14

Arquivos e fluxos

OBJETIVOS

Neste capítulo, você aprenderá:

- **Como criar, ler, gravar e atualizar arquivos.**
- **Como utilizar a classe `File` para recuperar informações sobre arquivos e diretórios.**
- **A hierarquia de classes para fluxo de entrada/saída do Java.**
- **As diferenças entre arquivos de texto e arquivos binários.**
- **Processamento de arquivos de acesso seqüencial e de acesso aleatório.**
- **Como utilizar as classes `Scanner` e `Formatter` para processar arquivos de texto.**
- **Como utilizar as classes `FileInputStream` e `FileOutputStream`.**
- **Como utilizar um diálogo de `JFileChooser`.**
- **Como utilizar as classes `FileInputStream` e `FileOutputStream`.**
- **Como utilizar a classe `RandomAccessFile`.**

14.1 Introdução

- **O armazenamento dos dados em variáveis e arrays é temporário.**
- **Computadores utilizam arquivos para armazenamento de longo prazo de grandes volumes de dados, mesmo depois de os programas que criaram os dados terminarem.**
- ***Dados persistentes* – existem além da duração da execução do programa.**
- **Arquivos armazenados nos *dispositivos de armazenamento secundários*.**
- ***Fluxo* – dados ordenados lidos de ou gravados em um arquivo.**

14.2 Hierarquia de dados

- **Computadores processam todos os itens de dados como combinações de zeros e uns.**
- **Bit, o menor item de dados em um computador, pode ter valores 0 ou 1.**
- **Byte – 8 bits.**
- **Caracteres – o maior item de dados.**
  - **Consistem em dígitos decimais, letras e símbolos especiais.**
  - **Conjunto de caracteres – o conjunto de todos os caracteres utilizados para escrever programas e representar itens de dados.**
    - **Unicode – caracteres compostos de dois bytes.**
    - **ASCII.**

14.2 Hierarquia de dados

- **Um campo – um grupo de caracteres ou bytes que carregam um significado.**
- **Registro – um grupo de campos relacionados.**
- **Arquivo – um grupo de registros relacionados.**
- **Os itens de dados processados pelos computadores formam uma hierarquia de dados que se torna maior e mais complexa partindo dos *bits* até os arquivos.**
- **Chave de registro – identifica um registro como pertencente a uma pessoa ou entidade particular – utilizada para fácil recuperação de registros específicos.**
- **Arquivo seqüencial – arquivo em que os registros são armazenados pela ordem do campo chave de registro.**
- **Banco de dados – um grupo de arquivos relacionados.**
- **Sistema de gerenciamento de bancos de dados – uma coleção dos programas projetada para criar e gerenciar bancos de dados.**

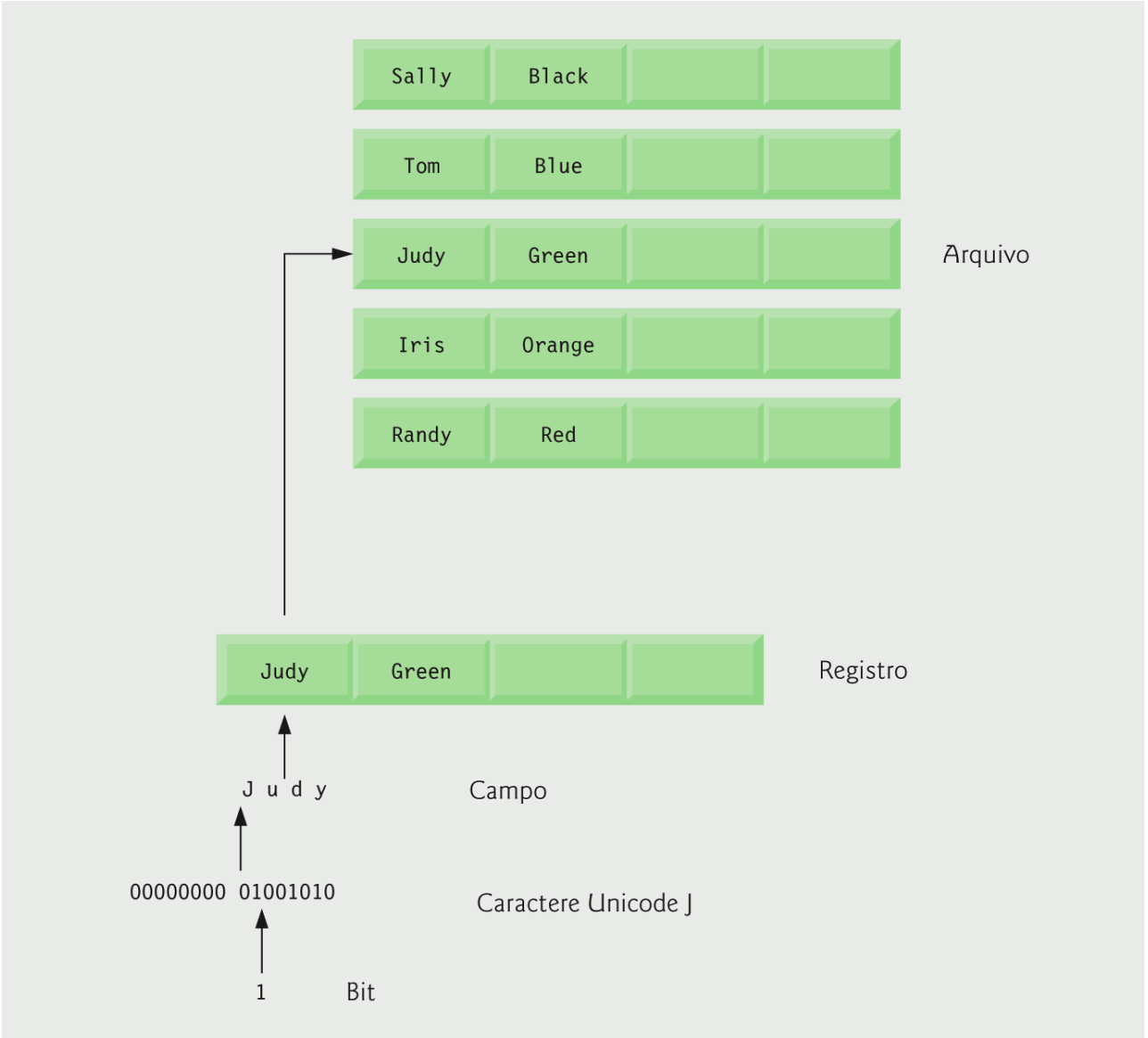


Figura 14.1 | Hierarquia de dados.

