Capítulo 2

Introdução a Orientação a Objetos

Há algo mais importante que a lógica: a imaginação.
Alfred Hitchcock, cineasta, mestre dos filmes de suspense.

Este capítulo descreve tópicos relevantes que apresentam a idéia inicial sobre o paradigma de programação orientada a objetos a partir da programação procedural e estruturada. A idéia é que o aluno entenda a orientação a objetos como uma nova forma de modelagem tomando como base conhecimentos já adquiridos em cursos anteriores como de Algoritmos e Lógica de Programação e de Estrutura de Dados. Desta forma, espera-se que o aluno possa compreender melhor os conceitos iniciais do paradigma de programação orientada a objetos.

2.1 Mudança de Paradigma

O termo *paradigma de programação* é utilizado no sentido de estilo ou visão da forma em que o programador irá adotar para estruturação e execução de um programa. Desta forma, o paradigma de programação está relacionado com a forma em que o programador irá utilizar para desenvolver seus códigos.

A programação procedural e estruturada é um paradigma de programação que envolve técnicas nas quais os programadores tentam utilizar estruturas para representar as informações e sub-rotinas (funções ou procedimentos) em seus programas. Em grande parte dos cursos introdutórios de programação, os alunos aprendem a programar por meio deste paradigma.

Problema 2.1 : *Escreva um programa que realize a leitura de informações sobre um cartão de crédito: número* [1]*, bandeira (Visa, Mastercard, American Express) e saldo inicial (total da dívida). Sabe-se que uma certa quantidade de bônus (pontuação) é acumulada a cada compra conforme a regra da Tabela 2.1. Faça com que seu programa realize uma compra de R$ 100,00 e que imprima as informações do cartão.*

Bandeira do Cartão	Pontos por Compra
VISA	1
MASTERCARD	2
AMERICAN EXPRESS	5

Tabela 2.1: Regra de compras do cartão

Uma implementação simples seguindo o paradigma não estruturado para o Problema 2.1, é apresentada conforme o Exemplo 2.1.

Nas linhas 1, 2, 3 e 4 são definidas as variáveis. A leitura das informações e inicialização do bônus é realizada nas linhas 6, 7, 8 e 9. A ação de compra é caracterizada por uma simples soma do valor com o saldo na linha 11 e o acumulo de bônus pelas linhas 12 até 17, conforme as condições descritas na Tabela 2.2.1. As linhas 19 até 27, realizam a impressão das informações do cartão.

A implementação apresentada pelo Exemplo 2.1 é pequena e simples consistindo apenas de um programa principal no qual utiliza-se de uma seqüência de comandos ou declarações que modificam os dados necessários para a solução do problema. Apesar de estar correta, este estilo de programação, não estruturada, tem várias desvantagens no caso de programas grandes. Por exemplo, se a mesma seqüência é necessária em localizações diferentes, logo ela estará replicada em diversas partes. Além disso, o programa não está organizado de uma forma limpa e clara, tornando mais complexo seu entendimento.

[1] Um número de cartão é composto por diversos dígitos. Para simplificação, adotaremos um valor do tipo inteiro sem levar em uma descreve uma ação que soluciona uma parte do consideração a quantidade de dígitos.

```
01  int numCartao, bonus;
02  String bandeira;
03  float saldo;
04  Scanner leitor = new Scanner(System.in);
05
06  numCartao = leitor.nextInt();
07  bandeira = leitor.nextLine();
08  saldo = leitor.nextFloat();
09  bonus = 0;
10
11  saldo = saldo + 100.0;
12  if(bandeira.equals("visa"))
13    bonus++;
14  if(bandeira.equals("master"))
15    bonus = bonus + 2;
16  if(bandeira.equals("american"))
17    bonus = bonus + 5;
18
19  System.out.println("No Cartão:"+numCartao);
20  System.out.println("Saldo:"+saldo);
21  System.out.println("Bonus:"+bonus);
22  if(bandeira.equals("visa"))
23    System.out.println("VISA");
24  if(bandeira.equals("master"))
25    System.out.println(“MASTERCARD”); ;
26  if(bandeira.equals("american"))
27    System.out.println(“AMERICAN EXPRESS”); ;
```

Exemplo 2.1: Implementação não estruturada do problema.

