CAPÍTULO

2 Paradigmas de programação

Um paradigma de programação está intimamente relacionado à forma de pensar do programador e como ele busca a solução para os problemas. É o paradigma que permite ou proíbe a utilização de algumas técnicas de programação. Ele é capaz, ainda, de mostrar como o programador analisou e abstraiu o problema a resolver. Existem vários paradigmas de programação: estruturado, orientado a objetos, lógico, funcional, dentre outros. Vamos analisar com mais detalhe os paradigmas estruturado e orientado a objetos.

Pelo *paradigma estruturado* (também conhecido como *imperativo* ou *procedural*), qualquer problema pode ser quebrado em problemas menores, de mais fácil solução, chamados de sub-rotinas ou funções. Cada sub-rotina ou função pode receber valores de entrada, submetê-los a um processo capaz de gerar um valor de saída para quem fez uso da sub-rotina ou função. O paradigma estruturado preconiza, ainda, que todo processamento pode ser realizado pelo uso de três tipos de estrutura: sequencial, condicional e iterativa (de repetição).

Já o *paradigma orientado a objetos* compreende o problema como uma coleção de objetos interagindo por meio de trocas de mensagem. Os objetos são estruturas de dados contendo estado (dados) e comportamento (lógica). Dessa maneira, um conjunto de objetos com informações comuns e com o mesmo comportamento dá origem a uma classe.

Além disso, um programador que utilize o paradigma estruturado analisa o problema tentando relacionar as ações que deverão ser executadas e como poderão ser subdivididas em módulos. Um programador que utilize o paradigma orientado a objetos analisa o mesmo problema tentando identificar os objetos que compõem essa realidade e como eles interagem entre si.

É importante destacar que o paradigma de programação está ligado à forma de pensar do programador. É possível ver, por exemplo, que o uso de uma linguagem com suporte nativo à orientação a objetos não implica, necessariamente, a criação de uma aplicação orientada a objetos. Também, podem-se encontrar facilmente soluções não estruturadas, construídas a partir de linguagens de programação com suporte à estruturação.

Verificaremos, agora, por meio de um exemplo, a aplicação dos dois paradigmas na resolução de um mesmo problema no qual devemos calcular a área e o perímetro de um retângulo. Para isso, deverá existir uma interface com o usuário pela qual serão informadas as medidas dos lados do retângulo e visualizado o resultado de cada cálculo realizado. Trata-se de um problema simples, mas como resolvê-lo?

Segundo o paradigma estruturado, devemos detalhar as ações necessárias para chegar à resposta desejada. Nesse sentido, devemos:

1. Obter o valor da altura do retângulo.
2. Obter o valor da largura do retângulo.
3. Calcular a área.
4. Calcular o perímetro.
5. Mostrar os cálculos realizados.

Posteriormente, devemos analisar a melhor forma de modularizar a solução. A ideia é que cada módulo realize uma tarefa bem específica, maximizando, assim, a possibilidade de ser reutilizado.

Cada módulo poderá receber valores e, também, devolver um valor a quem o solicitou. Nesse exemplo, nossa solução será composta por quatro módulos: o principal, pelo qual a execução começará; o calculaArea, responsável por calcular e devolver o valor da área do retângulo; o calculaPerimetro, responsável por calcular e devolver o valor do perímetro do retângulo; e o mostrarMensagem, responsável por mandar para a interface com o usuário o resultado dos cálculos realizados.

Em PASCAL, usando o paradigma estruturado, a solução ficaria da seguinte forma:

```
program exemplo;
uses crt;
var altura, largura, area, perimetro: real;

function calculaArea(a, b:real):real;
begin
calculaArea := a * b;
end;

function calculaPerimetro(a, b:real):real;
begin
calculaPerimetro := 2*a + 2*b;
end;

procedure mostraMensagem(msg: string; vlr:real);
begin
writeln(msg, vlr:5:2);
end;

begin
clrscr;
write('Digite o valor da altura do retângulo: ');
readln(altura);
write('Digite o valor da largura do retângulo: ');
readln(largura);
area := calculaArea(altura, largura);
perimetro := calculaPerimetro(altura, largura);
mostraMensagem('O valor da área , ', area);
mostraMensagem('O valor do perímetro , ', perimetro);
readln;
end.
```

