8

TEXTO BASE

ANÁLISE E MODELAGEM DE SISTEMAS

Faculdade IMPACTA

Texto base

8

O.O. Classes

Definição, elementos e modelagem

Prof. Renato de Tarso Silva

Resumo

Este texto traz a tona definições de classes. São elementos essenciais na aplicação da orientação a objetos. Aqui serão conhecidos os tipos, os elementos, e os principais tipos de relacionamentos entre classes. O bom entendimento de classes viabiliza a criação de interfaces e padrões de projetos de software.

8.1. Definição

As classes são como padrões, funcionam como moldes que têm a capacidade, e a responsabilidade, de formar objetos que sejam do tipo de entidade compostas pelos atributos, operações e relacionamentos da classe em questão. Uma classe pode representar uma entidade ou um conceito, que pode conter em suas propriedades algumas características, ou seja, seus atributos, e comumente contém métodos, que são suas funcionalidades/operações. Concebidas no levantamento de requisitos, as classes mantém de forma estruturada os elementos sistêmicos, identificados no negócio, implementados como participantes de um software. Por exemplo: *Entidades: Agenda, Produto, Serviço, Contrato, Consulta, Pagamento, etc;* Atores/Usuários do sistema: *Paciente, Dentista, Secretária, Usuário, Cliente, etc.*

De acordo com Booch (2005),

> Se 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.

Uma classe representa um conjunto de objetos, e não um objeto individual. Pense no conceito de “parede” como uma classe. Objetos desta classe contêm determinadas propriedades comuns, como: altura, largura, espessura, capacidade de

suportar ou não pesos, e, assim, por diante. E, então, considerar instâncias individuais de paredes, como “a parede leste da sala A”, contendo suas distintas propriedades.

8.2. Tipos de Classe

Há dois tipos básicos de classes: elas podem ser concretas ou abstratas. Perceba suas definições e principais diferenças:

- **Uma classe concreta**, em si, implementa seus métodos e permite criação de objetos através dela (instâncias desta classe). A classe concreta não possui métodos abstratos, ou, se os contêm, são métodos abstratos herdados de uma superclasse abstrata.
- **Uma classe abstrata** representa uma entidade ou um conceito não concreto, abstraíndo uma ou mais classes concretas. Por isso, é sempre uma superclasse e não pode ser instanciada. Para deixar claro por meio da especificação se uma classe é abstrata, seu nome é escrito em itálico.

Veja a figura 8.2.1, que contém três classes especificadas em um simples diagrama de classes. São elas: ContaBancária, ContaCorrente e ContaPoupança. Atente-se ao nome da classe ContaBancária, que está denotada em itálico, um sinal de que ela é uma classe abstrata. Enquanto “ContaCorrente” e “ContaPoupanca” estão escritas normalmente, significando que elas são classes concretas.

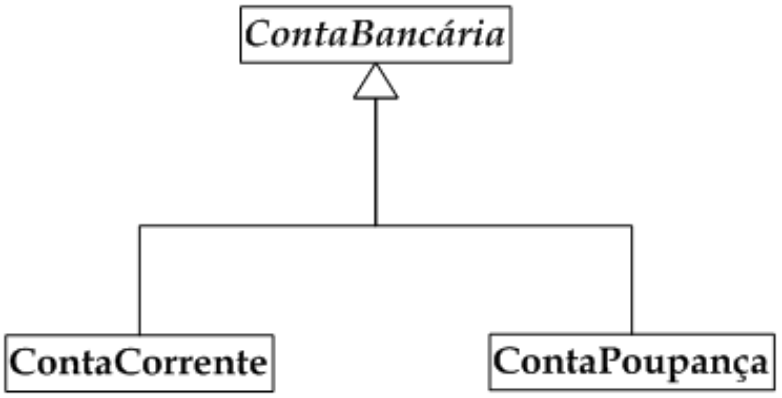

Figura 8.2.1. Classe - Concreta/Abstrata

Assim, a classe ContaBancária não conterá as implementações das possíveis funcionalidades (operações) existentes. São operações das classes ContaCorrente e ContaPoupança que conterão a implementação concreta.

8.3. Especificação

A UML contém recomendações para se especificar bem uma classe. Os detalhes são importantes na especificação da classe para se manter a clareza na sua notação. Note a figura 1.3.1, um exemplo de uma classe com suas propriedades especificadas.

