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`.**

Sumário

14.1 Introdução
14.2 Hierarquia de dados
14.3 Arquivos e fluxos
14.4 Classe `File`
14.5 Arquivos de texto de acesso seqüencial
14.5.1 Criando um arquivo de texto de acesso seqüencial
14.5.2 Lendo dados a partir de um arquivo de texto de acesso seqüencial
14.5.3 Estudo de caso: Um programa de consulta de crédito
14.5.4 Atualizando arquivos de acesso seqüencial

Sumário

14.6 Serialização de objeto

14.6.1 Criando um arquivo de acesso seqüencial com a serialização de objeto

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

14.7 Arquivos de acesso aleatório

14.7.1 Criando um arquivo de acesso aleatório

14.7.2 Gravando dados aleatoriamente em um arquivo de acesso aleatório

14.7.3 Lendo dados seqüencialmente de um arquivo de acesso aleatório

14.7.4 Estudo de caso: Um programa de processamento de transação

14.8 Classes `java.io` adicionais

14.9 Abrindo arquivos com `JFileChooser`

14.10 Conclusão

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.**

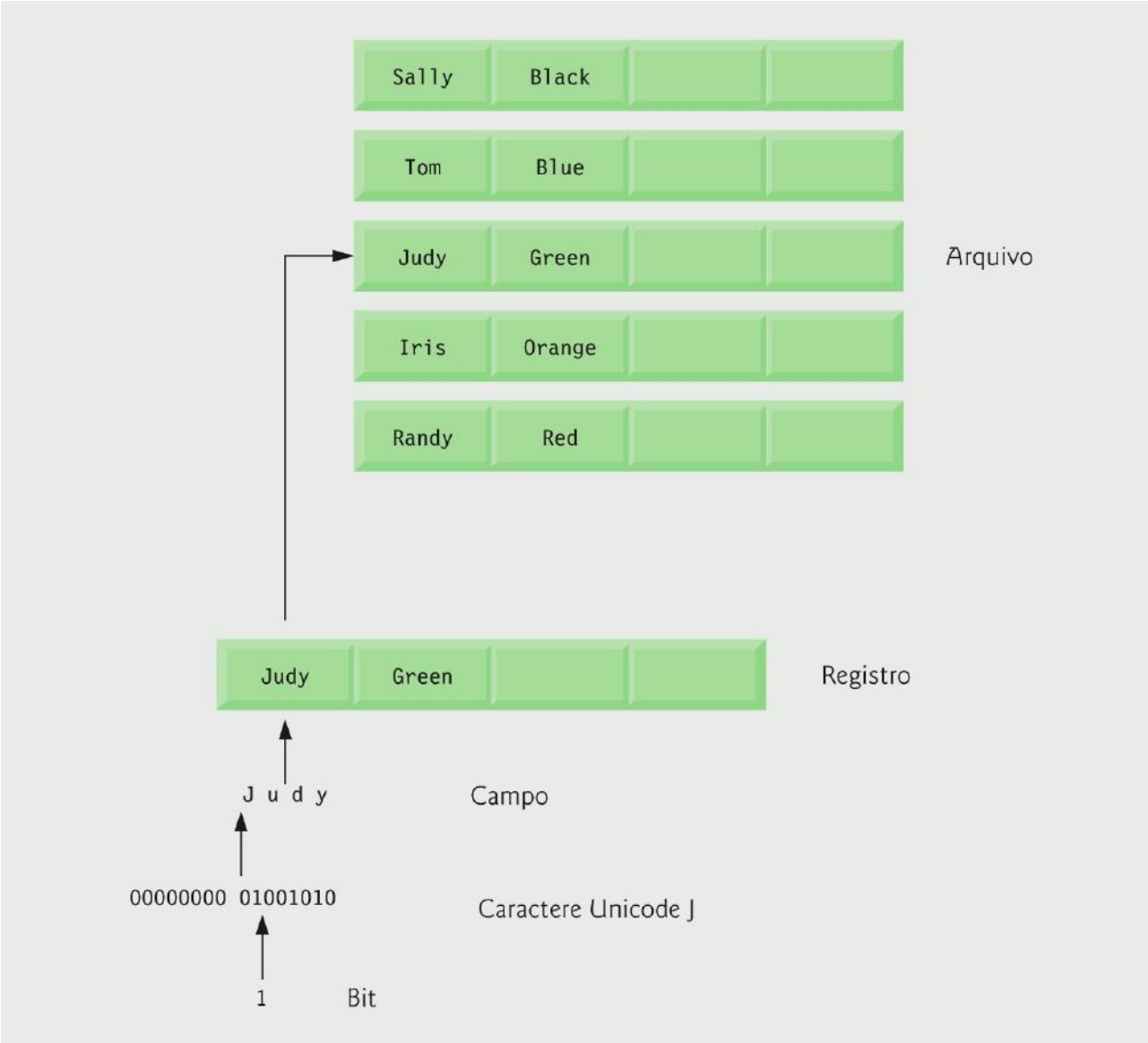


Figura 14.1 | Hierarquia de dados.

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 b
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
18             name.getName(), " e
19             ( name.isFile() ? "is a fi
20             ( name.isDirectory() ? "is a
21                "is not a directory" ),
22             ( name.isAbsolute() ? "is abso
23                "is not absolute path" ), "
24             name.lastModified(), "Length:
25             "Path: ", name.getPath(), "Abs
26             name.getAbsolutePath(), "Paren
27
```

Cria um novo objeto `File`; o usuário especifica o nome de arquivo e caminho

Retorna `true` se `name` for um diretório, não um

Retorna `true` se `path` for um caminho absoluto

Recupera a data/hora

Recupera o caminho inserido como uma string

Recupera o caminho absoluto do arquivo ou diretório

Recupera o diretório-pai (caminho onde o arquivo ou diretório do objeto `File` ou pode ser localizado)

Resumo

```
28          if ( name.isDirectory() ) // gera listagem de diretório
29          {
30             String directory[] = name.list();
31             System.out.println( "\n\nDirectory
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

```
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
```

AccountRecord.java

(3 de 3)

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

```
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 e
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
```

`CreateTextFile.java`

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

CreateTextFile .java

(4 de 4)

```
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

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
```

14.5.3 Estudo de caso: Um programa de consulta de crédito

- **Para recuperar dados seqüencialmente de um arquivo, os programas normalmente começam a ler a partir do início do arquivo e lêem todos os dados consecutivamente até que os dados desejados sejam encontrados.**
- **A classe `Scanner` não fornece nenhuma maneira de voltar ao começo do arquivo.**
- **Em vez disso, o arquivo é fechado e reaberto.**

