Projeto de Software

UNIDADE 4

FASE DE ELABORAÇÃO
Disciplina de Requisitos: Modelos de Classe de Projeto

Autores:

Profa. Dra Rosana Terezinha Vaccare Braga
Prof. Dr. Paulo Cesar Masiero

ICMC/USP

Professor da Disciplina:
Prof. Dr. Valter Camargo
Prof. Esp. Alexandre Pedroso Fernandes

7.1 – Introdução

Segundo o PU, na última etapa da fase de projeto deve-se reunir todas as informações presentes nos diagramas de colaboração, principalmente sobre classes e mensagens trocadas, e juntá-las às informações obtidas durante a análise do sistema, com o objetivo de compor as classes de projeto do sistema, que são as classes que serão efetivamente implementadas utilizando uma linguagem de programação orientada a objetos. O Diagrama de Classes de Projeto é composto de classes relacionadas por associações comuns, de herança ou de agregação. Cada classe possui atributos e métodos. Além disso, no projeto deve-se conhecer de antemão quais classes manterão referências a outras classes, por isso é necessário estabelecer a visibilidade entre classes, explicada na seção 7.2. Depois, com base nos diagramas de colaboração e no modelo conceitual do sistema pode-se finalmente criar os diagramas de classes de projeto (seção 7.3).

7.2 – Visibilidade entre classes

Visibilidade é a capacidade de um objeto ver ou fazer referência a outro: para que um objeto A envie uma mensagem para o objeto B, é necessário que B seja visível para A. Existem quatro tipos de visibilidade: por atributo (B é um atributo de A), por parâmetro (B é um parâmetro de um método de A), localmente declarada (B é declarado como um objeto local em um método de A) e global (B é, de alguma forma, globalmente visível).

7.2.1 – Visibilidade Por Atributo

A visibilidade por atributo ocorre sempre que uma classe possui um atributo que referencia um ou mais objetos de outra classe. Por exemplo, na Figura 7.1 pode-se entender que a classe Empréstimo contém um atributo leitor, que é uma referência para o objeto da classe Leitor, mais especificamente para o leitor que efetuou o empréstimo. Assim, objetos da classe Empréstimo podem enviar mensagens ao leitor sempre que precisarem. Dizemos que Empréstimo tem visibilidade por atributo à classe Leitor. No código do método emprestarCopia() da classe Emprestimo (Figura 7.1) pode-se ver que é enviada a mensagem calcularDataDevolução() ao objeto leitor, porque o objeto empréstimo o conhece e portanto pode comunicar-se com ele.

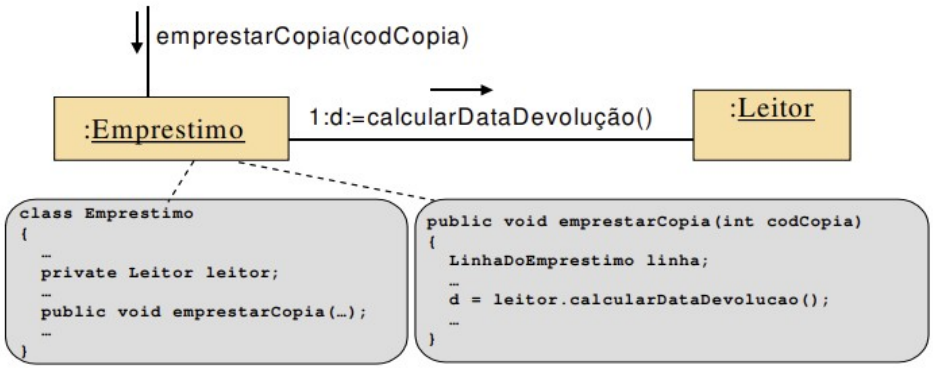

Figura 7.1 – Visibilidade por atributo

A visibilidade por atributo é a forma mais comum de visibilidade. A visibilidade persiste por muito tempo, pois considera-se que o objeto origem terá visibilidade para o destino sempre que precisar. A visibilidade só deixa de existir quando o objeto origem for removido.

Geralmente, a visibilidade por atributo se deve às associações existentes no modelo conceitual, ou seja, para muitas das associações existe visibilidade entre as classes. Resta saber em que sentido deve-se considerar a visibilidade. Por exemplo, existe uma associação de muitos para um entre Empréstimo e Leitor (um empréstimo é feito a um leitor, um leitor faz vários empréstimos). Em tempo de projeto deve-se decidir se a visibilidade entre Empréstimo e Leitor será nos dois sentidos ou se será somente em um sentido (nesse caso, em qual deles). Se considerarmos a visibilidade de Empréstimo para Leitor, a classe Empréstimo terá um atributo leitor, denotando o leitor que fez o empréstimo. Já se considerarmos a visibilidade de Leitor para Empréstimo, a classe Leitor deverá ter um atributo empréstimos (provavelmente do tipo vetor ou outro tipo de conjunto), denotando o conjunto de empréstimos realizados por um determinado leitor.

Se no diagrama de colaboração há troca de mensagens entre dois objetos que possuem visibilidade por atributo, o acoplamento é mais forte do que se a visibilidade for dos demais tipos (ver seções seguintes).