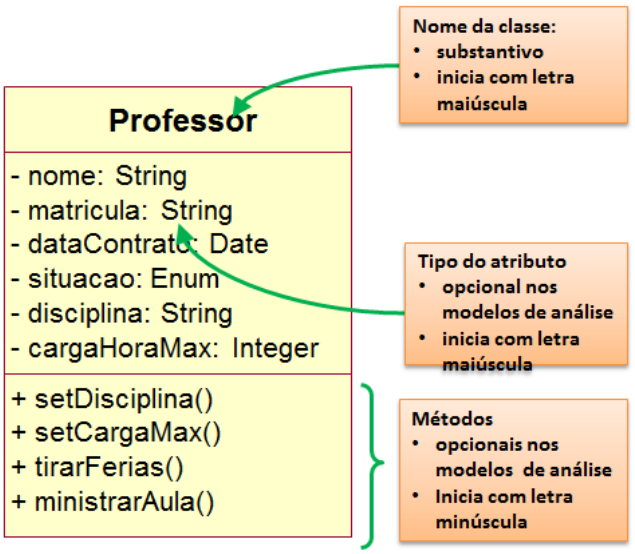

Figura 8.3.1. Classe - Especificação

Uma representação mais completa de uma classe mostra três de suas principais propriedades, e a forma de representá-las contém critérios:

- **Nome da Classe:** Deve refletir uma concepção substantiva, por isso se utiliza um termo substantivo, no padrão *UpperCamelCase*[1], para se nomear classes. Como, por exemplo: *Professor, ContaBancaria, Projeto, NotaFiscal, etc.*
- **Nome de Atributo:** Representa o tipo de informações que podem ser armazenadas pelos objetos instanciados naquela classe. A notação de um atributo é feita utilizando o padrão *lowerCamelCase*, por exemplo: *nome, dataContrato, cargaHoraria, dataEntrega, situacao, dataNescimento, etc.* O nome de um atributo deve ser seguido do seu tipo de dado, que é representado iniciando com letra maiúscula e mostra como a informação será armazenada na classe, por exemplo: *String(caracteres), Integer(números inteiros), Date(datas), Array(vetores/pilhas), etc.*
- **Método (Operação):** Representa uma função que objetos (instâncias da classe) podem utilizar para executar um comportamento no software. Do mesmo modo dos atributos, os métodos devem ser nomeados no padrão *lowerCamelCase*, por exemplo: *inserirProduto, calcularFrete, excluir, ativarProduto, etc*. São os serviços/comportamentos levantados em análises de Casos de Uso.

Os valores dos atributos são as informações que variam individualmente entre os objetos quando instanciados, ou seja, ao operar dados de uma entidade no sistema. Os métodos são os mesmos para os objetos de uma classe específica, e ao representá-los, deve-se focar no objetivo final da operação, logo, não se define as etapas que os métodos percorrerão quando chamados, esta é uma visão especificada em outras notações UML.

[1] CamelCase: Denominação em inglês para a prática de escrever as palavras compostas ou frases, onde cada palavra é iniciada com maiúsculas e unidas, sem espaços.

8.4. Visibilidade

Na especificação de classes, podem-se definir as visibilidades de atributos e métodos, são ferramentas que servem para proteger ou para esconder propriedades de outras classes. Os sinais que representam estas formas de visibilidade nas classes UML são: “+”, “-” e “#”. Tais sinais determinam a propriedade será ou não acessível para outras classes, e se classes herdeiras podem utilizar, ou não, tal propriedade. Conheça três tipos de visibilidades e as definições de cada um:

- **-** (privado): Acessível somente pelo próprio objeto;
- **+** (público): Acessível por quaisquer objetos;
- **#** (protegido): Acessível por objetos de classes herdeiras.

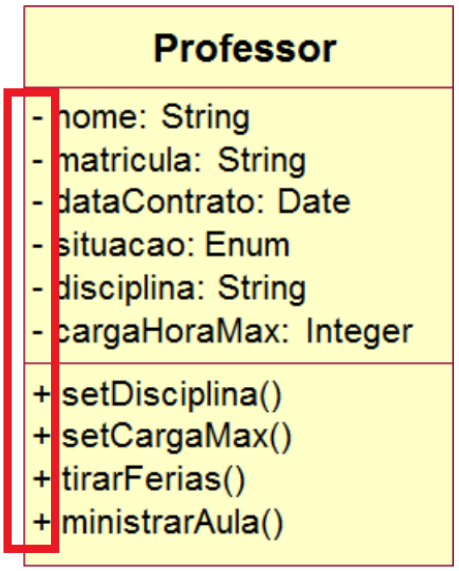

Figura 8.4.1. Classe - Visibilidade

A figura 8.4.1, por exemplo, mostra a classe Professor, que foi modelada de forma que todos os atributos somente serão acessíveis por objetos da própria classe, pois são privados em sua visibilidade. Enquanto seus métodos são todos públicos, ou seja, são acessíveis a partir de objetos de qualquer classe.

8.5. Relacionamentos

As classes são estruturas que comumente necessitam de propriedades de outras classes, então podem se associar. O relacionamento entre classes é um conceito dos mais fundamentais na orientação a objetos. [..] *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.* Book (2005).