Em JAVA, usando o paradigma estruturado, a solução ficaria como se segue:

```
import java.io.*;
import java.util.*;
class Retangulo {
  public static void main(String[] args) {
    float altura, largura, area, perimetro;
    Scanner entrada;
    entrada = new Scanner(System.in);
    System.out.print("Digite o valor da altura do retângulo: ");
    altura = entrada.nextFloat();
    System.out.print("Digite o valor da largura do retângulo: ");
    largura = entrada.nextFloat();
    area = calculaArea(altura, largura);
    perimetro = calculaPerimeto(altura, largura);
    mostraMensagem("O valor da área é ", area);
    mostraMensagem("O valor do perímetro é ", perimetro);
```

```
  }
  public static float calculaArea(float a, float b) {
  return a * b;
  }
  public static float calculaPerimetro(float a, float b) {
    return 2 *a + 2* b;
  }
  public static void mostraMensagem(String msg, float vlr) {
    System.out.println(msg+vlr);
  }
}
```

Em C++, fazendo uso do paradigma estruturado, a solução ficaria assim:

```
#include <stdio.h>
float calculaArea(float a, float b)
{ return a*b;
}
float calculaPerimetro(float a, float b)
{ return 2*a + 2*b;
}
void mostraMensagem(char *msg, float vlr)
{ printf("%s %5.2f", msg, vlr);
 }
int main()
{ float altura, largura, area, perimetro;
  printf("Digite o valor da altura do retângulo: ");
  scanf("%f%*c", &altura);
  printf("Digite o valor da largura do retângulo: ");
  scanf("%f%*c", &largura);
  area = calculaArea(altura, largura);
  perimetro = calculaPerimetro(altura, largura);
  mostraMensagem("O valor da área é ", area);
  mostraMensagem("O valor do perímetro é ", perimetro);
  return 0;
 }
```

Mesmo sendo um exemplo bastante simples, pode-se ver que a sub-rotina mostrarMensagem, presente nas três versões, é posta em execução por duas vezes, variando-se os parâmetros informados. Isso representa uma das bases do paradigma estruturado, em que o problema é quebrado em pequenos módulos, para aumentar a capacidade de ser reutilizado (ou seja, escreve-se uma vez e utiliza-se quantas vezes forem necessárias).

O paradigma orientado a objetos afirma que a solução de qualquer problema pode ser encontrada por meio das seguintes etapas:

1. Procurar por objetos existentes no problema.
2. Determinar as características e responsabilidades de cada objeto.
3. Estabelecer como ocorrerá a interação entre os objetos identificados.

Assim, pelo que foi apresentado e analisado no exemplo, observamos a existência de dois objetos: o retângulo e a interface com o usuário.

O objeto retângulo tem a obrigação de armazenar e manipular o valor da altura e da largura, além de calcular a área e o perímetro.

A janela tem a obrigação de receber os valores iniciais (altura e largura) e enviá-los para o retângulo. Depois disso, deve solicitar os valores da área e do perímetro ao objeto retângulo para mostrá-los.

A comunicação entre os objetos janela e retângulo é conhecida como troca de mensagens.
Em PASCAL, usando o paradigma orientado a objetos, a solução ficaria da seguinte forma:

```
program exemplo;
uses crt;
type retangulo = object
       altura, largura: real;
       procedure Inicializar_altura;
       procedure Inicializar_largura;
       function CalculaArea(alt, lar: real):real;
       function CalculaPerimetro(alt, lar: real):real;
       procedure MostrarMensagem(texto: string; valor:real);
       end;
procedure retangulo.Inicializar_altura;
begin
 writeln('Digite o valor da altura do retângulo:');
 readln(altura);
end;

procedure retangulo.Inicializar_largura;
begin
 writeln('Digite o valor da largura do retângulo:');
 readln(largura);
end;
function retangulo.CalculaArea(alt,lar: real):real;
begin
 CalculaArea := alt * lar;
end;

function retangulo.CalculaPerimetro(alt,lar: real):real;
begin
 CalculaPerimetro := 2*alt + 2*lar;
end;

procedure retangulo.MostrarMensagem(texto: string; valor: real);
begin
 writeln(texto,valor:5:2);
end;

var ret: retangulo; {instanciando um objeto da classe retangulo}

begin { programa principal}
 clrscr;
 ret.Inicializar_altura;
 ret.Inicializar_largura;
 ret.MostrarMensagem('O valor da área = ',ret.CalculaArea(ret.altura, ret.largura));
 ret.MostrarMensagem('O valor do perímetro = ',ret.CalculaPerimetro(
 ret.altura,ret.largura));
 readln;
 end.
```

Em C++, usando o paradigma orientado a objetos, a solução utilizaria dois arquivos: um com a extensão .hpp, correspondendo à definição da classe Retangulo, e outro correspondendo classe Janela seria representado por um arquivo com a extensão .cpp.

