Arquivos

Sumário

- Introdução;
- A Classe **`File`**;
- Criando um arquivo de texto de acesso seqüencial;
- Exemplos de Interface;
- Lendo dados a partir de um arquivo de texto de acesso aleatório;
- Bibliografia;

Introdução

- O armazenamento de dados em variáveis é temporário;
- Computadores utilizam arquivos para armazenamento do longo prazo de grandes volumes de dados;
- Chamados os dados mantidos em arquivos de dados persistentes porque eles existem além da duração da execução do programa;

Introdução

- Arquivo é um grupo de registros relacionados;
- Um grupo de arquivos relacionados costuma ser chamado de banco de dados;
- Uma coleção de programas projetados para criar e gerenciar banco de dados é chamada sistema de gerenciamento de banco de dados – SGBD;

Introdução

- O Java vê cada arquivo como um fluxo seqüencial de bytes;
- Programas Java realizam o processamento de arquivos utilizando as classes no pacote `java.io`;
- Esse pacote inclui definições para classes de fluxo, como:
  - **`FileInputStream;`**
  - **`FileOutputStream;`**
  - **`FileReader;`**
  - **`FileWriter;`**

A Classe File

- Primeiro exemplo:

FileDemonstration.java | FileDemonstrationTest.java

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

A Classe File

- Continuação do exemplo:

```
24
25          if (name.isDirectory()) // listagem de diretório de saída
26          {
27             String directory[] = name.list();
28             System.out.println( "\n\nDirectory contents:\n" );
29
30             for ( String directoryName : directory )
31                System.out.printf( "%s\n", directoryName );
32          } // fim do else
33       } // fim do if externo
34       else // não for arquivo ou diretório, gera saída da mensagem de erro
35       {
36          System.out.printf( "%s %s", path, "does not exist." );
37       } // fim do else
38    } // fim do método analyzePath
39 } // fim da classe FileDemonstration
40
  1:1   INS
```

A Classe File

- Programa de Teste:

FileDemonstration.java | FileDemonstrationTest.java

```
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
16
```

A Classe File

- Saída no console do programa anterior:

```
Saída - FileDemonstration (run)
compile:
run:
C:\Documents and Settings\Leonardo\Desktop\teste.txt
Enter file or directory name here: teste.txt exists
is a file
is not a directory
is absolute path
Last modified: 1194950662203
Length: 10
Path: C:\Documents and Settings\Leonardo\Desktop\teste.txt
Absolute path: C:\Documents and Settings\Leonardo\Desktop\teste.txt
Parent: C:\Documents and Settings\Leonardo\Desktop
EXECUTADO COM SUCESSO (tempo total: 1 minuto 48 segundos)
```

Criando um arquivo de texto de acesso seqüencial

- Vejamos a criação de um arquivo que poderia ser utilizado em um sistema de contas a receber;
- Esse programa pode ajudar a monitorar os valores devidos pelos seus clientes em uma empresa;
- Para cada cliente, o programa obtém do usuário:
  - Número da conta;
  - Nome do cliente;
  - Saldo do cliente;

Criando um arquivo de texto de acesso seqüencial

- A classe AccountRecord

AccountRecord.java *

```
1  public class AccountRecord
2  {
3     private int account;
4     private String firstName;
5     private String lastName;
6     private double balance;
7
8     // construtor sem argumentos chama outro construtor com valores padrão
9     public AccountRecord()
10    {
11       this( 0, "", "", 0.0 ); // chama o construtor com quatro argumentos
12    } // fim do construtor de AccountRecord sem argumentos
13
14    // inicializa um registro
15    public AccountRecord( int acct, String first, String last, double bal )
16    {
17       setAccount( acct );
18       setFirstName( first );
19       setLastName( last );
20       setBalance( bal );
21    } // fim do construtor de AccountRecord com quatro argumentos
22
```

Criando um arquivo de texto de acesso seqüencial

- Continuação da classe AccountRecord;

AccountRecord.java *

```
23      // configura o número de conta
24      public void setAccount( int acct )
25      {     account = acct;   }
26      public int getAccount()
27      {       return account;    }
28      public void setFirstName( String first )
29      {       firstName = first;   }
30      public String getFirstName()
31      {       return firstName;    }
32      public void setLastName( String last )
33      {       lastName = last;   }
34      public String getLastName()
35      {       return lastName;    }
36      public void setBalance( double bal )
37      {       balance = bal;   }
38      public double getBalance()
39      {       return balance;    }
40    }
41
```

