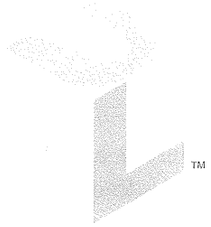

CAPÍTULO 4

Diagrama de Classes

O diagrama de classes é um dos mais importantes e mais utilizados da UML. Seu principal enfoque está em permitir a visualização das classes que comporão o sistema com seus respectivos atributos e métodos, bem como em demonstrar como as classes do diagrama se relacionam, complementam e transmitem informações entre si. Esse diagrama apresenta uma visão estática de como as classes estão organizadas, preocupando-se em como definir a estrutura lógica das mesmas. O diagrama de classes serve ainda como base para a construção da maioria dos outros diagramas da linguagem UML.

Basicamente, o diagrama de classes é composto por suas classes e pelas associações existentes entre elas, ou seja, os relacionamentos entre as classes. Alguns métodos de desenvolvimento de software, como o Processo Unificado, recomendam que se utilize o diagrama de classes ainda durante a fase de análise, produzindo-se um modelo conceitual a respeito das informações necessárias ao software. No modelo conceitual, o engenheiro preocupa-se apenas em representar as informações que o software necessitará, em termos de classes e seus atributos, bem como as associações entre as classes, não modelando características como os métodos que as classes poderão conter nessa etapa (os métodos já fazem parte do "como" o software será desenvolvido). Somente na fase de projeto toma-se o modelo conceitual do diagrama de classes e produz-se o modelo de domínio, que já enfoca a solução do problema. Os métodos necessários às classes são descobertos a partir da modelagem de diagramas de interação, como o diagrama de sequência, que será estudado nos próximos capítulos.

4.1 Atributos e Métodos

Classes costumam ter atributos, que, como já explicado no capítulo 2, armazenam os dados dos objetos da classe, além de métodos, também chamados operações, que são as funções que uma instância da classe pode executar. Os valores dos atributos podem variar de uma instância para outra. Graças a essa

característica, aliás, é possível identificar cada objeto individualmente, ao passo que os métodos são idênticos para todas as instâncias de uma classe específica.

Embora os métodos sejam declarados no diagrama de classes, identificando os possíveis parâmetros que são por eles recebidos e os possíveis valores por eles retornados, o diagrama de classes não se preocupa em definir as etapas que tais métodos deverão percorrer quando forem chamados, sendo esta uma função atribuída a outros diagramas, como o diagrama de atividade, que será analisado nos capítulos seguintes. A figura 4.1 apresenta um exemplo de classe contendo atributos e métodos.

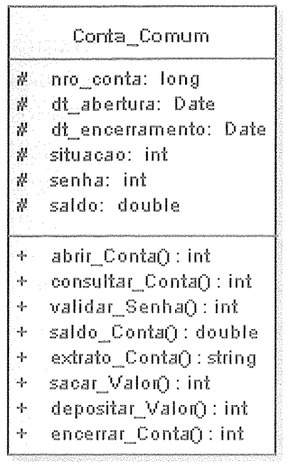

Figura 4.1 – Classe.

Como já foi explicado no capítulo 2, uma classe, na linguagem UML, é representada como um retângulo com até três divisões, descritas a seguir:

- A primeira contém a descrição ou nome da classe, que nesse exemplo é `Conta_Comum`.
- A segunda armazena os atributos e seus tipos de dados (o formato que os dados devem ter para serem armazenados em um atributo). No exemplo da figura 4.1 a classe `Conta_Comum` contém os atributos `nro_conta`, do tipo `long`; `dt_abertura` e `dt_encerramento`, do tipo `Date` (este não é um tipo primitivo e se refere a uma classe); `situacao` e `senha`, do tipo `int`; e `saldo`, do tipo `double`.
- Finalmente, a terceira divisão lista os métodos da classe. Nesse exemplo a classe `Conta_Comum` contém os métodos `abrir_Conta`, `consultar_Conta`, `validar_Senha`, `saldo_Conta`, `extrato_Conta`, `sacar_Valor`, `depositar_Valor` e `encerrar_Conta`.