14.3 Arquivos e fluxos

- **O Java vê cada arquivo como um *fluxo* seqüencial de bytes.**
- **O sistema operacional fornece um mecanismo para determinar o final do arquivo.**
  - **Como um marcador de fim do arquivo ou uma contagem do total de bytes no arquivo que é registrado nos dados mantidos na estrutura do sistema administrativo.**
  - **Um programa Java que processa um fluxo de bytes recebe uma indicação do sistema operacional sobre quando o programa alcança o final do fluxo.**

14.3 Arquivos e fluxos

- **Fluxos de arquivos:**
  - **Fluxos baseados em bytes – representam dados no formato binário.**
    - **Arquivos binários – criados a partir de fluxos baseados em bytes, lidos por um programa que converte os dados em formato legível por humanos.**
  - **Fluxos baseados em caracteres – armazenam os dados como uma seqüência de caracteres.**
    - **Arquivos de texto – criados a partir de fluxos baseados em caracteres, eles podem ser lidos por editores de textos.**
- **O Java abre o arquivo criando um objeto e associando um fluxo a ele.**
- **Fluxos-padrão – cada fluxo pode ser redirecionado:**
  - **`System.in` – objeto do fluxo de entrada-padrão, ele pode ser redirecionado com o método `setIn`.**
  - **`System.out` – objeto do fluxo de saída-padrão, ele pode ser redirecionado com o método `setOut`.**
  - **`System.err` – objeto do fluxo de erro-padrão, ele pode ser redirecionado com o método `setErr`.**

14.3 Arquivos e fluxos (*Cont.*)

- Classes `java.io`:
  - `FileInputStream` e `FileOutputStream` – E/S baseada em bytes.
  - `FileReader` e `FileWriter` – E/S baseada em caracteres.
  - `ObjectInputStream` e `ObjectOutputStream` – os objetos dessas classes podem ser utilizados para E/S de objetos ou variáveis de tipos de dados primitivos.
  - `File` – útil para obter informações sobre arquivos e diretórios.
- Classes `Scanner` e `Formatter`
  - `Scanner` – pode ser utilizada para ler facilmente os dados em um arquivo.
  - `Formatter` – pode ser utilizada para gravar facilmente dados em um arquivo.

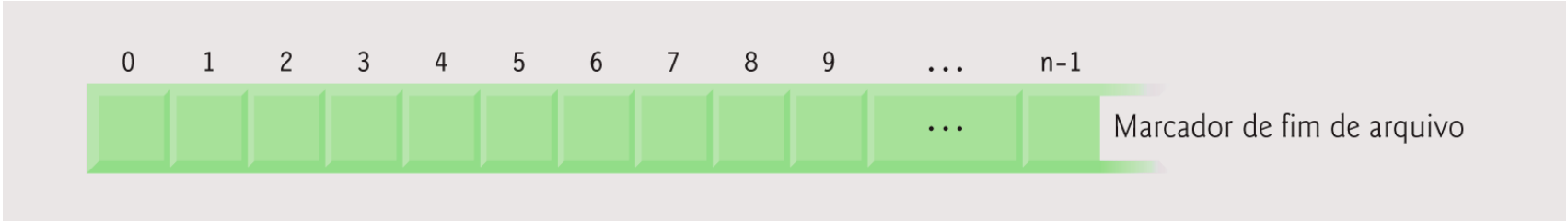


Figura 14.2 | Visualização do Java de um arquivo de *n* bytes.

14.4 Classe `File`

- **Classe `File` – útil para recuperar informações sobre arquivos e diretórios no disco.**
- **Os objetos da classe `File` não abrem arquivos nem fornecem capacidades de processamento de arquivos.**

Criando objetos `File`

- **A classe `File` fornece quatro construtores:**
  1. **Recebe `String` que especifica nome e caminho (localização do arquivo no disco).**
  2. **Recebe duas `String`s: a primeira especificando o caminho e a segunda especificando o nome do arquivo.**
  3. **Recebe o objeto `File` que especifica o caminho e `String` que especifica o nome do arquivo.**
  4. **Recebe o objeto `URI` que especifica o nome e a localização do arquivo.**
- **Diferentes tipos de caminhos:**
  - ***Caminho absoluto* – contém todos os diretórios desde o diretório-raiz que levam a um arquivo ou diretório específico.**
  - ***Caminho relativo* – normalmente inicia do diretório em que o aplicativo começou a execução.**

Método	Descrição
`boolean canRead()`	Retorna `true` se um arquivo for legível pelo aplicativo atual.
`boolean canWrite()`	Retorna `true` se um arquivo for gravável pelo aplicativo atual.
`boolean exists()`	Retorna `true` se o nome especificado como o argumento para o construtor File for um arquivo ou diretório no caminho especificado.
`boolean isFile()`	Retorna `true` se o nome especificado como o argumento para o construtor `File` for um arquivo.
`boolean isDirectory()`	Retorna `true` se o nome especificado como o argumento para o construtor `File` for um diretório.
`boolean isAbsolute()`	Retorna `true` se os argumentos especificados para o construtor `File` indicarem um caminho absoluto para um arquivo ou diretório.

Figura 14.3 | Métodos `File`. (Parte 1 de 2.)

Método	Descrição
`String getAbsolutePath()`	Retorna uma string com o caminho absoluto do arquivo ou diretório.
`String getName()`	Retorna uma string com o nome do arquivo ou diretório.
`String getPath()`	Retorna uma string com o caminho do arquivo ou diretório.
`String getParent()`	Retorna uma string com o diretório-pai do arquivo ou diretório (isto é, o diretório em que o arquivo ou diretório pode ser localizado).
`long length()`	Retorna o comprimento do arquivo, em bytes. Se o objeto `File` representar um diretório, 0 é retornado.
`long lastModified()`	Retorna uma representação dependente de plataforma da data/hora em que o arquivo ou diretório for modificado pela última vez. O valor retornado é útil somente para comparação com outros valores retornados por esse método.
`String[] list()`	Retorna um array de strings que representam o conteúdo de um diretório. Retorna `null` se o objeto `File` não representar um diretório.

Figura14.3 | Métodos `File`. (Parte 2 de 2.)

Dica de prevenção de erro 14.1

O método `File` utiliza `isFile` para determinar se um objeto `File` representa um arquivo (não um diretório) antes de tentar abrir o arquivo.

Demonstrando a classe `File`

- **Métodos `File` comuns:**
  - **`exists` – retorna `true` se o arquivo existir onde especificado.**
  - **`isFile` – retorna `true` se `File` for um arquivo, não um diretório.**
  - **`isDirectory` – retorna `true` se `File` for um diretório.**
  - **`getPath` – retorna o caminho de arquivo como uma string.**
  - **`list` – recupera o conteúdo de um diretório.**
- **Caractere separador – utilizado para separar diretórios e arquivos em um caminho.**
  - **O Windows utiliza `\`.**
  - **O UNIX utiliza `/`.**
  - **O Java processa ambos os caracteres. `File.pathSeparator` pode ser utilizado para obter o caractere separador adequado do computador local**

Resumo

FileDemonstration

.java

(1 de 2)

```
1  // Fig. 14.4: FileDemonstration.java
2  // Demonstrando a classe File.
3  import java.io.File;
4
5  public class FileDemonstration
6  {
7     // exibe informações sobre o arquivo que o usuário especifica
8     public void analyzePath( String path )
9     {
10       // cria o objeto File com base
11       File name = new File( path );
12
13       if ( name.exists() ) // se o nome existir, gera saída das informações sobre ele
14       {
15          // exibe informações sobre o arquivo (ou diretório)
16          System.out.printf(
17             "%s%s\n%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s",
18             name.getName(), " exists",
19             ( name.isFile() ? "is a file" : "is not a file" ),
20             ( name.isDirectory() ? "is a directory" :
21                "is not a directory" ),
22             ( name.isAbsolute() ? "is absolute path" :
23                "is not absolute path" ), "Last modified: ",
24             name.lastModified(), "Length: ", name.length(),
25             "Path: ", name.getPath(), "Absolute path: ",
26             name.getAbsolutePath(), "Parent: ", name.getParent() );
27
```

