# 9 Arquivos

*Onde uma nova estrutura de dados é apresentada e como ela permite o armazenamento de dados e informação de forma persistente.*

| Pré-requisito(s): | Objetivos (ao final você deverá ser capaz de): |
|---|---|
| • Saber declarar e utilizar classes como registros em Java<br>• Saber criar e utilizar métodos em Java | • Armazenar dados e informação em arquivos<br>• Realizar leitura e gravação de dados em arquivos<br>• Manipular dados armazenados<br>• Diferenciar arquivos de acesso seqüencial e arquivos de acesso direto<br>• Programar co arquivos em Java |

Até então, vínhamos realizando o armazenamento de dados para resolução de problemas em variáveis simples ou compostas. Entretanto, esse armazenamento é temporário, uma vez que essas variáveis representam endereços da memória principal (memória interna), que é volátil. Ou seja, no momento em que a variável sai do escopo ou o programa é encerrado, todos os dados são perdidos. Uma outra limitação das estruturas de armazenamento temporário é o volume de dados que são capazes de armazenar, que estão restritos ao tamanho da memória principal disponível na máquina. Normalmente, ao escrevermos um documento, salvamos as informações digitadas em um dispositivo de armazenamento secundário, como o HD, CDs, DVDs, pendrives, etc. Isso permite que possamos recuperar as informações que digitamos no momento em que quisermos, independentemente, inclusive, do computador ter sido desligado ou não. Este capítulo mostra como podemos proceder de maneira semelhante para salvar dados e informações de um programa de maneira persistente. O capítulo mostra ainda como podemos

recuperar esses dados e informações e manipulá-los em um programa de computador.

# 9.1 Armazenamento Persistente

A estrutura de dados que utilizamos para salvar dados e informações é usualmente conhecida como **arquivo**. Os dados mantidos em arquivos são dados persistente, uma vez que eles persiste além da duração da execução do programa. A capacidade de processar arquivos é importantíssima em linguagens de programação que sejam utilizadas para o desenvolvimento de aplicações comerciais, por exemplo, que em geral lidam com volumes maciços de dados.

## 9.1.1 Organização de arquivos

Um arquivo é normalmente composto por um conjunto de registros logicamente organizados e armazenados em um dispositivo de memória secundária. Como vimos, cada registro compreende um conjunto de dados representados por campos.

Considere como exemplo, um programa que deve armazenar dados sobre todos os alunos da Universidade Federal de Sergipe, desde o momento de sua matrícula da instituição. Um aluno é representado por seu número de matrícula, que o identifica unicamente dentre todos os demais, seu nome, endereço, data de ingresso, curso, disciplinas cursadas até o momento, média geral ponderada, entre outros. A figura 9.1 ilustra a **relação existente entre arquivo, registro e campos** para esse exemplo.

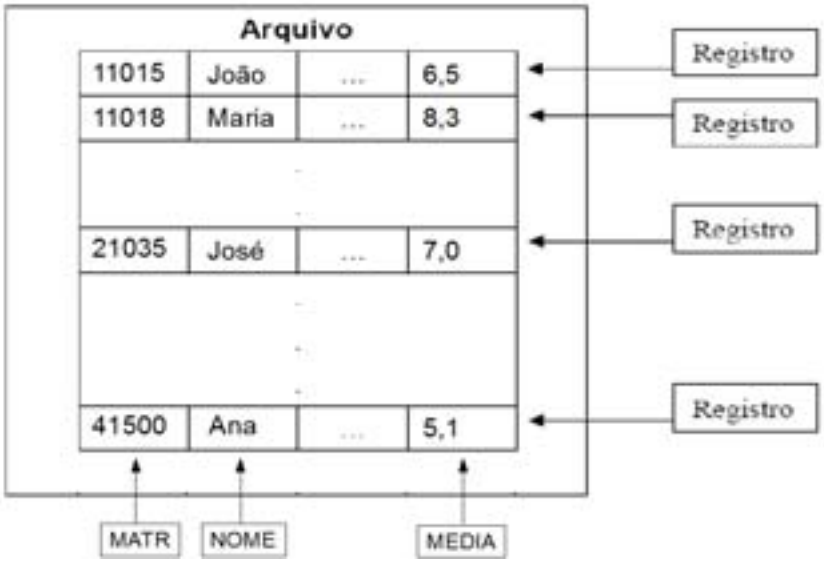


**Figura 9.1** – Relação entre arquivos registros e campos.

Uma peculiaridade da estrutura arquivo é não possuir tamanho limite pré-definido. Como vimos, cada tipo primitivo ou composto (inteiro, real, lógico, registro, vetor, etc) possui limite de tamanho especificado no momento da declaração. Os dados são adicionados, modificados ou excluídos de arquivos sem esse tipo de preocupação. De fato, o único limite existente é imposto pela capacidade de armazenamento do dispositivo de memória secundária utilizado.

