3. PARADIGMA ORIENTADO A OBJETOS

Este paradigma é o que mais reflete os problemas atuais. Linguagens orientada a objetos (OO) são projetadas para implementar diretamente a abordagem OO na solução dos problemas. Tal abordagem tornou-se uma ferramenta importante para solucionar muitos tipos de problemas através da simulação. Um programa OO consiste em objetos que enviam mensagens uns para os outros. Estes objetos no programa correspondem diretamente aos objetos atuais, tais como pessoas, máquinas, departamentos, documentos e assim por diante [DER 90].

O conceito de programação orientada a objetos teve início na linguagem *Simula 67*, que foi a primeira LP a implementar o conceito de objeto. Porém, estes conceitos foram aperfeiçoados e refinados na linguagem *Smalltalk 80*, que continua a servir como protótipo de implementação de modelos OO. Apesar de existirem outras linguagens OO hoje em dia, tais como *Eiffel* e *Java*, muitos ainda consideram que *Smalltalk* é a única LP puramente orientada a objetos. A capacidade de trabalhar com orientação a objetos é freqüentemente adicionada às linguagens imperativas para formar uma linguagem que fornece muitas das características do modelo OO. A mais popular destas linguagens híbridas é o *C++* [DER 90, SEB 99].

Neste capítulo são apresentados os principais conceitos e características da programação orientada a objetos. Na seção 3.1 as propriedades e os componentes deste modelo são introduzidos, e na seqüência classes e objetos são definidos. As seções seguintes descrevem hierarquia de classes, polimorfismo, método e mensagem.

3.1. Introdução

Linguagens orientadas a procedimentos, que foi o paradigma de programação que mais se desenvolveu em 1970, possuem como foco principal os subprogramas e bibliotecas de subprogramas. Neste caso, dados são enviados para serem processados nos subprogramas. Por exemplo, um vetor de valores inteiros que precisam ser ordenados é enviado como parâmetro para um subprograma que ordena-o.

Programação "baseada em dados" tem como foco os tipos abstratos de dados. Neste paradigma, o processamento de um objeto de dado é especificado através da chamada de subprogramas associados com o objeto. Se um objeto vetor necessita ser ordenado, a operação de ordenação é definida no tipo abstrato de dado para o vetor. O processo de ordenação é realizado através da chamada desta operação no objeto vetor específico [SEB 99].

O paradigma de programação OO tornou-se popular apenas na década de 80. Uma linguagem OO deve fornecer suporte para três características “chave”: tipos abstratos de dados, herança e um tipo particular de amarração dinâmica. Um modelo OO tem como entidade fundamental o “objeto”, que recebe e envia mensagens, executa processamentos e possui um estado local que ele pode modificar. Problemas são resolvidos através de objetos que enviam mensagens uns para os outros. Assim, pode-se dizer que um modelo OO é formado por quatro componentes básicos: objetos, mensagens, métodos e classes, descritos a seguir [DER 90, SEB 99].

- **Objeto**: Um objeto consiste em um conjunto de operações encapsuladas (métodos) e um “estado” (determinado pelo valor dos atributos) que grava e recupera os efeitos destas operações. Em outras palavras um objeto possui tudo o que é necessário para conhecer a si próprio. Um objeto executa uma operação em resposta a uma mensagem recebida que é associada com aquela operação. O resultado da operação depende tanto do conteúdo da mensagem recebida como do estado do objeto quando ele recebe a mensagem. Um objeto também pode, como parte da sua operação, enviar mensagens para outros objetos e para si mesmo. Para ilustrar, pode-se dizer que objetos representam uma coleção de dados relacionados com um tema em comum, como exemplificado na figura 3.1.