Retorna `true` se o arquivo ou diretório especificado existir

Resumo

```
28          if ( name.isDirectory() ) // gera listagem de diretório
29          {
30             String directory[] = name.list();
31             System.out.println( "\n\nDirectory contents:\
32
33             for ( String directoryName : directory )
34                System.out.printf( "%s\n", directoryName );
35          } // fim de
36       } // fim do if externo
37       else // não for arquivo ou diretório, gera saída da mensagem de erro
38       {
39          System.out.printf( "%s %s", path, "does not exist." );
40       } // fim de else
41    } // fim do método analyzePath
42 } // fim da classe FileDemonstration
```

Retorna `true` se `File` for um diretório, não um arquivo

Recupera e exibe o conteúdo do diretório

(2 de 2)

Resumo

FileDemonstrationTest.java

(1 de 3)

```
1  // Fig. 14.5: FileDemonstrationTest.java
2  // Testando a classe FileDemonstration.
3  import java.util.Scanner;
4
5  public class FileDemonstrationTest
6  {
7     public static void main( String args[] )
8     {
9        Scanner input = new Scanner( System.in );
10       FileDemonstration application = new FileDemonstration();
11
12       System.out.print( "Enter file or directory name here: " );
13       application.analyzePath( input.nextLine() );
14    } // fim de main
15 } // fim da classe FileDemonstrationTest
```

```
Enter file or directory name here: C:\Program Files\Java\jdk1.5.0\demo\jfc
jfc exists
is not a file
is a directory
is absolute path
Last modified: 1083938776645
Length: 0
Path: C:\Program Files\Java\jdk1.5.0\demo\jfc
Absolute path: C:\Program Files\Java\jdk1.5.0\demo\jfc
Parent: C:\Program Files\Java\jdk1.5.0\demo

Directory contents:

CodePointIM
FileChooserDemo
Font2DTest
Java2D
Metalworks
Notepad
SampleTree
Stylepad
SwingApplet
SwingSet2
TableExample
```

Resumo

FileDemonstration

Test.java

(2 de 3)

```
Enter file or directory name here:
C:\Program Files\Java\jdk1.5.0\demo\jfc\Java2D\readme.txt
readme.txt exists
is a file
is not a directory
is absolute path
Last modified: 1083938778347
Length: 7501
Path: C:\Program Files\Java\jdk1.5.0\demo\jfc\Java2D\readme.txt
Absolute path: C:\Program Files\Java\jdk1.5.0\demo\jfc\Java2D\readme.txt
Parent: C:\Program Files\Java\jdk1.5.0\demo\jfc\Java2D
```

Resumo

FileDemonstration

Test.java

(3 de 3)

Erro comum de programação 14.1

Utilizar \ como um separador de diretório em vez de \\ em uma literal de string é um erro de lógica. Uma \ simples indica que a \ seguida pelo próximo caractere representa uma seqüência de escape. Utilize \\ para inserir uma \ em uma literal de string.

14.5 Arquivos de texto de acesso seqüencial

- **Os registros são armazenados na ordem por campo de chave de registro.**
- **Podem ser criados como arquivos de texto ou arquivos binários.**

14.5.1 Criando um arquivo de texto de acesso seqüencial

- **O Java não impõe nenhuma estrutura a um arquivo; registros não existem como parte da linguagem Java.**
- **O programador deve estruturar os arquivos.**
- **A classe `Formatter` pode ser utilizada para abrir um arquivo de texto para gravar:**
  - **Passa o nome de arquivo para o construtor.**
  - **Se o arquivo não existir, ele será criado.**
  - **Se o arquivo já existir, o conteúdo será truncado (descartado).**
  - **Utiliza o método `format` para gravar texto formatado no arquivo.**
  - **Utiliza o método `close` para fechar o objeto `Formatter` (se esse método não for chamado, o SO normalmente fecha o arquivo quando o programa é fechado).**

14.5.1 Criando um arquivo de texto de acesso seqüencial (*Cont.*)

- **Possíveis exceções:**
  - `SecurityException` – **ocorre ao abrir o arquivo utilizando o objeto** `Formatter`**, se o usuário não tiver permissão para gravar dados no arquivo.**
  - `FileNotFoundException` – **ocorre ao abrir o arquivo utilizando o objeto** `Formatter`**, se o arquivo não puder ser localizado e um novo arquivo não puder ser criado.**
  - `NoSuchElementException` – **ocorre quando uma entrada inválida é lida por um objeto** `Scanner`**.**
  - `FormatterClosedException` – **ocorre quando é feita uma tentativa de gravar em um arquivo utilizando um objeto** `Formatter` **já fechado.**

Resumo

AccountRecord.java

(1 de 3)

```
1  // Fig. 14.6: AccountRecord.java
2  // Uma classe que representa um registro das informações.
3  package com.deitel.jhtp6.ch14; // empacotada para reutilização
4
5  public class AccountRecord
6  {
7     private int account;
8     private String firstName;
9     private String lastName;
10    private double balance;
11
12    // construtor sem argumentos chama outro construtor com valores padrão
13    public AccountRecord()
14    {
15       this( 0, "", "", 0.0 ); // chama o construtor com quatro argumentos
16    } // fim do construtor de AccountRecord sem argumentos
17
18    // inicializa um registro
19    public AccountRecord( int acct, String first, String last, double bal )
20    {
21       setAccount( acct );
22       setFirstName( first );
23       setLastName( last );
24       setBalance( bal );
25    } // fim do construtor de AccountRecord de quatro argumentos
26
```

Resumo

AccountRecord.java

(2 de 3)

```
27    // configura o número de conta
28    public void setAccount( int acct )
29    {
30       account = acct;
31    } // fim do método setAccount
32
33    // obtém número da conta
34    public int getAccount()
35    {
36       return account;
37    } // fim do método getAccount
38
39    // configura o nome
40    public void setFirstName( String first )
41    {
42       firstName = first;
43    } // fim do método setFirstName
44
45    // obtém o nome
46    public String getFirstName()
47    {
48       return firstName;
49    } // fim do método getFirstName
50
51    // configura o sobrenome
52    public void setLastName( String last )
53    {
54       lastName = last;
55    } // fim do método setLastName
56
```

Resumo

AccountRecord.java

(3 de 3)

```
57    // configura o sobrenome
58    public String getLastName()
59    {
60       return lastName;
61    } // fim do método getLastName
62
63    // obtém o saldo
64    public void setBalance( double bal )
65    {
66       balance = bal;
67    } // fim do método setBalance
68
69    // obtém o saldo
70    public double getBalance()
71    {
72       return balance;
73    } // fim do método getBalance
74 } // fim da classe AccountRecord
```

Resumo

```
1  // Fig. 14.7: CreateTextFile.java
2  // Gravando dados em um arquivo de texto com classe Formatter.
3  import java.io.FileNotFoundException;
4  import java.lang.SecurityException;
5  import java.util.Formatter;
6  import java.util.FormatterClosedException;
7  import java.util.NoSuchElementException;
8  import java.util.Scanner;
9
10 import com.deitel.jhtp6.ch14.AccountRecord;
11
12 public class CreateTextFile
13 {
14    private Formatter output; // objeto usado p/ gerar saída de texto p/ o arquivo
15
16    // permite ao usuário abrir o arquivo
17    public void openFile()
18    {
19       try
20       {
21          output = new Formatter( "clientes.txt" );
22       } // fim de try
23       catch ( SecurityException securityException )
24       {
25          System.err.println(
26             "You do not have write access to this file." );
27          System.exit( 1 );
28       } // fim de catch
```