## 9.1.2 Declaração e Manipulação de Arquivos

A declaração de um arquivo é feita através da seguinte sintaxe:

*<id_arquivo>: arquivo composto de <tipo>;*

onde, <id_arquivo> é o identificador da variável que representa o arquivo físico e <tipo> representa o tipo dos valores armazenados no arquivo; normalmente um tipo registro.

A figura 9.2 ilustra a declaração de um arquivo para o exemplo do cadastro de alunos.

```
1.  ...
2.  Tipos
3.     TDisciplina = registro
4.         codigo: inteiro;
5.           nome: literal;
6.          media: real;
7.     fimregistro;
8.     VDisciplinas = vetor[70] de TDisciplina;
9.     TAluno = registro
10.        matr: inteiro;
11.        nome: literal;
12.         end: literal;
13.        ingr: literal;
14.        sexo: lógico;
15.       curso: literal;
16.        hist: VDisciplinas;
17.        MGP: real;
18.    fimregistro;
19.    TArquivo = arquivo composto de TAluno;
20. Variáveis
21.    arqAlunos: TArquivo;
22. Início
23.    ...
24. Fim.
```

**Figura 9.2** – Declaração de uma variável do tipo arquivo.

Observe que `arqAlunos` é o identificador da variável que representa a estrutura do arquivo durante a execução do programa. O tipo dessa variável é o tipo `TArquivo`, definido como sendo um arquivo composto de registros de alunos

(`TAluno`) (linha 19), que por sua vez possui campos que caracterizam cada aluno da instituição (`matr`, `nome`, `endr`, etc) (linhas 9 à 18).

Várias operações estão associadas com a manipulação de arquivos. Como é sabido, podemos **criar** um novo arquivo, **abrir** ou **excluir** um arquivo existente, **procurar** por uma informação armazenada, **consultar** essa informação, **modificá-la**, **removê-la**, **adicionar** novas informações, e **fechar** um arquivo.

Para podermos utilizar as informações contidas no arquivo ou para guardá-las nesse arquivo devemos primeiramente abrir o arquivo através do comando:

```
abra (<id_arquivo>,<path_arquivo_fisico>);
```

onde `<id_arquivo>` é o nome da variável de arquivo previamente definida que será associada a um arquivo físico armazenado na memória secundária, e cujo nome é `<path_arquivo_fisico>` (por exemplo, `"c:/caminho/Arquivo.txt"`; ou simplesmente `"Arquivo.txt"`, se o arquivo se encontrar no mesmo diretório do programa desenvolvido). Após a operação de abertura, a informação que estará disponível é sempre a do primeiro registro que foi armazenado, para onde o apontador de registro estará sinalizando.

Após utilizar as informações de um arquivo recomenda-se fechá-lo para garantir a integridade dos dados armazenados:

```
feche (<id_arquivo>);
```

Para se excluir por completo o arquivo físico do dispositivo de armazenamento, utiliza-se o seguinte comando:

```
apaga(<id_arquivo>);
```

Uma vez que temos um arquivo aberto, podemos podemos proceder à leitura dos registros já armazenados no arquivo ou gravação de novos registros no arquivo. Quando efetuamos a leitura de um registro de um arquivo, transferimos os dados do registro do arquivo, armazenados em memória secundária, para a memória principal do computador. Para isto, utilizamos a seguinte notação:

```
leia(<id_arquivo>, <id_registro>);
```

onde, `<id_arquivo>` é o nome da variável do tipo de arquivo definido previamente e `<id_registro>` é o nome de uma variável do tipo registro de formato igual àquele que compõe o arquivo.

Quando gravamos (adicionamos) um novo registro em um arquivo, estamos transferindo os dados do registro, armazenados em memória principal, para a memória secundária do computador. Utilizamos o seguinte comando:

```
escreva(<id_arquivo>, <id_registro>);
```

onde, `<id_arquivo>` é o nome da variável do tipo de arquivo definido previamente e `<id_registro>` é o nome de uma variável do tipo registro definida

para compor o arquivo.

