



# ARQUIVOS E FLUXOS (CAPÍTULO 14 DEITEL)

Carla Merkle Westphall  
INE-CTC-UFSC  
E-Mail: [carlamw@inf.ufsc.br](mailto:carlamw@inf.ufsc.br)  
URL: <http://moodle.ufsc.br>  
INE5605-Turma 0238A

# OBJETIVOS

## **Neste capítulo você vai aprender:**

- Criar, ler, escrever e atualizar arquivos.
- Usar a classe File para recuperar informações sobre arquivos e diretórios.
- A hierarquia de classes de stream de entrada/saída.
- A diferença entre arquivos de texto e binários.
- Processar arquivos de acesso seqüencial.
- Usar a classe Scanner e Formatter para processar arquivos de texto.
- Usar as classes FileInputStream e FileOutputStream.
- Usar as classes ObjectInputStream e ObjectOutputStream.
- Usar o diálogo JFileChooser.
- Arquivos de acesso aleatório: criação, gravação, leitura
- Estudo de caso: Um programa de processamento de transação



# Introdução

- **Armazenamento de dados em variáveis e arrays é temporário**
- **Arquivos são usados para manter a persistência de uma grande quantidade de dados por longos períodos**
  - **Mesmo após o término do programa que criou os dados**
  - **Persistência de dados** – dados persistem além da duração da execução do programa



# Hierarquia de dados

- **Computadores processam todos os itens de dados como combinações de zeros ou uns**
  - **Bit** – menor item de dado no computador, pode assumir o valor 0 ou 1
  - **Byte** – 8 bits
  - **Caracteres** – maior item de dados
    - **Consiste de dígitos decimais, letras e símbolos especiais**
    - **Conjunto de caracteres** – conjunto de todos os caracteres usados para escrever programas e representar itens de dados
      - **Unicode** – caracteres compostos de dois bytes
      - **ASCII**



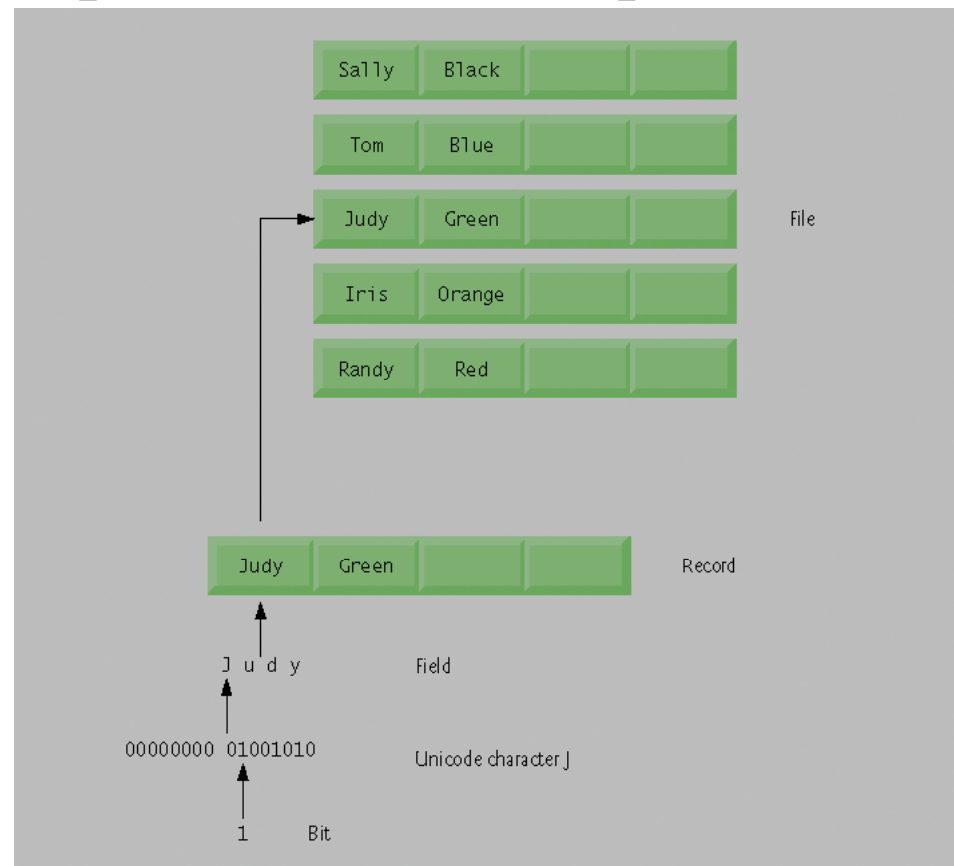
# Hierarquia de dados

- **Fields (Campos)** – um grupo de caracteres ou bytes que possuem um significado
  - Ex.: as letras minúsculas e maiúsculas que compõem o nome de um empregado
- **Record (Registro)** – um grupo de campos relacionados
  - Ex.: nome, número da carteira de trabalho, salário de um empregado
  - Chave de registro– identifica um registro como pertencendo a uma pessoa ou entidade em particular – usado para facilitar a recuperação de registros específicos
- **File (Arquivo)** – um grupo de registros relacionados
  - Ex.: arquivo contendo os cadastros dos empregados de uma empresa
  - De maneira mais ampla um arquivo pode armazenar seqüências de bytes e a forma de interpretar estes bytes é definido pelo programador.



# Hierarquia de dados

- Itens de dados processados pelo computador formam uma hierarquia de dados que se tornam maiores e mais complexos partido de bits a arquivos



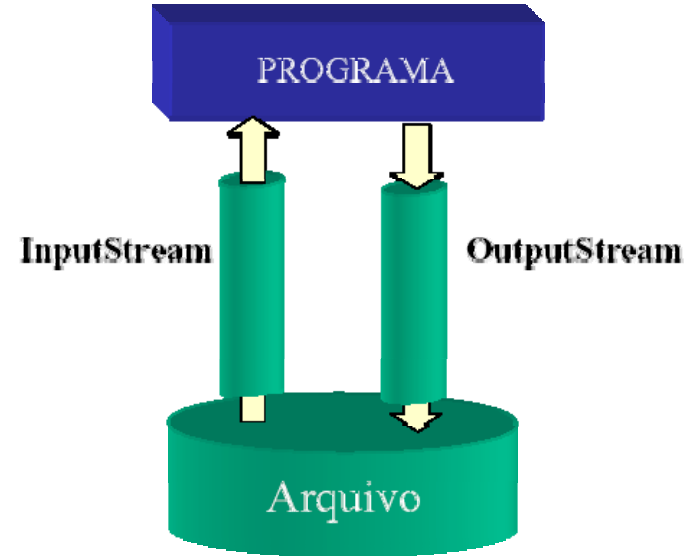
# Hierarquia de dados

- **Há várias maneiras de organizar registros em um arquivo**
  - **Arquivo seqüencial – forma mais usual de organizar registros em um arquivo**
  - **Base de dados – um grupo de arquivos relacionados**



# Arquivos e Streams

- Programa Java abre um arquivo através da criação de um objeto e a associação de um fluxo (stream) de bytes ou caracteres a este objeto.
  - Stream é um caminho atravessado pelos dados em um programa;
    - Permite ler e escrever uma Sequência de bytes/caractere em memória para um Arquivo de dados
- Fluxo (stream) pode ser associado à dispositivos
  - System.in (objeto de fluxo entrada padrão – ex. teclado)
  - System.out (objeto de fluxo saída padrão – ex. tela)
  - System.err (objeto de fluxo de erro padrão – ex. tela)





# Arquivos e Streams (Fluxos)

- Java vê cada arquivo como um **fluxo (stream)** seqüencial de bytes



- Sistema operacional oferece os mecanismos para determinar o final do arquivo
  - Programa Java processando um stream de bytes recebe uma indicação do sistema operacional quando o programa alcança o fim do stream
  - Marcador de fim-de-arquivo (End-of-file marker)
  - Contagem do total de bytes no arquivo



# Arquivos e Streams

- **Streams (fluxos)**
  - **Fluxos baseado em byte**
    - Fluxos de entrada e saída de bytes para arquivos
    - Armazenando dados no formato binário
    - Arquivos binários – criado a partir de fluxos baseados em byte, lido pelo programa que converte o dado em um formato legível para humanos
  - **Fluxos baseados em caracteres**
    - Fluxos de entrada e saída de caracteres para arquivos
    - armazena dados como uma sequência de caracteres
    - Arquivos de texto – criado a partir de fluxos baseados em caracteres, pode ser lido com um editor de texto



# Arquivos e Streams

- **Classes `java.io`**
  - `FileInputStream` e `FileOutputStream` – E/S baseado em byte
  - `FileReader` e `FileWriter` – E/S baseado em caractere
  - `ObjectInputStream` e `ObjectOutputStream` – usado para entrada e saída 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 usado para facilitar a leitura de dados no arquivo
  - `Formatter` – pode ser usado para facilitar a escrita de dados no arquivo



# Classe File

- **Classe File é útil para recuperar informações sobre arquivos e diretórios no disco**
- **Objetos da classe File não abrem arquivos ou oferecem qualquer capacidade de processamento de arquivo**