Utilizado para gravar dados no arquivo

Utilizado para recuperar a entrada do usuário

Objeto utilizado para gerar saída para arquivo

Abre `clientes.txt` para gravação

`CreateTextFile`

`.java`

(1 de 4)

Resumo

CreateTextFile

.java

```
29       catch ( FileNotFoundException filesNotFoundException )
30       {
31          System.err.println( "Error creating file." );
32          System.exit( 1 );
33       } // fim de catch
34    } // fim do método openFile
35
36    // adiciona registros ao arquivo
37    public void addRecords()
38    {
39       // objeto a ser gravado no arquivo
40       AccountRecord record = new AccountRecord();
41
42       Scanner input = new Scanner( System.in );
43
44       System.out.printf( "%s\n%s\n%s\n%s\n\n",
45          "To terminate input, type the end-of-file indicator ",
46          "when you are prompted to enter input.",
47          "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
48          "On Windows type <ctrl> z then press Enter" );
49
50       System.out.printf( "%s\n%s",
51          "Enter account number (> 0), first name, last name and balance.",
52          "? " );
53
```

Cria `AccountRecord` para ser preenchido com a entrada do usuário

Cria `Scanner` para recuperar a entrada do usuário

(2 de 4)

Resumo

CreateTextFile

.java

```
54       while ( input.hasNext() ) // faz um loop até o indicador de fim de arquivo
55       {
56          try // gera saída dos valores para o arquivo
57          {
58             // recupera os dados para saída
59             record.setAccount( input.nextInt() ); // lê o número de conta
60             record.setFirstName( input.next() ); // lê o nome
61             record.setLastName( input.next() ); // lê o sobrenome
62             record.setBalance( input.nextDouble() ); // lê o saldo
63
64             if ( record.getAccount() > 0 )
65             {
66                // grava novo registro
67                output.format( "%d %s %s %.2f\n", record.getAccount(),
68                   record.getFirstName(), record.getLastName(),
69                   record.getBalance() );
70             } // fim de if
71             else
72             {
73                System.out.println(
74                   "Account number must be greater than 0." );
75             } // fim de else
76          } // fim de try
77          catch ( FormatterClosedException formatterClosedException )
78          {
79             System.err.println( "Error writing to file." );
80             return;
81          } // fim de catch
```

Loop enquanto o usuário insere a entrada

Recupera entrada, armazena dados em `AccountRecord`

Grava informações de `AccountRecord` no arquivo

Arquivo fechado ao tentar gravar nele

Resumo

```
82          catch ( NoSuchElementException elementException )
83          {
84             System.err.println( "Invalid input. Please try again." );
85             input.nextLine(); // descarta entr
86          } // end catch
87
88          System.out.printf( "%s %s\n%s", "Enter account number (>0),",
89             "first name, last name and balance.", "? " );
90       } // fim de while
91    } // fim do método addRecords
92
93    // fecha o arquivo
94    public void closeFile()
95    {
96       if ( output != null )
97          output.close();
98    } // fim do método closeFile
99 } // fim da classe CreateTextFile
```

Erro com entrada inserida pelo usuário

Arquivo fechado

CreateTextFile

.java

(4 de 4)

Sistema operacional	Combinação de teclas
UNIX/Linux/Mac OS X	***<return> <ctrl> d***
Windows	***<ctrl> z***

Figura14.8 | Combinações de teclas de fim de arquivo para vários sistemas operacionais famosos.

Resumo

CreateTextFileTest.java

(1 de 2)

```
1  // Fig. 14.9: CreateTextFileTest.java
2  // Testando a classe CreateTextFile.
3
4  public class CreateTextFileTest
5  {
6     public static void main( String args[] )
7     {
8        CreateTextFile application = new CreateTextFile();
9
10       application.openFile();
11       application.addRecords();
12       application.closeFile();
13    } // fim de main
14 } // fim da classe CreateTextFileTest
```

```
To terminate input, type the end-of-file indicator
when you are prompted to enter input.
On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
On Windows type <ctrl> z then press Enter

Enter account number (> 0), first name, last name and balance.
? 100 Bob Jones 24.98
Enter account number (> 0), first name, last name and balance.
? 200 Steve Doe -345.67
Enter account number (> 0), first name, last name and balance.
? 300 Pam White 0.00
Enter account number (> 0), first name, last name and balance.
? 400 Sam Stone -42.16
Enter account number (> 0), first name, last name and balance.
? 500 Sue Rich 224.62
Enter account number (> 0), first name, last name and balance.
? ^Z
```

Resumo

CreateTextFileTest.java

(2 de 2)

Dados de exemplo			
100	Bob	Jones	24.98
200	Steve	Doe	-345.67
300	Pam	White	0.00
400	Sam	Stone	-42.16
500	Sue	Rich	224.62

Figura14.10 | Dados de exemplo para o programa na Figura 14.7.

14.5.2 Lendo dados a partir de um arquivo de texto de acesso seqüencial

- **Os dados são armazenados em arquivos de modo que eles possam ser recuperados para processamento quando necessário.**
- **O objeto `Scanner` pode ser utilizado para ler dados seqüencialmente em um arquivo de texto:**
  - **Passa o objeto `File`, que representa o arquivo a ser lido, para o construtor `Scanner`.**
  - **`FileNotFoundException` ocorre se o arquivo não puder ser localizado.**
  - **Os dados são lidos no arquivo utilizando os mesmos métodos como entrada de teclado – `nextInt`, `nextDouble`, `next` etc.**
  - **`IllegalStateException` ocorre se for feita uma tentativa de ler um objeto `Scanner` fechado.**

Resumo

ReadTextFile.java

(1 de 3)

```
1  // Fig. 14.11: ReadTextFile.java
2  // Esse programa lê um arquivo de texto e exibe cada registro.
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.lang.IllegalStateException;
6  import java.util.NoSuchElementException;
7  import java.util.Scanner;
8
9  import com.deitel.jhtp6.ch14.AccountRecord;
10
11 public class ReadTextFile
12 {
13    private Scanner input;
14
15    // permite ao usuário abrir o arquivo
16    public void openFile()
17    {
18       try
19       {
20          input = new Scanner( new File( "clients.txt" ) );
21       } // fim de try
22       catch ( FileNotFoundException fileNotFoundException )
23       {
24          System.err.println( "Error opening file." );
25          System.exit( 1 );
26       } // fim do catch
27    } // fim do método openFile
28
```

Abre `clients.txt` para leitura

Resumo

ReadTextFile.java

(2 de 3)

```
29    // lê registro a partir do arquivo
30    public void readRecords()
31    {
32       // objeto a ser gravado na tela
33       AccountRecord record = new AccountRecord();
34
35       System.out.printf( "%-10s%-12s%-12s%
36          "First Name", "Last Name", "Balan
37
38       try // lê registros do arquivo usand
39       {
40          while ( input.hasNext() )
41          {
42             record.setAccount( input.nextInt() ); // lê número da conta
43             record.setFirstName( input.next() ); // lê o nome
44             record.setLastName( input.next() ); // lê o sobrenome
45             record.setBalance( input.nextDouble() ); // lê o saldo
46
47             // exibe o conteúdo do registro
48             System.out.printf( "%-10d%-12s%-12s%10.2f\n",
49                record.getAccount(), record.getFirstName(),
50                record.getLastName(), record.getBalance() );
51          } // fim de while
52       } // fim de try
```