Criando um arquivo de texto de acesso seqüencial

- Vejamos a classe de criação do Arquivo:

AccountRecord.java | CreateTextFile.java * | CreateTextFileTest.java

```
1  import java.io.FileNotFoundException;
2  import java.lang.SecurityException;
3  import java.util.Formatter;
4  import java.util.FormatterClosedException;
5  import java.util.NoSuchElementException;
6  import java.util.Scanner;
7
8  public class CreateTextFile
9  {
10    private Formatter output; // objeto utilizado para gerar saída de texto no arquivo
11
12    // permite ao usuário abrir o arquivo
13    public void openFile()
14    {
15       try
16       {
17          output = new Formatter( "clients.txt" );
18       } // fim do try
19       catch ( SecurityException securityException )
20       {
21          System.err.println(
22             "You do not have write access to this file." );
23          System.exit( 1 );
24       } // fim do catch
```

Criando um arquivo de texto de acesso seqüencial

- Continuação:

AccountRecord.java | CreateTextFile.java * | CreateTextFileTest.java

```
25          catch ( FileNotFoundException filesNotFoundException )
26          {
27             System.err.println( "Error creating file." );
28             System.exit( 1 );
29          } // fim do catch
30       } // fim do método openFile
31
32       // adiciona registros ao arquivo
33       public void addRecords()
34       {
35          // objeto a ser gravado no arquivo
36          AccountRecord record = new AccountRecord();
37
38          Scanner input = new Scanner( System.in );
39
40          System.out.printf( "%s\n%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             "? " );
```

Criando um arquivo de texto de acesso seqüencial

- Continuação:

AccountRecord.java | CreateTextFile.java * | CreateTextFileTest.java

```
50        while (input.hasNext()) // faz um loop até o indicador de fim de arquivo
51        {
52           try // gera saída dos valores para o arquivo
53           {
54              // recupera os dados para saída
55              record.setAccount( input.nextInt() ); // lê o número de conta
56              record.setFirstName( input.next() ); // lê o primeiro nome
57              record.setLastName( input.next() ); // lê o sobrenome
58              record.setBalance( input.nextDouble() ); // lê o saldo
59
60              if ( record.getAccount() > 0 )
61              {
62                 // grava um novo registro
63                 output.format( "%d %s %s %.2f\n", record.getAccount(),
64                    record.getFirstName(), record.getLastName(),
65                    record.getBalance() );
66              } // fim do if
67              else
68              {
69                 System.out.println(
70                    "Account number must be greater than 0." );
71              } // fim do else
72           } // fim do try
```

Criando um arquivo de texto de acesso seqüencial

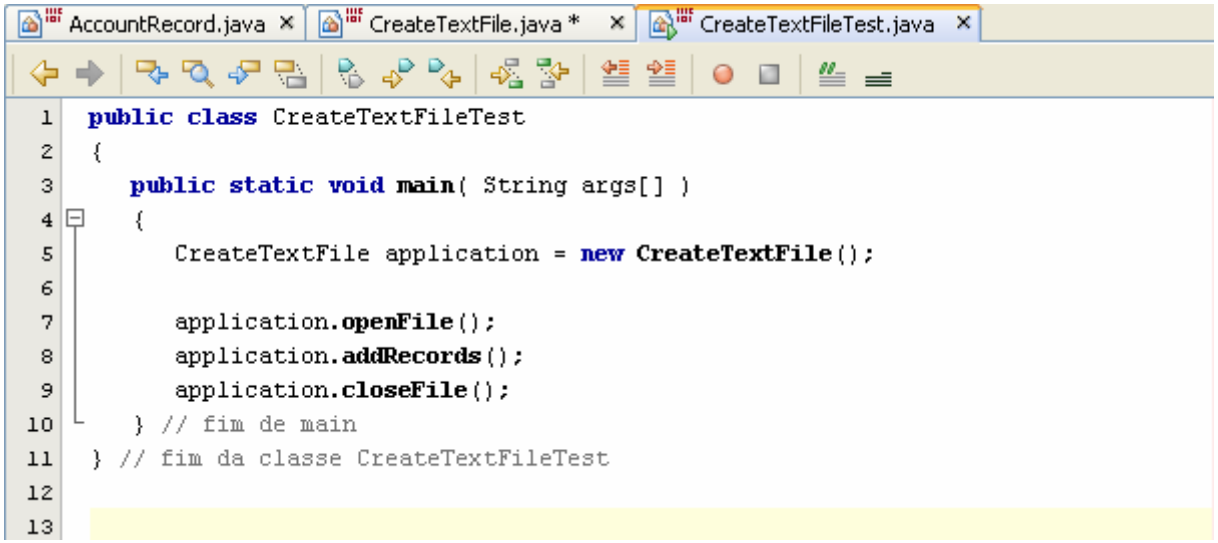
- Continuação:

AccountRecord.java | CreateTextFile.java * | CreateTextFileTest.java

```
73          catch ( FormatterClosedException formatterClosedException )
74          {
75             System.err.println( "Error writing to file." );
76             return;
77          } // fim do catch
78          catch ( NoSuchElementException elementException )
79          {
80             System.err.println( "Invalid input. Please try again." );
81             input.nextLine(); // descarta entrada para o usuário tentar de novo
82          } // fim do catch
83
84          System.out.printf( "%s %s\n%s", "Enter account number (>0),",
85             "first name, last name and balance.", "? " );
86       } // fim do while
87    } // fim do método addRecords
88
89    // fecha o arquivo
90    public void closeFile()
91    {
92       if ( output != null )
93          output.close();
94    } // fim do método closeFile
95 } // fim da classe CreateTextFile
```

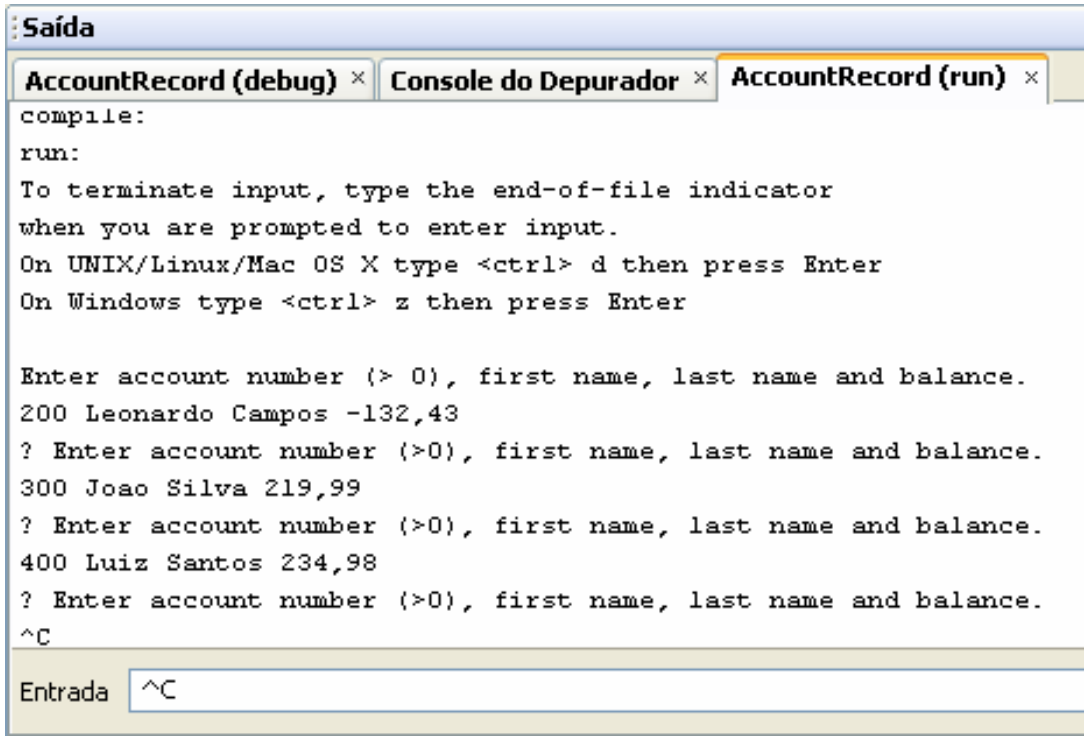
Criando um arquivo de texto de acesso seqüencial

- A classe CreateTextFileTest:



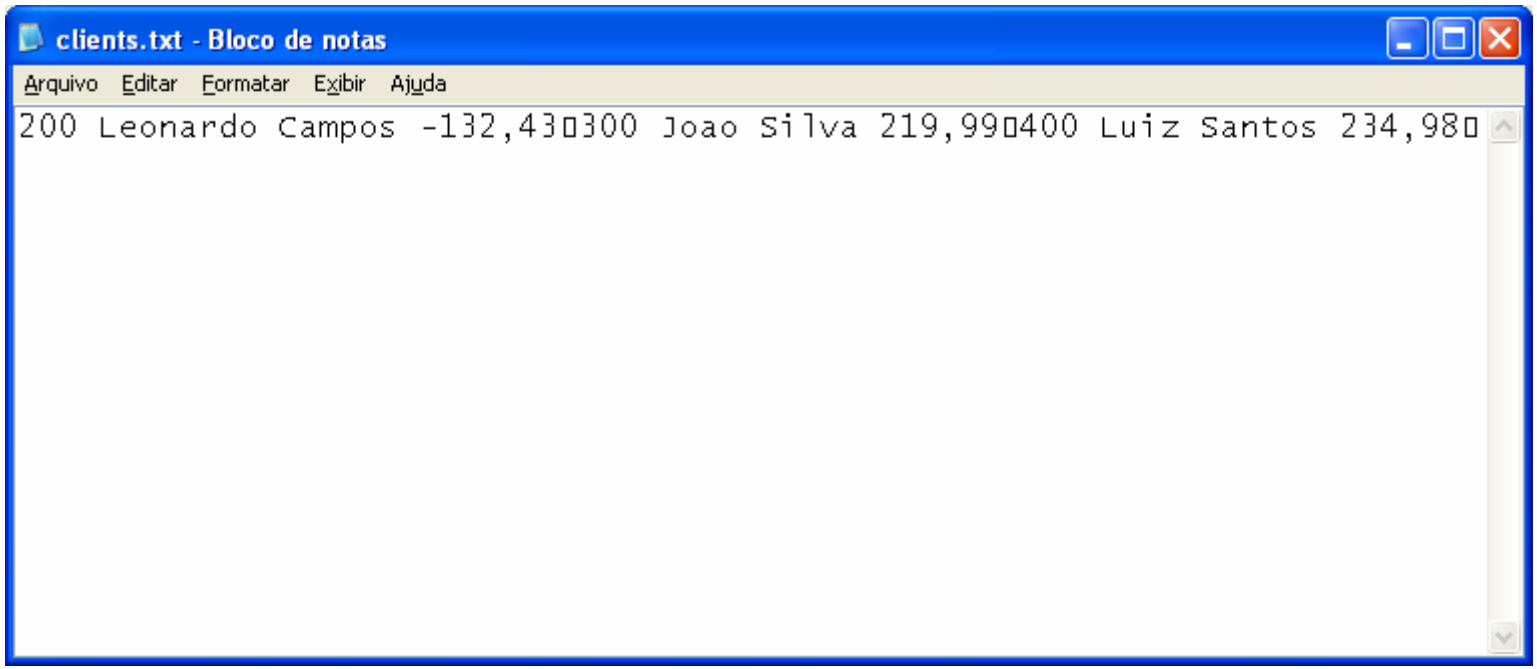
Criando um arquivo de texto de acesso seqüencial

- Saída no Console para o programa anterior:



Criando um arquivo de texto de acesso seqüencial

- Saída armazenada no arquivo:



Variáveis, Métodos e Classes `final`

- Variáveis declaradas como **`final`** indicam que elas não podem ser modificadas depois de declaradas e que devem ser inicializadas quando são declaradas;

- Métodos declarados como **`final`** em uma superclasse não pode ser sobrescrito em uma subclasse;
  - Métodos **`private`** e **`static`** são implicitamente **`final`**, pois não podem ser sobrescritos;

- Classes declaradas como **`final`** não podem ser superclasse:
  - Todos os métodos em uma classe **`final`** são implicitamente **`final`**

Criando e Utilizando Interfaces

- Uma interface em Java descreve um conjunto de métodos que podem ser chamados em um objeto, para instruir o objeto a realizar alguma tarefa ou retornar algumas informações;

- Uma declaração de interface inicia-se com a palavra-chave interface e contém somente constantes e métodos abstrat:

```
public interface Pagamento
{       /* corpo */      }
```