# *Criando objetos File*

**Classe File oferece quatro construtores:**

- 1. public File(String nome)**
  - nome especificando o nome e caminho (localização do arquivo no disco)
- 2. public File( String pathToName, String name )**
  - primeiro especificando o caminho e a segunda o nome do arquivo
- 3. public File( File directory, String name )**
  - objeto File especificando o caminho e name especificando o nome do arquivo
- 4. public File( URI uri )**
  - Uma argumento objeto URI especificando o nome e a localização do arquivo (file:/C:/data.txt)

**Diferentes tipos de caminhos**

- Caminho absoluto – contém todos os diretórios, partindo do diretório raiz, que leva ao arquivo ou diretório específico
- Caminho relativo – normalmente parte do diretório em que a aplicação está



Método	Descrição
<code>boolean canRead()</code>	Retorna <code>true</code> se um arquivo pode ser lido pela aplicação; senão retorna <code>false</code> .
<code>boolean canWrite()</code>	Retorna <code>true</code> se a aplicação pode escrever no arquivo; senão retorna <code>false</code> .
<code>boolean exists()</code>	Retorna <code>true</code> se o nome especificado como argumento no construtor de <code>File</code> é um arquivo ou diretório no caminho especificado; senão retorna <code>false</code> .
<code>boolean isFile()</code>	Retorna <code>true</code> se o nome especificado como argumento no construtor de <code>File</code> é um arquivo; senão retorna <code>false</code> .
<code>boolean isDirectory()</code>	Retorna <code>true</code> se o nome especificado como argumento no construtor de <code>File</code> é um diretório; senão retorna <code>false</code> .
<code>boolean isAbsolute()</code>	Retorna <code>true</code> se o argumento especificado no construtor de <code>File</code> indica um caminho absoluto para um arquivo ou diretório; senão retorna <code>false</code> .

### Métodos File. (Parte 1 de 2)



Método	Descrição
<code>String getAbsolutePath()</code>	Retorna uma string com o caminho absoluto do arquivo ou diretório.
<code>String getName()</code>	Retorna uma string com o nome do arquivo ou diretório.
<code>String getPath()</code>	Retorna uma string com o caminho do arquivo ou diretório.
<code>String getParent()</code>	Retorna uma string com o diretório pai do arquivo ou diretório (i.e., o diretório na qual o arquivo ou diretório pode ser encontrado).
<code>Long length()</code>	Retorna o tamanho do arquivo, em bytes. Se o objeto <code>File</code> object é um diretório, 0 é retornado.
<code>Long lastModified()</code>	Retorna uma representação dependente de plataforma do tempo na qual o arquivo ou diretório foi modificado. O valor retornado é útil apenas para comparação com outros valores retornados por este método.
<code>String[] list()</code>	Retorna um array de strings representando o conteúdo de um diretório. Retorna <code>null</code> se o objeto <code>File</code> não representa um diretório.

### Métodos File. (Parte 2 de 2)



# *Demonstrando a classe `File`*

- **Métodos `File` comuns**
  - `exists` – retorna `true` se o arquivo existe onde ele foi especificado
  - `isFile` – retorna `true` se `File` é um arquivo, não um diretório
  - `isDirectory` – retorna `true` se `File` é um diretório
  - `getPath` – retorna o caminho do arquivo como `string`
  - `list` – obtém conteúdo de um diretório
- **Caractere separador – usado para separar diretórios e arquivos em um caminho**
  - **Windows** usa `\`
  - **UNIX** usa `/`
  - **Java** processa os dois caracteres, `File`. `pathSeparator` pode ser usado para obter o caractere separados local do computador apropriado





## Resumo

FileDemonstration  
Test.java

(1 de 1)

```
1 // Arquivo: 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 do main
15 } // fim da classe FileDemonstrationTest
```



## Resumo

FileDemonstration  
.java

(1 de 2)

```

1 // Arquivo: FileDemonstration.java
2 // Demonstrando a classe File.
3 import java.io.File;
4
5 public class FileDemonstration
6 {
7     // apresenta informações sobre um arquivo especificado pelo usuário
8     public void analyzePath( String path )
9     {
10         // cria objeto File baseado no caminho especificado
11         File name = new File( path );
12
13         if ( name.exists() ) // se o nome existe, imprime informações sobre ele
14         {
15             // apresenta informações sobre o arquivo (ou diretório)
16             System.out.printf(
17                 "%s\n%s\n%s\n%s\n%s\n%s\n",
18                 name.getName(), "exists",
19                 ( name.isFile() ) ? "is a file" : "is not a file",
20                 ( name.isDirectory() ) ? "is a directory" : "is not a directory",
21                 ( name.isAbsolute() ) ? "is absolute path" : "is not absolute path",
22                 name.lastModified(), "Length: ",
23                 "Path: ", name.getPath(), "Absolute Path: ",
24                 name.getAbsolutePath(), "Parent Path: " );
25
26             // recupera o caminho digitado na forma de string
27             // recupera o tamanho do arquivo em bytes
28             // recupera o caminho absoluto do arquivo ou diretório
29             // recupera o diretório pai (caminho onde o arquivo ou diretório do objeto File pode ser encontrado)
30         }
31     }
32 }

```

Cria um novo objeto File; usuário

Retorna true se o arquivo ou diretório especificado existe

Returns true if name is a

Retorna true se name é um diretório, não um arquivo

Retorna true se path é um caminho absoluto

Recupera a data de últi

Recupera tamanho do arquivo

Recupera o caminho digitado na forma de string

bytes

Recupera caminho absoluto do arquivo ou diretório

Recupera diretório pai (caminho onde o arquivo ou diretório do objeto File pode ser encontrado)

tion,  
ved.

## Resumo

### FileDemonstration

```
28     if ( name.isDirectory() ) // lista conteúdo do 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 do else
36 } // fim do if exterior
37 else // não é arquivo ou diretório, mensagem de erro
38 {
39     System.out.printf( "%s %s", path, "does not exist." );
40 } // fim do else
41 } // fim do método analyzePath
42 } // fim da classe FileDemonstration
```

Retorna true se File é um diretório, não um arquivo

Recupera e apresenta o conteúdo do diretório

(2 de 2)



## Resumo

### FileDemonstration

### Test.java

(2 de 3)

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

Metal works

Notepad

SampleTree

Styl ePad

SwingAppl et

SwingSet2

TableExamp le



## Resumo

FileDemonstration

Test.java

(3 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



# Criando um Arquivo de Texto de Acesso Sequencial

- **Classe `Formatter` pode ser usada para criar um arquivo de texto para escrita**
  - **Passado o nome do arquivo no construtor**
  - **Se o arquivo não existir, ele será criado**
  - **Se o arquivo já existir, o conteúdo será descartado**
  - **Use método `format` para escrever texto formatado no arquivo**
  - **Use o método `close` para fechar o objeto `Formatter`**
    - **se não for chamado, o SO fecha o arquivo quando o programa terminar**



# Criando um Arquivo de Texto de Acesso Seqüencial

- **Exceções possíveis**
  - **SecurityException** – ocorre se o usuário não tiver permissão para escrever dados no arquivo sendo aberto pelo objeto **Formatter**
  - **FileNotFoundException** – ocorre quando da abertura do arquivo usando o objeto **Formatter**, se o arquivo não pode ser encontrado e um novo arquivo não pode ser criado
  - **NoSuchElementException** – ocorre quando uma entrada inválida é lida por um objeto **Scanner**
  - **FormatterClosedException** – ocorre quando da tentativa de se escrever em um arquivo usando um objeto **Formatter** já fechado.



## Resumo

### RegistroConta.java

(1 de 3)

```
1 // Arquivo: RegistroConta.java
2 // Uma classe que representa um registro de informação.
3 package com.deitel.jhtp6.ch15; // empacotado para reuso
4
5 public class RegistroConta
6 {
7     private int numeroConta;
8     private String nome;
9     private String sobrenome;
10    private double saldo;
11
12    // construtor sem argumento chama outro construtor com valores default
13    public RegistroConta ()
14    {
15        this( 0, "", "", 0.0 ); // chama construtor com quatro argumentos
16    } // fim do construtor sem argumento AccountRecord
17
18    // Inicializa um registro
19    public RegistroConta(int numero, String nome, String sobrenome, double saldo )
20    {
21        setNumeroConta(numero);
22        setNome( nome );
23        setSobrenome( sobrenome );
24        setSaldo( saldo );
25    } // fim do construtor com quatro argumentos RegistroConta
26
```





## Resumo

RegistroConta.java

(2 de 3)

```
27 // define número da conta
28 public void setNumeroConta( int conta )
29 {
30     this.numeroConta = conta;
31 } // fim do método setNumeroConta
32
33 // obter número da conta
34 public int getNumeroConta ()
35 {
36     return numeroConta;
37 } // fim do método getNumeroConta
38
39 // atribui nome
40 public void setNome ( String nome )
41 {
42     this.nome = nome;
43 } // fim do método setNome
44
45 // obtém nome
46 public String getNome()
47 {
48     return nome;
49 } // fim do método getNome
50
51 // atribuir sobrenome
52 public void setSobrenome( String sobrenome )
53 {
54     this.sobrenome = sobrenome;
55 } // fim do método setSobrenome
56
```