Resumo

MenuOption.Java

```
1  // Fig. 14.13: MenuOption.java
2  // Define um tipo enum para as opções do programa de consulta de crédito.
3
4  public enum MenuOption
5  {
6     // declara conteúdo do tipo enum
7     ZERO_BALANCE( 1 ),
8     CREDIT_BALANCE( 2 ),
9     DEBIT_BALANCE( 3 ),
10    END( 4 );
11
12    private final int value; // opção atual de menu
13
14    MenuOption( int valueOption )
15    {
16       value = valueOption;
17    } // fim do do construtor do enum de MenuOptions
18
19    public int getValue()
20    {
21       return value;
22    } // fim do método getValue
23 } // fim do enum de MenuOption
```

Resumo

CreditInquiry.java

(1 de 6)

```
1  // Fig. 14.14: CreditInquiry.java
2  // Esse programa lê um arquivo seqüencialmente e exibe o
3  // conteúdo baseado no tipo de conta que o usuário solicita
4  // (saldo credor, saldo devedor ou saldo zero).
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import java.lang.IllegalStateException;
8  import java.util.NoSuchElementException;
9  import java.util.Scanner;
10
11 import com.deitel.jhtp6.ch14.AccountRecord;
12
13 public class CreditInquiry
14 {
15    private MenuOption accountType;
16    private Scanner input;
17    private MenuOption choices[] = {
18       MenuOption.CREDIT_BALANCE, MenuOption.DEBIT_BALANCE,
19       MenuOption.END };
20
21    // lê registros de arquivo e exibe somente os registros do tipo apropriado
22    private void readRecords()
23    {
24       // objeto a ser gravado no arquivo
25       AccountRecord record = new AccountRecord();
26
```

`Scanner` é utilizada para ler os dados no arquivo

`AccountRecord` armazena o registro sendo lido a partir do arquivo

```
27    try // lê registros
28    {
29       // abre o arquivo para leitura a partir do início
30       input = new Scanner( new File(
31
32       while ( input.hasNext() ) // 
33       {
34          record.setAccount( input.nextInt() ); // lê o número da conta
35          record.setFirstName( input.next() ); // lê o nome
36          record.setLastName( input.next() ); // lê o sobrenome
37          record.setBalance( input.nextDouble() ); // lê o saldo
38
39          // se o tipo for a conta adequada
40          if ( shouldDisplay( record.getBal
41             System.out.printf( "%-10d%-12s%-12s%10.2f\n",
42                record.getAccount(), record.getFirstName(),
43                record.getLastName(), record.getBalance() );
44       } // fim do while
45    } // fim do try
46    catch ( NoSuchElementException elementExce
47    {
48       System.err.println( "File improperly formed." );
49       input.close();
50       System.exit( 1 );
51    } // fim do catch
```

Abre `clients.txt` para leitura

Enquanto houver dados a ler no arquivo

Verifica se o registro é do tipo solicitado

a entrada, armazena dados em `AccountRecord`

Exibe os dados do registro na tela

Fecha `Scanner`

`CreditInquiry.java`

(2 de 6)

Resumo

`CreditInquiry.java`

(3 de 6)

```
52       catch ( IllegalStateException stateException )
53       {
54          System.err.println( "Error reading from file." );
55          System.exit( 1 );
56       } // fim do catch
57       catch ( FileNotFoundException fileNotFoundException )
58       {
59          System.err.println( "File cannot be found." );
60          System.exit( 1 );
61       } // fim do catch
62       finally
63       {
64          if ( input != null )
65             input.close(); // close the Scanner and the file
66       } // fim do finally
67    } // fim do método readRecords
68
69    // usa tipo de registro para determinar se o registro deve
70    private boolean shouldDisplay( double balance )
71    {
72       if ( ( accountType == MenuOption.CREDIT_BALANCE )
73          && ( balance < 0 ) )
74          return true;
75
76       else if ( ( accountType == MenuOption.DEBIT_BALANCE )
77          && ( balance > 0 ) )
78          return true;
79
```

Arquivo fechado

O método determina se o registro é do tipo adequado

Resumo

CreditInquiry.java

(4 de 6)

```
80          else if ( ( accountType == MenuOption.ZERO_BALANCE )
81             && ( balance == 0 ) )
82             return true;
83
84          return false;
85       } // fim do método shouldDisplay
86
87       // obtém solicitação do usuário
88       private MenuOption getRequest()
89       {
90          Scanner textIn = new Scanner( System.in );
91          int request = 1;
92
93          // exibe opções de solicitação
94          System.out.printf( "\n%s\n%s\n%s\n%s\n%s\n",
95             "Enter request", " 1 - List accounts with zero balances",
96             " 2 - List accounts with credit balances",
97             " 3 - List accounts with debit balances", " 4 - End of run" );
98
99          try // tenta inserir a escolha de menu
100         {
101            do // insere a solicitação de usuári
102            {
103               System.out.print( "\n? " );
104               request = textIn.nextInt();
105            } while ( ( request < 1 ) || ( request > 4 ) );
106         } // fim do try
```

Faz um loop até o usuário inserir uma solicitação válida

Recupera a solicitação inserida

Resumo

CreditInquiry.java

(5 de 6)

```
107      catch ( NoSuchElementException elementException )
108      {
109         System.err.println( "Invalid input." );
110         System.exit( 1 );
111      } // end catch
112
113      return choices[ request - 1 ]; // retorna o valor enum da opção
114   } // fim do método getRequest
115
116   public void processRequests()
117   {
118      // obtém a solicitação do usuário (por exemplo, saldo zero, credor ou devedor)
119      accountType = getRequest();
120
121      while ( accountType != MenuOption.END )
122      {
123         switch ( accountType )
124         {
125            case ZERO_BALANCE:
126               System.out.println( "\nAccounts with zero balances:\n" );
127               break;
```

Resumo

```
128            case CREDIT_BALANCE:
129               System.out.println( "\nAccounts with credit balances:\n" );
130               break;
131            case DEBIT_BALANCE:
132               System.out.println( "\nAccounts with debit balances:\n" );
133               break;
134         } // fim do switch
135
136         readRecords();
137         accountType = getRequest();
138      } // fim do while
139   } // fim do método processRequests
140} // fim da classe CreditInquiry
```

Lê o arquivo, exibe os registros adequados

CreditInquiry.java

(6 de 6)