As operações realizadas com registros em uma estrutura de dados do tipo arquivo podem ser resumidas em: inserção de um novo registro (inclusão), obtenção de um registro do arquivo (consulta), modificação dos dados armazenados no arquivo (alteração) ou remoção de um registro do arquivo (exclusão). Dependendo do tipo de problema estas operações poderão ocorrer em maior ou menor escala.

Neste caso é necessário que seja analisada qual é a melhor forma de acessar esses registros. Basicamente existem duas formas diferentes de acesso aos registros de um arquivo: o **acesso seqüencial** e o **acesso direto** (também conhecido como randômico).

## 9.2 Acesso Seqüencial

No momento em que criamos um novo arquivo, devemos estabelecer o modo pelo qual os registros são armazenados no arquivo, ou seja, a estruturação do arquivo. Se os **registros são gravados no arquivo de forma contínua**, uma após o outro, dizemos que o arquivo foi criado para acesso seqüencial.

Nessa forma de gravação de registro, não especificamos a priori a posição do registro no arquivo. Sendo assim, quando desejamos ter acesso aos dados de um determinado registro e não sabemos a localização desse registro no arquivo devemos vasculhar o arquivo desde o início em busca do registro desejado até o final do arquivo se necessário. Esta operação deverá ser repetida seqüencialmente, isto é, avançando pelos registros armazenados até que se encontre o registro procurado. Isto significa que para acessar um registro específico precisamos obedecer a ordem com que os registros foram armazenados no arquivo, o que implica em percorrer todos os registros que o antecedem.

O laço para controlar o avanço pelos registros armazenados em um arquivo pode ser definido como ilustrado abaixo:

```
repita
    avance(<id_arquivo>);
  ...
até fda(<id_arquivo>);
```

onde `avance<id_arquivo>` avança para o registro que está na posição consecutiva no arquivo e `fda(<id_arquivo>)` é o comando que retorna verdadeiro quando o ponteiro de registro estiver apontando para o final do arquivo.

Considere, por exemplo, um arquivo organizado seqüencialmente chamado `Alunos.txt` salvo no HD de uma máquina, contendo uma série de registros do

tipo `TAluno` já gravados. O pseudocódigo da figura 9.3 deverá permitir a consulta da média de um determinado aluno, dado seu número de matrícula.

```
1.  Algoritmo LeituraArquivoAlunos;
2.  Tipos
3.     TDisciplina = registro
4.         codigo: inteiro;
5.           nome: literal;
6.          media: real;
7.     fimregistro;
8.     VDisciplinas = vetor[70] de TDisciplina;
9.     TAluno = registro
10.        matr: inteiro;
11.        nome: literal;
12.         end: literal;
13.        ingr: literal;
14.        sexo: lógico;
15.       curso: literal;
16.        hist: VDisciplinas;
17.        MGP: real;
18.    fimregistro;
19.    TArquivo = arquivo composto de TAluno;
20. Variáveis
21.       arq: TArquivo;
22.       reg: TAluno;
23.    codigo: inteiro;
24.     achou: lógico;
25. Início
26.    abra (arq,"Alunos.txt");
27.    achou <- falso;
28.    leia(codigo);
29.    repita
30.      leia(arq,reg);
31.      se (reg.codigo = codigo) então
32.        início
33.          escreva("Média do aluno = ", reg.media);
34.          achou <- verdadeiro;
35.        fim;
36.      avance(arq);
37.    até (fda(arq)) ou (achou = verdadeiro);
38.    se (achou = falso) então
39.       escreva("Aluno não encontrado!");
40.    feche(arq);
41. Fim.
```

**Figura 9.3** – Acesso seqüencial a dados gravados em um arquivo texto.

A variável de arquivo `arq` é associada a um arquivo físico armazenado no HD da máquina e o comando para abertura desse arquivo é executado na linha 26. O usuário do programa é então solicitado a fornecer um código de matrícula para consulta (linha 28) nesse arquivo. A partir daí o laço definido na linha 29 irá avançar registro por registro (linha 36) realizando a leitura de cada um (linha 30) até encontrar um registro cujo código seja igual ao solicitado ou atinja o fim do

arquivo (linhas 30 e 37). Note que ao se realizar a leitura de um registro, todo o seu conteúdo é automaticamente copiado para uma variável auxiliar criada para este fim `(reg)`.