## Resumo

RegistroConta.java

(3 de 3)

```
57 // obtém sobrenome
58 public String getSobrenome()
59 {
60     return sobrenome;
61 } // fim do método getSobrenome
62
63 // define o saldo da conta
64 public void setSaldo( double valor )
65 {
66     saldo = valor;
67 } // fim do método setSaldo
68
69 // define o saldo da conta
70 public double getSaldo ()
71 {
72     return saldo;
73 } // fim do método detSaldo
74 } // fim da classe RegistroConta
```



## Resumo

TesteCriaArquivo  
Texto.java

(1 de 1)

```
1 // Arquivo: TesteCriaArquivoTexto.java
2 // Testando a classe CriaArquivoTexto.
3
4 public class TesteCriaArquivoTexto
5 {
6     public static void main( String args[] )
7     {
8         CriaArquivoTexto aplicacao = new CriaArquivoTexto ();
9
10        aplicacao.abreArquivo();
11        aplicacao.adicionaRegistros();
12        aplicacao.fecharArquivo();
13    } // fim do main
14 } // fim da classe TesteCriaArquivoTexto
```



## Resumo

Cri aArqui voTexto

.j ava

(1 de 4)

// Escreve dados em um arquivo de texto com a classe Formatter.

import java.io.FileNotFoundException;

import java.lang.SecurityException;

import java.util.Formatter; ←

Usado para escrever dados no arquivo

import java.util.FormatterClosedException;

import java.util.NoSuchElementException;

import java.util.Scanner; ←

import com.deitel.jhtp6.ch15.RegistroConta;

Usado para ler entradas do usuário

public class CriaArquivoTexto {

private Formatter saida; // objeto usado para escrever dados no arquivo

// permite ao usuário abrir o arquivo

public void abreArquivo() {

try {

saida = new Formatter( "clientes.txt" );

Abre arquivo Cl i entes. txt para  
escrita

} // fim do try

catch ( SecurityException securit ←

System.err.println(

"Voce não tem acesso a escrita neste arquivo. ",

Objeto usado para saída de dados para o  
arquivo

System.exit( 1 );

} // fim do catch

catch ( FileNotFoundException filesNotFoundException ) {

System.err.println( "Erro na criacao do arquivo." );

System.exit( 1 );

} // fim do catch

} // fim do método abreArquivo



## Resumo

```
// adiciona registro ao arquivo
public void adicionaRegistros()
```

```
{
```

```
    // objeto a ser escrito no arquivo
```

```
    RegistroConta registro = new RegistroConta();
```

```
    Scanner entrada = new Scanner( System.in );
```

```
    System.out.printf( "%s\n%s",
```

```
        "Digite numero da conta (> 0 - 0 para sair), nome, sobrenome e saldo",
```

```
    try {
```

```
        int numConta = entrada.nextInt() ; // possível excessão nao tratada
```

```
        while ( numConta!=0 ) // loop até o indicador de fim de arquivo
```

```
        {
```

```
            // lê dados a serem salvos no arquivo
```

```
            registro.setNumeroConta(numConta ); // lê numero da conta
```

```
            registro.setNome (entrada.next() ); // lê nome
```

```
            registro.setSobrenome(entrada.next() ); // lê sobrenome
```

```
            registro.setSaldo(entrada.nextDouble() ); // lê saldo
```

```
            // Escreva novo registro
```

```
            saida.format( "%d %s %s %.2f\n", registro.getNumeroConta(),
```

```
                registro.getNome(), registro.getSobrenome(),
```

```
                registro.getSaldo());
```

Cria RegistroConta para ser preenchido com entradas do usuário

Cria Scanner para ler dados de entrada do usuário

Lê entrada, armazena dados em RegistroConta

Escreve informações em RegistroConta no arquivo



## Resumo

CriarArquivoTexto

Java

Arquivo fechado e se  
tentou escrever nele

(3 de 4)

Erro com entradas digitadas pelo  
usuário

Fecha arquivo

```

        System.out.printf( "%s %s\n%s", "Digite numero da conta (>0),",
                                "nome, sobrenome e saldo.", "? " );
        numConta = entrada.nextInt();
    } // fim do while
}
catch ( FormatterClosedException formatterClosedException )
{
    System.err.println( "Erro de escrita no arquivo." );
    return;
} // fim do catch
catch ( NoSuchElementException elementException )
{
    System.err.println( "Entrada inválida. Tente novamente." );
    entrada.nextLine(); // descarta dados digitados
} // fim do catch
} // fim do método setRegistro

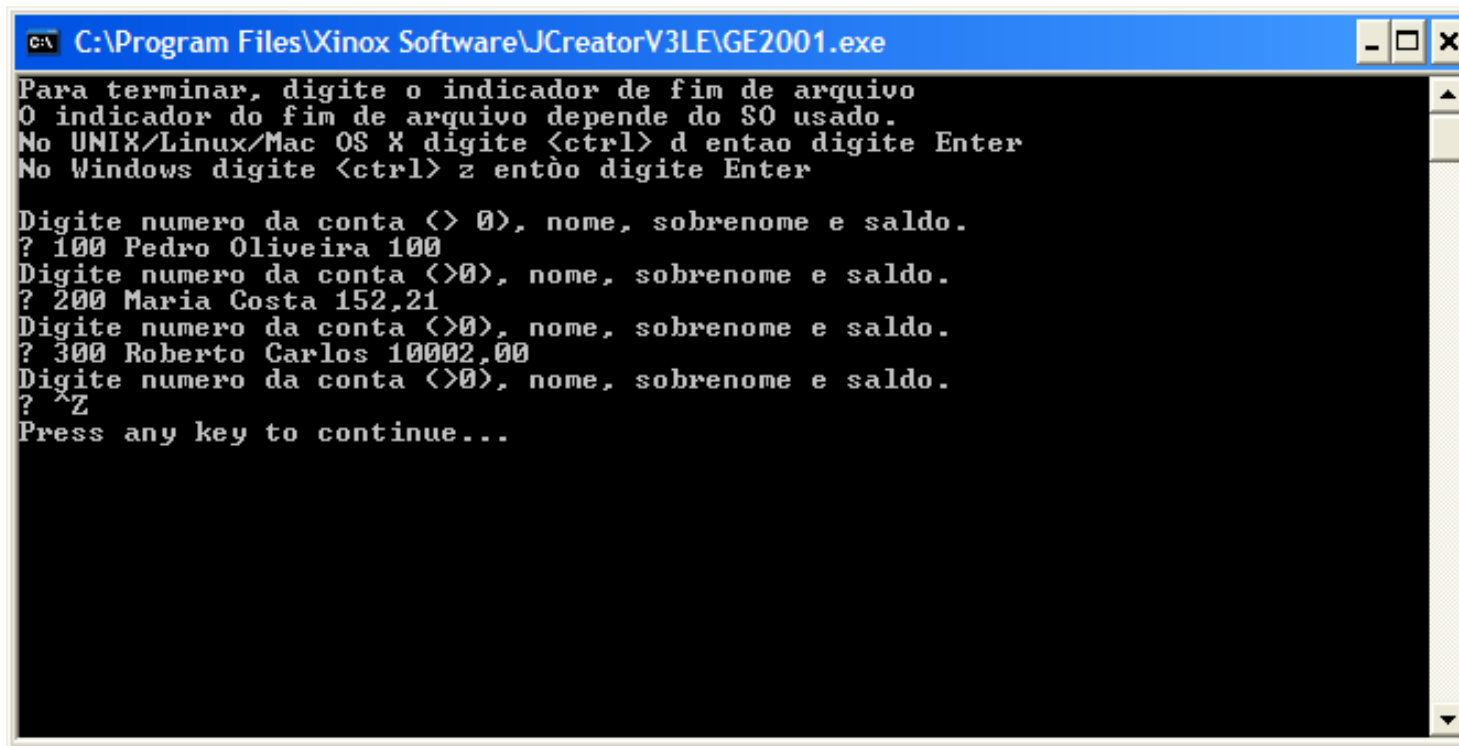
// fecha arquivo
public void fecharArquivo()
{
    if ( saida != null )
        saida.close();
} // fim do método fecharArquivo
} // fim da classe CriarArquivoTexto

