Dia 21 - Arquivo

Conteúdo

O dispositivo de memória principal de um computador é volátil, não permitindo que as informações fiquem armazenadas permanentemente. Por este motivo os dados armazenados em variáveis são perdidos sempre que um programa terminar sua execução normalmente ou devido a algum problema. A solução para este tipo de situação é utilizar uma estrutura de dados que permita a gravação de informações em dispositivos de memória secundária como disquete, disco rígido, entre outros. Nas linguagens de programação, a estrutura de dados que guarda informações em dispositivos de armazenamento secundário é chamada de arquivo.

Um arquivo é uma abstração utilizada para uniformizar a interação entre o ambiente de execução (um programa) e os dispositivos externos (o disco rígido, por exemplo). Os computadores utilizam os arquivos como estruturas de dados para armazenamento de longo prazo de grandes volumes de dados, mesmo depois de os programas que criaram os dados terminarem sua execução. Dados mantidos em arquivos são chamados de dados persistentes porque eles existem além da duração da execução do programa em dispositivos de armazenamento secundário.

Neste artigo, será abordado como os programas Java criam, recuperam, atualizam e processam arquivos de dados. O processamento de arquivos representa no Java um subconjunto das capacidades de processamento que permitem a um programa armazenar e processar volumes maciços de dados persistentes.

Fluxos de dados (streams)

O Java enxerga um arquivo como um fluxo sequencial de caracteres ou bytes finalizados por uma marca de final de arquivo ou pelo número total de bytes registrados.

Os fluxos de entrada e saída baseados em bytes, denominados de arquivos binários, armazenam e recuperam dados no formato binário. Nos arquivos binários os dados são organizados como sendo uma sequência de bytes. Um byte é composto de oito bits agrupados. Um bit, abreviação de “binary digit”, é o menor item de dados manipulado em um computador e pode assumir o valor 0 ou o valor 1. Por exemplo, se o valor 9 fosse armazenado utilizando um fluxo baseado em bytes, ele seria armazenado no formato binário do seu valor numérico 9, ou 1001.

Os fluxos de entrada e saída baseados em caracteres, denominados de arquivos de texto, armazenam e recuperam dados como uma sequência de caracteres dividida em linhas terminadas por um caractere de fim de linha (\n).

Dia 21 - Arquivo

O sistema de entrada e saída em Java fornece uma “interface” consistente ao programador, independente do dispositivo real (teclado, disco, monitor, rede de comunicação, entre outros) que é acessado, estabelecendo um nível de abstração entre o programa e o dispositivo utilizado. Essa abstração, representada na Figura 1, é chamada de stream e o dispositivo real é chamado de arquivo.

Java: Arquivos e fluxos de dados

Figura 1. Representação dos fluxos de entrada e saída de dados (streams)

Um programa Java abre um arquivo instanciando, criando e associando um objeto ao fluxo de dados. As classes utilizadas para criar esses objetos serão discutidas mais adiante. De concreto, o Java cria três objetos de fluxo que são associados a dispositivos de entrada ou saída sempre que um programa inicia a execução:

System.in, objeto de fluxo de entrada padrão, normalmente utilizado pelo programa para obter dados a partir do teclado;

System.out, objeto de fluxo de saída padrão, normalmente utilizado pelo programa para enviar resultados para a tela do computador; e

System.err, objeto de fluxo de erro padrão, normalmente utilizado pelo programa para gerar saída de mensagens de erro na tela.

Programas Java implementam o processamento de arquivos utilizando as classes do pacote java.io. A hierarquia de classes oferecida por este pacote, apresentada de forma parcial na Figura 2, é relativamente grande e complexa, oferecendo mais de 50 classes distintas para o processamento de entrada e saída em arquivos baseados em bytes e caracteres e arquivos de acesso aleatório. Os arquivos são abertos criando-se objetos através de uma das classes de fluxo, citando:

FileInputStream: para entrada baseada em bytes de um arquivo;

FileOutputStream: para saída baseada em bytes para um arquivo;

RandomAccessFile: para entrada e saída baseada em bytes de e para um arquivo;

FileReader: para entrada baseada em caracteres de um arquivo;

FileWriter: para saída baseada em caracteres para um arquivo.