Embora uma visibilidade por atributo venha a ser implementada posteriormente como um atributo na classe origem, isso não deve ser mostrado no diagrama de classes. Considera-se que a associação representa esse atributo, e pode-se dar um nome ao papel desempenhado pelo extremo da associação, para que esse nome venha a ser utilizado durante a programação da classe.

7.2.2 – Visibilidade Por Parâmetro

A visibilidade por parâmetro ocorre sempre que uma classe recebe um objeto como parâmetro em alguma invocação de método e, portanto, conhece o objeto enviado por parâmetro e pode enviar-lhe mensagens. Por exemplo, na Figura 7.2 pode-se entender que a classe Empréstimo recebe o objeto fita como parâmetro quando da invocação do seu método adiciona. Assim, durante a execução desse método específico (adiciona), é possível enviar mensagens ao objeto fita, ou seja, fita é visível para emprestimoCorrente somente temporariamente, por tê-lo recebido como parâmetro. (ver Figura 7.2). No caso do exemplo, o código fonte mostra que não há invocação de nenhum método do objeto fita, mas ele é passado adiante como parâmetro na invocação de um outro método, no caso o método associaFita da classe ItemDoEmprestimo.

A visibilidade por parâmetro é relativamente temporária, ou seja, ela persiste enquanto persistir o método. Portanto, terminada a execução do método adiciona, o objeto emprestimoCorrente não terá mais visibilidade para o objeto fita.

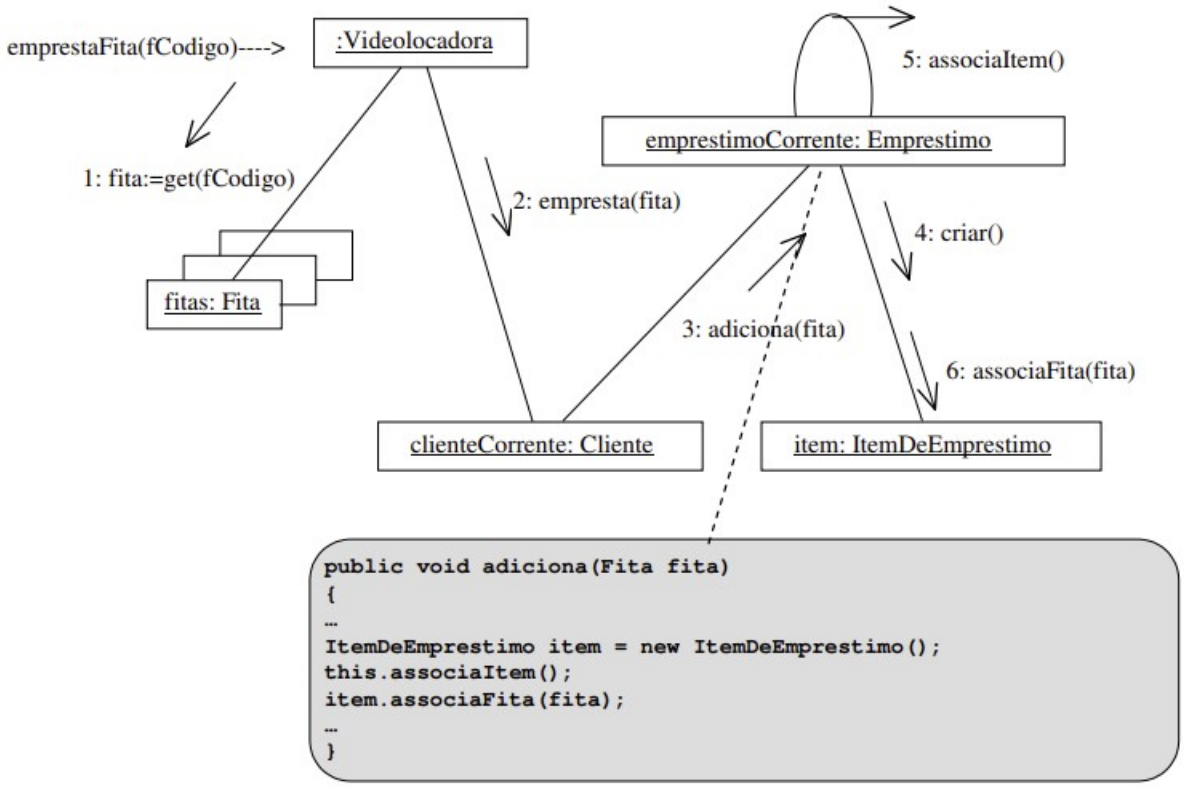

Figura 7.2 – Visibilidade por parâmetro

7.2.3 – Visibilidade Localmente declarada

A visibilidade é dita localmente declarada se o objeto tiver referência a outro que declarou localmente, por meio de uma variável local. Há duas formas de fazer isso: criar uma nova instância local e atribuí-la a uma variável local, ou atribuir o objeto retornado pela invocação de um método a uma variável local. No primeiro caso, considere a Figura 7.2. A variável local item, declarada em um método pertencente à classe Empréstimo, faz referência a um objeto da classe ItemDeEmprestimo. Portanto, dizemos que a classe Empréstimo possui visibilidade local para a classe ItemDeEmprestimo.