Caso fosse necessário alterar algum dado de um determinado aluno, como o endereço, por exemplo, o registro referente àquele aluno deve ser localizado no arquivo, o valor do campo `end` alterado para o novo valor, e então o registro completo com o valor do campo alterado é gravado no arquivo exatamente na mesma posição em que se encontrava. Esse procedimento é feito no pseudocódigo da figura 9.4.

```
1.  ...
2.  Início
3.     abra (arq,"Alunos.txt");
4.     achou <- falso;
5.     leia(codigo);
6.     repita
7.       leia(arq,reg);
8.       se (reg.codigo = codigo) então
9.         início
10.          escreva("Novo endereço = ");
11.          leia(reg.end);
12.          escreva(arq,reg);
13.          achou <- verdadeiro;
14.        fim;
15.      avance(arq);
16.    até (fda(arq)) ou (achou = verdadeiro);
17.    se (achou = falso) então
18.       escreva("Aluno não encontrado!");
19.    feche(arq);
20. Fim.
```

**Figura 9.4** – Manipulação de dados em arquivos com acesso seqüencial.

Uma vez que o registro com o código solicitado é encontrado (linha 8), o novo endereço é solicitado ao usuário do programa (linhas 10 e 11), e o comando de gravação de registros no arquivo é executado (linha 12), já contemplando a alteração do valor do campo `end`.

## 9.3 Acesso direto

Uma outra forma de estruturação de arquivos é a que permite que os **registros armazenados sejam acessados de forma direta**, sem a necessidade de se percorrer todos os registros que antecedem o registro procurado.

Neste caso, para acessarmos o registro desejado é necessário que saibamos a sua posição física, que é definida no instante da gravação. Para que isto seja possível pelo menos um dos campos do registro precisa armazenar a informação

do número da posição física do registro dentro do arquivo; este campo é nossa já famosa **chave.** Essa chave naturalmente deve ser única, pois dois registros diferentes não podem ter mesma localização.

Para posicionarmos o ponteiro de registro exatamente para a posição do mesmo gravada no arquivo utilizamos o comando abaixo:

```
posicione (<id_arquivo>,<chave>);
```

onde `<id_arquivo>` representa a variável que identifica o arquivo no programa, e a chave é um campo de valor inteiro do registro que indica a posição no arquivo em que o registro se encontra.

No exemplo dos registros de alunos da universidade ilustrado anteriormente, o campo `matr`, que representa a matrícula do aluno na instituição, serve como chave para acesso direto ao arquivo uma vez que é do tipo inteiro e identifica unicamente o aluno na instituição.

O pseudocódigo da figura 9.5 ilustra um programa que permite a inclusão, consulta, alteração e exclusão de registros de um arquivo já existente utilizando acesso direto. O código para cada uma das operações está organizado em um módulo separado (figuras 9.6, 9.7, 9.8, 9.9).

```
1.  Algoritmo AcessoDiretoArquivoAlunos;
2.  Tipos
3.     TDisciplina = registro
4.         codigo: inteiro;
5.           nome: literal;
6.          media: real;
7.     fimregistro;
8.     VDisciplinas = vetor[70] de TDisciplina;
9.     TAluno = registro
10.        matr: inteiro;
11.        nome: literal;
12.         end: literal;
13.        ingr: literal;
14.        sexo: lógico;
15.       curso: literal;
16.        hist: VDisciplinas;
17.        MGP: real;
18.    fimregistro;
19.    TArquivo = arquivo composto de TAluno;
20. Variáveis
21.       arq: TArquivo;
22.       reg: TAluno;
23.    codigo, opcao: inteiro;
24. Início
25.    abra (arq,"Alunos.txt");
26.    repete
27.      leia(opcao);
28.      se (opcao<>5) então
29.        caso opcao igual
30.          1: inclusao;
```

```
31.           2: consulta;
32.           3: alteracao;
33.           4: exclusao;
34.         fimcaso;
35.     até (opcao = 5);
36. Fim.
```

**Figura 9.5** – Inclusão, consulta, alteração e exclusão de registros de um arquivo já existente utilizando acesso direto.

```
1.  modulo inclusao;
2.    op: literal;
3.    se tamanho(arq) > 0 então
4.         posicione (arq, tamanho(arq+1));
5.    repita
6.       reg.codigo <- tamanho(arq);
7.       leia(reg.nome);
8.       leia(reg.e_mail);
9.       escreva(arq,reg);
10.      avance(arq);
11.      escreva('Deseja Continuar <S/N> ? ');
12.      repita
13.         leia(op);
14.      até (op='S') ou (op='N');
15.   até op = "N";
16. fimmodulo;
```