Cria `AccountRecord` para armazenar entrada proveniente de um arquivo

Enquanto houver dados a ler no arquivo

Lê os dados no arquivo, armazena-os em `AccountRecord`

Exibe o conteúdo de `AccountRecord`

Resumo

ReadTextFile.java

(3 de 3)

```
53       catch ( NoSuchElementException elementException )
54       {
55          System.err.println( "File improperly formed." );
56          input.close();
57          System.exit( 1 );
58       } // fim de catch
59       catch ( IllegalStateException stateException )
60       {
61          System.err.println( "Error reading from file." );
62          System.exit( 1 );
63       } // fim de catch
64    } // fim do método readRecords
65
66    // fecha o arquivo e termina o aplicativo
67    public void closeFile()
68    {
69       if ( input != null )
70          input.close(); // fecha o arquivo
71    } // fim do método closeFile
72 } // fim da classe ReadTextFile
```

Arquivo fechado

Resumo

ReadTextFileTest

.java

```
1  // Fig. 14.12: ReadTextFileTest.java
2  // Este programa testa a classe ReadTextFile.
3
4  public class ReadTextFileTest
5  {
6     public static void main( String args[] )
7     {
8        ReadTextFile application = new ReadTextFile();
9
10       application.openFile();
11       application.readRecords();
12       application.closeFile();
13    } // fim do main
14 } // fim da classe ReadTextFileTest
```

```
Account   First Name  Last Name      Balance
100       Bob         Jones            24.98
200       Steve       Doe            -345.67
300       Pam         White            0.00
400       Sam         Stone          -42.16
500       Sue         Rich           224.62
```

14.5.4 Atualizando arquivos de acesso seqüencial

- **Os dados em muitos arquivos seqüenciais não podem ser modificados sem o risco de destruir outros dados no arquivo.**
- **Dados antigos não podem ser sobrescritos se os novos dados não tiverem o mesmo tamanho.**
- **Registros em arquivos de acesso seqüencial normalmente não são atualizados no local. Em vez disso, geralmente o arquivo inteiro é regravado.**

14.6 Serialização de objeto

- **Com arquivos de texto, as informações do tipo de dados são perdidas.**
- **Serialização de objeto – o mecanismo para ler ou gravar um objeto inteiro em um arquivo.**
- **Objeto serializado – o objeto representado como uma seqüência de bytes, incluindo os dados do objeto e as informações sobre o objeto.**
- **Desserialização – recria um objeto na memória a partir dos dados no arquivo.**
- **A serialização e a desserialização são realizadas com as classes `ObjectInputStreams` e `ObjectOutputStream`, métodos `readObjects` e `writeObject`.**

14.6.1 Criando um arquivo de acesso seqüencial com a serialização de objeto
Definindo a classe `AccountRecordSerializable`

- **Interface `Serializable` – os programadores precisam declarar uma classe para que possam implementar a interface `Serializable` ou os objetos dessa classe não poderão ser gravados em um arquivo.**
- **Para abrir um arquivo para gravar objetos, crie um `FileOutputStream` empacotado por um `ObjectOutputStream`.**
  - **`FileOutputStream` fornece os métodos para gravar a saída baseada em bytes em um arquivo.**
  - **`ObjectOutputStream` utiliza `FileOutputStream` para gravar objetos em um arquivo.**
  - **O método `writeObject` de `ObjectOutputStream` grava um objeto no arquivo de saída.**
  - **O método `close` de `ObjectOutputStream` fecha os dois objetos.**

Resumo

AccountRecord

Serializable.java

```
1  // Fig. 14.17: AccountRecordSerializable.java
2  // Uma classe que representa um registro de informações.
3  package com.deitel.jhtp6.ch14; // empacotada para reutilização
4
5  import java.io.Serializable;
6
7  public class AccountRecordSerializable implements Serializable
8  {
9     private int account;
10    private String firstName;
11    private String lastName;
12    private double balance;
13
14    // construtor sem argumentos chama outro construtor
15    public AccountRecordSerializable()
16    {
17       this( 0, "", "", 0.0 );
18    } // fim do construtor de AccountRecordSerializable com quatro argumentos
19
20    // construtor com quatro argumentos inicializa um registro
21    public AccountRecordSerializable(
22       int acct, String first, String last, double bal )
23    {
24       setAccount( acct );
25       setFirstName( first );
26       setLastName( last );
27       setBalance( bal );
28    } // fim do construtor de AccountRecordSerializable com quatro argumentos
29
```

Interface `Serializable` especifica que objetos `AccountRecordSerializable` podem ser gravados em um arquivo

```
30    // configura o número de conta
31    public void setAccount( int acct )
32    {
33       account = acct;
34    } // fim do método setAccount
35
36    // obtém número da conta
37    public int getAccount()
38    {
39       return account;
40    } // fim do método getAccount
41
42    // configura o nome
43    public void setFirstName( String first )
44    {
45       firstName = first;
46    } // fim do método setFirstName
47
48    // obtém o nome
49    public String getFirstName()
50    {
51       return firstName;
52    } // fim do método getFirstName
53
54    // configura o sobrenome
55    public void setLastName( String last )
56    {
57       lastName = last;
58    } // fim do método setLastName
59
```

Resumo

AccountRecord

Serializable.java

(2 de 3)

```
60    // obtém o nome
61    public String getLastName()
62    {
63       return lastName;
64    } // fim do método getLastName
65
66    // configura o saldo
67    public void setBalance( double bal )
68    {
69       balance = bal;
70    } // fim do método setBalance
71
72    // obtém o saldo
73    public double getBalance()
74    {
75       return balance;
76    } // fim do método getBalance
77 } // fim da classe AccountRecordSerializable
```

Resumo

AccountRecord

Serializable.java

(3 de 3)

sumo

equential

File.java

(1 de 4)

```
1  // Fig. 14.18: CreateSequentialFile.java
2  // Gravando objetos seqüencialmente em um arquivo com a classe ObjectOutputStream.
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  import java.io.ObjectOutputStream;
6  import java.util.NoSuchElementException;
7  import java.util.Scanner;
8
9  import com.deitel.jhtp6.ch14.AccountRecordSerializable;
10
11 public class CreateSequentialFile
12 {
13    private ObjectOutputStream output; // gera saída de dados no arquivo
14
15    // permite que o usuário especifique o nome de arquivo
16    public void openFile()
17    {
18       try // abre o arquivo
19       {
20          output = new ObjectOutputStream(
21             new FileOutputStream( "clients.ser" ) );
22       } // fim do try
23       catch ( IOException ioException )
24       {
25          System.err.println( "Error opening file." );
26       } // fim do catch
27    } // fim do método openFile
28
```

A classe utilizada para criar fluxo de saída baseado em bytes

A classe utilizada para criar a saída para dados do objeto no fluxo baseado em bytes

Abre o arquivo `clients.ser` para gravação

```
29    // adiciona registros ao arquivo
30    public void addRecords()
31    {
32       AccountRecordSerializable record; // objeto a ser gravado no arquivo
33       int accountNumber = 0; // número da conta para o objeto de registro
34       String firstName; // nome para o objeto de registro
35       String lastName; // sobrenome para o objeto de registro
36       double balance; // saldo para o objeto de registro
37
38       Scanner input = new Scanner( System.in );
39
40       System.out.printf( "%s\n%s\n%s\n%s\n\n",
41          "To terminate input, type the end-of-file indicator ",
42          "when you are prompted to enter input.",
43          "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
44          "On Windows type <ctrl> z then press Enter" );
45
46       System.out.printf( "%s\n%s",
47          "Enter account number (> 0), first name, last name and balance.",
48          "? " );
49
50       while ( input.hasNext() ) // faz loop até o indicador de fim de arquivo
51       {
52          try // gera saída dos valores para o arquivo
53          {
54             accountNumber = input.nextInt(); // lê número da conta
55             firstName = input.next(); // lê o nome
56             lastName = input.next(); // lê o sobrenome
57             balance = input.nextDouble(); // lê o saldo
58
```