Os símbolos de sustenido (#) e mais (+) na frente dos atributos e métodos representam a visibilidade dos mesmos, o que determina quais objetos de quais classes podem utilizar o atributo ou o método em questão. Os tipos de visibilidade já foram explicados no capítulo 2.

Não é realmente obrigatório que uma classe apresente as três divisões, pois pode haver classes que não tenham atributos ou que não contenham métodos, ou pode acontecer ainda que seus atributos e métodos não precisem ser apresentados no diagrama, já que é recomendado apresentar apenas atributos relevantes ao diagrama para evitar, por exemplo, tornar o diagrama muito poluído. Assim, é possível encontrar classes com somente duas divisões ou mesmo com apenas uma, no caso, aquela que contém a descrição da classe, porque esta é obrigatória.

Métodos podem receber valores como parâmetros e retornar valores (da mesma forma que as funções implementadas na linguagem de programação C), que podem tanto ser o resultado produzido pela execução do método quanto simplesmente um valor representado se o método foi realizado com sucesso ou não. Nessa classe, por exemplo, podemos perceber que o retorno do método `abrir_Conta` é um `long`, que conterá o número da nova conta gerada. Por sua vez, o retorno dos métodos `consultar_Conta`, `validar_Senha`, `sacar_Valor`, `depositar_Valor` e `encerrar_Conta` é um inteiro (`int`), e esse valor é utilizado para determinar se o método foi concluído com sucesso ou não. Por padrão o retorno 1 significa verdadeiro (ou sucesso) e 0 falso (ou fracasso). Já os métodos `saldo_Conta` e `extrato_Conta` retornam um `double` e uma `String`, que contêm o resultado da execução desses métodos. Para o método `abrir_Conta`, se o valor retornado for igual a 1 significará que o método foi concluído com sucesso e que uma nova conta foi aberta. Já se o retorno for igual a zero, saberemos que o método não foi concluído da forma esperada e que algum problema ocorreu.

Na figura 4.1 os métodos foram apresentados sem o detalhamento de quais argumentos (parâmetros) eles deveriam receber (a lista de argumentos de um método, junto com seu valor de retorno, é chamada de assinatura da operação). Esse detalhamento é opcional, e isso foi feito propositadamente para explicar os diagramas que serão apresentados no decorrer do capítulo. Alguns métodos podem ter muitos parâmetros, e o detalhamento destes em um diagrama que contenha muitas classes tornará esse diagrama muito extenso e será difícil visualizá-lo claramente como um todo, porque as classes ficarão largas. Assim, é melhor, quando se trata de apresentar um diagrama de classes composto por muitas classes, apresentar somente o nome dos métodos da classe, sem especificar os argumentos que ele receberá. O detalhamento dos métodos de cada classe poderá ser feito individualmente em um diagrama separado, conforme apresentado na figura 4.2.

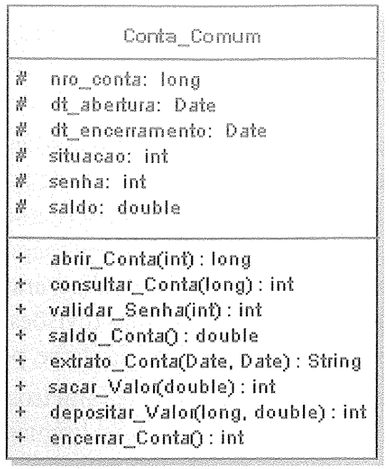

Figura 4.2 – Detalhamento das Assinaturas das Operações.