**Figura 9.6** – Módulo para inclusão de registros.

```
1.  modulo consulta;
2.     op: literal;
3.     num_reg: inteiro;
4.     repita
5.       repita
6.          leia (num_reg);
7.       até (num_reg>=0) e (num_Reg<=tamanho(arq));
8.       posicione(arq,num_reg);
9.       leia(arq,reg);
10.      escreva(reg.codigo,reg.nome.reg.e_mail);
11.      escreva('Deseja Continuar <S/N> ? ');
12.      repita
13.         leia(op);
14.      até (op='S') ou (op='N');
15.    até op='N';
16. fimmodulo;
```

**Figura 9.7** – Módulo para consulta de registros.

```
1.  modulo alteracao;
2.     op, aux: literal;
3.     num_reg: inteiro;
4.   repita
5.     repita
6.        leia (num_reg);
7.     até (num_reg >= 0) e (num_Reg < tamanho(arq));
```

```
8.     posicione(arq, num_reg);
9.     leia(arq,reg);
10.    escreva(reg.codigo, reg.nome, reg.e_mail);
11.    leia (aux);
12.    se aux <> '' então
13.       reg.nome <- aux;
14.    leia(aux);
15.    se aux <> '' então
16.       reg.e_mail <- aux;
17.    posicione(arq, num_reg);
18.    escreva(arq,reg);
19.    escreva("Deseja Continuar <S/N> ? ");
20.    repita
21.       leia(op);
22.    até (op = "S") ou (op = "N");
23.  até op='N';
24. fimmodulo;
```

**Figura 9.8** – Módulo para alteração de registros.

```
1.  modulo exclusao;
2.     op: literal;
3.     num_reg: inteiro;
4.     repita
5.       repita
6.         leia (num_reg);
7.       até (num_reg >= 0) e (num_Reg < tamanho(arq));
8.       posicione(arq, num_reg);
9.       leia(arq, reg);
10.      se reg.nome <> "*****" então
11.         início
12.            escreva(reg.codigo,reg.nome.reg.e_mail);
13.            escreva ("Confirma a Exclusão <S/N> ?");
14.            repita
15.              leia(op);
16.            até (op = "S") ou (op = N");
17.            se op = "S" então
18.                início
19.                  reg.nome <- "*****";
20.                  posicione(arq, num_reg);
21.                  escreva(arq, reg);
22.                fim;
23.            fimse;
24.          fim;
25.       senão
26.          escreva("Código já excluído !!");
27.       escreva("Deseja Continuar <S/N> ? ");
28.       repita
29.         leia(op);
30.       até (op = "S") ou (op = "N");
31.     até op= "N";
32. Fim;
```

**Figura 9.9** – Módulo para exclusão de registros.

# 9.4 Trabalhando com arquivos em Java

A linguagem Java trata um arquivo como um fluxo sequencial de bytes ou um fluxo sequencial de caracteres. Esses fluxos de dados são conhecidos pelo termo **stream** e representam na verdade uma grande abstração criada pela linguagem para leitura/escrita de dados de uma maneira geral - uma vez que estamos lendo (ou escrevendo) dados em algum stream, não precisamos saber se estes estão vindo (ou indo) de algum dispositivo de rede (ou Internet), de um arquivo texto, ou mesmo do console.

Arquivos criados com base nos fluxos de bytes são chamados de **arquivos binários** enquanto que arquivos criados com base nos fluxos de caracteres são chamados de **arquivos de texto**. O conteúdo de arquivos de texto pode ser visto tranquilamente por editores de texto padrão, enquanto que arquivos binários não são legíveis por humanos.

Um programa Java abre um determinado arquivo criando e associando um objeto ao fluxo de bytes ou caracteres. As classes utilizadas para criar esses objetos estão presentes no pacote de classes chamada `java.io`.

## 9.4.1 Arquivos de Texto com Acesso Seqüencial

Como dito anteriormente, em arquivos de acesso seqüencial, os registros são armazenados na ordem em que são gravados, não havendo uma posição específica reservada para cada um. Para recuperar os dados gravados, os programas devem começar a ler os registros consecutivamente a partir do início do arquivo até que a informação desejada seja encontrada.