Esse tipo de visibilidade é relativamente temporária, pois só existe durante a execução do método, da mesma forma que na visibilidade por parâmetro (seção 7.2.2.). Outro exemplo de visibilidade local é quando utiliza-se uma variável local para armazenar o retorno da invocação de um método de outra classe. Observe, por exemplo, o código da Figura 7.3, em que a variável emprestimoCorrente é declarada em um método da classe VideoLocadora. Essa variável é utilizada para armazenar o objeto que retorna da invocação do método clienteCorrente, da classe Cliente, que por sua vez retorna um objeto da classe Empréstimo. Portanto, dizemos que VideoLocadora tem visibilidade local para Empréstimo.

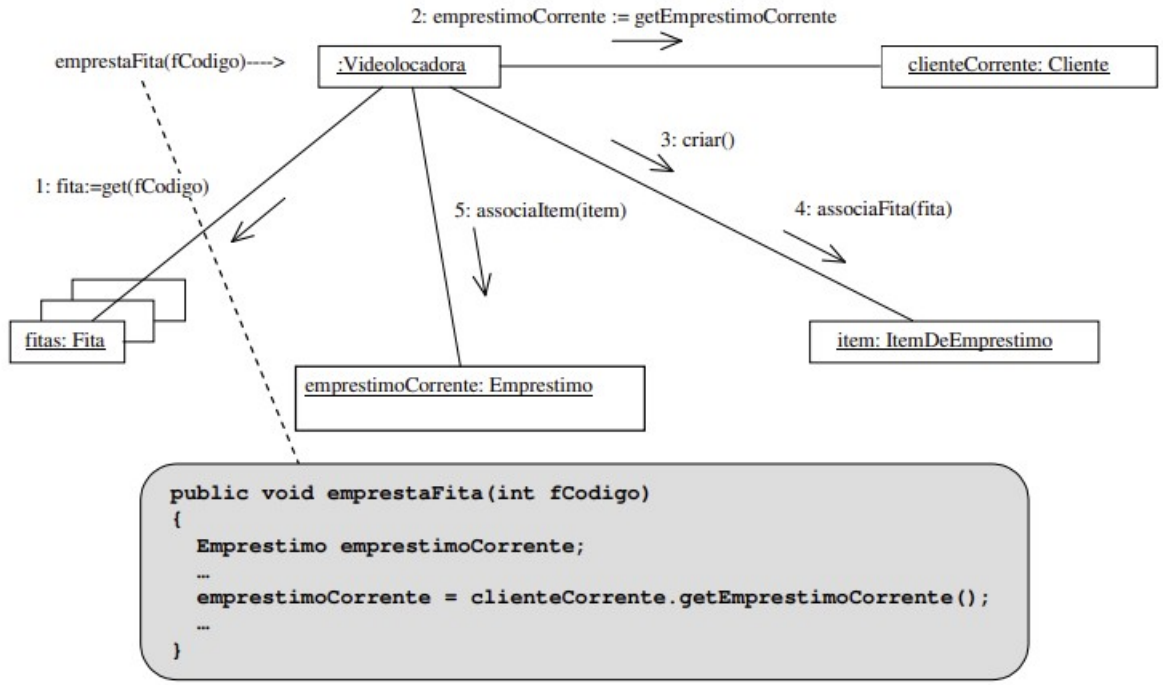

Figura 7.3 – Visibilidade localmente declarada

7.2.4 – Visibilidade Global

O tipo de visibilidade menos comum é a global, que ocorre quando um objeto de uma certa classe é sempre visível a quaisquer outros objetos do sistema. Esse tipo de visibilidade é relativamente permanente, pois persiste enquanto o objeto existir. Uma forma de implementar a visibilidade local é por meio do padrão Singleton [Gamma, 1995], descrito no Capítulo 9. Uma forma óbvia e menos desejável de conseguir visibilidade global é por meio da atribuição de uma instância ou objeto a uma variável global.

7.2.5 – Notação UML para Visibilidade

Em geral, só são mostradas nos diagramas de classes as visibilidades por atributo, por meio de uma seta direcionada na relação de associação entre as classes, partindo da classe que tem visibilidade para a classe para a qual ela tem referência. Isso será mostrado na seção 7.3.

Pode-se também usar estereótipos da UML para denotar a visibilidade nos diagramas de colaboração, conforme indicado na Figura 7.4. Isso é uma indicação explícita do tipo de visibilidade, mais comumente utilizada quando é difícil descobrir o tipo de visibilidade somente pelas informações presentes no diagrama. Em geral, o raciocínio é o seguinte:

- se não houver nenhuma indicação, subentende-se que a visibilidade é por atributo;
- se o objeto X recebe um outro objeto Y como parâmetro, entende-se que ele pode invocar mensagens desse objeto Y; e
- se o objeto cria um novo objeto (usualmente pela mensagem criar() ou novo()), a visibilidade é local.

