIGURA 7.5 Definindo uma interface requerida em um pacote de cliente.

QUANDO USAR DIAGRAMAS DE PACOTES

Considero os diagramas de pacotes extremamente úteis em sistemas de grande porte, para obter uma visão das dependências entre os principais elementos de um sistema. Esses diagramas correspondem bem às estruturas usuais de programação. A representação de diagramas de pacotes e dependências o ajuda a manter as dependências de uma aplicação sob controle.

Os diagramas de pacotes representam um mecanismo de agrupamento em tempo de compilação. Para mostrar como os objetos são compostos em tempo de execução, use um diagrama de estrutura composta (página 132).

ONDE ENCONTRAR MAIS INFORMAÇÕES

A melhor discussão que conheço sobre pacotes e como utilizá-los é [Martin]. Há algum tempo, Robert Martin tem uma obsessão quase patológica com dependências e escreve muito bem sobre como prestar atenção a elas para que você possa controlá-las e minimizá-las.