Resumo

CreditInquiryTest.java

```
1  // Fig. 14.15: CreditInquiryTest.java
2  // Este programa testa classe CreditInquiry.
3
4  public class CreditInquiryTest
5  {
6     public static void main( String args[] )
7     {
8        CreditInquiry application = new CreditInquiry();
9        application.processRequests();
10    } // fim do main
11 } // fim da classe CreditInquiryTest
```

Resumo

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run

? 1

Accounts with zero balances:

300       Pam         White              0.00

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run

? 2

Accounts with credit balances:
200       Steve       Doe             -345.67
400       Sam         Stone            -42.16

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run

? 3

Accounts with debit balances:
100       Bob         Jones             24.98
500       Sue         Rich             224.62

? 4
```

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 construt
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

Resumo

CreateSequentialFile.java

(2 de 4)

```
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

```
59          if ( accountNumber > 0 )
60          {
61             // cria o novo registro
62             record = new AccountRecordSerializable( accoun
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

Cria AccountRecord com base na entrada do usuário

ntial

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

CreateSequential

FileTest.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
```

14.7 Arquivos de acesso aleatório

- **Arquivos de acesso seqüencial impróprios para aplicações de acesso instantâneo.**
- **Aplicações de acesso instantâneo são aplicações nas quais as informações desejadas precisam ser localizadas instantaneamente.**
- **Acesso instantâneo é possível com arquivos de acesso aleatório (também chamados arquivos de acesso direto) e bancos de dados.**
- **Os dados podem ser inseridos em um arquivo de acesso aleatório sem destruir outros dados.**
- **Diferentes técnicas para criar arquivos de acesso aleatório.**
  - **A mais simples: Exigir que todos os registros em um arquivo tenham o mesmo comprimento fixo.**
    - **Fácil calcular (como uma função do tamanho do registro e da chave de registro) a localização exata de quaisquer registros em relação ao começo do arquivo.**

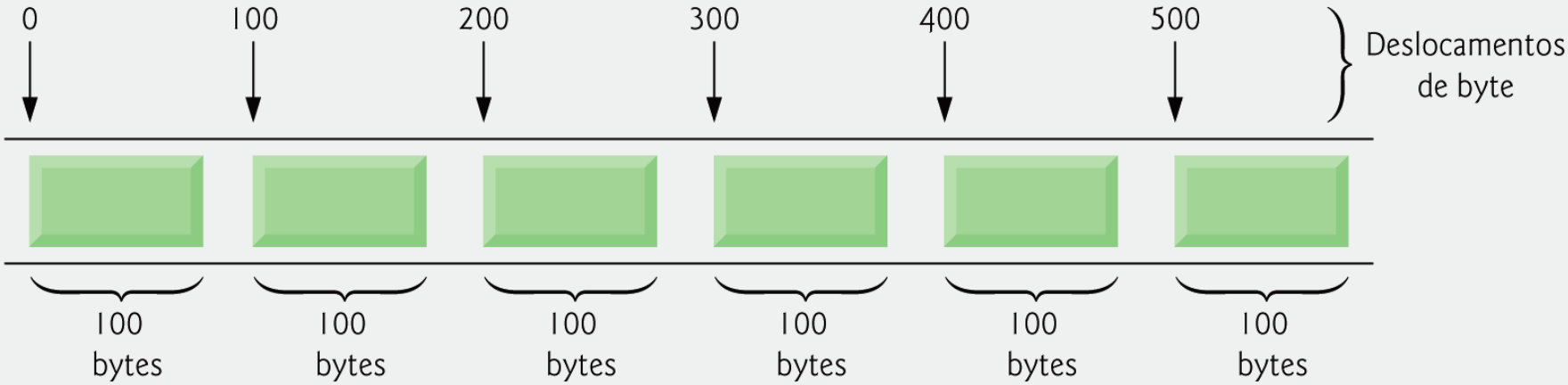


Figura 14.22 | Visualização do Java de um arquivo de acesso aleatório.

14.7.1 Criando um arquivo de acesso aleatório

- Classe `RandomAccessFile`:
  - Inclui todas as capacidades de `FileInputStream` e `FileOutputStream`.
  - Inclui capacidades de leitura e gravação dos valores do tipo primitivo, arrays e strings de bytes.
  - Utilizando `RandomAccessFile`, o programa pode ler ou gravar os dados começando em um local especificado pelo ponteiro de posição de arquivo.
  - Manipula todos os dados como tipos primitivos.
  - Métodos `readInt`, `readDouble`, `readChar` utilizados para ler dados `integer`, `double` e `char` do arquivo.
  - Métodos `writeInt`, `writeDouble`, `writeChars` utilizados para gravar dados `integer`, `double` e `string` no arquivo.
  - Modo File-open – especifica se o arquivo é aberto para leitura (“`r`”) ou para leitura e gravação (“`rw`”). O modo File-open é especificado como o segundo argumento para o construtor `RandomAccessFile`

14.7.1 Criando um arquivo de acesso aleatório (*Continuação*)

- **Classe `StringBuffer` – permite manipular strings dinamicamente.**
  - **Objetos string são imutáveis; `StringBuffer` é utilizada para que strings possam ser alteradas dinamicamente.**
  - **Pode armazenar um número de caracteres especificado por capacidade.**
  - **Se a capacidade for excedida, a capacidade é expandida.**
  - **O número de caracteres no conjunto `StringBuffer` com o método `setLength`.**
- **Cria um arquivo de acesso aleatório gravando registros em branco ou vazios no arquivo de acordo com a quantidade de registros que será necessária.**

Resumo

```
1  // Fig. 14.23: RandomAccessAccountRecord.java
2  // Subclasse de AccountRecord para programas com arquivos de acesso aleatório.
3  package com.deitel.jhtp6.ch14; // empacotada para reutilização
4
5  import java.io.RandomAccessFile;
6  import java.io.IOException;
7
8  public class RandomAccessAccountRecord extends AccountRecord
9  {
10    public static final int SIZE = 72;
11
12    // construtor sem argumentos chama outro construtor com valores padrão
13    public RandomAccessAccountRecord()
14    {
15       this( 0, "", "", 0.0 );
16    } // fim do construtor de RandomAccessAccountRecord sem argumentos
17
18    // inicializa uma RandomAccessAccountRecord
19    public RandomAccessAccountRecord( int account, String firstName,
20       String lastName, double balance )
21    {
22       super( account, firstName, lastName, balance );
23    } // fim do construtor de quatro argumentos da classe RandomAccessAccountRecord
24
```

Utilizado para gravação e leitura nos arquivos de acesso aleatório

ccess

AccountRecord.java

(1 de 3)