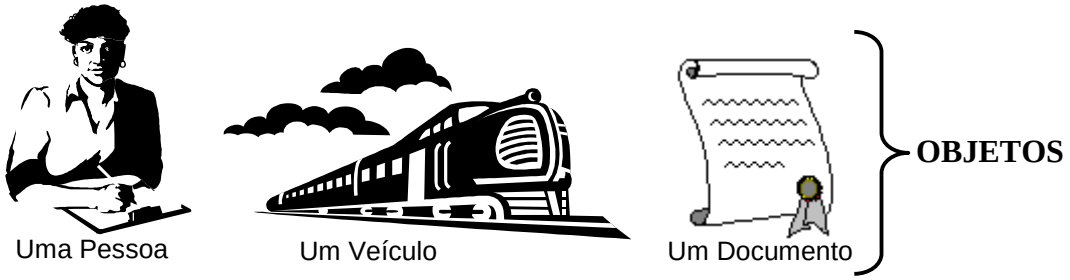

Figura 3.1 – Exemplos de objetos

- **Mensagem**: Mensagens são requisições enviadas de um objeto para outro, para que o objeto "receptor" forneça algum resultado desejado através da execução de uma operação. A natureza das operações realizadas para alcançar o resultado requerido é determinada pelo objeto "receptor". Mensagens também podem ser acompanhadas por parâmetros que são aceitos pelo objeto "receptor" e podem ter algum efeito nas operações realizadas. Estes parâmetros são os próprios objetos. Resumindo, trata-se de um ciclo completo onde uma mensagem é enviada a um objeto, operações são executadas dentro desse e uma mensagem contendo o resultado da operação é enviada ao objeto solicitante. Funciona como uma fábrica ilustrada na figura 3.2, que recebe uma ordem de produção (mensagem de solicitação), processa essa ordem (operações) utilizando matéria-prima (atributos) e gera um produto final (mensagem de resposta).

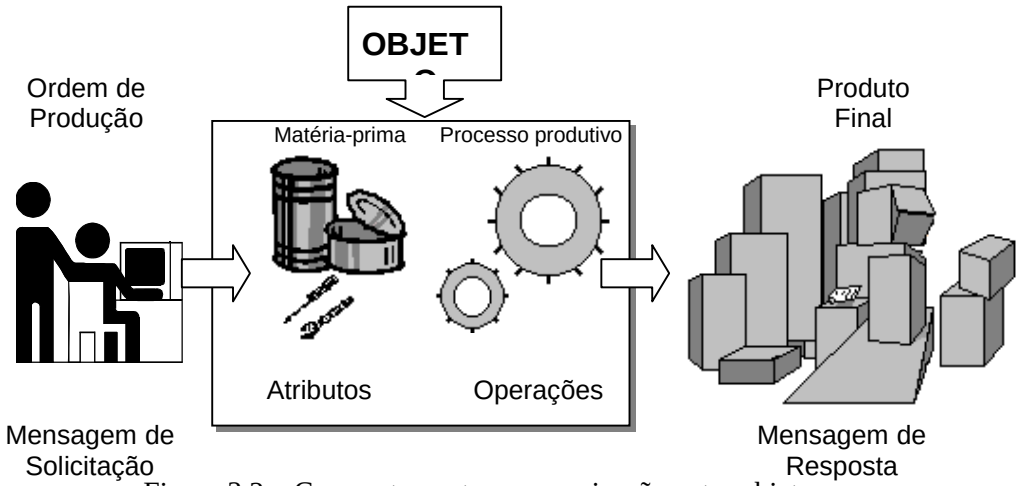

Figura 3.2 – Comportamento e comunicação entre objetos

- **Métodos**: Métodos são similares a procedimentos e funções e consistem nas descrições das operações que um objeto executa quando recebe uma mensagem. Há, portanto, uma correspondência um para um entre mensagens e métodos que são executados quando a mensagem é recebida através de um dado objeto. A mesma mensagem pode resultar em métodos diferentes quando for enviada para objetos diferentes. O método associado com a mensagem é pré-definido.
- **Atributos**: Um atributo consiste em um dado ou informação de estado, para o qual cada objeto de uma classe tem seu próprio valor. Existem dois tipos de atributos em um sistema orientado a objetos: os atributos de objetos e os atributos de classes. O primeiro, trata-se dos atributos que descrevem valores (estados) mantidos em um objeto. Diferentes objetos de uma mesma classe não compartilham os atributos de objetos, ou seja, cada um possui sua própria cópia do atributo. Os atributos de classe são aqueles cujo valor todos os seus objetos devem compartilhar.
- **Classes**: Uma classe define as características de uma coleção de objetos. Classes consistem em descrições de métodos e atributos que objetos que pertencem à classe irão possuir. Uma classe é similar a um tipo abstrato de dado no sentido que ela define uma estrutura interna e um conjunto de operações que todos os objetos que são instâncias daquela classe irão possuir. Desta forma é desnecessário redefinir cada objeto que possui as mesmas propriedades. Uma classe é também um próprio objeto e por isso pode aceitar mensagens e ter métodos e um estado interno. Em outras palavras uma classe representa uma idéia ou um conceito simples e categoriza objetos que possuem propriedades similares.