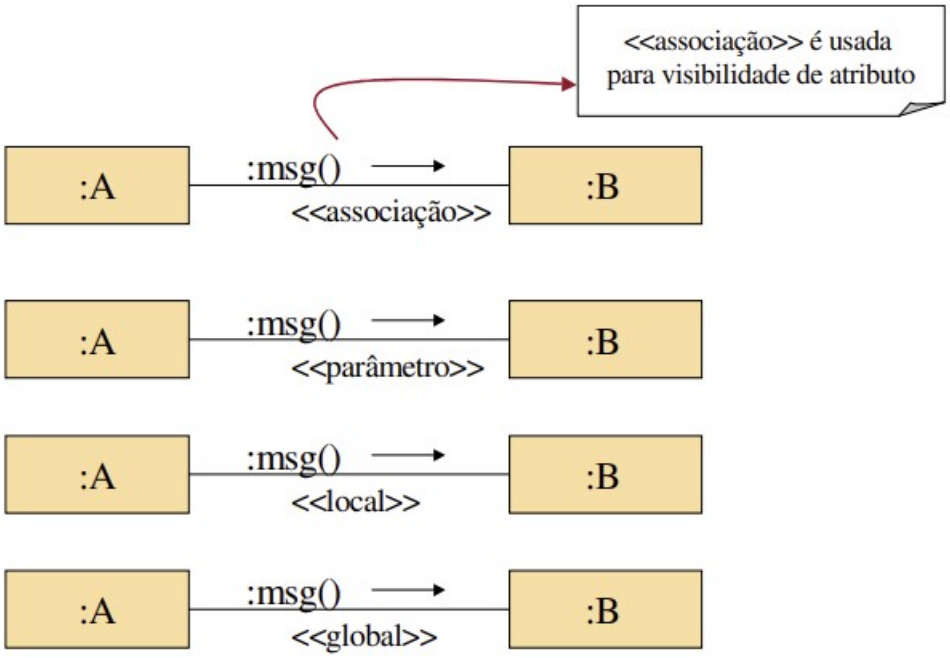

Figura 7.4 – Visibilidade na UML

7.3 – Diagrama de classes de Projeto

O Diagrama de Classes de Projeto apresenta as especificações para as classes de software e respectivas interfaces (por exemplo interfaces Java) a serem implementadas em uma aplicação. As informações típicas contidas em um diagrama de classes são: classes, associações e atributos (como no modelo conceitual, substituindo-se os conceitos por classes); interfaces, com operações e constantes (trata-se da assinatura de todos os métodos que cada classe oferece, ou seja, os métodos que podem ser invocados por outras classes que tenham visibilidade para ela); métodos (ou seja, as linhas de código que implementam o comportamento esperado da classe); tipos dos atributos (cada atributo deve ter um tipo, que pode ser primitivo, composto, definido pelo usuário, referência a outro objeto ou coleções de objetos, etc); navegabilidade (indica a visibilidade por atributo de uma classe para outra); e dependências (indica visibilidade dos outros três tipos: por parâmetro, local ou global).

7.3.1 – Modelo Conceitual X Diagrama de Classes de Projeto

No Modelo Conceitual são feitas abstrações de conceitos ou objetos do mundo real, também chamados de classes conceituais. A idéia e mostrar os objetos que compõem o sistema, para dar uma visão geral, de forma esquemática e fácil de entender. Já o Diagrama de Classes de Projeto apresenta a definição de classes como componentes de software, ou seja, as classes que o compõem serão efetivamente implementadas usando uma linguagem de programação OO.

Na prática, o digrama de classes pode ser construído à medida que a fase de projeto avança, a partir dos diagramas de colaboração. Cada classe que aparece no diagrama de colaboração automaticamente é incluída no diagrama de classes de projeto. Neste livro preferimos fazer primeiro os diagramas de colaboração, para depois reunir as informações necessárias para o diagrama de classes de projeto.

7.3.2 – Como identificar as classes e atributos do Diagrama de Classes de Projeto

Todas as classes que aparecerem nos diagramas de colaboração devem ser incluídas no diagrama de classes de projeto, porque são classes que receberão e invocarão mensagens para satisfazer o comportamento esperado do sistema. Por exemplo, observando-se os diagramas de colaboração das Figuras 6.15 e 6.20 (Capítulo 6), chega-se às seguintes classes: Leitor, Empréstimo, LinhaDoEmpréstimo e CopiaDoLivro. Analisando o modelo conceitual da biblioteca (Figura 4.15 do Capítulo 4), pode-se deduzir quais são os atributos dessas classes. Pode-se acrescentar tipos de atributos também, se for o caso, para guiar a futura implementação. Se uma ferramenta CASE for utilizada para geração automática de código, os tipos detalhados são necessários. Por outro lado, se o diagrama for usado exclusivamente por desenvolvedores de software, o excesso de informação pode "poluir" o diagrama e dificultar seu entendimento. Além dos atributos identificados durante o projeto, podem ser incluídos outros atributos mais específicos ou para satisfazer restrições técnicas de projeto. Portanto, pode-se começar a esboçar o diagrama de classes de projeto, conforme ilustrado na Figura 7.5. É claro que, no caso do sistema de biblioteca, se outros diagramas de colaboração fossem observados, muitas outras classes seriam identificadas.