O exemplo a seguir, ilustra como podemos criar um arquivo de texto com acesso seqüencial em Java, gravar, ler e atualizar dados nesse arquivo. O programa ilustrado na figura 9.10 permite a um gerente de crédito de um determinada empresa obter listas de clientes com saldo zero, listas de clientes credores e listas de clientes devedores. Uma conta é representada como uma classe Java na figura 9.11.

```
1.    import java.io.File;
2.    import java.util.Scanner;
3.    import java.util.Formatter;
4.
5.    public class Credito
6.    {
7.      public static void criaClientes() {
8.        Formatter output = null;
9.        // permite ao usuário abrir o arquivo
10.       try { output = new Formatter( "clientes.txt" ); }
11.       catch ( Exception e ) { System.err.println("Erro na criacao do arquivo
12.                               de saida" ); System.exit(1); }
```

```
13.       // adiciona registros ao arquivo
14.       RConta conta = new RConta();
15.       Scanner input = new Scanner( System.in );
16.       System.out.println("Entre com o número da conta (> 0), nome, sobrenome e
17.                           balanco (ou digite  <ctrl>+z+Enter  para finalizar): ");
18.       while (input.hasNext()) // faz um loop até o indicador de fim de arquivo
19.       {
20.          try // gera saída dos valores para o arquivo
21.          {
22.            // recupera os dados para saída
23.            conta.numero = input.nextInt(); // lê o número de conta
24.            conta.nome = input.next(); // lê o primeiro nome
25.            conta.sobrenome = input.next(); // lê o sobrenome
26.            conta.balanco = input.nextFloat(); // lê o saldo
27.            if ( conta.numero > 0 ) { // grava um novo registro
28.                output.format( "%d %s %s %.2f\n", conta.numero,
29.                               conta.nome, conta.sobrenome,
30.                               conta.balanco);
31.            } else { System.out.println("Número da conta deve ser maior que
32.                                        0." ); }
33.            } catch ( Exception e ) { System.err.println( "Erro!" ); return; }
34.            System.out.println("Entre com o número da conta (> 0), nome,
35.                               sobrenome e balanco (ou digite  <ctrl>+z+Enter
36.                               para finalizar): ");
37.       }
38.       //fecha o arquivo de saída
39.       if ( output != null )
40.         output.close();
41.    }
42.
43.    public static void consultaClientes() {
44.       int tipoConta = 1;
45.       Scanner arquivo = null;
46.       // obtém a solicitação do usuário (por exemplo, saldo zero, credor ou
47.          devedor)
48.       Scanner textIn = new Scanner( System.in );
49.       System.out.printf( "\n%s\n%s\n%s\n%s\n%s\n%s\n",
50.            "Entre com o pedido", " 1 - Listar contas com balanço zero",
51.            " 2 - Listar contas de credores",
52.            " 3 - Listar contas de devedores",
53.            " 4 - Fim" );
54.         System.out.print( "\n? " );
55.       tipoConta = textIn.nextInt();
56.       while ( tipoConta != 4 ) {
57.          switch ( tipoConta ) {
58.          case 1: System.out.println( "\nContas com balanços zero:\n" ); break;
59.          case 2: System.out.println( "\nContas de credores:\n" ); break;
60.          case 3: System.out.println( "\nContas de devedores:\n" ); break;
61.          }
62.          // objeto a ser gravado no arquivo
63.          RConta conta = new RConta();
64.          try {
65.             // abre o arquivo para leitura a partir do início
66.             arquivo = new Scanner( new File( "clientes.txt" ) );
67.             while (arquivo.hasNext()) // insere os valores do arquivo
68.             {
69.                conta.numero = arquivo.nextInt(); // lê o número de conta
70.                conta.nome = arquivo.next(); // lê o primeiro nome
71.                conta.sobrenome = arquivo.next(); // lê o sobrenome
72.                conta.balanco = arquivo.nextFloat(); // lê o saldo
73.                // se o tipo for a conta adequada, exibe o registro
74.                if ( ((tipoConta == 2) && (conta.balanco > 0)) ||
75.                     ((tipoConta == 3) && (conta.balanco < 0)) ||
76.                     ((tipoConta == 1) && (conta.balanco== 0)) )
77.                   System.out.printf( "%-10d%-12s%-12s%10.2f\n",
78.                                      conta.numero, conta.nome,
79.                                      conta.sobrenome, conta.balanco );
80.             }
81.          }
82.          catch ( Exception e ) { System.err.println( "Erro!" ); }
83.          finally { if ( arquivo != null ) arquivo.close(); }
84.          System.out.print( "\n? " );
85.
```

```
86.          tipoConta = textIn.nextInt();
87.       }
88.    }
89.
90.    public static void main( String args[] ) {
91.       int opcao = 0;
92.       Scanner textIn = new Scanner( System.in );
93.       System.out.print( "O que deseja fazer?\n1 - Adicionar contas em um novo
94.                          arquivo\n2 - Consultar contas");
95.       System.out.print( "\n? " );
96.       opcao = textIn.nextInt();
97.       switch (opcao) {
98.          case 1: criaClientes(); break;
99.          case 2: consultaClientes(); break;
100.      }
101.   }
102. }
```