Resumo

```
25    // lê um registro em um RandomAccessFile especificado
26    public void read( RandomAccessFile file ) throws IOException
27    {
28       setAccount( file.readInt() );
29       setFirstName( readName( file ) );
30       setLastName( readName( file ) );
31       setBalance( file.readDouble() );
32    } // fim do método read
33
34    // assegura que o nome tenha um comprimento adequado
35    private String readName( RandomAccessFile file ) throws IOException
36    {
37       char name[] = new char[ 15 ], temp;
38
39       for ( int count = 0; count < name.length; count++ )
40       {
41          temp = file.readChar();
42          name[ count ] = temp;
43       } // fim do for
44
45       return new String( name ).replace( '\0', ' ' );
46    } // fim do método readName
47
```

O método lê os dados do tipo primitivo e armazena-os no objeto `RandomAccessAccountRecord`

Lê caracteres, agregando-os em uma string

AccountRecord.java

(2 de 3)

Resumo

```
48    // grava um registro no RandomAccessFile especificado
49    public void write( RandomAccessFile file ) throws IOException
50    {
51       file.writeInt( getAccount() );
52       writeName( file, getFirstName() );
53       writeName( file, getLastName() );
54       file.writeDouble( getBalance() );
55    } // fim do método write
56
57    // grava um nome no arquivo; máximo de 15 caracteres
58    private void writeName( RandomAccessFile file, String name )
59       throws IOException
60    {
61       StringBuffer buffer = null;
62
63       if ( name != null )
64          buffer = new StringBuffer( name );
65       else
66          buffer = new StringBuffer( 15 );
67
68       buffer.setLength( 15 );
69       file.writeChars( buffer.toString() );
70    } // fim do método writeName
71 } // fim da classe RandomAccessAccountRecord
```

O método grava os dados do tipo primitivo com base nos dados provenientes do objeto `RandomAccessAccountRecord`

`AccountRecord.java`

Grava o primeiro e o último nome no arquivo, certificando-se de que cada um tem 15 caracteres

Resumo

CreateRandomFile.java

(1 de 2)

```
1  // Fig. 14.24: CreateRandomFile.java
2  // Cria arquivo de acesso aleatório gravando 100 registros vazios no disco.
3  import java.io.IOException;
4  import java.io.RandomAccessFile;
5
6  import com.deitel.jhtp6.ch14.RandomAccessAccountRecord;
7
8  public class CreateRandomFile
9  {
10    private static final int NUMBER_RECORDS = 100;
11
12    // permite ao usuário selecionar o arquivo a abrir
13    public void createFile()
14    {
15       RandomAccessFile file = null;
16
17       try // abre arquivo para ler e gravar
18       {
19          file = new RandomAccessFile( "clients.dat", "rw
20
21          RandomAccessAccountRecord blankRecord =
22             new RandomAccessAccountRecord();
23
24          // grava 100 registros em branco
25          for ( int count = 0; count < NUMBER_RECORDS; count++ )
26             blankRecord.write( file );
27
```

Abre o arquivo para leitura e gravação

Cria um registro em branco

Gera saída do registro em branco 100 vezes, para cada possível conta

```
28          // exibe uma mensagem de que o arquivo foi criado
29          System.out.println( "Created file clients.dat." );
30
31          System.exit( 0 ); // termina o programa
32       } // fim do try
33       catch ( IOException ioException )
34       {
35          System.err.println( "Error processing file." );
36          System.exit( 1 );
37       } // fim do catch
38       finally
39       {
40          try
41          {
42             if ( file != null )
43                file.close(); // fecha o arquivo
44          } // fim do try
45          catch ( IOException ioException )
46          {
47             System.err.println( "Error closing file." );
48             System.exit( 1 );
49          } // fim do catch
50       } // fim do finally
51    } // fim do método createFile
52 } // fim da classe CreateRandomFile
```

Arquivo fechado

Resumo

CreateRandomFile

.java

(2 de 2)

Resumo

CreateRandomFile

Test.java

```
1  // Fig. 14.25: CreateRandomFileTest.java
2  // Testando a classe CreateRandomFile.
3
4  public class CreateRandomFileTest
5  {
6     public static void main( String args[] )
7     {
8        CreateRandomFile application = new CreateRandomFile();
9        application.createFile();
10    } // fim do main
11 } // fim da classe CreateRandomFileTest
```

```
Created file clients.dat.
```

14.7.2 Gravando dados aleatoriamente em um arquivo de acesso aleatório

- **O método `RandomAccessFile` busca no ponteiro de posição de arquivo as posições de uma localização específica em um arquivo em relação ao começo do arquivo.**
- **O tamanho de cada registro é conhecido, assim a localização no arquivo de um registro específico pode ser localizada multiplicando o tamanho do registro pelo número do registro.**
- **Depois que a localização é conhecida, novos dados de registro podem ser gravados sem a necessidade de se preocupar com o restante do arquivo, uma vez que cada registro tem sempre o mesmo tamanho.**

Resumo

WriteRandomFile.java

(1 de 4)

```
1  // Fig. 14.26: WriteRandomFile.java
2  // Esse programa recupera informações do usuário no
3  // teclado e grava essas informações em um arquivo de acesso aleatório.
4  import java.io.File;
5  import java.io.IOException;
6  import java.io.RandomAccessFile;
7  import java.util.NoSuchElementException;
8  import java.util.Scanner;
9
10 import com.deitel.jhtp6.ch14.RandomAccessAccountRecord;
11
12 public class WriteRandomFile
13 {
14    private RandomAccessFile output;
15
16    private static final int NUMBER_RECORDS = 100;
17
18    // permite ao usuário escolher o arquivo a abrir
19    public void openFile()
20    {
21       try // abre o arquivo
22       {
23          output = new RandomAccessFile( "clients.dat", "rw" );
24       } // fim do try
25       catch ( IOException ioException )
26       {
27          System.err.println( "File does not exist." );
28       } // fim do catch
29    } // fim do método openFile
30
```

Abre o arquivo para leitura e gravação

```
31    // fecha o arquivo e termina o aplicativo
32    public void closeFile()
33    {
34       try // fecha o arquivo e encerra
35       {
36          if ( output != null )
37             output.close();
38       } // fim do try
39       catch ( IOException ioException )
40       {
41          System.err.println( "Error closing file." );
42          System.exit( 1 );
43       } // fim do catch
44    } // fim do método closeFile
45
46    // adiciona registros ao arquivo
47    public void addRecords()
48    {
49       // objeto a ser gravado no arquivo
50       RandomAccessAccountRecord record = new RandomAccessAccountRecord();
51
52       int accountNumber = 0; // número da conta para o objeto AccountRecord
53       String firstName; // nome para o objeto AccountRecord
54       String lastName; // sobrenome para o objeto AccountRecord
55       double balance; // saldo para o objeto AccountRecord
56
```