Arquivo **Retangulo.hpp**

```
class Retangulo
{ private:
   float altura;
   float largura;
    public:
        Retangulo()
        { altura = 0;
          largura = 0;
        }
        float getAltura()
        { return altura;
        }
        void setAltura(float a)
        { altura = a;
        }
        float getLargura()
        { return largura;
        }
        void setLargura(float l)
        { largura = l;
        }
        float calculaArea()
        { return altura * largura;
        }
        float calculaPerimetro()
        { return 2*altura + 2*largura;
        }
};
```

Arquivo **Janela.cpp**

```
#include <stdio.h>
#include <iostream>
#include "Retangulo.hpp"
using namespace std;
void mostraMensagem(string msg, float vlr)
{
 cout << msg << vlr << "\n";
}
int main()

{ Retangulo r;
  float altura, largura, area, perimetro;
  printf("Digite o valor da altura do retângulo: ");
  scanf("%f%*c", &altura);
  printf("Digite o valor da largura do retângulo: ");
  scanf("%f%*c", &largura);
  r.setAltura(altura);
  r.setLargura(largura);
  mostraMensagem("O valor da área é ", r.calculaArea());
  mostraMensagem("O valor do perímetro é ", r.calculaPerimetro());
  return 0;
}
```

Em JAVA, usando o paradigma orientado a objetos, a solução ficaria conforme descrito a seguir.

Um arquivo Retangulo.java, contendo todos os atributos e comportamentos esperados em um retângulo qualquer.

```
public class Retangulo
{ private float altura;
  private float largura;
public Retangulo() {
  altura = 0; largura = 0;
}
public float getAltura() {
  return altura;
}
public void setAltura(float a) {
  altura = a;
}
public float getLargura() {
  return largura;
}
public void setLargura(float l) {
  largura = l;
}
public float calculaArea() {
  return altura * largura;
}
public float calculaPerimetro() {
  return 2*altura + 2*largura;
}
}
```

Um arquivo chamado Janela.java, contendo o que se percebeu como necessário para a interface gráfica com o usuário.

```
import java.io.*;
import java.util.*;
class Janela {
  public static void main(String[] args) {
  Retangulo r;
  r = new Retangulo();
  float altura, largura, area, perimetro;
  Scanner entrada;
  entrada = new Scanner(System.in);
  System.out.print("Digite o valor da altura do retângulo: ");
  r.setAltura(entrada.nextFloat());
  System.out.println("Digite o valor da largura do retângulo: ");
  r.setLargura(entrada.nextFloat());
  mostraMensagem("O valor da área é ", r.calculaArea());
  mostraMensagem("O valor do perímetro é ", r.calculaPerimetro());
  }
  public static void mostraMensagem(String msg, float vlr) {
  System.out.println(msg+vlr);
  }
}
```

Cada linguagem de programação atende a pelo menos um paradigma. Alguns autores consideram que qualquer paradigma pode ser implementado em qualquer linguagem (inclusive *assembly*), pois depende apenas da forma de pensar do programador e de sua habilidade de programar. De forma inversa, também se pode afirmar que o fato de a linguagem dar suporte nativo a determinado paradigma não significa que ele foi utilizado.

Além disso, deve-se observar que o paradigma orientado a objetos não exclui o estruturado, pelo contrário, eles trabalham juntos, uma vez que toda a lógica embutida nos objetos segue o pensamento estruturado.

Por fim, uma observação importantíssima precisa ser feita neste momento: este livro não se destina ao estudo da orientação a objetos nem tem a pretensão de discutir vantagens e desvantagens dos diferentes paradigmas. Temos como objetivo o desenvolvimento da lógica em programadores iniciantes, junto com a utilização das estruturas de controle, das estruturas de dados e dos recursos de modularização disponíveis nas linguagens PASCAL, C/C++ e JAVA. Por isso, grande parte das soluções aqui apresentadas segue o paradigma estruturado. Nos capítulos finais deste livro, serão abordados alguns conceitos do paradigma orientado a objetos.