Por outro lado, podemos notar que existem 3 blocos onde cada um resolve um sub-problema do problema original. Temos a leitura das informações (linhas 6, 7. 8 e 9), a ação de compra (linhas 11 até 17) e de impressão das informações (linhas 19 até 27). Assim, uma outra forma de implementação é apresentada pelo Exemplo 2.2. O mesmo programa é quebrado em 3 rotinas (funções), onde cada uma descreve uma ação que soluciona uma parte do problema. A solução do problema passa a ser encarada como uma seqüência de chamadas de funções, linhas 35, 36 e 37.

```
01  int numCartao, bonus;
02  String bandeira;
03  float saldo;
04  Scanner leitor = new Scanner(System.in);
05
06  void leitura() {
07    numCartao = leitor.nextInt();
08    bandeira = leitor.nextLine();
09    saldo = leitor.nextFloat();
10    bonus = 0;
11  }
12
13  void compra(float valor) {
14    saldo = saldo + 100.0;
15    if (bandeira.equals("visa"))
16      bonus++;
17    if (bandeira.equals("master"))
18      bonus = bonus + 2;
19    if (bandeira.equals("american"))
20      bonus = bonus + 5;
21  }
22
23  void imprime() {
24    System.out.println("No Cartão:"+numCartao);
25    System.out.println("Saldo:"+saldo);
26    System.out.println("Bonus:"+bonus);
27    if (bandeira.equals("visa"))
28      System.out.println("VISA");
29    if (bandeira.equals("master"))
30      System.out.println(“MASTERCARD”);
31    if (bandeira.equals("american"))
32      System.out.println(“AMERICAN EXPRESS”);
33  }
34
35  leitura();
36  compra(100.0f);
37  imprime();
```

Exemplo 2.2 : Implementação procedural do problema.

Este estilo, também chamado de procedural, é geralmente uma escolha melhor que a implementação puramente seqüencial e não

estruturada. Para problemas que envolvem uma complexidade média, a facilidade de manutenção pode ser claramente percebida com a divisão dos sub-problemas por meio das funções. Desta forma, podemos reutilizar o mesmo código em diferentes lugares no programa sem copiá-lo. Além disso, torna-se mais fácil organizar o fluxo do programa em comparação a uma simples seqüência de comandos que podem transformar a solução do problema em um programa grande e complicado de ser entendido.

Uma implementação ainda mais elegante pode ser implementada utilizando o conceito de TAD. Em computação, um TAD (Tipo Abstrato de Dado) é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados. Com isso, podemos reduzir a informação necessária para a criação e solução do problema por meio da abstração das variáveis envolvidas em uma única entidade fechada, disponibilizando operações próprias à sua natureza conforme podemos notar pela implementação apresentada pelo Exemplo 2.3.

A criação das TAD's possibilitou um agrupamento das informações que estão relacionadas entre si e que descrevem uma entidade. Este agrupamento é descrito pelas linhas 1 até 6. As funções agora recebem como parâmetro um valor contendo a estrutura definida para que seja possível manipular as informações de forma agrupada. Além disso, proporciona um acesso às operações claras e bem definidas.

Apesar de conter conceitos mais sofisticados que os TAD's, o paradigma orientado a objetos possui uma forma semelhante de estruturação do programa, baseada na criação de classes, as quais descrevem as características por meio das informações e ações que se deseja que uma entidade tenha, conforme apresentado pelo Exemplo 2.4.

O paradigma também apresenta uma estrutura denominada objeto, que é a instância desta entidade criada a partir das classes, possibilitando a manipulação dos dados e a execução das ações.