Ao examinarmos a figura 4.2 podemos perceber que os métodos `abrir_Conta` e `validar_Senha` recebem um inteiro como parâmetro; o método `consultar_Conta`, um `long`; `sacar_Valor`, um `double`; e `depositar_Valor`, um `long` e um `double`. Já os outros métodos não recebem argumento algum. A partir do modelo da classe `conta_Comum` apresentada na figura 4.2 é possível gerar o código correspondente a ele. A seguir, será apresentado o código correspondente a essa classe implementado em Java.

```
public class Conta_Comum {
    protected long nro_conta;
    protected Date dt_abertura;
    protected Date dt_encerramento;
    protected int situacao;
    protected int senha;
    protected double saldo;

    public Conta_Comum(){
    }
    public void finalize() throws Throwable {
    }
    public long abrir_Conta(int senha){
        return 0;
    }
    public int consultar_Conta(long nro_conta){
        return 0;
    }
```



```
    public int validar_Senha(int senha){
        return 0;
    }
    public double saldo_Conta(){
        return 0;
    }
    public String extrato_Conta(){
        return "";
    }
    public int sacar_Valor(double valor){
        return 0;
    }
    public int depositar_Valor(long nro_conta, double valor){
        return 0;
    }
    public int encerrar_Conta(){
        return 0;
    }
}
```

Obviamente esse código é apenas um esqueleto da classe. Como pode-se perceber, não há nenhum detalhamento de como os métodos irão desempenhar sua função. Além disso, será preciso criar uma classe chamada `Date` para que o compilador reconheça os atributos desse tipo. Observe que o valor de retorno em Java é declarado antes do nome do método, enquanto na UML ele é definido depois do nome do método e dos parâmetros que ele receberá.

Os atributos de uma classe podem ainda ter características extras, entre as quais podemos citar valores iniciais, multiplicidade e se o atributo é derivado, ou seja, se seus valores são produzidos por meio de algum tipo de cálculo, conforme apresentado na figura 4.3.

Ao estudarmos esse exemplo podemos verificar que o valor inicial dos atributos `situacao` e `saldo` será respectivamente 1 e 0 quando da instanciação de um objeto dessa classe durante a abertura de uma nova conta. Dessa forma, sempre que uma nova conta for aberta sua situação inicial terá o valor 1 (está ativa), e o seu saldo permanecerá com valor 0 até que um depósito seja realizado.

Pode-se perceber também que o atributo `dt_encerramento`, após a definição de seu tipo (`Date`), contém os valores [0..1]. Isso é chamado multiplicidade e, nesse contexto, significa que existirá no mínimo nenhuma (0) e no máximo uma (1) data de encerramento para a conta, uma vez que a conta pode estar aberta ainda e, portanto, não poderá ter uma data de encerramento e, se ela tiver sido encerrada, não poderá ter mais do que uma.

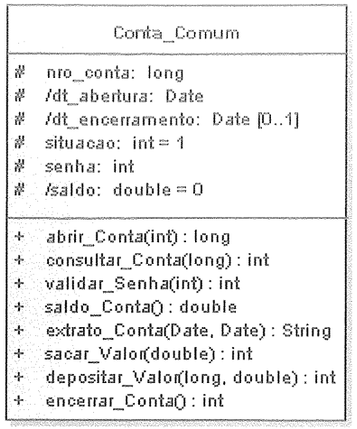

Figura 4.3 – Detalhamento dos Atributos.

Finalmente, os atributos `dt_abertura`, `dt_encerramento` e `saldo` têm uma barra (/) antes de seus nomes, significando que os valores desses atributos sofrem algum tipo de cálculo. No caso das datas, quando for realizada a operação de abertura de conta, o valor da data de abertura será tomado da data do sistema, o mesmo ocorrendo com o valor da data de encerramento quando do encerramento da conta. A rigor, não necessariamente isso poderia ser considerado um cálculo e sim uma simples atribuição, sendo que alguns poderiam considerar isso como sendo o valor inicial dos atributos. No entanto, principalmente no caso da data de encerramento, que será deixada indefinida até que a conta seja encerrada, isso não seria verdadeiro, e o valor desse atributo seria definido em uma operação posterior à criação do objeto. Já no caso do saldo, ele precisa ser recalculado sempre que uma operação de saque ou depósito for realizada.