- **Encapsulamento**: Cada objeto é visto como o encapsulamento do seu estado interno, suas mensagens e seus métodos. A estrutura do estado e dos pares "mensagem - método" são todas definidas através da classe à qual o objeto pertence. O valor atual do estado interno é determinado através dos métodos que o objeto executa em resposta às mensagens recebidas. Encapsulamento constitui-se no "coração" da programação orientada a objetos, a união da ação e da informação em novos tipos de dados que "sabem" o que supostamente devem fazer, separando, assim, um serviço da sua implementação.
- **Polimorfismo**: É a propriedade que permite que a mesma mensagem seja enviada a diferentes objetos e que cada objeto execute a operação que é apropriada à sua classe. Mais importante, o objeto "emissor" não precisa conhecer a classe do objeto "receptor" e como este objeto irá responder à mensagem. Isto significa que, por exemplo, a mensagem *print* pode ser enviada para um objeto sem a necessidade de saber se o objeto é um caracter, um inteiro ou um *string*. O objeto "receptor" responde com o método que é apropriado à classe a qual ele pertence.
- **Herança**: É uma das propriedades mais importantes do modelo de orientação a objetos. A herança é permitida através da definição de uma hierarquia de classes, isto é, uma "árvore" de classes onde cada uma possui zero ou mais subclasses. Herança refere-se ao fato de que uma subclasse herda todos os componentes da classe pai, incluindo uma estrutura de estado interna e pares "mensagem - método". Qualquer propriedade herdada pode ser redefinida na definição da subclasse, substituindo a definição herdada. Mas, componentes que permanecem iguais não requerem redefinição. Esta propriedade encoraja a definição de novas classes sem uma grande duplicação do código [DER 90].

Para ilustrar, a tabela abaixo contém um resumo e um exemplo de todos estes conceitos.

Palavra-Chave	Breve Definição	Exemplo
Classe	Agrupamento de objetos similares que apresentam os mesmos atributos e operações	Indivíduo, caracterizando as pessoas do mundo
Atributo	Característica particular de uma ocorrência da classe	Indivíduo possui nome, sexo, data de nascimento
Operações	Lógica contida em uma classe para designar-lhe um comportamento	Cálculo da idade de uma pessoa em uma classe (Indivíduo)
Encapsulamento	Combinação de atributos e operações de uma classe	Atributo: data de nascimento Operação: cálculo da idade
Herança	Compartilhamento pela subclasse dos atributos e operações da classe pai	Subclasse (Eucalipto) compartilha atributos e operações da classe (Árvore)
Subclasse	Característica particular de uma classe	Classe (Árvore) → Subclasses (Ipê, Eucalipto, Jacarandá, etc.)
Instância de Classe	Uma ocorrência específica de uma classe. É o mesmo que objeto	Uma pessoa, uma organização ou um equipamento
Objeto	Elemento do mundo real (natureza). Sinônimo de instância de classe	Pessoa "Fulano de Tal", Organização "ACM", Equipamento "Extintor"
Mensagem	Uma solicitação entre objetos para invocar certa operação	Informar idade da pessoa "Fulano de Tal"
Polimorfismo	Habilidade para usar a mesma mensagem para invocar comportamentos diferentes do objeto	Chamada da operação: "Calcular Saldo" de correntista. Invoca as derivações correspondentes para cálculo de saldo de poupança, renda fixa, etc.

Como pode-se observar, o encapsulamento é a base de toda a abordagem, pois ele contribui para a redução dos efeitos das mudanças, colocando uma "parede" de código em volta de cada parte dos dados. Todo o acesso aos dados é manipulado por métodos que foram colocados para mediar o acesso aos dados. Em outras palavras, como a figura 3.4 ilustra, na programação convencional, os dados são passados para funções, as quais podem devolver dados novos ou modificados. Na programação orientada a objetos, uma classe encapsula (inclui) atributos e métodos, tornando sua inter-relação parte da estrutura do programa [SWA 93].