A programação orientada a objetos aproveita a modelagem estruturada por meio das TAD's e acrescenta novos conceitos mais sofisticados produzindo um estilo que proporciona, entre diversos benefícios, a facilidade de manutenção e a reutilização dos programas.

```
01  def_estrutura Cartao
02  {
03    int numCartao, bonus;
04    String bandeira;
05    float saldo;
06  }
07  Scanner leitor = new Scanner(System.in);
08
09  void leitura(Cartao c) {
10    c.numCartao = leitor.nextInt();
11    c.bandeira = leitor.nextLine();
12    c.saldo = leitor.nextFloat();
13    c.bonus = 0;
14  }
15
16  void compra(Cartao c, float valor) {
17    c.saldo = c.saldo + 100.0;
18    if (c.bandeira.equals("visa"))
19      c.bonus++;
20    if (c.bandeira.equals("master"))
21      c.bonus = c.bonus + 2;
22    if (c.bandeira.equals("american"))
23      c.bonus = c.bonus + 5;
24  }
25
26  void imprime(Cartao c) {
27    System.out.println("No Cartão:"+c.numCartao);
28    System.out.println("Saldo:"+c.saldo);
29    System.out.println("Bonus:"+c.bonus);
30    if (c.bandeira.equals("visa"))
31      System.out.println("VISA");
32    if (c.bandeira.equals("master"))
33      System.out.println(“MASTERCARD”);
34    if (c.bandeira.equals("american"))
35      System.out.println(“AMERICAN EXPRESS”);
36  }
37
38  Cartao c1;
39  leitura(c1);
40  compra(c1,100.0f);
41  imprime(c1);
```

Exemplo 2.3: Implementação estruturada TAD.

```
01  class Cartao {
02    int numCartao, bonus;
03    String bandeira;
04    float saldo;
05    Scanner leitor = new Scanner(System.in);
06
07    void leitura() {
08      numCartao = leitor.nextInt();
09      bandeira = leitor.nextLine();
10      saldo = leitor.nextFloat();
11      bonus = 0;
12    }
13    void compra(float valor) {
14      saldo = saldo + 100.0;
15      if (bandeira.equals("visa"))
16        bonus++;
17      if (bandeira.equals("master"))
18        bonus = bonus + 2;
19      if (bandeira.equals("american"))
20        bonus = bonus + 5;
21    }
22    void imprime() {
23      System.out.println("No Cartão:"+numCartao);
24      System.out.println("Saldo:"+saldo);
25      System.out.println("Bonus:"+bonus);
26      if (bandeira.equals("visa"))
27        System.out.println("VISA");
28      if (bandeira.equals("master"))
29        System.out.println(“MASTERCARD”);
30      if (bandeira.equals("american"))
31        System.out.println(“AMERICAN EXPRESS”);
32    }
33  }
34
35  Cartao c1, c2;
36  c1 = new Cartao();
37  c2 = new Cartao();
38  c1.leitura();
39  c2.leitura();
40  c1.compra(100.0f);
41  c2.compra(20.0f);
42  c1.imprime();
43  c2.imprime();
```

Exemplo 2.4: Implementação Orientada a Objetos (em Java).

2.2 Classes e Objetos para solução de problemas

Na programação orientada a objetos, os conceitos de classe, objeto, método, atributo e instância são fundamentais para o bom entendimento deste paradigma de programação. Entretanto, as definições sobre cada um desses conceitos talvez não seja a melhor maneira de compreendê-los. Desta forma, utilizamos um problema exemplo para apresentar esses conceitos.

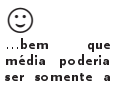
Problema 2.2 : *Escreva um programa que realize a leitura de informações sobre um aluno: nome e duas notas. Seu programa deve calcular a média aritmética das notas e apresentar essa média e as informações do aluno.*

Uma implementação não orientada a objetos é apresentada conforme o Exemplo 2.5.

```
01  String nome;
02  float nota1, nota2, media;
03  Scanner leitor = new Scanner(System.in);
04
05  System.out.print("Digite o nome:");
06  nome = leitor.nextLine();
07  System.out.print("Digite nota 1:");
08  nota1 = leitor.nextFloat();
09  System.out.print("Digite nota 2:");
10  nota2 = leitor.nextFloat();
11
12  media = (nota1 + nota2) /2;
13
14  System.out.println("Aluno:"+nome);
15  System.out.println("Nota 1:"+nota1);
16  System.out.println("Nota 2:"+nota2);
17  System.out.println("Média:"+media);
```

Exemplo 2.5 : Implementação não orientada a objetos.

Nas linhas 1, 2 e 3 são declaradas as variáveis. A leitura das informações é realizada nas linhas 5 até 10. O cálculo da média é



realizado na linha 12. As linhas 14 a 17 realizam a impressão das informações do aluno. Podemos notar que as variáveis representam informações e que existem 3 funcionalidades relacionadas à uma entidade Aluno. Na orientação a objetos chamamos a descrição dessas características e funcionalidades da entidade Aluno como *classe*. As características são chamadas de *atributos* e as funcionalidades de *métodos*.