```



## Resumo

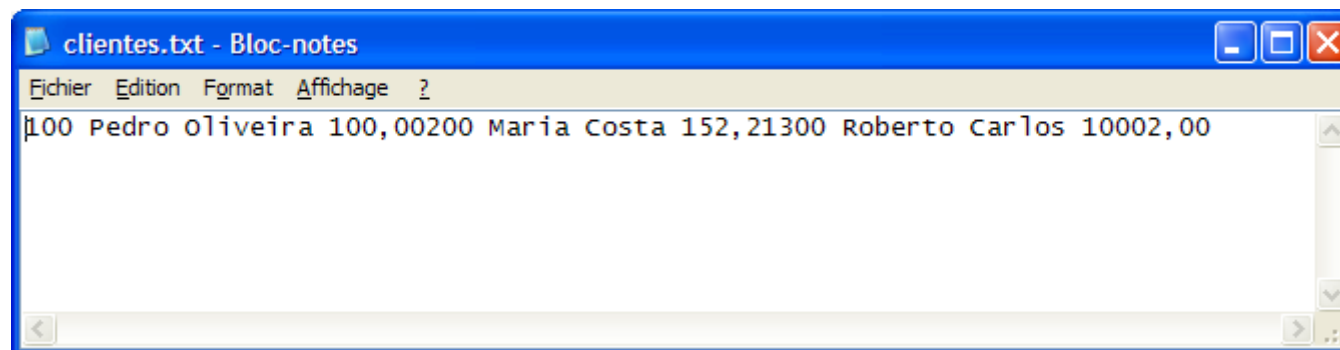
Execução de  
TesteCri aArqui vo  
Texto.java



```
C:\Program Files\Xinox Software\JCreatorV3LE\GE2001.exe

Para terminar, digite o indicador de fim de arquivo
O indicador do fim de arquivo depende do SO usado.
No UNIX/Linux/Mac OS X digite <ctrl> d entao digite Enter
No Windows digite <ctrl> z então digite Enter

Digite numero da conta (> 0), nome, sobrenome e saldo.
? 100 Pedro Oliveira 100
Digite numero da conta (>0), nome, sobrenome e saldo.
? 200 Maria Costa 152,21
Digite numero da conta (>0), nome, sobrenome e saldo.
? 300 Roberto Carlos 10002,00
Digite numero da conta (>0), nome, sobrenome e saldo.
? ^Z
Press any key to continue...
```



```
clientes.txt - Bloc-notes
Fichier Edition Format Affichage ?

100 Pedro oliveira 100,00200 Maria Costa 152,21300 Roberto Carlos 10002,00
```



# Lendo dados de um arquivo de texto de acesso seqüencial

- Os dados são armazenados nos arquivos e eles podem ser recuperados para processamento quando necessário
- Objeto Scanner pode ser usado para ler os dados seqüencialmente do arquivo de texto
  - Passar um objeto File representando o arquivo a ser lido para o construtor Scanner
    - FileNotFoundException ocorre se o arquivo não for encontrado
  - Dados do arquivo podem ser lidos da mesma forma que fizemos até agora para leitura de dados do teclado – nextInt(), nextDouble(), next(), etc.
    - IllegalStateException ocorre se tentamos ler dados de um objeto Scanner fechado



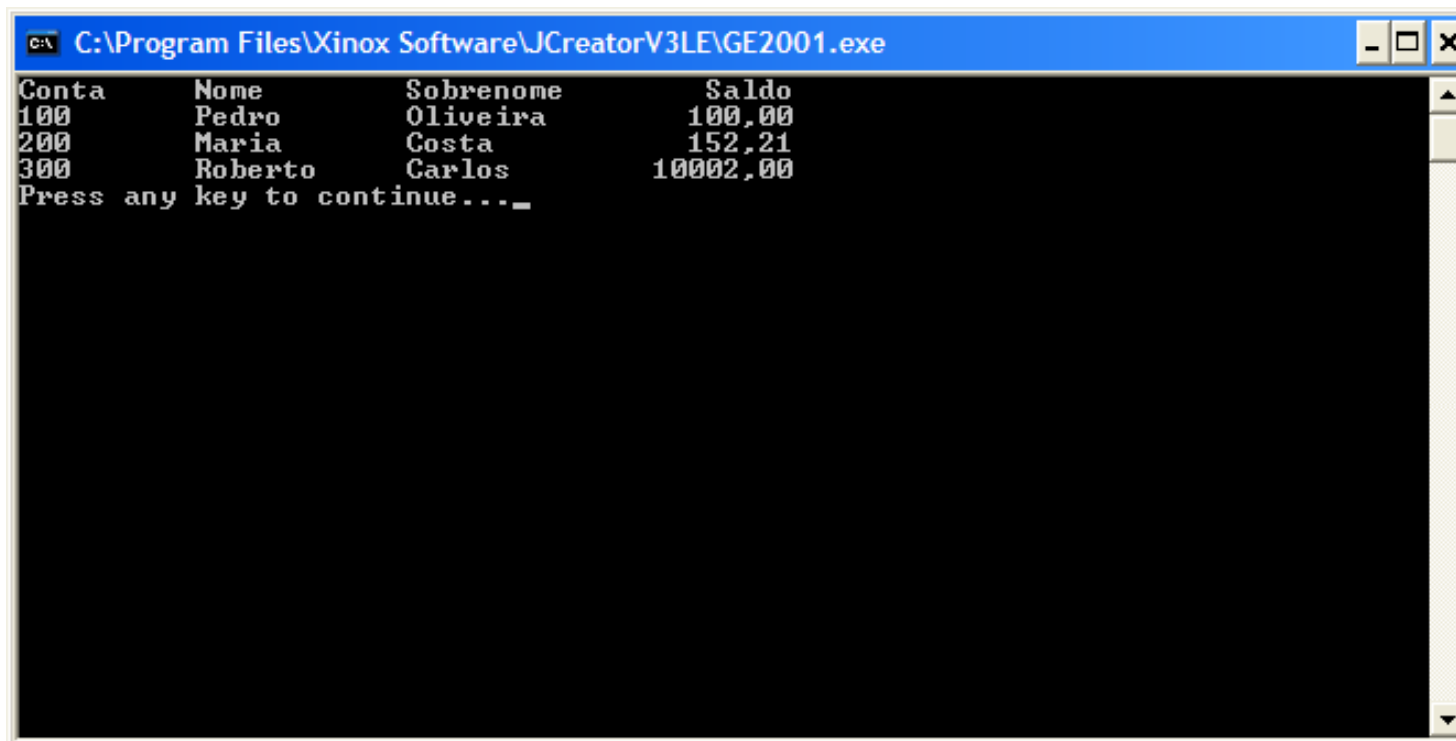


## Resumo

TesteLei tura

Arqui voTexto. j ava

```
1 // Arqui vo: TesteLei turaArqui voTexto. j ava
2 // Este programa testa a classe Lei turaArqui voTexto.
3
4 public class TesteLei turaArqui voTexto
5 {
6     public static void main( String args[] )
7     {
8         Lei turaArqui voTexto apl i cacao = new Lei turaArqui voTexto();
9
10        apl i cacao. abreArqui vo();
11        apl i cacao. lerRegi stros();
12        apl i cacao. fecharArqui vo();
13    } // fim do mai n
14 } // fim da classe TesteLei turaArqui voTexto
```



Conta	Nome	Sobrenome	Saldo
100	Pedro	Oliveira	100,00
200	Maria	Costa	152,21
300	Roberto	Carlos	10002,00

Press any key to continue...\_



## Resumo

LeituraArquivo

Texto.java

(1 de 3)

```
1 // Arquivo: LeituraArquivoTexto.java
2 // Este programa lê um arquivo de texto e apresenta 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.RegistroConta;
10
11 public class LeituraArquivoTexto
12 {
13     private Scanner entrada;
14
15     // permite ao usuário abrir o arquivo
16     public void abreArquivo()
17     {
18         try
19         {
20             entrada = new Scanner( new File( "clientes.txt" ) );
21         } // end try
22         catch ( FileNotFoundException fileNotFoundException )
23         {
24             System.err.println( "Erro de abertura de arquivo." );
25             System.exit( 1 );
26         } // fim do catch
27     } // fim do método abreArquivo
28
```

Abre arquivo clientes.txt para  
leitura



## Resumo

LeituraArquivoText  
o.java

```

29 // lê registros do arquivo
30 public void lerRegistros()
31 {
32     // objeto a ser escrito na tela
33     RegistroConta registro = new RegistroConta();
34
35     System.out.printf( "%-10s%-12s%-12s%10.2f\n",
36         "Nome", "Sobrenome", "Saldo" );
37
38     try // lê registro do arquivo usando
39     {
40         while ( entrada.hasNext() )
41         {
42             registro.setNumeroConta( entrada.nextInt() ); // lê número da conta
43             registro.setNome( entrada.next() ); // lê nome
44             registro.setSobrenome( entrada.next() ); // lê sobrenome
45             registro.setSaldo( entrada.nextDouble() ); // lê saldo
46
47             // apresenta conteúdo do registro
48             System.out.printf( "%-10d%-12s%-12s%10.2f\n",
49                 registro.getNumeroConta(), registro.getNome(),
50                 registro.getSobrenome(), registro.getSaldo() );
51         } // fim do while
52     } // fim do try

```

Cria RegistroConta para  
armazenar dados lidos do  
arquivo

Enquanto houver dados a serem lidos do  
arquivo faça...