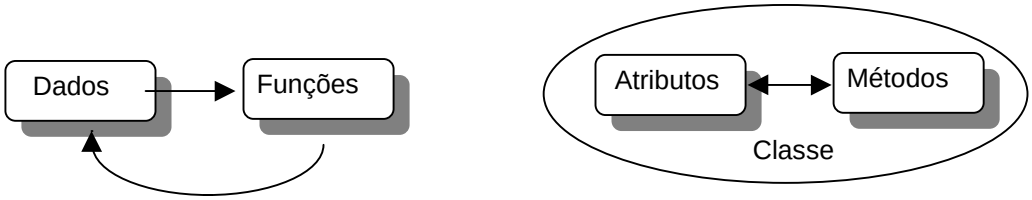

Figura 3.4 – Programação Convencional X Programação Baseada em Objetos

Entre os principais objetivos das LP OO, pode-se destacar a reutilização de código e a restrição de acesso a detalhes dos componentes do *software*. A restrição de acesso é necessária para garantir que os clientes usem os componentes de tal maneira que fiquem independentes dos seus detalhes de implementação e que qualquer alteração na implementação do componente terá apenas um efeito local.

Entretanto, deve-se atentar para o fato de que estes dois objetivos de restrição de acesso e modificação do código para reutilização podem ser, algumas vezes, mutuamente incompatíveis. Por exemplo, para adicionar operações em uma pilha, é necessário acessar detalhes da implementação interna da estrutura de dados da pilha. Restrição de acesso também pode ser complementar à necessidade de modificação, desde que força as operações a serem definidas de uma forma abstrata, que facilita a modificação da sua implementação sem trocar a definição.

O controle sobre o acesso pode ter muitas formas. Uma maneira consiste em listar explicitamente todas as operações e dados que são acessíveis a clientes em uma lista de exportação. A segunda, consiste em listar as propriedades de um componente que são inacessíveis em uma parte privada, como em *Ada*. O controle mais explícito de todos, consiste em declarar explicitamente tanto os membros públicos como os privados.

Também é possível declarar que elementos de um componente sejam parcialmente acessíveis, isto é, acessíveis à alguns, mas não a todos os componentes externos. Isto pode ser feito tanto explicitamente, nomeando os componentes externos, ou implicitamente, permitindo acesso a um grupo de componentes. Da mesma forma, pode-se requisitar que os clientes declarem explicitamente as utilizações de um componente através de uma lista de importação, como em *Modula 2*, ou eles podem estar disponíveis para usar um componente implicitamente, que deve ser localizado automaticamente pela linguagem ou através de instruções para um verificador de interfaces.

Mecanismos para restrição de acesso a detalhes internos é conhecido por vários nomes e é determinado através de especificadores de acesso, que podem ser diferentes em cada LP. Alguns autores chamam de mecanismos de encapsulamento, enquanto outros referem-se a eles como mecanismos que escondem a informação. Outro nome, aqui utilizado, é **mecanismos de proteção** [LOU 93].

Concluindo, com base no que já foi apresentado, pode-se observar que o enfoque tradicional de modelagem para a construção de sistemas baseia-se na compreensão desse sistema como um conjunto de programas que, por sua vez, executam processos sobre dados. Já o enfoque de modelagem por objetos, como ilustrado na figura 3.5, vê o mundo como uma coletânea de objetos que interagem entre si, apresentam características próprias que são representadas pelos seus atributos (dados) e operações (processos). Resumidamente, então, a mudança de enfoque é justificada pelo fato de que objetos existem na natureza muito antes de haver qualquer tipo de aplicação: equipamento, pessoas, minerais, petróleo, etc., existem por si só, apresentam características peculiares representadas pelos seus atributos e pelo seu comportamento no mundo real [FUR 98].