Em Java, descrevemos a classe por meio da palavra reservada class. Os atributos são descritos como variáveis declaradas dentro do bloco da classe, seguido dos métodos conforme apresentado pelo Exemplo 2.6.

Neste caso, teríamos a classe Aluno descrevendo os atributos: NOME, NOTA1, NOTA2 E MEDIA; e os métodos LERDADOS(), CALCULAMEDIA() e IMPRIME(). Note que a classe *Aluno* apenas descreve o comportamento de um aluno. Para que possamos armazenar informações nos atributos e realizar a execução dos métodos, precisamos que estas definições sejam concretizadas de alguma forma, ou seja, que uma estrutura computacional contendo as especificações descritas pela classe seja alocada em memória. Esta estrutura caracteriza o que chamamos de *objeto*.

Na linguagem Java, um *objeto* é criado a partir de uma *classe* por meio da palavra reservada NEW. O termo *instanciar um objeto* caracteriza o ato de criar um objeto. Além disso, assumimos um objeto como uma instância de uma certa classe.

...o termo INSTÂNCIA vêm do idioma inglês INSTANCE que significa EXEMPLO...na OO indica a concretização de uma classe

```
01  class Aluno {
02    String nome;
03    float nota1;
04    float nota2;
05    float media;
06
07    void lerDados() {
08      Scanner leitor = new Scanner(System.in);
09      System.out.print("Digite o nome:");
10      nome = leitor.nextLine();
11      System.out.print("Digite nota 1:");
12      nota1 = leitor.nextFloat();
13      System.out.print("Digite nota 2:");
14      nota2 = leitor.nextFloat();
15    }
16
17    void calculaMedia() {
18      media = (nota1 + nota2) /2;
19    }
20
21    void imprime() {
22      System.out.println("Aluno:"+nome);
23      System.out.println("Nota 1:"+nota1);
24      System.out.println("Nota 2:"+nota2);
25      System.out.println("Média:"+media);
26    }
27  }
```

Exemplo 2.6: Implementação da Classe Aluno.

Para que possamos simular o comportamento de um objeto especificado pela classe Aluno, construímos uma classe extra. O Java disponibiliza a execução de classes que tenham definido um método específico, chamado MAIN(). Com isso, o Exemplo 2.7 descreve uma classe PROG que possibilita a simulação do comportamento de um objeto instanciado da classe Aluno.

☺
"Uma longa caminhada começa com o primeiro passo", *um filósofo*
"void main(){", *um programador*

Na linha 5 declaramos duas variáveis a1 e a2 para armazenar objetos da classe Aluno. Os objetos são instanciados e armazenados em a1 e a2 respectivamente pelas linhas 7 e 8. Após isto, são realizadas as chamadas pelos métodos de cada um dos objetos.

Note que, apesar dos objetos armazenados em a1 e a2 serem iguais nas características, pois foram instanciados da mesma classe Aluno, ambos são independentes, podendo armazenar valores diferentes em seus atributos.

```
01  class Prog {
02    public static void main(String args[]) {
03      Aluno a1, a2;
04
05      a1 = new Aluno();
06      a2 = new Aluno();
07
08      a1.lerDados();
09      a1.calculaMedia();
10      a1.imprime();
11
12      a2.lerDados();
13      a2.calculaMedia();
14      a2.imprime();
15    }
16  }
```

Exemplo 2.7 : IImplementação da Classe Prog.

A Figura 2.2 apresenta uma possível representação dos objetos durante a execução do método main() da classe Prog.

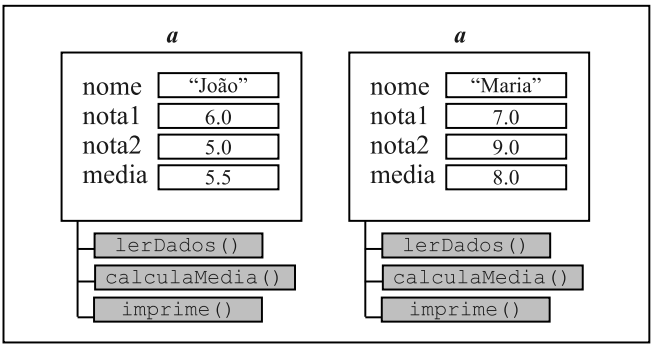


Figura 2.1: Representação dos objetos armazenados nas variáveis a1 e a2.