Resumo

`WriteRandomFile.java`

(2 de 4)

Resumo

WriteRandomFile

.java

(3 de 4)

```
57       Scanner input = new Scanner( System.in );
58
59       System.out.printf( "%s\n%s\n%s\n%s\n\n",
60          "To terminate input, type the end-of-file indicator ",
61          "when you are prompted to enter input.",
62          "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
63          "On Windows type <ctrl> z then press Enter" );
64
65       System.out.printf( "%s %s\n%s", "Enter account number (1-100),",
66          "first name, last name and balance.", "? " );
67
68       while ( input.hasNext() ) // faz um loop até o indicador de fim de arquivo
69       {
70          try // gera saída de valores no arquivo
71          {
72             accountNumber = input.nextInt(); // lê número da conta
73             firstName = input.next(); // lê o nome
74             lastName = input.next(); // lê o sobrenome
75             balance = input.nextDouble(); // lê o saldo
76
77             if ( accountNumber > 0 && accountNumber <= NUMBER_RECORDS )
78             {
79                record.setAccount( accountNumber );
80                record.setFirstName( firstName );
81                record.setLastName( lastName );
82                record.setBalance( balance );
83
```

Armazena os dados de entrada em RandomAccessAccountRecord

```
84                output.seek( ( accountNumber - 1 ) * // posição para a localização
85                   RandomAccessAccountRecord.SIZE ); // adequada do arquivo
86                record.write( output );
87             } // fim do if
88             else
89                System.out.println( "A                100." );
90          } // fim do try
91          catch ( IOException ioException )
92          {
93             System.err.println( "Error writing to file." );
94             return;
95          } // fim do catch
96          catch ( NoSuchElementException elementException )
97          {
98             System.err.println( "Invalid input. Please try again." );
99             input.nextLine(); // descarta entrada p/ o usuário tentar novamente
100         } // fim do catch
101
102         System.out.printf( "%s %s\n%s", "Enter account number (1-100),",
103            "first name, last name and balance.", "? " );
104      } // fim do while
105   } // fim do método addRecords
106} // fim da classe WriteRandomFile
```

Gera a saída do novo registro para o arquivo

Calcula a localização do novo registro

Resumo

WriteRandomFile

.java

(4 de 4)

Resumo

WriteRandomFile

Test.java

```
1  // Fig. 14.27: WriteRandomFileTest.java
2  // Este programa testa a classe WriteRandomFile.
3
4  public class WriteRandomFileTest
5  {
6     public static void main( String args[] )
7     {
8        WriteRandomFile application = new WriteRandomFile();
9        application.openFile();
10       application.addRecords();
11       application.closeFile();
12    } // fim do main
13 } // fim da classe WriteRandomFileTest
```

```
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 (1-100), first name, last name and balance.
? 37 Doug Barker 0.00
Enter account number (1-100), first name, last name and balance.
? 29 Nancy Brown -24.54
Enter account number (1-100), first name, last name and balance.
? 96 Sam Stone 34.98
Enter account number (1-100), first name, last name and balance.
? 88 Dave Smith 258.34
Enter account number (1-100), first name, last name and balance.
? 33 Stacey Dunn 314.33
Enter account number (1-100), first name, last name and balance.
? ^Z
```

14.7.3 Lendo dados seqüencialmente de um arquivo de acesso aleatório

- **Abre o arquivo com o modo de abertura de arquivo "r" para leitura.**
- **Ignora os registros vazios (normalmente, aqueles com o número de conta de zero) ao ler no arquivo.**
- **Registros armazenados pelo número da conta nos arquivos de acesso aleatório têm o bônus extra de poderem ser classificados, uma vez que os dados de cada registro só podem ser colocados em uma parte específica do arquivo.**
- **Classificar com as técnicas de acesso direto é extremante rápido — a velocidade é alcançada tornando o arquivo suficientemente grande a fim de que ele contenha cada registro possível.**
  - **Troca espaço/tempo.**

Boa prática de programação 14.1

Abra um arquivo com o modo de abertura do arquivo "r" para entrada se o conteúdo não deve ser modificado. Isso evita modificação não intencional do conteúdo do arquivo. Esse é outro exemplo do princípio do menor privilégio.

Resumo

ReadRandomFile

.java

(1 de 3)

```
1  // Fig. 14.28: ReadRandomFile.java
2  // Este programa lê um arquivo de acesso aleatório seqüencialmente e
3  // exibe o conteúdo um registro por vez em campos de texto.
4  import java.io.EOFException;
5  import java.io.IOException;
6  import java.io.RandomAccessFile;
7
8  import com.deitel.jhtp6.ch14.RandomAccessAccountRecord;
9
10 public class ReadRandomFile
11 {
12    private RandomAccessFile input;
13
14    // permite que o usuário selecione o arquivo a abrir
15    public void openFile()
16    {
17       try // abre o arquivo
18       {
19          input = new RandomAccessFile( "clients.dat", "r" );
20       } // fim do try
21       catch ( IOException ioException )
22       {
23          System.err.println( "File does not exist." );
24       } // fim do catch
25    } // fim do método openFile
26
```

Abre o arquivo para leitura

Resumo

ReadRandomFile

.java

(2 de 3)

```
27    // lê e exibe registros
28    public void readRecords()
29    {
30       RandomAccessAccountRecord record = new RandomAccessAccountRecord();
31
32       System.out.printf( "%-10s%-15s%-15s%10s\n", "Account",
33          "First Name", "Last Name", "Balance" );
34
35       try // lê um registro e exibe
36       {
37          while ( true )
38          {
39             do
40             {
41                record.read( input );
42             } while ( record.getAccount() == 0 );
43
44             // exibe conteúdo do registro
45             System.out.printf( "%-10d%-12s%-12s%10.2f\n",
46                record.getAccount(), record.getFirstName(),
47                record.getLastName(), record.getBalance() );
48          } // fim do while
49       } // fim do try
50       catch ( EOFException eofException ) // clos
51       {
52          return; // fim do arquivo foi alcançado
53       } // fim do catch
```

Lê até o registro não em brando ser encontrado

A exceção ocorre quando o final do arquivo é alcançado

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

Resumo

ReadRandomFile

.java

(3 de 3)