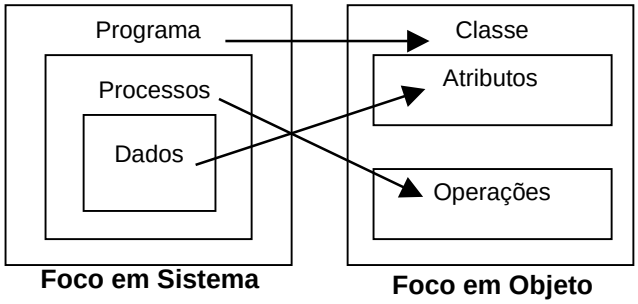

Figura 3.5 –Enfoque baseado em sistema X Enfoque baseado em objeto

É interessante comentar que, para uma melhor visualização e entendimento da hierarquia de classes, normalmente utiliza-se uma notação gráfica para a sua representação. Ente as várias notações disponíveis para representação de sistemas orientados a objetos, pode-se citar *Booch*, OMT e UML [FUR 98]. Para exemplificar, a metodologia OMT - *Object Modeling Technique* [RUM 91] será brevemente descrita. O modelo de objetos desta metodologia descreve os objetos no sistema e seus relacionamentos da seguinte maneira: as classes são representadas por retângulos divididos em três partes, sendo que na parte superior é colocado o nome da classe, na parte intermediária os seus atributos e na parte inferior as operações suportadas pela classe (Figura 3.6). As subclasses são representadas por uma divisão em níveis, a partir de um triângulo que representa a generalização, ou a partir de um losango que representa uma agregação.

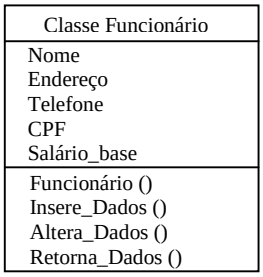

Figura 3.6 – Exemplo da definição de uma classe usando notação gráfica

Para finalizar, os conceitos de programação orientada a objetos descritos até agora são ilustrados no programa apresentado a seguir, escrito na LP *C++*.

```
class Pessoa {                      // Definição de uma superclasse
    private:                        // Mecanismo de restrição de acesso
        char nome[31];              //  \
        char endereco[40];          //  / Atributos da classe
    public :                        // Mecanismo de restrição de acesso
        void setar(void);           //  \
        void mostrar(void);         //   > Métodos da classe
        ~Pessoa(void);              //  /
};
class Funcionario : public Pessoa { // Definição de uma subclasse
    private:                        // Mecanismo de restrição de acesso
        int faltas;                 //  \
        date data_admissao;         //   > Atributos da classe
        float salario_base;         //  /
    public :                        // Mecanismo de restrição de acesso
        Funcionario(void);          //  \
        void setar(void);           //   \
        void mostrar(void);         //   / Métodos da classe
        ~Funcionario(void);         //  /
};
```

```
main () {
   clrscr();
   Funcionario func;                    // Declaração de um objeto
   func.setar();
   func.mostrar();
}
void Pessoa :: setar (void) {
   char newline;
   cout << "\n Digite o Nome: ";
   cin.get(nome,30,'\n');
   cin.get(newline);
   cout << "\n Digite o Endereco: ";
   cin.get(endereco,40,'\n');
   cin.get(newline);
}
void Pessoa :: mostrar(void) {
      cout << "\n Nome: " << nome  << "\n";
      cout << "\n Endereco: " << endereco  << "\n";
}
Pessoa::~Pessoa(void) {      // Destrutor
      cout << "\n Liberando a Memoria Alocada para Pessoa";
}

void Funcionario :: setar (void) {
      char newline;
      cout << "ENTRADA COM OS DADOS DO FUNCIONARIO \n";
      Pessoa::setar();
      cout << "\n Digite o Salario Base : ";
      cin  >> salario_base;
}
void Funcionario :: mostrar(void) {
      cout << "\n\n SAIDA DE DADOS DO FUNCIONARIO \n";
      Pessoa::mostrar();
      cout << "\n Salario Base: " << salario_base  << "\n";
      cout << "\n Faltas no Periodo: " << faltas << "\n";
      cout << "\n Data de Admissao: " << (int) data_admissao.da_day << '\\' << (int) data_admissao.da_mon
           << '\\' << data_admissao.da_year << "\n";
      getch();
}
Funcionario::~Funcionario(void) {    // Destrutor
      cout << "\nLiberando a Memoria Alocada para Funcionário";
}
Funcionario::Funcionario(void) {     // Construtor
      faltas = 0; getdate(&data_admissao);
}
```