Lê dado do arquivo,  
armazena em  
RegistroConta

Apresenta conteúdo de  
RegistroConta



## Resumo

LeituraArquivoText  
o.java

(3 de 3)

```
53 catch ( NoSuchElementException elementException )
54 {
55     System.err.println( "Arquivo formatado de maneira inapropriada." );
56     entrada.close();
57     System.exit( 1 );
58 } // fim do catch
59 catch ( IllegalStateException stateException )
60 {
61     System.err.println( "Erro de leitura do arquivo." );
62     System.exit( 1 );
63 } // fim do catch
64 } // fim do lerRegistros
65
66 // fecha arquivo e termina aplicação
67 public void fecharArquivo()
68 {
69     if ( entrada != null )
70         entrada.close(); // fecha arquivo
71 } // fim do método fecharArquivo
72 } // fim da classe LerArquivoTexto
```

Fecha arquivo



## Atualizando arquivos de acesso seqüencial

- **Dados em vários arquivos sequenciais não podem ser modificados sem risco de destruir outros dados no arquivo**
- **Dados atuais no arquivo não podem ser sobreescritos se um novo dado não tem o mesmo tamanho**
- **Registros em arquivos de acesso sequencial não são usualmente atualizados. Em vez disso, o arquivo inteiro é normalmente reescrito**



# Estudo de Caso

- **Agenda pessoal**

Pessoa
- nome : String - endereco : String - telefone : String
+ Pessoa(pNome : String, pEndereco : String, pTelefone : String) + Pessoa() + Pessoa(p : Pessoa) + setNome(pNome : String) : void + setEndereco(pEndereco : String) : void + setTelefone(pTelefone : String) : void + getNome() : String + getEndereco() : String + getTelefone() : String + toString() : String

**Conterá dados sobre pessoas**



- Leitura dos dados cadastrados
- Inclusão de novos dados



## Pessoa.java

(1 de 2)

```
public class Pessoa {  
    private String nome;  
    private String endereco;  
    private String telefone;  
  
    public Pessoa(String pNome, String pEndereco,  
                  String pTelefone ) {  
        setNome(pNome);  
        setEndereco(pEndereco);  
        setTelefone(pTelefone);  
    }  
    public Pessoa() {  
        this("", "", "");  
    }  
    public Pessoa(Pessoa p) {  
        setNome(p.getNome());  
        setEndereco(p.getEndereco());  
        setTelefone(p.getTelefone());  
    }  
    public void setNome(String pNome) {  
        nome = new String(pNome);  
    }  
    public void setEndereco(String pEndereco) {  
        endereco = new String(pEndereco);  
    }  
}
```



## Pessoa.java

(2 de 2)

```
public void setTelefone(String pTelefone) {  
    telefone = new String(pTelefone);  
}  
public String getNome() {  
    return nome;  
}  
public String getEndereco() {  
    return endereco;  
}  
public String getTelefone() {  
    return telefone;  
}  
public String toString() {  
    return String.format("Nome: %s\nEndereco:  
                        %s\nTelefone: %s\n\n",  
                        getNome(), getEndereco(), getTelefone());  
}  
}
```





## Agenda.java

(1 de 6)

```
import java.util.*;
import java.io.*;
public class Agenda {
    private Vector<Pessoa> listaPessoas;
    private Formatter saida;
    private Scanner entrada;
    private Scanner teclado = new Scanner(System.in);
    private static final String NOMEARQUIVO = "agenda.txt";

    public static void main( String args[] ) {
        Agenda agenda = new Agenda();
        agenda.abreArquivo(); // abre e lê arquivo agenda.txt
        agenda.fecharArquivo(); // fecha arquivo agenda.txt
        agenda.menuAgenda(); // apresenta menu da agenda
    }
    /** Abre o arquivo agenda.txt e lê dados
    public void abreArquivo() {
        try {
            Scanner entrada = new Scanner(new File(NOMEARQUIVO));
            lerDados(entrada);
        } catch (FileNotFoundException fileNotFoundException) {
            System.err.println( "Agenda vazia..." );
            criarArquivo(); // arquivo criado caso não existir
        }
    }
}
```



```
/** Cria o arquivo agenda.txt */
public void criarArquivo() {
    try {
        saida = new Formatter( NOMEARQUIVO );
    }
    catch ( SecurityException securityException )
    {
        System.err.println(
            "Voce não tem acesso a escrita neste arquivo." );
        System.exit( 1 );
    }
    catch ( FileNotFoundException filesNotFoundException )
    {
        System.err.println( "Erro na criação do arquivo." );
        System.exit( 1 );
    }
}

/** fecha arquivo para leitura e escrita */
public void fecharArquivo() {
    if ( entrada != null )
        entrada.close(); // fecha arquivo
    if ( saida != null )
        saida.close();
}
```

## Agenda.java

(2 de 6)



## Agenda.java

(3 de 6)

```
/** Leitura dos dados do arquivo */
public void lerDados(Scanner entrada) {
    listaPessoas = new Vector<Pessoa>();
    try {
        while ( entrada.hasNext() ) {
            Pessoa pessoa = new Pessoa();
            pessoa.setNome( entrada.nextLine() );
            pessoa.setEndereco( entrada.nextLine() );
            pessoa.setTelefone( entrada.nextLine() );
            listaPessoas.add(pessoa);
        } // fim do while
    } // fim do try
    catch ( NoSuchElementException elementException )
    {
        System.err.println( "Arquivo formatado de maneira
            inapropriada. " );
        entrada.close();
        System.exit( 1 );
    } // fim do catch
    catch ( IllegalStateException stateException )
    {
        System.err.println( "Erro de leitura do arquivo. " );
        System.exit( 1 );
    } // fim do catch
} // fim do lerDados
```



```
/** Apresenta Menu e realiza operação solicitada */
public void menuAgenda() {
    int comando = lerComando();
    while (comando!=3) {
        if (comando==1)
            cadastrarPessoa();
        else
            listarPessoas();
        comando = lerComando();
    }
}

/** Cadastra nova pessoa */
public void cadastrarPessoa() {
    Pessoa pessoa = new Pessoa();
    System.out.print("Nome: ");
    pessoa.setNome(teclado.nextLine());
    System.out.print("Endereco: ");
    pessoa.setEndereco(teclado.nextLine());
    System.out.print("Telefone: ");
    pessoa.setTelefone(teclado.nextLine());
    listaPessoas.add(pessoa);
    salvarPessoas();
    fecharArquivo();
}
```

## Agenda.java

(5 de 6)



```
/** Leitura dos comandos do usuário
public int lerComando() {
    int comando=0;
    while ((comando<1)||comando>3) {
        try {
            System.out.println("Digite 1: para cadastramento,
                                2: para listar pessoas,
                                3: para sair");

            System.out.print("COMANDO: ");
            comando = teclado.nextInt();
        } catch ( NoSuchElementException elementException ) {
            System.err.println( "Entrada inválida.
                                Tente novamente." );

            entrada.nextLine(); // descarta dados digitados
        } // fim do catch
    }
    teclado.nextLine();
    return comando;
}
```

## Agenda.java

(4 de 6)



## Agenda.java

(6 de 6)

```
/** Salva dados em listaPessoas no arquivo agenda.txt */
public void salvarPessoas() {
    criarArquivo(); // cria agenda.txt (reescrevendo)
    for (Pessoa pessoa: listaPessoas)
        saída.format( "%s\n%s\n%s\n", pessoa.getNome(),
                      pessoa.getEndereco(), pessoa.getTelefone() );
    fecharArquivo();
}

/** Lista pessoas cadastradas */
public void listarPessoas() {
    if (!listaPessoas.isEmpty()) {
        System.out.println("Relação de pessoas
                           cadastradas\n");
        for (Pessoa pessoa: listaPessoas)
            System.out.println(pessoa);
    } else
        System.out.println("Não há dados cadastrados");
}
}
```



# Serialização de objetos

- **Arquivos de texto,**
  - informação do tipo de dado é perdido
- **Serialização de objetos**
  - mecanismo para ler ou escrever um objeto inteiro em um arquivo
  - **Objeto serializado** – objeto representado como uma sequência de bytes, incluindo dados do objeto e a informação de tipo do objeto
- **Deserialização**
  - recriar objeto na memória a partir do dado no arquivo
- **Serialização e deserialização**
  - são realizadas com as classes `ObjectInputStream` e `ObjectOutputStream`, e métodos `readObject` and `writeObject`



## Criando um arquivo de acesso seqüencial usando serialização de objeto :

### Definindo a classe *RegistroContaSerializavel*

- Interface *Serializavel* – programador deve declarar uma classe que implemente a interface *Serializavel*, ou senão os objetos não podem ser escritos em um arquivo
- Para abrir um arquivo onde serão escritos objetos, crie um *FileOutputStream* ligado a um *ObjectOutputStream*
  - *FileOutputStream* oferece métodos para escrever saída baseada em byte para um arquivo
  - *ObjectOutputStream* usa *FileOutputStream* para escrever objetos no arquivo
    - Método *writeObject* de *ObjectOutputStream* escreve objetos no arquivo de saída
    - Método *close* de *ObjectOutputStream* fecha os dois objetos





## Resumo

```

1 // Arquivo: RegistroContaSerializavel.java
2 // Uma classe que representa um registro de informação
3 package com.deitel.jhtp6.ch15; // empacotado para reuso
4
5 import java.io.Serializable;
6
7 public class RegistroContaSerializavel implements Serializable
8 {
9     private int numeroConta;
10    private String nome;
11    private String sobrenome;
12    private double saldo;
13
14    // construtor sem argumentos chama outro construtor com valores default
15    public RegistroContaSerializavel ()
16    {
17        this( 0, "", "", 0.0 );
18    } // fim do construtor sem argumentos
19
20    // construtor com 4 argumentos que inicializa o registro
21    public RegistroContaSerializavel (
22        int numero, String nome, String sobrenome, double valor )
23    {
24        setNumeroConta(numero);
25        setNome( nome );
26        setSobrenome( sobrenome );
27        setSaldo( valor );
28    } // fim do construtor com 4 argumentos
29