Hierarquia parcial de classes do pacote java.io

Figura 2. Hierarquia parcial de classes do pacote java.io

Classe File

Inicialmente será apresentada a classe File, utilizada para recuperar informações sobre arquivos ou diretórios em disco. Os objetos da classe File não abrem arquivos de dados e também não fornecem capacidades de processamento de arquivos, apenas são utilizados para

especificar arquivos ou diretórios. Para instanciar um objeto da classe File deve-se escolher entre uma das quatro formas de construtores:

public File (String name) - especifica o name de um arquivo ou diretório para associar com o objeto instanciado;

public File (String pathtoname, String name) - utiliza o argumento pathtoname para localizar o arquivo ou diretório especificado por name;

public File (File directory, String name) - utiliza o objeto directory existente para localizar o arquivo ou diretório especificado por name;

public File (URI uri) - utiliza o objeto uri para localizar o arquivo. Um Uniform Resource Identifier (URI) é uma cadeia de caracteres usada para identificar um arquivo como um endereço da internet.

A classe File possui vários métodos que permitem recuperar informações sobre o objeto instanciado, como o tipo (arquivo ou diretório) do argumento informado, tamanho em bytes, data da última modificação, se existe permissão para leitura e gravação, citando:

boolean canRead() - retorna true se o aplicativo pode ler o arquivo especificado; false, caso contrário;

boolean canWrite() - retorna true se o aplicativo pode modificar o arquivo especificado; false, caso contrário;

boolean delete() - exclui o arquivo ou diretório especificado pelo aplicativo retornando true se a exclusão foi realizada com sucesso; false, caso contrário;

boolean exits() - retorna true se o nome usado como argumento no construtor File indica um arquivo ou diretório existente; false, caso contrário;

String getAbsolutePath() - retorna uma String com o caminho absoluto do arquivo ou diretório. Um caminho absoluto contém o caminho completo com todos os diretórios desde o diretório-raiz até o arquivo ou o diretório especificado;

String getName() - retorna uma String com o nome do arquivo ou diretório;

String getParent() - retorna uma String com o diretório-pai do arquivo ou diretório;

String getPath() - retorna uma String com o caminho do arquivo ou diretório;

boolean isFile() - retorna true se o nome usado como argumento no construtor File é um arquivo; false, caso contrário;

boolean isDirectory() - retorna true se o nome usado como argumento no construtor File é um diretório; false, caso contrário;

long lastModified() - retorna um inteiro longo que representa a data/hora em que o arquivo ou diretório foi modificado pela última vez;

Dia 21 - Arquivo

long length() - retorna o comprimento do arquivo em bytes. Se for um diretório, o valor 0 será retornado;

String[] list() - retorna um array de strings com os nomes de arquivos e diretórios que representam o conteúdo de um diretório. Retorna null se o objeto File for um arquivo;

boolean mkdir() - cria o diretório especificado pelo aplicativo retornando true se a operação foi realizada com sucesso; false, caso contrário;

boolean mkdirs() - cria toda a estrutura do diretório especificado pelo aplicativo retornando true se a operação foi realizada com sucesso; false, caso contrário.

A classe Exemplo1, apresentada na Listagem 1, utiliza um objeto instanciado a partir da classe File para recuperar informações de um arquivo ou diretório.

```
import java.io.File;
import java.util.Scanner;

public class Exemplo1 {

 public static void main(String[] args) {
  Scanner ler = new Scanner(System.in);

  System.out.printf("Informe o nome de um arquivo ou diretório:\n");
  String nome = ler.nextLine();

  File objFile = new File(nome);
  if (objFile.exists()) {
    if (objFile.isFile()) {
      System.out.printf("\nArquivo (%s) existe - tamanho: %d bytes\n",
        objFile.getName(),  objFile.length());
    }
    else {
     System.out.printf("\nConteúdo do diretório:\n");
     String diretorio[] = objFile.list();
     for (String item: diretorio) {
       System.out.printf("%s\n", item);
```

```
        }
       }
    } else System.out.printf("Erro: arquivo ou diretório informado não existe!\n");
  }
}
```