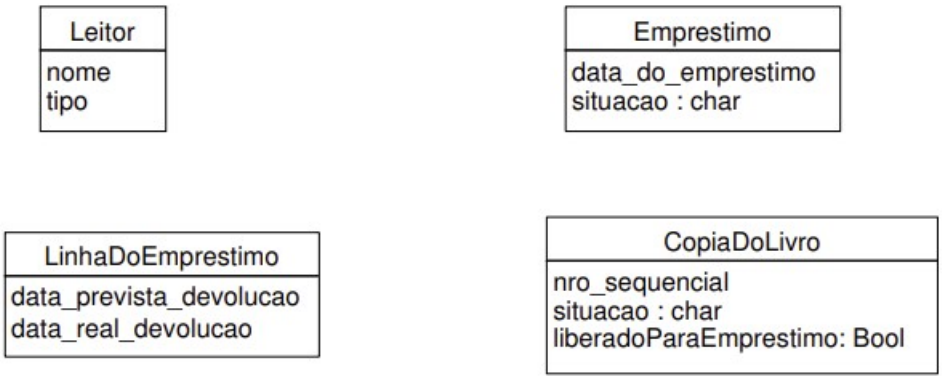

Figura 7.5 – Esboço inicial das classes da biblioteca

7.3.3 – Como identificar as associações e navegabilidade no Diagrama de Classes de Projeto

Para fazer as **associações** entre as classes identificadas, deve-se novamente observar todos os diagramas de colaboração construídos durante a fase de projeto. Todas as mensagens devem ser analisadas. Quando houver visibilidade por atributo de uma classe para outra, subentende-se que há uma associação entre essas classes. Para descobrir a multiplicidade, pode-se observar o modelo conceitual e, se ele não for suficiente, deve-se examinar a lógica dos métodos que são invocados de uma classe para a outra para descobrir qual é a multiplicidade da associação. Os nomes das associações também podem ser derivados a partir do modelo conceitual. Caso a associação não exista no modelo conceitual, deve-se dar um nome a ela durante o projeto.

Para exemplificar a identificação de associações, considere a Figura 7.6, em que se mostram as associações entre as classes identificadas anteriormente para o sistema de Biblioteca (Figura 7.5). Foram incluídas as associações de acordo apenas com os diagramas de colaboração das Figuras 6.15 e 6.20. As multiplicidades e os nomes das associações foram extraídos do Modelo Conceitual da Figura 4.15. A navegabilidade foi determinada da seguinte forma: se A envia mensagem para B, ou se A cria B, ou ainda se A precisa manter uma conexão com B, então nota-se um indício de que A deve ter visibilidade para B, ou seja, deve-se desenhar no diagrama de classes uma seta na associação entre A e B com a seta posicionada no lado de B. Deve-se verificar o envio de mensagens de objetos que possuem visibilidade por atributo, pois para os demais tipos de visibilidade não é necessário mostrar no diagrama de classes, para não torná-lo demasiadamente complexo sem necessidade. Em alguns casos, pode-se mostrar esses outros tipos de visibilidade por meio de dependência, como explicado na Seção 7.3.5.

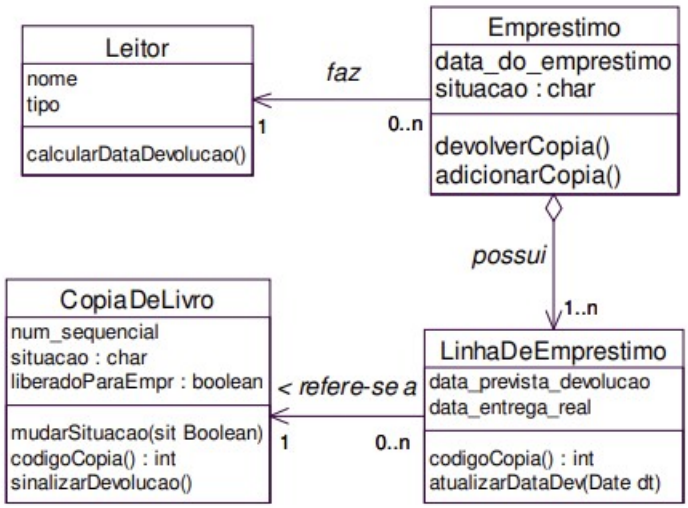

Figura 7.6– Diagrama de classes parcial para a biblioteca

Examinando-se o diagrama de colaboração da Figura 6.15, conclui-se que há navegabilidade entre as classes Emprestimo e Leitor (no sentido de Empréstimo para Leitor) e Emprestimo e LinhaDoEmprestimo, por haver visibilidade por atributo entre elas. Similarmente, examinando-se o

diagrama de colaboração da Figura 6.20, nota-se que há navegabilidade entre Emprestimo e LinhaDoEmprestimo e entre LinhaEmprestimo e CopiaDoLivro. Essas navegabilidades são denotadas pelas setas dirigidas na Figura 7.6.