```

### RegistroConta

Interface `Serializable` especifica que objetos `RegistroContaSerializavel` podem ser escritos em arquivo

(1 de 3)



## Resumo

RegistroConta

Serializavel.java

(2 de 3)

```
30 // define numero da conta
31 public void setNumeroConta( int numero )
32 {
33     numeroConta = numero;
34 }
35
36 // obter número da conta
37 public int getNumeroConta ()
38 {
39     return numeroConta;
40 }
41
42 // define nome
43 public void setNome( String nome )
44 {
45     this.nome = nome;
46 }
47
48 // obter nome
49 public String getNome()
50 {
51     return nome;
52 }
53
54 // define sobrenome
55 public void setSobrenome( String sobrenome )
56 {
57     this.sobrenome = sobrenome;
58 }
59
```



## Resumo

RegistroConta

Serializavel.java

(3 de 3)

```
60 // obter sobrenome
61 public String getSobrenome()
62 {
63     return sobrenome;
64 }
65
66 // definir saldo
67 public void setSaldo( double valor )
68 {
69     saldo = valor;
70 }
71
72 // obter saldo
73 public double getSaldo()
74 {
75     return saldo;
76 }
77 } // fim da classe RegistroContaSerializavel
```



## Resumo

TesteCri aArqui vo  
Sequenci al .j ava

```

1 // Arqui vo: TesteCri aArqui voSequenci al .j ava
2 // Testando a cl asse Cri aArqui voSequenci al .
3
4 publ ic cl ass TesteCri aArqui voSequenci al
5 {
6     publ ic stati c void mai n( Stri ng args[] )
7     {
8         Cri aArqui voSequenci al apl i cacao = new Cri aArqui voSequenci al ();
9
10        apl i cacao.abri rArqui vo();
11        apl i cacao.adi ci onarRegi stros();
12        apl i cacao.fecharArqui vo();
13    } // fim do mai n
14 } // Fim da cl asse TesteCri aArqui voSequenci al

```

Para terminar a entrada, digite o indicador de fim-de-arquivo  
Este indicador varia de acordo com o sistema operacional.  
No UNIX/Linux/Mac OS X type <ctrl> d entao digite Enter  
No Windows digite <ctrl> z entao digite Enter

Digite numero da conta (>0), nome, sobrenome e saldo.  
? Pedro Cabral  
Entrada invalida. Tente novamente.  
Digite numero da conta (>0), nome, sobrenome e saldo.  
? 100 Pedro Cabral 251,45  
Digite numero da conta (>0), nome, sobrenome e saldo.  
? 101 Mario Silva 100  
Digite numero da conta (>0), nome, sobrenome e saldo.  
? 200 Maria Fagundes 125,12  
Digite numero da conta (>0), nome, sobrenome e saldo.  
? ^Z  
Press any key to continue...\_

clientes.ser - Bloc-notes

Fichier Edition Format Affichage 2

-f |sr /com.deitel.jhttp6.ch14.RegistroContaSerializavelPN0aE0N- |I ,numeroContaD |saldoL ,nomet ;Ljava/lang/String;L  
sobrenomeq ~ ,xp d@onffffff |Pedrot -Cabralsq ~ e@Y t |Mariot |Silvasq ~ E@\_G@qzAht |Mariat qFagundes



Education,  
reserved.

```

1 // Arquivo: CriArquivoSerial.java
2 // Escreve objetos sequencialmente em um arquivo com 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.delitel.jhttp6.ch15.RegistroContaSerializavel;
10
11 public class CriArquivoSequencial
12 {
13     private ObjectOutputStream saida; // escreve dados para o arquivo
14
15     // permite ao usuário abrir o arquivo
16     public void abrirArquivo()
17     {
18         try // abre arquivo
19         {
20             saida = new ObjectOutputStream(
21                 new FileOutputStream("clientes.ser"));
22         }
23         catch (IOException ioException)
24         {
25             System.err.println("Erro na abertura do arquivo.");
26         }
27     } // fim do método abrirArquivo
28

```

Classe usada para criar stream de saída baseado em byte

Classe usada para criar stream de saída de dados de objeto baseado em byte

Abre arquivo clientes.ser para escrita

Sequencial.java

(1 de 4)



## Resumo

Cri aArqui vo

Sequenci al .j ava

(2 de 4)

```

29 // adicionar registros no arquivo
30 public void adicionarRegistros()
31 {
32     RegistroContaSerializavel registro; // objeto a ser escrito no arquivo
33     int numeroConta = 0; // numero de conta para o objeto registro
34     String nome; // nome para o objeto registro
35     String sobrenome; // sobrenome para o objeto registro
36     double saldo; // saldo para o objeto registro
37
38     Scanner entrada = new Scanner( System.in );
39     do { // loop até o usuario digitar conta=0
40         try
41         {
42             System.out.printf( "%s %s\n%s", "Di gi te numero da conta",
43                 (> 0 - 0 para sair), nome, sobrenome e saldo.", "?" );
44             numeroConta = entrada.nextInt(); // lê numero da conta
45             if (numeroConta>0)
46             {
47                 nome = entrada.next(); // lê nome
48                 sobrenome = entrada.next(); // lê sobrenome
49                 saldo = entrada.nextDouble(); // lê saldo
50                 registro = new RegistroContaSerializavel ( numeroConta,
51                     nome, sobrenome, saldo );
52                 saída.writeObject( registro ); // imprime registro
53             }
54         }
55     } while (numeroConta != 0);
56 }

```

Escreve objeto registro no arquivo

Cria Regi stroConta baseado na entrada do usuário



## Resumo

Cri aArqui vo

Sequenci al .j ava

(3 de 4)

```

59         catch ( IOException ioException )
60         {
61             System.err.println( "Erro de escri ta no arqui vo." );
62             return;
63         }
64         catch ( NoSuchElementException elementException )
65         {
66             System.err.println( "Entrada i nval ida. Tente novamente." );
67             entrada.nextLine(); // descarta valores já digi tados
68         }
69     } while ( numeroConta>0); // fim do do while
70 } // fim do método adici onaRegistros

```

```

88 // fecha arqui vo e termi na apl icação
89 public void fecharArqui vo()
90 {
91     try // fecha o arqui vo
92     {
93         if ( sai da != null )
94             sai da.close();
95     }
96     catch ( IOException ioException )
97     {
98         System.err.println( "Erro no fechamento do arqui vo." );
99         System.exit( 1 );
100     } // fim do catch
101 } // fim do método fecharArqui vo
102 } // fim da classe Cri aArqui voSerial

```



# Lendo e deserializando um arquivo de acesso seqüencial

- Para abrir um arquivo para ler os objetos, crie um `FileInputStream` ligado por um `ObjectInputStream`
  - `FileInputStream` oferece métodos para ler baseado em byte um arquivo
  - `ObjectInputStream` usa `FileInputStream` para ler objetos do arquivo
  - Método `readObject` de `ObjectInputStream` lê em `Object`, que é então downcast para o tipo apropriado
    - `EOFException` ocorre na tentativa de se ler depois do final do arquivo
    - `ClassNotFoundException` ocorre se a classe para o objeto sendo lido não pode ser localizada
  - Método `close` de `ObjectInputStream` fecha os dois objetos