Existem cinco tipos de relacionamentos entre classes. Suas formas de representação gráfica podem ser notadas na figura 8.5.1.

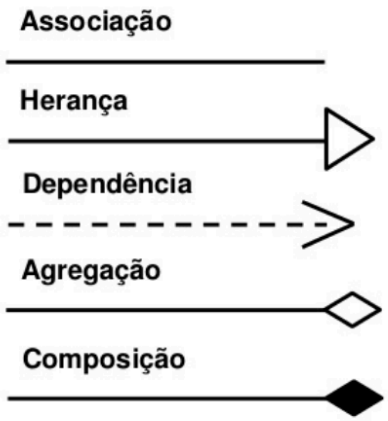

Figura 8.5.1. Relacionamentos entre Classes

Cada um fornece uma forma diferente de combinações e de abstrações. Conheça a definição de cada uma:

- **Associação**: representa relacionamentos estruturais entre objetos, uma notação de associação conectando classes. Significando tais classes estão conceitualmente em um mesmo nível, uma não é mais importante que a outra.
- **Herança(Generalização/Especialização)**: relaciona classes generalizadas (superclasses) às suas especializações (subclasses). As generalizações são lidas como "*é um tipo de*". Exemplo: um item *Cachorro* "é um tipo de" um item mais geral *Mamífero*, que por sua vez "é um tipo de" *Animal*.
- **Dependência**: um relacionamento de utilização entre classes, como: refinamento, rastreamento e/ou vínculos; Usa-se dependência sempre que se deseja indicar que algum item depende de um outro.
- **Agregação**: 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 "contém", um objeto do "todo" contém os objetos das "partes".
- **Composição**: uma forma de agregação mas com tempo de vida coincidente entre "partes" e "todo". Assim, as partes poderão ser criadas após a própria composição do todo. E, uma vez criadas, as partes podem ser "mortas" explicitamente antes da "morte" do todo, mas morrem sempre que "todo" morre.

Entre as associações de agregação e de Composição, há uma forma elementar de diferenciação. A agregação tem identidade própria, tem maior autonomia e é independe da classe agregada. Já na composição, a classe que compõe tem baixa identidade conceitual, e depende totalmente da classe por ela composta para existir. Em ambas a semântica padrão do relacionamento é de "posse", então não deve-se indicar nome nas representações de relacionamentos de agregação ou de composição.

8.5.1. Multiplicidade

Em muitas situações de modelagem, é importante determinar a quantidade de objetos que podem ser conectados pela instância da classe em um relacionamento. Essa "quantidade" é chamada de multiplicidade, um adorno do papel de uma associação, um intervalo de números inteiros que estabelece o tamanho possível do conjunto de objetos relacionados. Pode haver multiplicidades, como: zero ou um (0..1); zero ou muitos (0..*); um ou muitos (1..*), etc. Também, é possível fornecer um intervalo inteiro (como 2..5) de mínimos e máximos, e até determinar o número exato único de multiplicidade, como, por exemplo, "3", (mesmo que 3..3).

Veja a figura 8.5.2, que mostra um exemplo de associação contendo: o nome da associação (*trabalha para*); o sentido de leitura do relacionamento em seu papel (*pessoa trabalha para empresa*); e a multiplicidade do relacionamento entre as duas classes (*uma pessoa pode trabalhar para várias empresas, e uma empresa pode conter uma ou muitas pessoas trabalhando*).

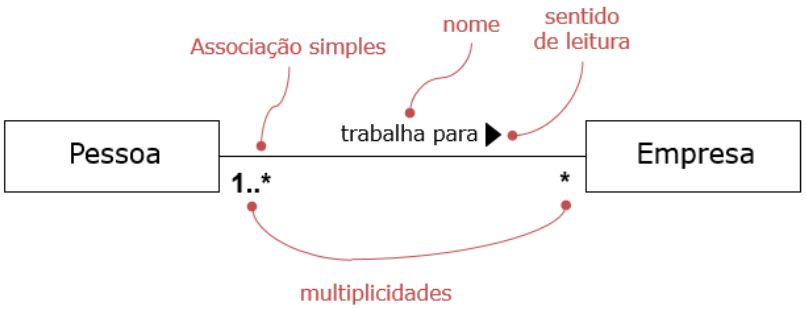

Figura 8.5.2. Adornos de associações

Como você poderá saber quando os objetos de uma determinada classe devem interagir com os objetos de outra classe? A resposta está na análise de casos de uso, onde deve-se considerar cenários estruturais e comportamentais. Assim, sempre que descobrir que duas ou mais classes interagem, especifique sua associação.

Referências

Booch G., Rumbaugh J., Jacobson I. “The Unified Modeling Language user Guide” 2ª Edição. 2005.