Listagem 1. Recuperando as informações de um arquivo ou diretório

A execução do código fonte da Listagem 1 solicita ao usuário final o nome do arquivo ou diretório, instancia o objeto File e verifica se o arquivo ou diretório informado existe. Se o objeto instanciado existe, são exibidas no fluxo de saída padrão as informações recuperadas, para um arquivo: nome e tamanho em bytes; para um diretório: caminho e a lista com os nomes dos arquivos e diretórios que representam o seu conteúdo. A Figura 3 ilustra a execução da classe Exemplo1 na recuperação de informações de um arquivo enquanto que a Figura 4 mostra informações recuperadas de um diretório.

Java: Arquivos e fluxos de dados

Figura 3. Executando a classe Exemplo1 na recuperação de informações de um arquivo

Java: Arquivos e fluxos de dados

Figura 4. Executando a classe Exemplo1 na recuperação de informações de um diretório

A seguir serão abordadas as classes de fluxo do pacote java.io que permitem a um programa Java realizar o processamento, ler e gravar dados, em arquivos.

Processamento em arquivos

A interação de um programa com um dispositivo através de arquivos passa por três etapas: abertura ou criação de um arquivo, leitura ou gravação de dados e fechamento do arquivo.

Os arquivos poderão ter os seus dados acessados para leitura ou gravação na forma sequencial, direta ou indexada. Para o propósito deste artigo não será abordada a forma de acesso indexada.

Em um arquivo de acesso sequencial o processo de leitura e gravação é feito de forma contínua, um byte ou caractere após o outro a partir do início. O acesso direto permite a localização imediata a cada um dos dados, desde que se conheça a sua posição no arquivo.

Fluxo de dados baseado em bytes (arquivo binário)

Dia 21 - Arquivo

Primeiramente será demonstrada a criação e a gravação de dados em um arquivo binário sequencial. Como mencionado anteriormente, esses são arquivos em que os dados são armazenados no formato binário de forma contínua a partir do seu início.

A Listagem 2 apresenta uma forma de utilizar as classes FileOutputStream e DataOutputStream na criação e gravação de dados baseada em bytes para um arquivo binário. Os métodos writeUTF(), writeChar(), writeInt() e writeDouble() da classe DataOutputStream são aplicados na gravação de bytes que representam valores do tipo literal, caractere, inteiro e real, respectivamente.

```
import java.io.DataOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

public class Exemplo2 {

  public static void main(String[] args) throws FileNotFoundException, IOException {
    Scanner ler = new Scanner(System.in);
    String nome;
    char sexo;
    int idade, altura;
    double pc; // peso corporal

    FileOutputStream arq = new FileOutputStream("d:\\arquivo.dat");
    DataOutputStream gravarArq = new DataOutputStream(arq);

    System.out.printf("Informe o seu nome:\n");
    nome = ler.nextLine();

    System.out.printf("\nInforme o seu sexo (M/F).....: ");
    sexo = (char)System.in.read();
```

```
   System.out.printf("Informe a sua idade.....: ");
   idade = ler.nextInt();

   System.out.printf("Informe o seu peso corporal (em kg): ");
   pc = ler.nextDouble();

   System.out.printf("Informe a sua altura (em cm).....: ");
   altura = ler.nextInt();

   gravarArq.writeUTF(nome);
   gravarArq.writeChar(sexo);
   gravarArq.writeInt(idade);
   gravarArq.writeDouble(pc);
   gravarArq.writeInt(altura);

   arq.close();

   System.out.printf("\nDados gravados com sucesso em \"d:\\arquivo.dat\".\n");
  }
}
```

Listagem 2. Gravando dados em um arquivo binário

No código fonte da Listagem 2, o arquivo externo arquivo.dat é aberto para operações de saída através do objeto arq instanciado e criado a partir da classe FileOutputStream. Já o objeto de gravação gravarArq é associado a um fluxo de saída de dados baseado em bytes através da classe DataOutputStream. Definido o arquivo binário externo, foram implementados comandos para a entrada dos dados: nome, sexo, idade, peso corporal e altura. A seguir estes dados são gravados (write) no arquivo, que é fechado através do método close(). A Figura 5 ilustra a execução da classe Exemplo2.