7.3.4 – Como identificar herança e agregação no Diagrama de Classes de Projeto

Associações de herança e agregação também se propagam para o diagrama de classes. Freqüentemente podem aparecer novas associações de herança durante o projeto, quando mais detalhes das classes passam a ser conhecidos e pode-se desejar abstraí-los para aumentar o reuso de código na fase seguinte. Por exemplo, na Biblioteca, percebe-se que tanto a reserva quanto o empréstimo tem vários atributos em comum, portanto pode-se decidir pela criação de uma classe que abstraia essas duas, por exemplo, uma classe denominada "Transação", com os atributos comuns tanto a Empréstimo quanto a Reserva, fazendo com que essas duas últimas herdem da primeira, conforme ilustrado na Figura 7.7.

7.3.5 – Como identificar métodos no Diagrama de Classes de Projeto

Além de mostrar os atributos de cada classe, o diagrama de classes de projeto deve mostrar também os métodos que serão implementados em cada classe. A idéia é mostrar apenas a assinatura do método, ou seja, o nome do método, seus parâmetros de entrada e suas saídas. Linguagens de programação distintas podem ter sintaxes distintas para métodos. Portanto, recomenda-se que seja utilizada a sintaxe básica UML para métodos, que é a seguinte: nomeMétodo(Par1, Par2, … Parn): Retorno, onde Par1, Par2,... Parn são os parâmetros de entrada do método e Retorno é o tipo de dado ou objeto retornado pelo método. Na UML, os métodos são exibidos no terceiro compartimento do retângulo que representa a classe, conforme ilustrado na Figura 7.6 (por exemplo, calcularDataDevolucao é um método da classe Leitor).

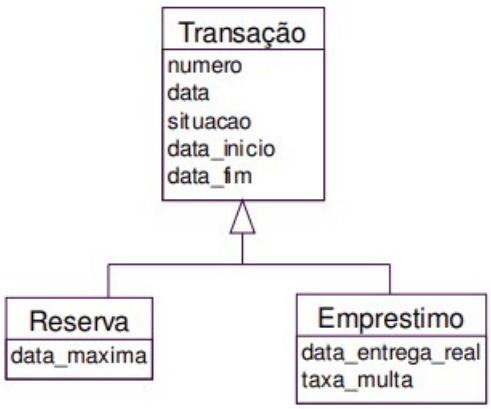

Figura 7.7– Herança no diagrama de classes

7.3.5.1 – Métodos que não se deve incluir no diagrama de classes

Mas como saber que métodos incluir em cada classe? Comecemos dizendo quais métodos não incluir nas classes. Em geral, os métodos de acesso, que são aqueles responsáveis por atribuir e recuperar valor de atributos da classe (como setNome, getNome, etc.), embora importantes, não devem ser incluídos para não sobrecarregar a classe com muitos métodos. Assim, considera-se que a classe deverá implementá-los, mas eles não são mostrados no diagrama de classes explicitamente.

Métodos referentes a mensagens enviadas a coleções de objetos também não devem ser mostrados no diagrama de classes, pois dependem da implementação. Por exemplo, o método próximo() na Figura 7.8 é enviado à coleção de linhas de empréstimo para obter a próxima linha a ser considerada durante a devolução da cópia. Não incluímos o método próximo() na classe LinhaDoEmprestimo e consideramos que ele será implementado pelo mecanismo escolhido para representar a coleção, seja ele uma lista, um arquivo, etc. Da mesma forma, não seriam mostrados métodos como buscar(), adicionar(), etc, enviados à coleções de objetos. O método criar() também não deve ser incluído no diagrama de classes de projeto, pois considera-se que a linguagem OO fornece o método criador de alguma forma. Por exemplo, em Java utiliza-se o construtor da classe e em Smalltalk a palavra chave new.

7.3.5.2 – Métodos a incluir no diagrama de classes

Passemos agora para os métodos que devem ser incluídos no diagrama de classes de projeto. Em primeiro lugar, as operações (identificadas por meio dos diagramas de seqüência do sistema, seção 5.1) devem ser incluídas nas classes controladoras, de acordo com o projeto OO realizado anteriormente, no qual foi feita a escolha do padrão Controlador mais adequado (seção 6.3.7). Por exemplo, na Figura 7.8, a operação devolverCopia, foi atribuída à classe controladora fachada Empréstimo, e portanto um método de mesmo nome é incluído na classe Empréstimo (ver Figura 7.6).