Resumo

CreateSequential File.java

(2 de 4)

Resumo

```
59            if ( accountNumber > 0 )
60            {
61               // cria o novo registro
62               record = new AccountRecordSerializable( accountNumber,
63                  firstName, lastName, balance );
64               output.writeObject( record ); // gera saída
65            } // fim do if
66            else
67            {
68               System.out.println(
69                  "Account number must be greater than 0." );
70            } // fim do else
71         } // fim do try
72         catch ( IOException ioException )
73         {
74            System.err.println( "Error writing to file." );
75            return;
76         } // fim do catch
77         catch ( NoSuchElementException elementException )
78         {
79            System.err.println( "Invalid input. Please try again." );
80            input.nextLine(); // descarta entrada para o usuário tentar de novo
81         } // fim do catch
82
83         System.out.printf( "%s %s\n%s", "Enter account number (>0),",
84            "first name, last name and balance.", "? " );
85      } // fim do while
86   } // fim do método addRecords
87
```

Grava o objeto de registro no arquivo

ential

File.java

(3 de 4)

```
88    // fecha o arquivo e termina o aplicativo
89    public void closeFile()
90    {
91       try // fecha o arquivo
92       {
93          if ( output != null )
94             output.close();
95       } // fim do try
96       catch ( IOException ioException )
97       {
98          System.err.println( "Error closing file." );
99          System.exit( 1 );
100      } // fim do catch
101   } // fim do método closeFile
102} // fim da classe CreateSequentialFile
```

Resumo

CreateSequential File.java

(4 de 4)

Resumo

CreateSequentialFileTest.java

```
1  // Fig. 14.19: CreateSequentialFileTest.java
2  // Testando a classe CreateSequentialFile.
3
4  public class CreateSequentialFileTest
5  {
6     public static void main( String args[] )
7     {
8        CreateSequentialFile application = new CreateSequentialFile();
9
10       application.openFile();
11       application.addRecords();
12       application.closeFile();
13    } // fim do main
14 } // fim da classe CreateSequentialFileTest
```

```
To terminate input, type the end-of-file indicator
when you are prompted to enter input.
On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
On Windows type <ctrl> z then press Enter

Enter account number (> 0), first name, last name and balance.
? 100 Bob Jones 24.98
Enter account number (> 0), first name, last name and balance.
? 200 Steve Doe -345.67
Enter account number (> 0), first name, last name and balance.
? 300 Pam White 0.00
Enter account number (> 0), first name, last name and balance.
? 400 Sam Stone -42.16
Enter account number (> 0), first name, last name and balance.
? 500 Sue Rich 224.62
Enter account number (> 0), first name, last name and balance.
? ^Z
```

Erro comum de programação 14.2

É um erro de lógica abrir um arquivo existente para saída quando, de fato, o usuário quer preservar o arquivo.

14.6.2 Lendo e desserializando dados a partir de um arquivo de acesso seqüencial

- **Para abrir um arquivo a fim de ler objetos, crie um `FileInputStream` empacotado por um `ObjectInputStream`.**
  - **`FileInputStream` fornece os métodos para ler a entrada baseada em bytes a partir de um arquivo.**
  - **`ObjectInputStream` utiliza `FileInputStream` para ler os objetos em um arquivo.**
  - **O método `readObject` de `ObjectInputStream` lê um objeto, que, então, sofre downcast para o tipo adequado.**
    - **`EOFException` ocorre se houver uma tentativa de ler depois do final do arquivo.**
    - **`ClassNotFoundException` ocorre se a classe para o objeto sendo lido não puder ser localizada.**
  - **O método `close` de `ObjectInputStream` fecha os dois objetos.**

Resumo

```
1  // Fig. 14.20: ReadSequentialFile.java
2  // Este programa lê um arquivo de objetos seqüencialmente
3  // e exibe cada registro.
4  import java.io.EOFException;
5  import java.io.FileInputStream;
6  import java.io.IOException;
7  import java.io.ObjectInputStream;
8
9  import com.deitel.jhtp6.ch14.AccountRecordSerializable;
10
11 public class ReadSequentialFile
12 {
13    private ObjectInputStream input;
14
15    // permite ao usuário selecionar o arquivo a abrir
16    public void openFile()
17    {
18       try // abre o arquivo
19       {
20          input = new ObjectInputStream(
21             new FileInputStream( "clients.ser" ) );
22       } // fim de try
23       catch ( IOException ioException )
24       {
25          System.err.println( "Error opening file." );
26       } // fim de catch
27    } // fim do método openFile
28
```

Classe utilizada para criar fluxo de entrada baseado em bytes

Classe utilizada para ler a entrada dos dados do objeto no fluxo baseado em bytes

Abre o arquivo `clients.ser` para leitura

(1 de 3)

Resumo

ReadSequentialFile.java

```
29    // lê registro a partir do arquivo
30    public void readRecords()
31    {
32       AccountRecordSerializable record;
33       System.out.printf( "%-10s%-12s%-12s%10s\n", "Account",
34          "First Name", "Last Name", "Balance" );
35
36       try // insere os valores a partir do arquivo
37       {
38          while ( true )
39          {
40             record = ( AccountRecordSerializable ) input.readObject();
41
42             // exibe o conteúdo do registro
43             System.out.printf( "%-10d%-12s%-12s%10.2f\n",
44                record.getAccount(), record.getFirstName(),
45                record.getLastName(), record.getBalance() );
46          } // fim do while
47       } // fim do try
48       catch ( EOFException endOfFileException )
49       {
50          return; // fim do arquivo foi alcançado
51       } // fim do catch
```

Lê o registro a partir do arquivo

(2 de 3)

Exibe as informações sobre o registro na tela

```
52       catch ( ClassNotFoundException classNotFoundException )
53       {
54          System.err.println( "Unable to create object." );
55       } // fim do catch
56       catch ( IOException ioException )
57       {
58          System.err.println( "Error during read from file." );
59       } // fim do catch
60    } // fim do método readRecords
61
62    // fecha arquivo e termina o aplicativo
63    public void closeFile()
64    {
65       try // fecha o arquivo e encerra
66       {
67          if ( input != null )
68             input.close();
69       } // fim do try
70       catch ( IOException ioException )
71       {
72          System.err.println( "Error closing file." );
73          System.exit( 1 );
74       } // fim do catch
75    } // fim do método closeFile
76 } // fim da classe ReadSequentialFile
```

Arquivo fechado

Resumo

ReadSequentialFile.java

(3 de 3)

Resumo

ReadSequentialFileTest.java

```
1  // Fig. 14.21: ReadSequentialFileTest.java
2  // Esse programa testa a classe ReadSequentialFile.
3
4  public class ReadSequentialFileTest
5  {
6     public static void main( String args[] )
7     {
8        ReadSequentialFile application = new ReadSequentialFile();
9
10       application.openFile();
11       application.readRecords();
12       application.closeFile();
13    } // fim do main
14 } // fim da classe ReadSequentialFileTest
```

```
Account   First Name  Last Name      Balance
100       Bob         Jones            24.98
200       Steve       Doe            -345.67
300       Pam         White            0.00
400       Sam         Stone          -42.16
500       Sue         Rich           224.62
```