Java: Arquivos e fluxos de dados

Figura 5. Executando a classe Exemplo2 na gravação de dados em um arquivo binário

Para Deitel & Deitel: os dados são armazenados em arquivos de forma que possam ser recuperados para processamento quando necessários. Nos arquivos binários os dados são

recuperados, de forma contínua a partir do seu início, no formato binário e traduzidos pelo sistema em valores numéricos ou literais.

A Listagem 3 apresenta uma forma de utilizar as classes FileInputStream e DataInputStream na abertura e leitura de dados baseada em bytes de um arquivo binário sequencial. Os métodos readUTF(), readChar(), readInt() e readDouble() da classe DataInputStream são aplicados na leitura de bytes que representam valores do tipo literal, caractere, inteiro e real, respectivamente.

```
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class Exemplo3 {

  public static void main(String[] args) throws IOException {
    String nome;
    char sexo;
    int idade, altura;
    double pc;  // peso corporal
    double GEB; // gasto energético basal

    FileInputStream arq = new FileInputStream("d:\\arquivo.dat");
    DataInputStream lerArq = new DataInputStream(arq);

    nome = lerArq.readUTF();
    sexo = lerArq.readChar();
    idade = lerArq.readInt();
    pc = lerArq.readDouble();
    altura = lerArq.readInt();

    // calculando o gasto energético basal
    if ((sexo == 'M') || (sexo == 'm'))
```

```
      GEB = 66.47 + (13.75 * pc) + (5 * altura) - (6.76 * idade);
    else
      GEB = 655.1 + (9.56 * pc) + (1.85 * altura) - (4.67 * idade);

    System.out.printf("Nome.....: %s\n", nome);
    System.out.printf("Sexo.....: %c\n", sexo);
    System.out.printf("Idade.....: %d anos\n", idade);
    System.out.printf("Peso Corporal.....: %.2f kgs\n", pc);
    System.out.printf("Altura.....: %d cm\n", altura);
    System.out.printf("Gasto Energético Basal: %.0f kcal/dia\n", GEB);

    arq.close();
  }
}
```

Listagem 3. Lendo dados de um arquivo binário

Na aplicação Java da Listagem 3, o arquivo externo arquivo.dat, criado no diretório raiz d, é aberto para operações de entrada através do objeto arq instanciado e criado a partir da classe FileInputStream. Já o objeto de leitura lerArq é associado a um fluxo de entrada de dados baseado em bytes através da classe DataInputStream. Definido o arquivo binário externo, foram implementados os comandos para leitura (read) dos dados: nome, sexo, idade, peso corporal e altura que estão gravados no arquivo. Em seguida estes dados são utilizados para calcular o gasto energético basal (GEB) através da fórmula desenvolvida por Haris e Benedict. O GEB estima a quantidade de energia diária em quilocalorias (kcal) para a manutenção das funções vitais do organismo de uma criança ou adulto. A Figura 6 ilustra a execução da classe Exemplo3.