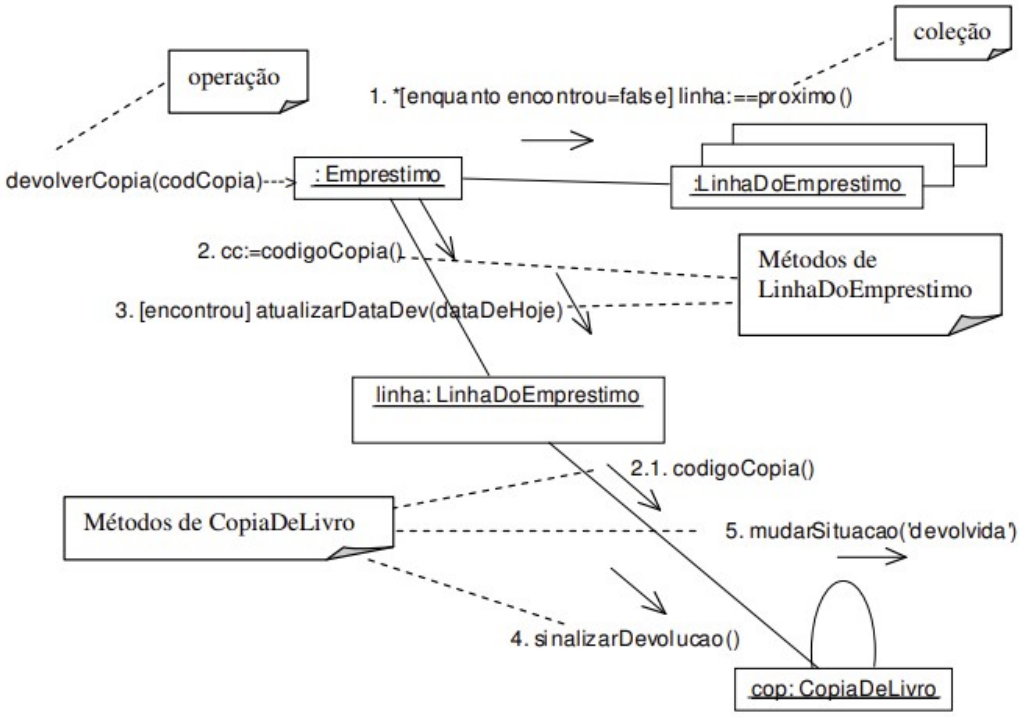

Figura 7.8– Identificação de métodos no diagrama de colaboração

A última regra é incluir os métodos nas classes que recebem a mensagem correspondente. Por exemplo, no diagrama de colaboração da Figura 7.8 pode-se deduzir a existência dos seguintes métodos de LinhaDoEmprestimo e CopiaDeLivro

- LinhaDoEmprestimo: codigoCopia(), atualizarDataDev() e
- CopiaDeLivro: mudarSituacao(), codigoCopia() e sinalizarDevolucao()

Todos os diagramas de colaboração produzidos durante o projeto OO devem ser analisados para identificar os métodos a serem incluídos no diagrama de classes de projeto do sistema. Para fins do exemplo da biblioteca, analisemos mais um diagrama de colaboração, mostrado na Figura 7.9. Por meio dele chegamos a mais dois métodos a serem incluídos no diagrama de classes: a operação adicionarCopia() é incluída como um método na classe controladora Emprestimo e o método calculaDataDevolucão é incluído na classe Leitor. O método criar() não é incluído, conforme explicado na seção 7.3.5.1. A Figura 7.6 mostra o diagrama de classes com os métodos já identificados.

7.3.6 – Dependência entre classes

Opcionalmente, pode-se incluir no diagrama de classes de projeto as dependências entre classes. Uma classe é dita dependente de outra se ela de alguma forma invoca métodos ou cria objetos dessa outra classe, mas não tem visibilidade por atributo para ela, ou seja, a classe tem visibilidade de outros tipos (por parâmetro, local ou global). A dependência é ilustrada em UML por

meio de uma linha tracejada com uma seta no sentido da classe dependente, ou seja, a ponta da seta fica na extremidade referente à classe da qual se depende.

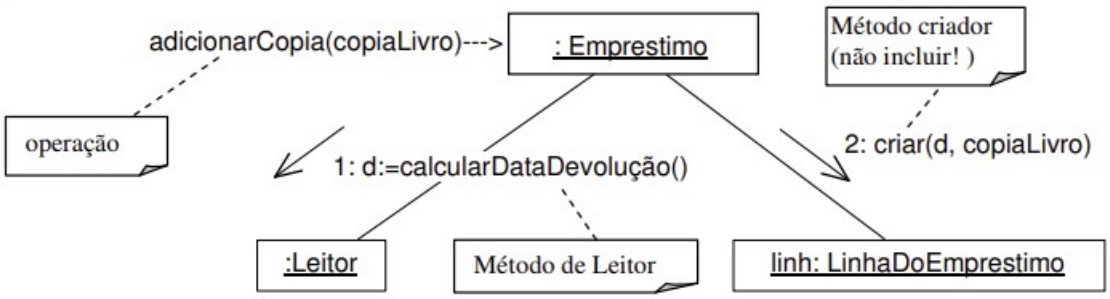

Figura 7.9– Identificação de métodos no diagrama de colaboração

Pode ser útil visualizar a dependência entre classes para, por exemplo, fazer manutenção no sistema, quando é preciso ter idéia do que uma certa mudança pode causar em outras classes. Sabendo-se que uma classe A depende de uma classe B, se algo for mudado na interface da classe B, pode ser que a classe A tenha que mudar a forma de invocar um certo método, por exemplo.