Resumo

ReadRandomFileTest.java

```
1  // Fig. 14.29: ReadRandomFileTest.java
2  // Testando a classe ReadRandomFile.
3
4  public class ReadRandomFileTest
5  {
6     public static void main( String args[] )
7     {
8        ReadRandomFile application = new ReadRandomFile();
9        application.openFile();
10       application.readRecords();
11       application.closeFile();
12    } // fim do main
13 } // fim da classe ReadRandomFileTest
```

```
Account   First Name      Last Name          Balance
29        Nancy           Brown               -24.54
33        Stacey          Dunn                314.33
37        Doug            Barker                0.00
88        Dave            Smith               258.34
96        Sam             Stone                34.98
```

14.7.4 Estudo de caso: Um programa de processamento de transação

- **Exemplo de processamento de acesso instantâneo.**
- **O usuário pode:**
  - **Exibir registros – ler do começo ao final, ignorando registros vazios.**
  - **Atualizar registros – solicitar o número da conta, somente permitindo que o usuário atualize se o registro não estiver vazio.**
  - **Adicionar novos registros – solicitar o número da conta, somente permitindo que o usuário adicione uma conta se o registro estiver vazio.**
  - **Excluir registros – solicitar o número da conta, somente excluir registros existentes (isto é, substituir um registro por um registro vazio).**

Resumo

```
Account   First Name      Last Name         Balance
29        Nancy           Brown              -24.54
33        Stacey          Dunn               314.33
37        Doug            Barker               0.00
88        Dave            Smith              258.34
96        Sam             Stone               34.98
```

Processador de transações: Exibe contas

Resumo

```
Enter account to update ( 1 - 100 ): 37
37        Doug            Barker              0.00

Enter charge ( + ) or payment ( - ): +87.99
37        Doug            Barker             87.99
```

Processador de transações: Atualiza contas

Resumo

```
Enter account number, first name, last name and balance.
(Account number must be 1 - 100)
? 22 Sarah Johnston 247.45
```

Processador de transações: Insere contas

Resumo

MenuOption.Java

```
1  // Fig. 14.33: MenuOption.java
2  // Define um tipo enum para as opções do programa de consulta de crédito.
3
4  public enum MenuOption
5  {
6     // declara o conteúdo do tipo enum
7     PRINT( 1 ),
8     UPDATE( 2 ),
9     NEW( 3 ),
10    DELETE( 4 ),
11    END( 5 );
12
13    private final int value; // opção atual de menu
14
15    MenuOption( int valueOption )
16    {
17       value = valueOption;
18    } // fim do construtor do enum de MenuOptions
19
20    public int getValue()
21    {
22       return value;
23    } // fim do método getValue
24 } // fim do enum de MenuOption
```

Resumo

FileEditor.java

(1 de 5)

```
1  // Fig. 14.34: FileEditor.java
2  // Esta classe declara os métodos que manipulam contas bancárias
3  // registra em um arquivo de acesso aleatório.
4  import java.io.EOFException;
5  import java.io.File;
6  import java.io.IOException;
7  import java.io.RandomAccessFile;
8  import java.util.Scanner;
9
10 import com.deitel.jhtp6.ch14.RandomAccessAccountRecord;
11
12 public class FileEditor
13 {
14    RandomAccessFile file; // referência ao arquivo
15    Scanner input = new Scanner( System.in );
16
17    // abre o arquivo
18    public FileEditor( String fileName ) throws IOExcepti
19    {
20       file = new RandomAccessFile( fileName, "rw" );
21    } // fim do construtor FileEditor
22
23    // fecha o arquivo
24    public void closeFile() throws IOException
25    {
26       if ( file != null )
27          file.close();
28    } // fim do método closeFile
29
```

Abre o arquivo para leitura e gravação

Arquivo fechado

Resumo

FileEditor.java

(2 de 5)

```
30    // obtém um registro do arquivo
31    public RandomAccessAccountRecord getRecord( int accountNumber )
32       throws IllegalArgumentException, NumberForma
33    {
34       RandomAccessAccountRecord record = new RandomAccessAccountRecord();
35
36       if ( accountNumber < 1 || accountNumber > 100 )
37          throw new IllegalArgumentException( "Out of range" );
38
39       // busca o registro apropriado no arquivo
40       file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );
41
42       record.read( file );
43
44       return record;
45    } // fim do método getRecord
46
47    // atualiza registro no arquivo
48    public void updateRecord( int accountNumber, double transaction )
49       throws IllegalArgumentException, IOException
50    {
51       RandomAccessAccountRecord record = getRecord( accountNumber );
52
53       if ( record.getAccount() == 0 )
54          throw new IllegalArgumentException( "Account does not exist" );
55
56       // busca registro apropriado no arquivo
57       file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );
58
```

Recupera o registro com base no número da conta

Posiciona o ponteiro de posição de arquivo no registro

Lê o registro a partir do arquivo

Recupera o registro com base no número da conta

Posiciona o ponteiro de posição de arquivo no registro

```
59       record = new RandomAccessAccountRecord(
60          record.getAccount(), record.getFirstName(),
61          record.getLastName(), recor
62
63       record.write( file ); // grava registro atualizado no arquivo
64    } // fim do método updateRecord
65
66    // adiciona o registr ao arquivo
67    public void newRecord( int accountNumber, String firstName,
68       String lastName, double balance )
69       throws IllegalArgumentException, IOException
70    {
71       RandomAccessAccountRecord record = getRecord( accountNumber );
72
73       if ( record.getAccount() != 0 )
74          throw new IllegalArgumentException( "Account already exists" );
75
76       // busca registro apropriado no arquivo
77       file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );
78
79       record = new RandomAccessAccountRecord( ac
80          firstName, lastName, balance );
81
82       record.write( file ); // grava registro no arquivo
83    } // fim do método newRecord
84
```

Modifica o registro com base na entrada

Grava um novo registro no arquivo

FileEditor.java

Recupera o registro com base no número da conta

Posiciona o ponteiro de posição de arquivo no

Cria um novo registro com base na entrada

Grava um novo registro no arquivo

FileEditor.java

(4 de 5)

```
85    // exclui registro do arquivo
86    public void deleteRecord( int accountNumber )
87       throws IllegalArgumentException, IOException
88    {
89       RandomAccessAccountRecord record = getRecord( accountNumber );
90
91       if ( record.getAccount() == 0 )
92          throw new IllegalArgumentException( "Account does not exist" );
93
94       // busca registro apropriado no arquivo
95       file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );
96
97       // cria um registro em branco a gravar no
98       record = new RandomAccessAccountRecord();
99       record.write( file );
100   } // fim do método deleteRecord
101
102   // lê e exibe registros
103   public void readRecords()
104   {
105      RandomAccessAccountRecord record = new RandomAccessAccountRecord();
106
107      System.out.printf( "%-10s%-15s%-15s%10s\n", "Account",
108         "First Name", "Last Name", "Balance" );
109
```