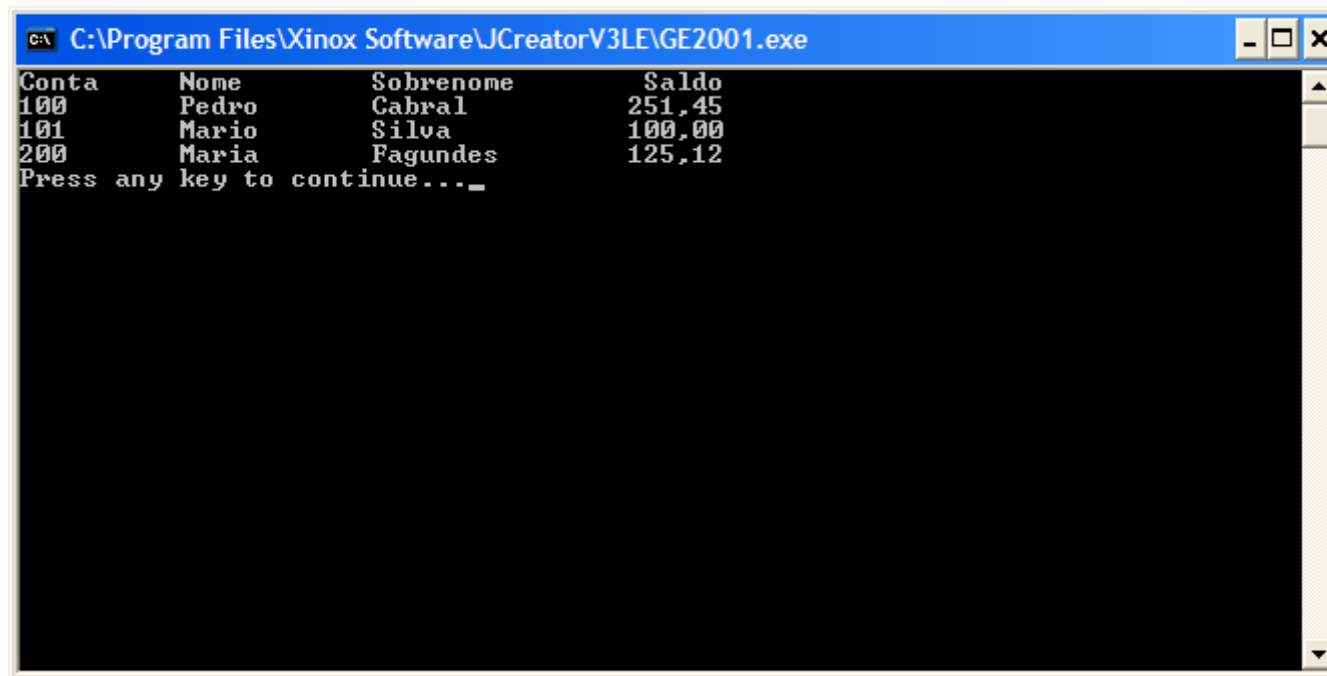
## Resumo

TesteLei tura  
Arqui voSequenci al  
.j ava

```

1 // Arqui vo: TesteLei turaArqui voSequenci al .j ava
2 // Este programa testa a classe Lei turaArqui voSequenci al .
3
4 public class TesteLei turaArqui voSequenci al
5 {
6     public static void main( String args[] )
7     {
8         Lei turaArqui voSequenci al apl i cacao = new Lei turaArqui voSequenci al ();
9
10        apl i cacao. abri rArqui vo();
11        apl i cacao. l erRegi stros();
12        apl i cacao. fecharArqui vo();
13    } // fim do mai n
14 } // fim da classe TesteLei turaArqui voSequenci al

```



Conta	Nome	Sobrenome	Saldo
100	Pedro	Cabral	251,45
101	Mario	Silva	100,00
200	Maria	Fagundes	125,12

Press any key to continue...\_



## Resumo

```

1 // Arquivo: LeituraArquivoSequencial.java
2 // Este programa lê um arquivo de objetos sequencialmente
3 // e apresenta 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.ch15.RegistroContaSerravallo;

```

```

10

```

```

11 public class LeituraArquivoSequencial

```

```

12 {

```

```

13     private ObjectInputStream entrada;

```

```

14

```

```

15     // abre o arquivo

```

```

16     public void abrirArquivo()

```

```

17     {

```

```

18         try // abre arquivo

```

```

19         {

```

```

20             entrada = new ObjectInputStream(

```

```

21                 new FileInputStream( "clientes.ser" ) );

```

```

22         }

```

```

23         catch ( IOException ioException )

```

```

24         {

```

```

25             System.err.println( "Erro de abertura de arquivo." );

```

```

26         }

```

```

27     } // fim do método abrirArquivo

```

```

28

```

Classe usada para criar um stream de entrada

Classe usada para ler objetos do fluxo de entrada baseado em byte

Arquivo  
i al .java

(1 de 3)

Abre arquivo clientes.ser para  
leitura



## Resumo

LeituraArquivo  
Sequencial.java

```
29 // Leitura de registros do arquivo
30 public void lerRegistros()
31 {
32     RegistroContaSerializavel registro;
33     System.out.printf( "%-10s%-12s%-12s%10s\n", "Conta",
34         "Nome", "Sobrenome", "Saldo" );
35
36     try // entra valores lidos do arquivo
37     {
38         while ( true )
39         {
40             registro = ( RegistroContaSerializavel ) entrada.readObject();
41
42             // apresenta conteúdo do registro
43             System.out.printf( "%-10d%-12s%-12s%10.2f\n",
44                 registro.getNumeroConta(), registro.getNome(),
45                 registro.getSobrenome(), registro.getSaldo() );
46         } // fim do while
47     } // fim do try
48     catch ( EOFException endOfFileException )
49     {
50         return; // fim do arquivo foi encontrado
51     } // fim do catch
```

Lê registro do arquivo

(2 de 3)

Imprime informações do registro  
na tela



## Resumo

LeituraArquivo  
Sequencial.java

(3 de 3)

```

52     catch ( ClassNotFoundException classNotFoundException )
53     {
54         System.err.println( "Nao foi possivel criar o objeto." );
55     } // fim do catch
56     catch ( IOException ioException )
57     {
58         System.err.println( "Erro durante a leitura do arquivo." );
59     } // fim do catch
60 } // fim do método LerRegistros
61
62 // fecha arquivo
63 public void fecharArquivo()
64 {
65     try // fecha arquivo
66     {
67         if ( entrada != null )
68             entrada.close();
69     } // fim do try
70     catch ( IOException ioException )
71     {
72         System.err.println( "Erro no fechamento do arquivo." );
73         System.exit( 1 );
74     } // fim do catch
75 } // fim do método fechar arquivo
76 } // fim da classe LeituraArquivoSequencial

```

Fecha arquivo



## Abrindo arquivos com JFileChooser

- **JFileChooser** – Classe usada para apresentar um diálogo que permite ao usuário facilmente selecionar arquivos
  - Método `setSelectionMode` especifica o que o usuário pode selecionar de **JFileChooser**
    - Constante `FILES_AND_DIRECTORIES` indica arquivos e diretórios
    - Constante `FILES_ONLY` indica arquivos apenas
    - Constante `DIRECTORIES_ONLY` indica diretório apenas
  - Método `showOpenDialog` apresenta **JFileChooser** com botões **Open** e **Cancel**
    - Constante `CANCEL_OPTION` permite verificar se o usuário clicou no botão **Cancel**
  - Método `getSelectedFile` obtém o arquivo ou diretório selecionado pelo usuário



## Resumo

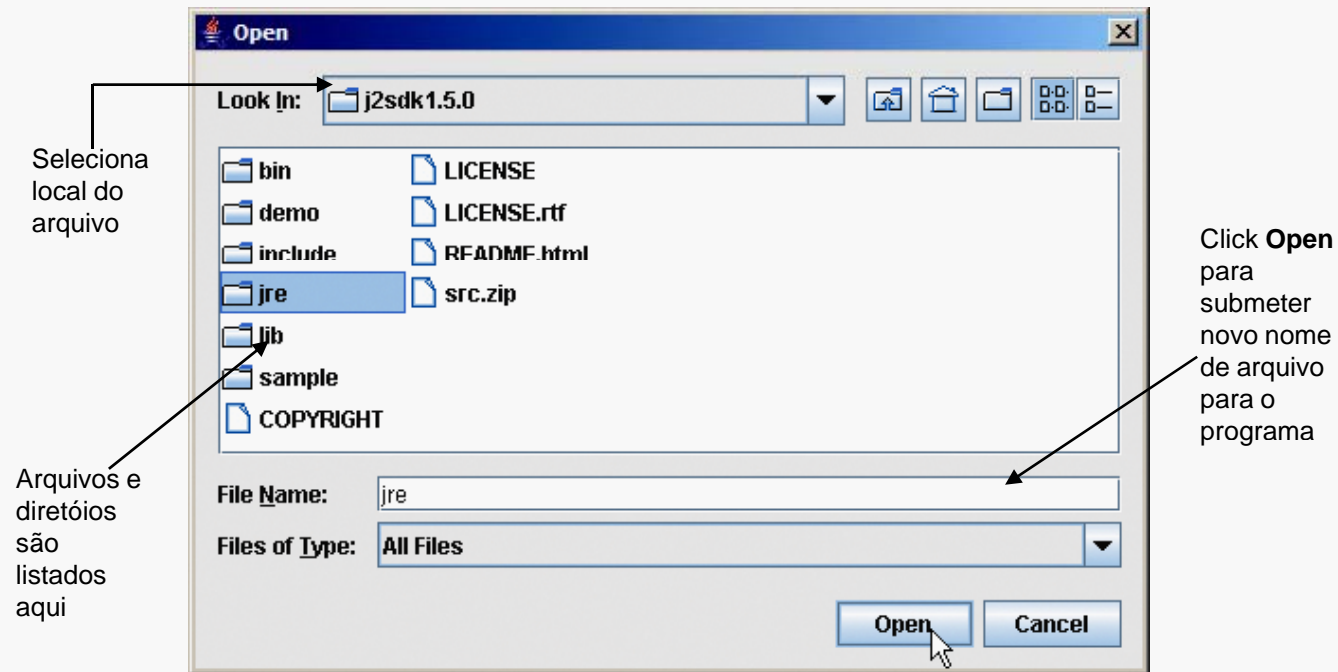
FileDemonstration  
Test.java

(1 de 2)

```

1 // Arquivo: FileDemonstrationTest.java
2 // Testando a classe FileDemonstration.
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 do main
12 } // fim da classe FileDemonstrationTest