14.8 Classes `java.io` adicionais
Interfaces e classes para entrada e saída baseada em bytes

- Classes `InputStream` e `OutputStream`:
  - Classes `abstract` que declaram os métodos para realizar entrada e saída baseada em bytes.
- Classes `PipedInputStream` e `PipedOutputStream`
  - Estabelecem pipes entre dois threads em um programa.
  - Pipes são canais de comunicação sincronizados entre threads.
- Classes `FilterInputStream` e `FilterOutputStream`:
  - Fornecem funcionalidade adicional ao fluxo, como agregar bytes de dados a unidades de tipo primitivo significativas.
- Classe `PrintStream`:
  - Gera a saída de texto para um fluxo especificado.
- Interfaces `DataInput` e `DataOutput`:
  - Para leitura e gravação de tipos primitivos em um arquivo.
  - `DataInput` é implementada pelas classes `RandomAccessFiles` e `DataInputStream`; `DataOutput` é implementada por `RandomAccessFile` e `DataOuputStream`.
- A classe `SequenceInputStream` permite a concatenação de vários `InputStreams` – o programa vê o grupo como um `InputStream` contínuo.

Interfaces e classes para entrada e saída baseada em bytes (*Cont.*)

- **Armazenamento em buffer (*buffering*) é uma técnica de aprimoramento do desempenho de E/S.**
  - **Aumenta significativamente a eficiência de uma aplicação.**
  - **Saída (utiliza a classe `BufferedOutputStream`).**
    - **Cada instrução de saída não necessariamente resulta em uma transferência física real dos dados ao dispositivo de saída – os dados são direcionados a uma região da memória chamada buffer (mais rápido que gravar em um arquivo).**
    - **Quando o buffer está cheio, a transferência real ao dispositivo de saída é realizada em uma grande *operação física de saída* (as operações físicas de saída também são chamadas de *operações lógicas de saída*).**
    - **Um buffer parcialmente preenchido pode ser esvaziado com o método `flush`.**
  - **Entrada (utiliza a classe `BufferedInputStream`):**
    - **Muitos fragmentos lógicos de dados em um arquivo são lidos como uma *operação física de entrada* (também chamada *operação lógica de entrada*).**
    - **Quando buffer está vazio, a próxima operação física de entrada é realizada.**
- **Classes `ByteArrayInputStream` e `ByteArrayOutputStream` são utilizadas para inserir a partir de arrays de `byte` na memória e enviá-los como saída para arrays de `byte` na memória.**

Dica de desempenho 14.1

E/S armazenada em buffer produz melhorias significativas de desempenho em relação a E/S não-armazenada em buffer.

As interfaces e classes para entrada e saída baseada em caracteres

- Classes abstratas `Reader` e `Writer`:
  - Unicode de dois bytes, fluxos baseados em caracteres.
- Classes `BufferedReader` e `BufferedWriter`:
  - Permitem armazenamento em buffer de fluxos baseados em caracteres.
- Classes `CharArrayReader` e `CharArrayWriter`:
  - Lêem e gravam fluxos de caracteres em arrays de caracteres.
- Classe `LineNumberReader`:
  - Fluxo de caracteres armazenado em buffer que monitora o número de leitura de linhas.
- Classes `PipedReader` e `PipedWriter`:
  - Implementam fluxos de caracteres redirecionados que podem ser utilizados para transferir informações entre threads.
- Classes `StringReader` e `StringWriter`:
  - Lêem caracteres e gravam caracteres em `String`s.

14.9 Abrindo arquivos com `JFileChooser`

- **`JFileChooser` – classe utilizada para exibir um diálogo que permite aos usuários selecionar arquivos facilmente.**
  - **O método `setFileSelectionMode` especifica o que o usuário pode selecionar em `JFileChooser`:**
    - **Constante `FILES_AND_DIRECTORIES` indica arquivos e diretórios.**
    - **Constante `FILES_ONLY` indica somente arquivos.**
    - **Constante `DIRECTORIES_ONLY` indica somente diretórios.**
  - **Método `showOpenDialog` exibe o diálogo `JFileChooser` intitulado Open, com os botões Open e Cancel (para abrir um arquivo/diretório ou fechar o diálogo, respectivamente).**
    - **Constante `CANCEL_OPTION` especifica que o usuário clicou no botão Cancel.**
  - **O método `getSelectedFile` recupera o arquivo ou diretório que o usuário selecionou.**

Resumo

FileDemonstration

java

(1 de 4)

```
1  // Fig. 14.37: FileDemonstration.java
2  // Demonstrando a classe File.
3  import java.awt.BorderLayout;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.io.File;
7  import javax.swing.JFileChooser;
8  import javax.swing.JFrame;
9  import javax.swing.JOptionPane;
10 import javax.swing.JScrollPane;
11 import javax.swing.JTextArea;
12 import javax.swing.JTextField;
13
14 public class FileDemonstration extends JFrame
15 {
16    private JTextArea outputArea; // utilizado para saída
17    private JScrollPane scrollPane; // utilizado para fornecer rolagem para saída
18
19    // configura a GUI
20    public FileDemonstration()
21    {
22       super( "Testing class File" );
23
24       outputArea = new JTextArea();
25
26       // adiciona outputArea ao scrollPane
27       scrollPane = new JScrollPane( outputArea );
28
29       add( scrollPane, BorderLayout.CENTER ); // adiciona scrollPane à GUI
30
```

Classe para exibir o diálogo `JFileChooser`

Resumo

FileDemonstration

.java

```
31       setSize( 400, 400 ); // configura o tamanho da GUI
32       setVisible( true ); // exibe a GUI
33
34       analyzePath(); // cria e analisa o objeto File
35    } // fim do construtor FileDemonstration
36
37    // permite que o usuário especifique nome do arquivo
38    private File getFile()
39    {
40       // exibe o diálogo de arquivo para o usuário escolher o arquivo a abrir
41       JFileChooser fileChooser = new JFileChooser();
42       fileChooser.setFileSelectionMode(
43          JFileChooser.FILES_AND_DIRECTORIES );
44
45       int result = fileChooser.showOpenDialog( this );
46
47       // se o usuário clicou no botão Cancel no diálogo, retorna
48       if ( result == JFileChooser.CANCEL_OPTION )
49          System.exit( 1 );
50
51       File fileName = fileChooser.getSelectedFile
52
53       // exibe erro se inválido
54       if ( ( fileName == null ) || ( fileName.getName().equals( "" ) ) )
55       {
56          JOptionPane.showMessageDialog( this, "Invalid File Name",
57             "Invalid File Name", JOptionPane.ERROR_MESSAGE );
58          System.exit( 1 );
59       } // fim de if
60
```