Recupera o registro com base no número da conta

Posiciona o ponteiro de posição de arquivo no registro

Grava um registro em branco no arquivo

Resumo

FileEditor.java

```
110     try // lê um registro e exibe
111     {
112        file.seek( 0 );
113
114        while ( true )
115        {
116           do
117           {
118              record.read( file );
119           } while ( record.getAccount() == 0
120
121           // exibe conteúdo do registro
122           System.out.printf( "%-10d%-15s%-15s%10.2f\n",
123              record.getAccount(), record.getFirstName(),
124              record.getLastName(), record.getBalance() );
125        } // fim do while
126     } // fim do try
127     catch ( EOFException eofException ) // fecha o arquivo
128     {
129        return; // fim do arquivo foi alcançado
130     } // fim do catch
131     catch ( IOException ioException )
132     {
133        System.err.println( "Error reading file." );
134        System.exit( 1 );
135     } // fim do catch
136  } // fim do método readRecords
137} // fim da classe FileEditor
```

Retorna ao começo do arquivo para ler todos os registros

Lê até que um registro não em branco seja localizado

Exibir um registro

Resumo

Transação

Processor.java

(1 de 7)

```
1  // Fig. 14.35: TransactionProcessor.java
2  // Um programa de processamento de transações c/ arquivos de acesso aleatório.
3  import java.io.IOException;
4  import java.util.NoSuchElementException;
5  import java.util.Scanner;
6
7  import com.deitel.jhtp6.ch14.RandomAccessAccountRecord;
8
9  public class TransactionProcessor
10 {
11    private FileEditor dataFile;
12    private RandomAccessAccountRecord record;
13    private MenuOption choices[] = { MenuOption.PRINT,
14       MenuOption.UPDATE, MenuOption.NEW,
15       MenuOption.DELETE, MenuOption.END };
16
17    private Scanner input = new Scanner( System.in );
18
19    // obtém o nome de arquivo e abre o arquivo
20    private boolean openFile()
21    {
22       try // tenta abrir o arquivo
23       {
24          // chama o método auxiliar para abrir o arquivo
25          dataFile = new FileEditor( "clients.dat" );
26       } // fim do try
```

```
27       catch ( IOException ioException )
28       {
29          System.err.println( "Error opening file." );
30          return false;
31       } // fim do catch
32
33       return true;
34    } // fim do método openFile
35
36    // fecha o arquivo e termina o aplicativo
37    private void closeFile()
38    {
39       try // fecha o arquivo
40       {
41          dataFile.closeFile();
42       } // fim do try
43       catch ( IOException ioException )
44       {
45          System.err.println( "Error closing file." );
46          System.exit( 1 );
47       } // fim do catch
48    } // fim do método closeFile
49
```

Resumo

Transação

Processor.java

(2 de 7)

Resumo

Transação

Processor.java

(3 de 7)

```
50    // cria, atualiza ou exclui o registro
51    private void performAction( MenuOption action )
52    {
53       int accountNumber = 0; // número de conta do registro
54       String firstName; // nome da conta
55       String lastName; // sobrenome da conta
56       double balance; // saldo da conta
57       double transaction; // valor monetário a alterar no saldo
58
59       try // manipular arquivos com base na opção selecionada
60       {
61          switch ( action ) // alterna com base na opção selecionada
62          {
63             case PRINT:
64                System.out.println();
65                dataFile.readRecords();
66                break;
67             case NEW:
68                System.out.printf( "\n%s%s\n%s\n%s",
69                   "Enter account number,",
70                   " first name, last name and balance.",
71                   "(Account number must be 1 - 100)", "? " );
72
73                accountNumber = input.nextInt(); // lê o número de conta
74                firstName = input.next(); // lê o nome
75                lastName = input.next(); // lê o sobrenome
76                balance = input.nextDouble(); // lê o saldo
77
```

Lê e exibir todos os registros

Solicita ao usuário novos dados no registro

Recupera novos dados no registro

Resumo

Transação Processor.java

(4 de 7)

```
78             dataFile.newRecord( accountNumber, firstName,
79                lastName, balance ); // cria novo registro
80             break;
81          case UPDATE:
82             System.out.print(
83                "\nEnter account to update ( 1 - 100 ): " );
84             accountNumber = input.nextInt();
85             record = dataFile.getRecord( accountNumber );
86
87             if ( record.getAccount() == 0 )
88                System.out.println( "Account does not exist." );
89             else
90             {
91                // exibe o conteúdo de registro
92                System.out.printf( "%-10d%-12s%-12s%10.2f\n\n",
93                   record.getAccount(), record.getFirstName(),
94                   record.getLastName(), record.getBalance() );
95
96                System.out.print(
97                   "Enter charge ( + ) or payment ( - ): " );
98                transaction = input.nextDouble();
99                dataFile.updateRecord( accountNumber, // atualiza registro
100                  transaction );
101
102               // recupera o registro atualizado
103               record = dataFile.getRecord( accountNumber );
104
```

Grava um novo registro no arquivo

Recupera um valor de transação

Atualiza o registro no arquivo

Inserir rodapé

Figura 14.35 Programa de processamente de transação. (Parte 1 de 7.)

Transação

```
105                // exibe o registro atualizado
106                System.out.printf( "%-10d%-12s%-12s%10.2f\n",
107                   record.getAccount(), record.getFirstName(),
108                   record.getLastName(), record.getBalance() );
109             } // fim do else
110             break;
111          case DELETE:
112             System.out.print(
113                "\nEnter an account to delete (1 - 100): " );
114             accountNumber = input.nextInt();
115
116             dataFile.deleteRecord( accountNumber ); // exclui o registro
117             break;
118          default:
119             System.out.println( "Invalid action." );
120             break;
121       } // fim do switch
122    } // fim do try
123    catch ( NumberFormatException format )
124    {
125       System.err.println( "Bad input." );
126    } // fim do catch
127    catch ( IllegalArgumentException badAccount )
128    {
129       System.err.println( badAccount.getMessage() );
130    } // fim do catch
```