**Figura 9.10** – Arquivo de texto com acesso seqüencial em Java

```
1.   public class RConta {
2.      int numero;
3.      String nome;
4.      String sobrenome;
5.      float balanco;
6.   }
```

**Figura 9.11** – Registro que compõe um arquivo texto Java.

## 9.4.2 Arquivos com Acesso Direto

Como dito anteriormente, em arquivos de acesso seqüencial, os registros são armazenados na ordem em que são gravados, não havendo uma posição específica reservada para cada um. Para recuperar os dados gravados, os programas devem começar a ler os registros consecutivamente a partir do início do arquivo até que a informação desejada seja encontrada.

O exemplo a seguir, ilustra como podemos criar um arquivo de texto com acesso direto em Java, gravar, ler e atualizar dados nesse arquivo. O programa ilustrado na figura 9.12 permite criar um arquivo com acesso direto, criar registros de contas de clientes e gravar nesse arquivo de forma direta, e realizar consulta a registros desse arquivo através de acesso direto à posição via chave informada pelo usuário.

```
1.    import java.io.*;
2.    import java.util.Scanner;
3.    public class AcessoDireto
4.    {
5.        public static void main(String[] args)
6.        {
7.            RandomAccessFile raf = null;
8.            RConta conta = null;
9.            Scanner input = null;
10.           conta = new RConta();
11.           input = new Scanner( System.in );
12.           try
```

```
13.          {
14.              // Cria um novo arquivo de acesso direto
15.              raf = new RandomAccessFile("arquivo.txt", "rw");
16.              System.out.println("Entre com o número da conta (> 0), nome,
17.                                  sobrenome e balanco (ou digite
18.                                  <ctrl>+z+Enter  para finalizar): ");
19.                  while (input.hasNext()) // faz um loop até o indicador de
20.                                          fim de arquivo
21.                  {
22.                       try // gera saída dos valores para o arquivo
23.                       {
24.                          // recupera os dados para saída
25.                          conta.numero = input.nextInt(); // lê o número de
26.                                                             conta
27.                          conta.nome = input.next(); // lê o primeiro nome
28.                          conta.sobrenome = input.next(); // lê o sobrenome
29.                          conta.balanco = input.nextFloat(); // lê o saldo
30.
31.                          if ( conta.numero > 0 ) { // grava um novo registro
32.                             raf.seek((conta.numero-1)*72);
33.                             raf.writeInt(conta.numero);
34.                             raf.writeUTF(conta.nome);
35.                             raf.writeUTF(conta.sobrenome);
36.                             raf.writeFloat(conta.balanco);
37.                          } else { System.out.println("Número da conta deve
38.                                                       ser maior que 0." ); }
39.                       } catch ( Exception e ) { System.err.println("Erro!" );
40.                                                 return; }
41.                       System.out.println("Entre com o número da conta (> 0),
42.                                           nome, sobrenome e balanco (ou digite
43.                                           <ctrl>+z+Enter  para finalizar): ");
44.                  }
45.                  if ( raf != null )
46.                raf.close();
47.                  // realiza consulta da conta desejada
48.                  raf = new RandomAccessFile("arquivo.txt", "r");
49.                  input = new Scanner( System.in );
50.              System.out.print("Entre com o número da conta desejada: ");
51.              int num = input.nextInt();
52.              System.out.print("numero lido: "+num);
53.              // Posiciona o ponteiro de registro para a posição (num-1)*72
54.              raf.seek((num-1)*72);
55.              System.out.println("Conta encontrada: ");
56.              System.out.println(raf.readInt());
57.              System.out.println(raf.readUTF());
58.              System.out.println(raf.readUTF());
59.              System.out.println(raf.readFloat());
60.              raf.close();
61.          } catch (IOException ex) { System.out.println(ex.toString()); }
62.      }
63.  }
```

**Figura 9.12** – Gravação e consulta de registros em um arquivo de texto com acesso direto em Java.

# Resumo

- O tipo de dado arquivo representa uma abstração de arquivos físicos armazenados na memória secundária do computador.
- Em um arquivo pode-se armazenar uma grande quantidade de registros de dados.
- Arquivos permitem operações de consulta, inclusão, exclusão e alteração de

registros.