2.2.1 Algumas definições

A seguir, apresentamos algumas definições encontradas na literatura para os conceitos de classe, objeto, método e atributo.

Classe:

- Abstração do objeto;
- Molde definindo os atributos e (comportamentos) métodos comuns compartilhados por um tipo de objeto;
- Descreve as características que definem um objeto: atributos e métodos;
- Molde para criação de objetos;
- Descrição dos objetos que a ela pertencem;
- Modelo de código utilizado para gerar objetos;
- Gabarito que define atributos e métodos de um objeto do mundo real.

Objeto:

- Instância da classe;
- Conjunto de atributos (características do objeto) e mensagens (ações que um objeto pode realizar) agrupados e armazenados em memória;
- Estrutura dinâmica originada com base em uma classe;
- Classe valorada.

Atributo:

- Representam as informações do objeto e definem a estrutura do mesmo;
- Características;
- Informações;
- Estado.

Método:

- Ações ou serviços que um objeto pode realizar;

- Ação;
- Comportamento;
- Mensagem;
- Operação.

2.3 Modelando classes simples

Esta Seção tem como objetivo exercitar a criação de classes simples.

Problema 2.3 : *Descreva uma classe chamada* ContaCorrente *contendo dois atributos:* número da conta *e* saldo. *Implemente os seguintes métodos:* CREDITA(FLOAT VALOR), DEBITA(FLOAT VALOR), *e* IMPRIME(). *Os métodos devem respectivamente adicionar um valor ao saldo da conta, retirar um valor do saldo, e imprimir as informações da conta. Sabe-se que um débito é apenas realizado se o saldo é suficiente para esta operação. Simule um objeto desta classe.*

```
01  class ContaCorrente {
02    int numero;
03    float saldo;
04    void credita(float valor) {
05      saldo = saldo + valor;
06    }
07    void debita(float valor) {
08      if (valor <= saldo)
09        saldo = saldo - valor;
10    }
11    void imprime() {
12      System.out.println("Conta:"+numero);
13      System.out.println("Saldo:"+saldo);
14    }
15  }
```

Exemplo 2.8 : Solução para o Problema 2.3.

```
01  class Prog {
02    public static void main(String args[]) {
03      ContaCorrente conta;
04      conta = new ContaCorrente();
05      conta.numero = 123;
06      conta.saldo = 100.0f;
07      conta.credita(23.0f);
08      conta.debita(43.0f);
09      conta.debita(90.0f);
10      conta.imprime();
11    }
12  }
```

Exemplo 2.9 : Solução para o Problema 2.3.

Problema 2.4 : *Descreva uma classe chamada Veiculo contendo como atributos: nome, marca, velocidade e um estado que controle a situação do veículo (ligado/desligado). Implemente os seguintes métodos: ligar(), desligar(), acelerar(), frear() e imprime(). Os métodos devem respectivamente ligar e desligar o veículo, acelerar e frear o veículo, e imprimir as informações do veículo. Sabe-se que a ação de acelerar/frear incrementa/decrementa 1 km/h à velocidade do veículo e somente é possível acelerar o veículo quando ele está ligado. Ao frear, a velocidade é zerada. Simule um objeto desta classe.*

Problema 2.5 : *Descreva uma classe chamada TicketEvento contendo como atributos: nome do evento, valor, número do acento e um estado que controle a situação do ticket (válido/não válido). Implemente os seguintes métodos: valida(), cancela() e imprime(). Os métodos devem respectivamente validar e cancelar o ticket, e imprimir as informações do Ticket. Simule um objeto desta classe.*

Problema 2.6 : *Acrescente um atributo para armazenar a potência do veículo na classe chamada Veiculo do problema 2. Modifique os métodos para que a ação de acelerar/frear incremente/decremente 1 km/h para veículos com potencia 1.0, 3 km/h para veículos com potencia 1.6, e 10 km/h para veículos com potencia 1.8. Simule um objeto desta classe.*