Como exemplo de dependência, a Figura 7.10 ilustra a dependência entre as classes CopiaDeLivro e Leitor. Imagine que o método sinalizarDevolucao precisasse regularizar a situação do leitor diante da devolução do livro. Como não tem visibilidade para ele, poderia receber como parâmetro o objeto leitor, e então poderia invocar algum de seus métodos para executar essa responsabilidade. Para que isso fosse cumprido, ao invocar o método atualizarDataDev de LinhaDeEmprestimo, o objeto de Empréstimo deveria passar leitor como parâmetro, para que este, por sua vez, pudesse passar para CopiaDeLivro.

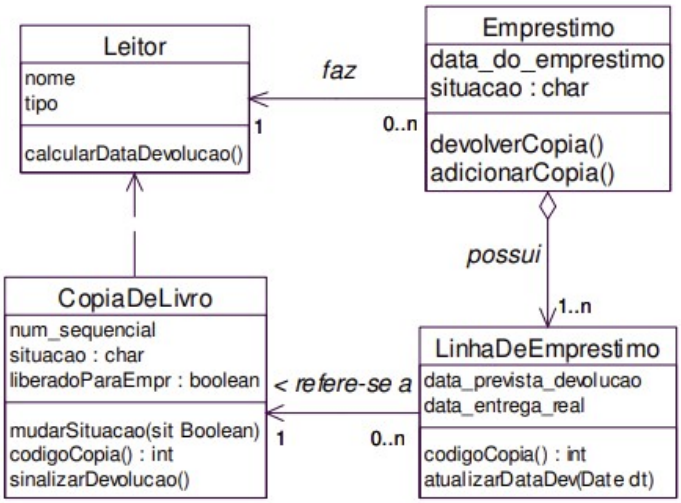

Figura 7.10– Diagrama de Classes com associação de dependência

7.3.7 – Mais detalhes sobre a notação para diagrama de classes

A Figura 7.11 mostra mais alguns conceitos sobre diagramas de classes que podem ser úteis/necessários durante o projeto OO. Um atributo pode ter seu valor inicial atribuído no diagrama (1). Se um atributo da classe for desejado, ao invés de um atributo dos objetos instanciados a partir da classe, sublinha-se o nome do atributo (2). Isso significa que o atributo pertencerá à classe e não a seus objetos individualmente. Pode ser interessante, em determinados projetos, ilustrar no diagrama de classes atributos que podem ser derivados ou calculados por outros meios, por exemplo, por chamada de métodos ou operações aritméticas. Isso é ilustrado colocando-se uma barra invertida antes do nome do atributo (3). Atributos derivados provavelmente não serão implementados na linguagem de programação e não serão armazenados na base de dados, mas pode ser útil durante o projeto saber que o atributo é inerente à classe. Métodos abstratos (4) são denotados com itálico. Um método é abstrato se seu comportamento não existe na superclasse, mas é definido apenas em suas subclasses. Métodos públicos são aqueles visíveis a quaisquer classes que precisem invocá-los. São denotados por um sinal de + (5). Já os métodos privados são visíveis apenas na classe na qual estão definidos e são denotados por um hífen (6). Por fim, os métodos protegidos, denotados pelo sinal #, são aqueles visíveis na classe em que estão definidos, bem como em quaisquer subclasses dela (7). Assim como para atributos, métodos também podem se referir à classe em si, sendo denotados em UML pelo sublinhado (8). Métodos da classe podem ser invocados diretamente à classe, ao invés de a instâncias da classe.

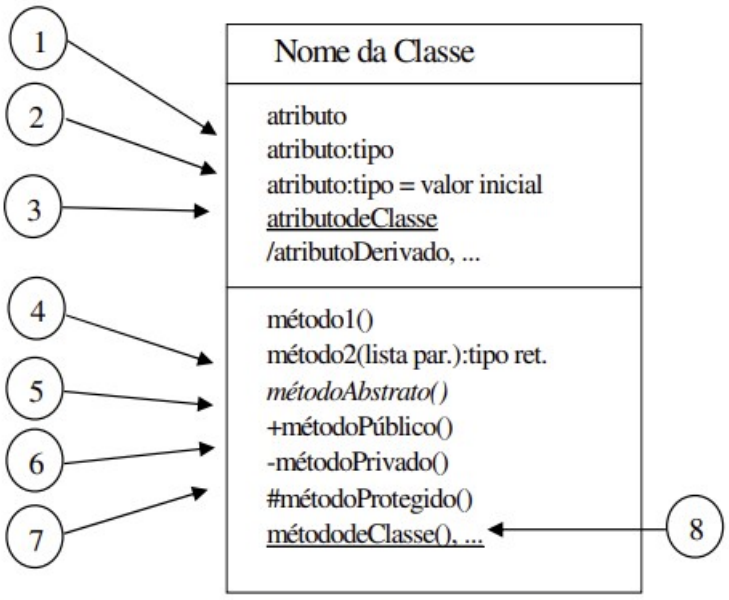

Figura 7.11 – Resumo da notação UML para Diagrama de Classes