Exercicio

Responda a alternativa correta de acordo com o exercicio

```
public class Exemplo4 {

  public static void main(String[] args) throws IOException {
    Scanner ler = new Scanner(System.in);
    int i, n;
```

Dia 21 - Arquivo

```
   System.out.printf("Informe o número para a tabuada:\n");
   n = ler.nextInt();

   FileWriter arq = new FileWriter("d:\\tabuada.txt");
   PrintWriter gravarArq = new PrintWriter(arq);

   gravarArq.printf("+--Resultado--+%n");
   for (i=1; i<=10; i++) {
     gravarArq.printf("| %2d * %d = %2d |%n", i, n, (i*n));
   }
   gravarArq.printf("+-------------+%n");

   arq.close();

   System.out.printf("\nTabuada do %d foi gravada com sucesso em \"d:\\tabuada.txt\".\n",
n);
  }
}
```

a) Não e aberto para operação através do objeto

b) No código fonte o arquivo externo linha.txt é aberto para operações de saída através do objeto

c) No código fonte o arquivo externo tabuada.txt é aberto para operações de saída através do objeto

d)nem uma das

2) Dado:

```
3. import java.util. *;
4. classe pública Magellan {
5. public static void main (String [] args) {
6. TreeMap <String, String> myMap = novo TreeMap <String, String> ();
7. myMap.put ("a", "apple"); myMap.put ("d", "data");
8. myMap.put ("f", "fig"); myMap.put ("p", "pêra");
9. System.out.println ("1º depois da manga:" + // sop 1
```

10. myMap.higherKey ("f"));

11. System.out.println ("1º depois da manga:" + // sop 2

12. myMap.ceilingKey ("f"));

13. System.out.println ("1º depois da manga:" + // sop 3

14. myMap.floorKey ("f"));

15. SortedMap <String, String> sub = new TreeMap <String, String> ();

16. sub = myMap.tailMap ("f");

17. System.out.println ("1º depois da manga:" + // sop 4

18. sub.firstKey ());

19.}

20.}

Qual das instruções System.out.println produzirá a saída primeiro após manga: p?

(Escolha todas as opções aplicáveis.)

A. sop 1

B. sop 2

C. sop 3

D. sop 4

3) Dado:

3. import java.util. *;

4. public class GeoCache {

5. public static void main (String [] args) {

6. String [] s = {"map", "pen", "marble", "key"};

7. Otelo o = novo Otelo ();

8. Arrays.sort (s, o);

Respostas do autoteste 659

660 Capítulo 7: Genéricos e coleções

9. para (String s2: s) System.out.print (s2 + "");

10. System.out.println (Arrays.binarySearch (s, "map"));

11.}

12. classe estática Othello implementa Comparator <String> {

13. public int compare (String a, String b) {return b.compareTo (a); }

14.}

15.}

Quais são verdadeiras? (Escolha todas as opções aplicáveis.)

A. A compilação falha

B. A saída conterá 1

C. A saída conterá um 2

D. A saída conterá um -1

Prova

1)

```
1. class Clidder {
2. private final void flipper() { System.out.println("Clidder"); }
3. }
4. public class Clidlet extends Clidder {
5. public final void flipper() { System.out.println("Clidlet"); }
6. public static void main(String [] args) {
7. new Clidlet().flipper();
8. } }
```

Qual é o resultado?

A. Clidlet

B. Clidder

C. Clidder

Clidlet

D. Clidlet

E. Compilador falha

2) Quais afirmações são verdadeiras? (Escolha todas as opções aplicáveis.)

A. Coesão é o princípio OO mais associado a ocultar detalhes de implementação

B. Coesão é o princípio OO mais associado a garantir que as classes saibam

sobre outras classes apenas por meio de suas APIs

C. Coesão é o princípio OO mais intimamente associado a garantir que uma classe seja

projetado com um propósito único e bem focado

D. Coesão é o princípio OO mais intimamente associado a permitir que um único objeto seja

visto como tendo muitos tipos

3)

Dado o seguinte,

```
1. classe X {void do1 () {}}
2. classe Y estende X {void do2 () {}}
3 -
4. classe Chrome {
    1.  public static void main (String [] args) {
    2.  X x1 = novo X ();
    3.  X x2 = novo Y ();
    9.  Y y1 = novo Y ();
    10. // insira o código aqui
    4.  }
    5.  }
```

Que, inserido na linha 9, irá compilar? (Escolha todas as opções aplicáveis.)

A. x2.do2 ();

B. (Y) x2.do2 ();

C. ((Y) x2) .do2 ();

D. Nenhuma das declarações acima irá compilar