- Um arquivo permite dois tipos de acesso a dados: seqüencial ou direto (randômico)
- Arquivos com acesso seqüencial possuem registros que foram armazenados um após o outro e, portanto, devem ser acessados da mesma maneira. Ou seja, para se consultar o 54º registro devemos passar por todos os outros 53 anteriores.
- Arquivos com acesso direto possuem registros que foram armazenados respeitando uma posição pré-determinada, normalmente a partir do valor de suas chaves. O acesso a esses registros também é feito de forma direta, sem a necessidade de leitura seqüencial dos demais registros armazenados.
- Java implementa os dois tipos de arquivos.

# Na próxima aula...

… veremos como construir programas Java de aparência mais amigável, com uso de botões, janelas e caixas de texto. Também veremos como executar programas Java a partir de uma página na Web.

# Referências e Sugestões de Leitura

Arquivos em pseudocódigo são tratados no capítulo 5 de

**FORBELLONE, A., EBERSPÄCHER, H. Lógica de Programação. Prentice Hall, 3ed, 2005.**

A utilização de arquivos em Java é tratada no capítulo 14 de

**DEITEL, H., DEITEL, P. Java como Programar, Prentice-Hall, 2005.**

# Exercícios Propostos

9.1 – Seja a seguinte estrutura de registro:

R.A.

NOME

Fazer um programa Java para cadastrar alunos em um arquivo novo. Parar o processo quando R.A. = “

9.2 – Seja a seguinte estrutura de registro:

NÚMERO DE INSCRIÇÃO

NOME

SEXO

CURSO

a) Cadastrar dados sobre candidatos ao vestibular em um arquivo novo, fazendo crítica para entrada de sexo. Parar o processo quando nome = ‘’.

b) Dado o número de inscrição mostrar na tela os dados do candidato (consulta de acesso seqüencial).

9.3 – A seção de controle de produção de uma fábrica mantém o arquivo de registros de produção por máquinas. Cada registro contém o número da máquina, a data e o número de peças produzidas no dia. Supondo que a fábrica possua três máquinas, escrever um algoritmo que separe o arquivo em três outros arquivos, um para cada máquina. O novo arquivo não precisa conter o número da máquina. Utilizar:

MAQ: número da máquina - numérico inteiro 1,2 ou 3

DATA: dd/mm/aa - 8 caracteres

PECAS: número de peças - numérico inteiro.

9.4 – Uma instituição de pesquisa recolheu amostras em três regiões a respeito do nível de vida da população. Cada amostra constitui um registro com os seguintes campos: sexo, idade, salário e grau de instrução. Em cada região os dados foram armazenados em um arquivo seqüencial. Escrever um algoritmo que junte estes arquivos em um único arquivo. O novo arquivo deverá conter em cada registro um campo com o número da região. Utilizar:

SEXO sexo

ID idade

SAL salário

GRAU grau de instrução

REGIAO número da região

9.5 – Considerando o arquivo contendo registros gravados com os seguintes campos:

CODMAT: código do material - numérico inteiro

NOMAT: nome do material

QTEST: quantidade em estoque - numérico inteiro

QTMIN: quantidade mínima - numérico inteiro

PRUNIT: preço unitário - numérico real.

a) Fazer a alteração automática do preço unitário de todos os materiais. O usuário deverá fornecer o índice de reajuste.

b) Exibir na tela o código e nome do material cuja quantidade em estoque seja inferior ou igual a quantidade mínima.

9.6 – Seja o arquivo contendo a seguinte estrutura de registro:

CODIGO: numérico inteiro

NOME: literal

SETOR: literal (letra)

FUNÇÃO:

SALÁRIO numérico real.

a) Permitir ao usuário escolher o código do funcionário e caso este esteja cadastrado, apresentar os dados na tela e permitir alterar um ou mais campos entre: nome, setor e função. O usuário poderá parar o processo quando desejar.

b) Criar e gravar um arquivo de cópia de segurança do arquivo original.

9.7 – Seja um arquivo contendo registros com a seguinte estrutura:

MARCA

MODELO

COR

ANO

PREÇO

a) Dado MARCA e MODELO, apresentar na tela todos os veículos cuja MARCA e MODELO coincidam com as escolhas.

b) Considerando o mesmo arquivo, dado o número do registro, mostrar as informações do veículo na tela e solicitar confirmação para exclusão. Em caso afirmativo regravar o registro com o campo MARCA contendo *** (exclusão lógica). Repetir o processo até que se digite N para opção "mensagem -> Continua exclusão <S/N> ?".