Cria `JFileChooser`

Permite que o usuário selecione tanto arquivos como diretórios

Exibe diálogo

O usuário clicou em Cancel

Recupere o arquivo ou diretório selecionado pelo usuário

Resumo

FileDemonstration

.java

(3 de 4)

```
61       return fileName;
62    } // fim do método getFile
63
64    // exibe informações sobre o arquivo que o usuário especifica
65    public void analyzePath()
66    {
67       // cria objeto File com base na entrada do usuário
68       File name = getFile();
69
70       if ( name.exists() ) // se o nome existir, gera saída das informações sobre ele
71       {
72          // exibe informações do arquivo (ou diretório)
73          outputArea.setText( String.format(
74             "%s%s\n%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s",
75             name.getName(), " exists",
76             ( name.isFile() ? "is a file" : "is not a file" ),
77             ( name.isDirectory() ? "is a directory" :
78                "is not a directory" ),
79             ( name.isAbsolute() ? "is absolute path" :
80                "is not absolute path" ), "Last modified: ",
81             name.lastModified(), "Length: ", name.length(),
82             "Path: ", name.getPath(), "Absolute path: ",
83             name.getAbsolutePath(), "Parent: ", name.getParent() ) );
84
```

Exibir informações sobre o arquivo

```
85          if ( name.isDirectory() ) // gera saída de listagem de diretório
86          {
87             String directory[] = name.list();
88             outputArea.append( "\n\nDirectory contents:\n" );
89
90             for ( String directoryName : directory )
91                outputArea.append( directoryName + "\n" );
92          } // fim de else
93       } // fim de if externo
94       else // não arquivo ou diretório, gera saída da mensagem de erro
95       {
96          JOptionPane.showMessageDialog( this, name +
97             " does not exist.", "ERROR", JOptionPane.ERROR_MESSAGE );
98       } // fim de else
99    } // fim do método analyzePath
100} // fim da classe FileDemonstration
```

Resumo

FileDemonstration.java

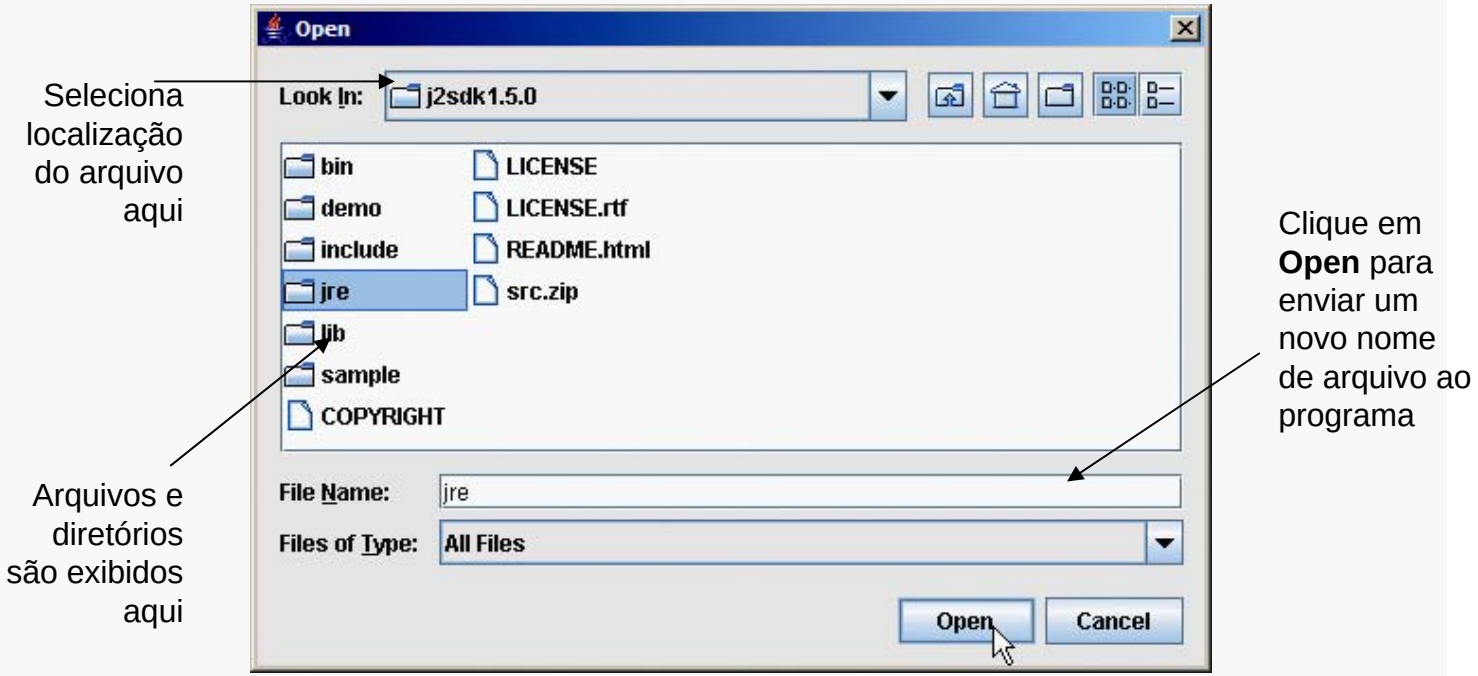
(4 de 4)

Resumo

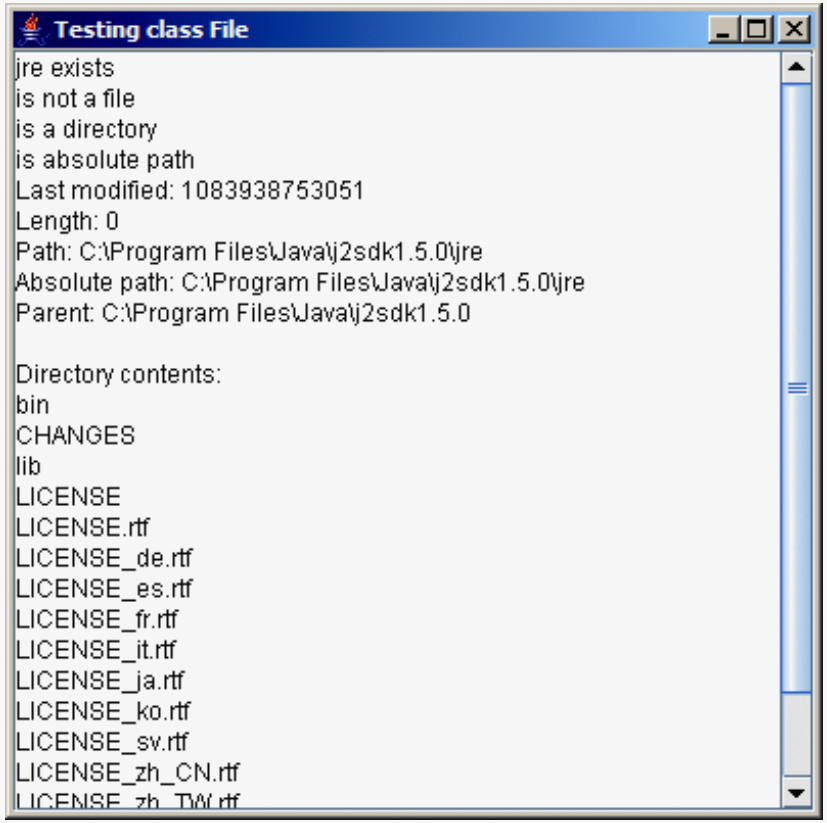
FileDemonstration Test.java

(1 de 2)

```
1  // Fig. 14.38: FileDemonstrationTest.java
2  // Testando a classe FileDmonstration.
3  import javax.swing.JFrame;
4
5  public class FileDemonstrationTest
6  {
7     public static void main( String args[] )
8     {
9        FileDemonstration application = new FileDemonstration();
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11    } // fim de main
12 } // fim da classe FileDemonstrationTest
```



Resumo



FileDemonstration

Test.java

(2 de 2)