Criando e Utilizando Interfaces

- Diferentemente das classes, todos os membros de interface devem ser **`public`** e as interfaces não podem especificar nenhum detalhe de implementação;
- Portanto, todos os métodos declarados em uma interface são implicitamente métodos **`public`** abstract e todos os campos são implicitamente **`public`**, **`static`** e **`final`**;
- Para utilizar uma classe concreta deve especificar que implementa a interface e deve declarar nesta cada método com a assinatura especificada na interface;

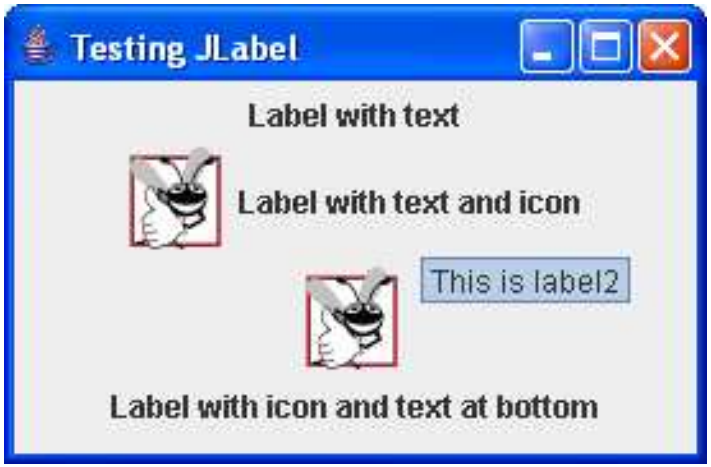
Criando e Utilizando Interfaces

- Uma classe que implementa todos os métodos da interface é uma classe abstrata e deve ser declarada como abstract;

- Vejamos alguns exemplos:

Exemplo 1

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo01/



Exemplo 1

- Considerações:
  - Um tipo de classe ImageIcon (pacote javax.swing), que suporta vários formatos de imagem, inclusive Graphics Interchange Format (GIF), Portable Network Graphics (PNG) e Joint Photographic Expert Group (JPEG);
  - **`FlowLayout`** é um tipo de gerenciador de layout que coloca os componentes GUIs em contêiner da esquerda para a direita de cima para baixo;
  - O método getResource retorna a localização da imagem com um URL (Uniform Resource Locator)

Exemplo 1

- Considerações:
  - O método `setDefaultCloseOperation` de **`LabelFrame`** (herdado da classe **`JFrame`**) com a constante `JFrame.EXIT_ON_CLOSE` como o argumento indica que o programa deve terminar quando a janela for fechada pelo usuário;
  - O método `setSize` de **`LabelFrame`** especifica a largura e a altura da janela;
  - Por fim, o método `setVisible` de **`LabelFrame`** com o argumento **`true`** exibe a janela na tela;

Exemplo 2

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo02/

Exemplo 2

- Considerações:
  - As interfaces gráficas com o usuário (Graphical User Interface - GUI) são baseadas em eventos. O código que realiza uma tarefa em resposta a um evento é chamado de handler de evento;
  - O processo de responder a eventos é conhecido como tratamento de evento e consiste em:
    - 1) Criar uma classe que represente o *handler* de evento;
    - 2) Implementar uma interface apropriada, conhecida como interface *listener* de evento, na classe do Passo 1;
    - 3) Indicar que um objeto da classe dos Passos 1 e 2 deve ser notificado quando o evento ocorrer;

Exemplo 2

- Considerações:
  - As classes aninhadas (classe dentro de outra classe) são freqüentemente utilizadas para tratamento de eventos;
  - Antes que um objeto de uma classe interna possa ser criado, deve ser primeiro um objeto da classe de primeiro nível;
  - Um objeto de classe interna tem implicitamente uma referência a um objeto de sua classe de primeiro nível e tem permissão para acessar todas as variáveis de instâncias e métodos da classe externa;