```

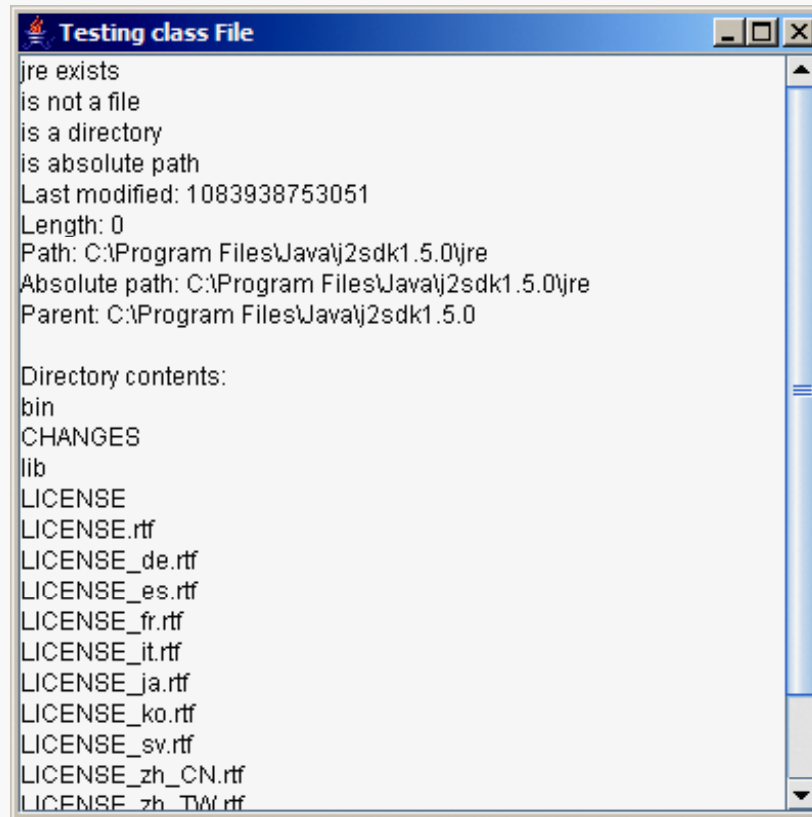


## Resumo

### FileDemonstration

### Test.java

(2 de 2)



## Resumo

FileDemonstration

ava

(1 de 4)

```

1 // Arquivo: 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; // usado para imprimir texto
17     private JScrollPane scrollPane; // usado para oferecer scrolling na saída
18
19     // define 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 ao JFrame
30

```

Classe para apresentar diálogo  
JFileChooser





## Resumo

### FileDemonstration

Java

```

31     setSize( 400, 400 ); // define tamanho do JFrame
32     setVisible( true ); // apresenta JFrame
33
34     analyzePath(); // cria e analisa objeto File
35 } // fim do construtor FileDemonstration
36
37 // permite ao usuário especificar o nome do arquivo
38 private File getFile()
39 {
40     // apresenta diálogo JFileChooser para o usuário selecionar arquivo
41     JFileChooser fileChooser = new JFileChooser();
42     fileChooser.setFileSelectionMode(
43         JFileChooser.FILES_AND_DIRECTORIES );
44
45     int result = fileChooser.showOpenDialog( this );
46
47     // se usuário clicou no botão Cancel, termina programa
48     if ( result == JFileChooser.CANCEL_OPTION )
49         System.exit( 1 );
50
51     File fileName = fileChooser.getSelectedFile();
52
53     // apresenta 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 do if
60

```

Cria JFileChooser

Permite ao usuário selecionar  
arquivos ou diretórios

Apresenta diálogo

Usuário clicou em Cancel

Recupera arquivo ou diretório  
selecionado pelo usuário





## Resumo

### FileDemonstration

.java

(4 de 4)

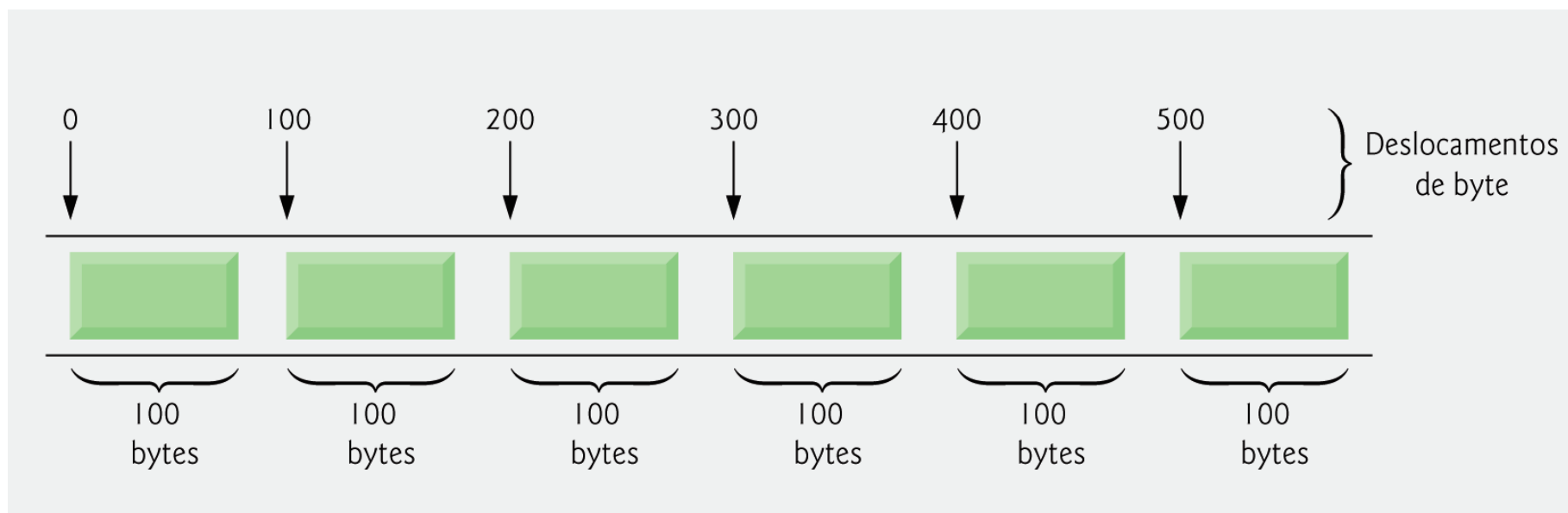
```
85         if ( name.isDirectory() ) // imprime listagem do diretório
86         {
87             String directory[] = name.list();
88             outputArea.append( "\n\nDirectory contents: \n" );
89
90             for ( String directoryName : directory )
91                 outputArea.append( directoryName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



# 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.





**Visualização do Java de um arquivo de acesso aleatório.**



# Criando um arquivo de acesso aleatório

- **Classe `RandomAccessFile` é:**
  - 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`



# 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

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

AccountRecord.java

(2 de 3)

```
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
```

Lê caracteres, agregando-os em uma string



## Resumo

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

AccountRecord.java

```
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
```

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



# Resumo

CreateRandomFile

.java

(2 de 2)

```
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

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.



# Gravando dados aleatoriamente em um arquivo de acesso aleatório

- O método `RandomAccessFile` e 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



# Resumo

WriteRandomFile

.java

(2 de 4)

```
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

(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



## Resumo

Calcula a localização do novo registro

Gera a saída do novo registro para o arquivo

WriteRandomFile

.java

(4 de 4)

```

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 iException )
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

```



# 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

```



# 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 extremamente 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

---

**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.delitel.jhttp6.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 ) // close
51     {
52         return; // fim do arquivo foi alcançado
53     } // fim do catch

```

Lê até o registro não em branco ser encontrado

A exceção ocorre quando o final do arquivo é alcançado



# Resumo

ReadRandomFile

.java

(3 de 3)

```
54     catch ( IOException iException )
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 iException )
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

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



# 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

## MenuOpti on. Java

```
1 // Fig. 14.33: MenuOpti on.java
2 // Define um tipo enum para as opções do programa de consul ta de crédi to.
3
4 public enum MenuOpti on
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    MenuOpti on( int val ueOpti on )
16    {
17        val ue = val ueOpti on;
18    } // fim do construtor do enum de MenuOpti ons
19
20    public int getVal ue()
21    {
22        return val ue;
23    } // fim do método getVal ue
24 } // fim do enum de MenuOpti on
```



# 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 IOException
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

```

30 // obtém um registro do arquivo
31 public RandomAccessAccountRecord getRecord( int accountNumber )
32     throws IllegalArgumentException, NumberFormatException
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

FileEditor.java

(2 de 5)

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(), record
62     );
63     record.write( file ); // grava registro atualizado no arquivo
64 } // fim do método updateRecord
65
66 // adiciona o registro 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



```

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%-15s10s\n", "Account",
108         "First Name", "Last Name", "Balance" );
109

```

Recupera o registro com base no número da conta

FileEditor.java

(4 de 5)

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
```



# Resumo

Transação

Processor.java

(2 de 7)

```
27     catch ( IOException iException )
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 iException )
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

(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\n%s\n%s\n",
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



```

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

```

Transação

Recupera um número de conta do  
registro a ser excluído

ava

(5 de 7)

Exclui um registro





# Resumo

Transação

Processor.java

(6 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

(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
```



# Classes java. i o 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 RandomAccessFile e DataInputStream; DataOutput é implementada por RandomAccessFile e DataOutputStream.
- **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 Strings.