Recupera um número de conta do registro a ser excluído

Exclui um registro

(5 de 7)

```
131      catch ( IOException ioException )
132      {
133         System.err.println( "Error writing to the file." );
134      } // fim do catch
135      catch ( NoSuchElementException elementException )
136      {
137         System.err.println( "Invalid input. Please try again." );
138         input.nextLine(); // descarta entrada para o usuário tentar de novo
139      } // fim do catch
140   } // fim do método performAction
141
142   // permite ao usuário inserir escolha de menu
143   private MenuOption enterChoice()
144   {
145      int menuChoice = 1;
146
147      // exibe opções disponíveis
148      System.out.printf( "\n%s\n%s\n%s\n%s\n%s\n%s",
149         "Enter your choice", "1 - List accounts",
150         "2 - Update an account", "3 - Add a new account",
151         "4 - Delete an account", "5 - End program\n? " );
152
153      try
154      {
155         menuChoice = input.nextInt();
156      }
```

Resumo

Transação

Processor.java

(6 de 7)

Resumo

Transação

Processor.java

(7 de 7)

```
157      catch ( NoSuchElementException elementException )
158      {
159         System.err.println( "Invalid input." );
160         System.exit( 1 );
161      } // fim do catch
162
163      return choices[ menuChoice - 1 ]; // retorna escolha do usuário
164   } // fim do enterChoice
165
166   public void processRequests()
167   {
168      openFile();
169
170      // obtém a solicitação do usuário
171      MenuOption choice = enterChoice();
172
173      while ( choice != MenuOption.END )
174      {
175         performAction( choice );
176         choice = enterChoice();
177      } // fim do while
178
179      closeFile();
180   } // fim do método processRequests
181} // fim da classe TransactionProcessor
```

Edita um arquivo com base na opção de menu selecionada pelo usuário

Resumo

Transação

ProcessorTest.java

```
1  // Fig. 14.36: TransactionProcessorTest.java
2  // Testando o processador de transação.
3
4  public class TransactionProcessorTest
5  {
6     public static void main( String args[] )
7     {
8        TransactionProcessor application = new TransactionProcessor();
9        application.processRequests();
10    } // fim do main
11 } // fim da classe TransactionProcessorTest
```

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

va

```
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

(3 de 4)

Resumo

```
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
```

FileDemonstration.java

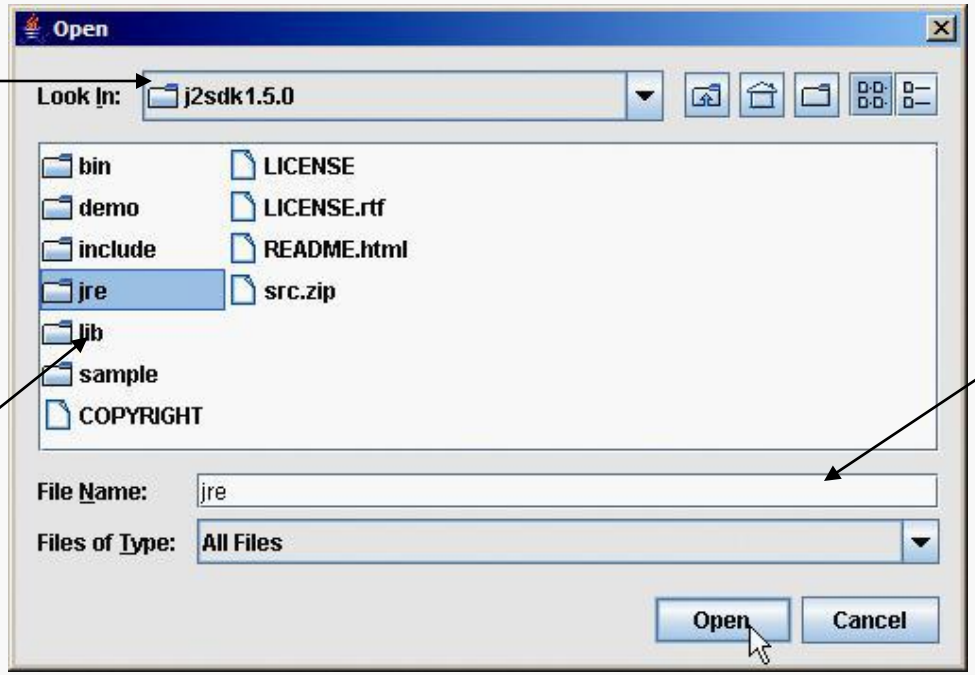
(4 de 4)

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
```

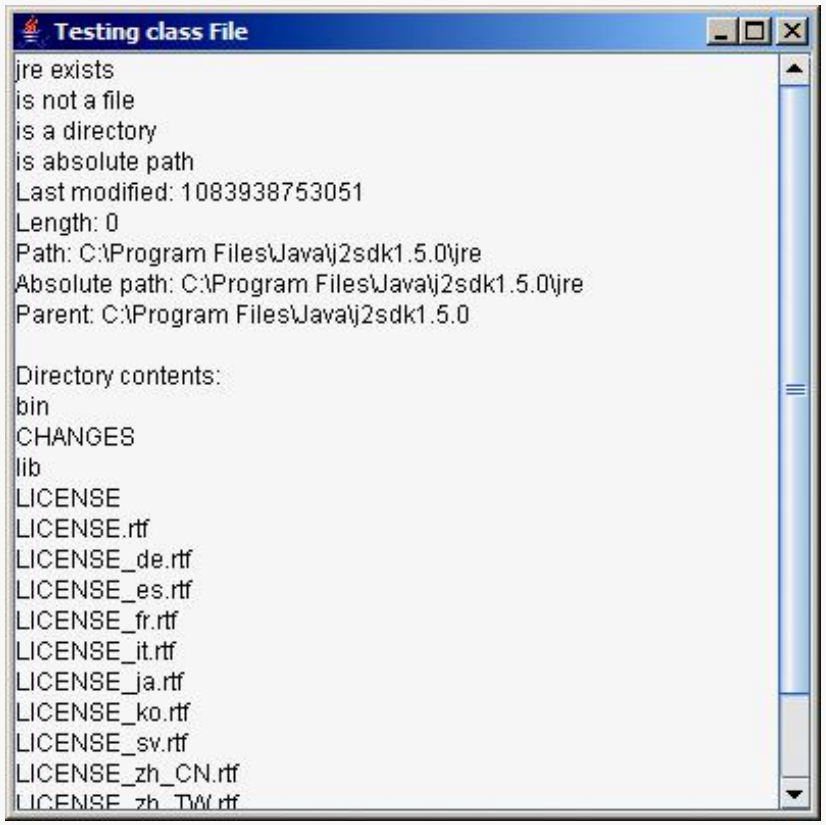


Seleciona localização do arquivo aqui

Arquivos e diretórios são exibidos aqui

Clique em **Open** para enviar um novo nome de arquivo ao programa

Resumo



FileDemonstration

Test.java

(2 de 2)