Exemplo 2

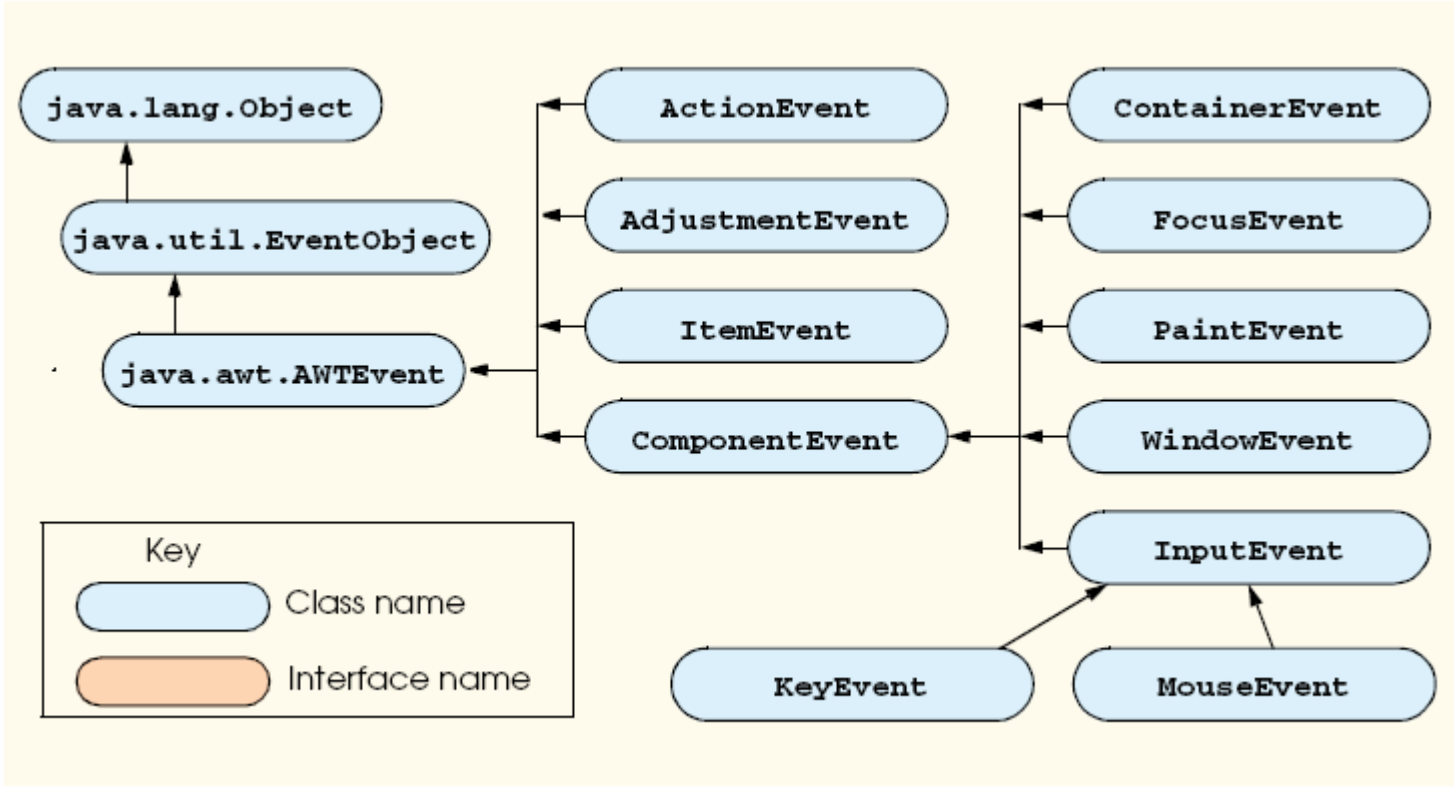
- Considerações:
  - Cada evento é representado por uma classe e pode ser processado apenas pelo tipo de *handler* de evento apropriado;
  - Quando o usuário pressiona *Enter* em um **`JTextField`** ou em um **`JPasswordField`**, o componente GUI gera um **`ActionEvent`** (pacote **`java.awt.event`**);
  - Um evento assim é processado por um objeto que implementa a interface **`ActionListener`**. O método **`actionPerformed`** especifica as tarefas a serem realizadas quando ocorrer um **`ActionEvent`**;

Exemplo 2

- Considerações:
  - O método **addActionListener** é usado para registrar o *handler* de evento para cada componente;
  - Esse método recebe como seu argumento um objeto **ActionListener**, que pode ser um objeto de qualquer classe que implemente **ActionListener**;
  - O método ActionEvent getSource retorna uma referência à origem do evento:
    - A pergunta é: textField é a origem do evento?

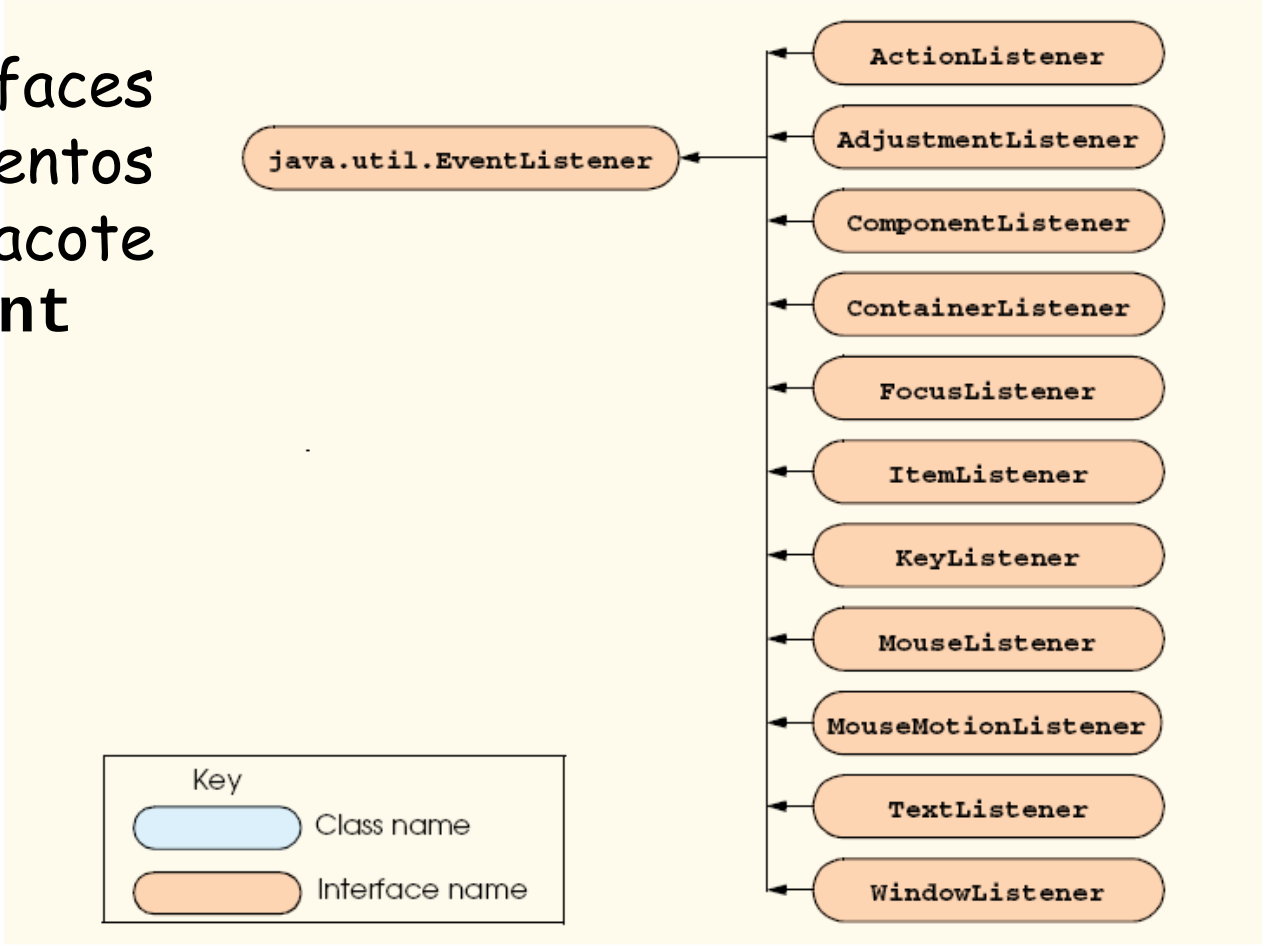
Exemplo 2

- Algumas classes de evento do pacote `java.awt.event`



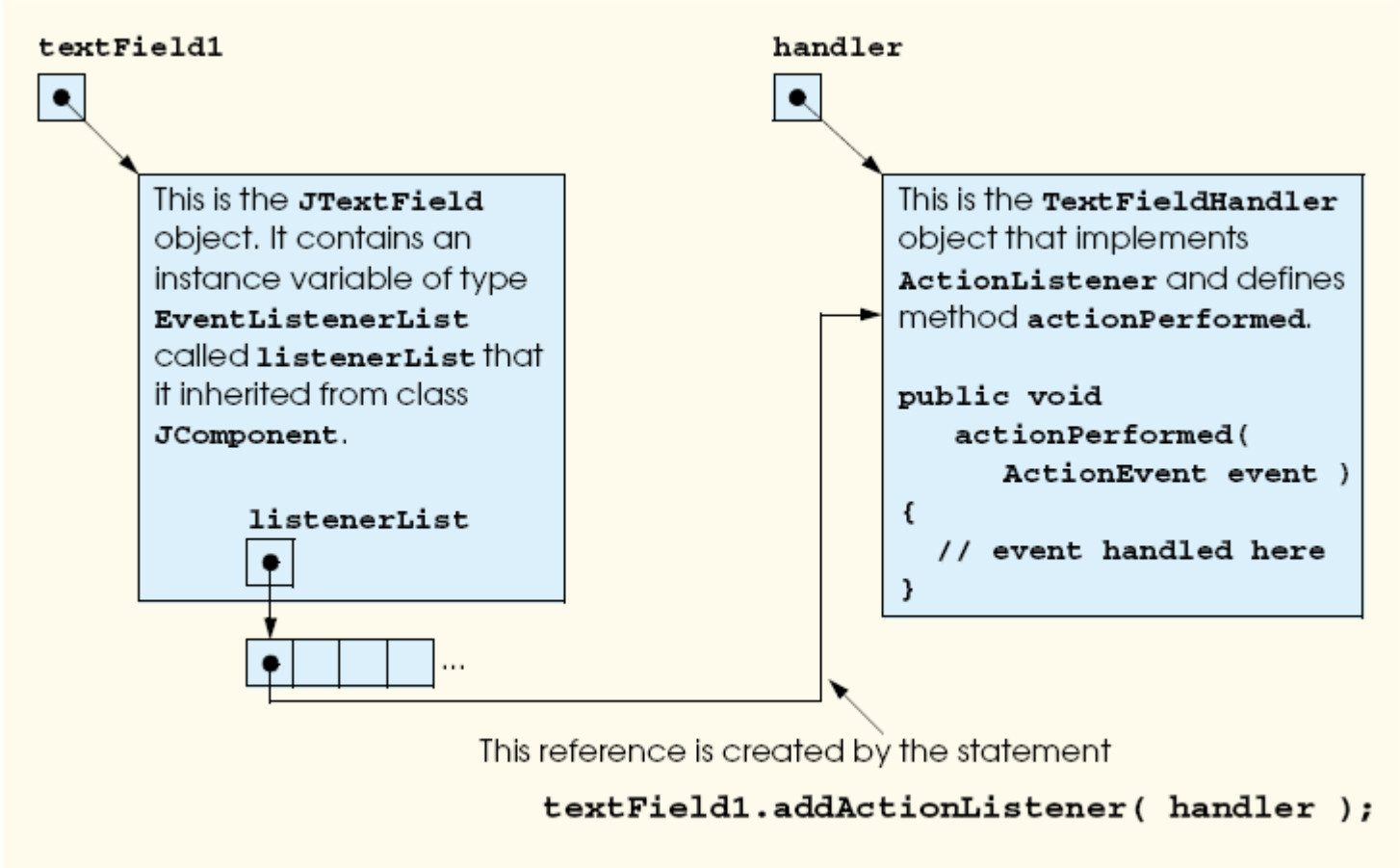
Exemplo 2

- Algumas interfaces ouvintes de eventos comuns do pacote **`java.awt.event`**



Exemplo 2

- Como o tratamento de evento funciona:



Exemplo 3

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo03/

Exemplo 3

- Considerações:
  - As interfaces `MouseListener` e `MouseMotionListener` tratam eventos do mouse; eles aceitam objetos `MouseEvent` como argumento;
  - Um objeto `MouseEvent` contém as informações sobre o evento do mouse que ocorreu, incluindo as coordenadas x e y da localização onde ele ocorreu;
  - Além disso, os métodos e constantes da classe `InputEvent` (a superclasse de `MouseEvent`) permitem que um aplicativo determine em que botão do mouse o usuário clicou;

Exemplo 3

- Considerações:
  - A classe `JPanel` (pacote **`javax.swing`**) fornece uma área em que podemos desenhar;
  - O layout padrão de conteúdo de um `JFrame` é o `BorderLayout`. Esse gerenciador de layout organiza componentes em cinco regiões: `NORTH`, `SOUTH`, `EAST`, `WEST` e `CENTER`;
  - O `JLabel` chamado de **`statusBar`** exibe uma string indicando a ação e as coordenadas x e y que ocorre o evento;

Exercício

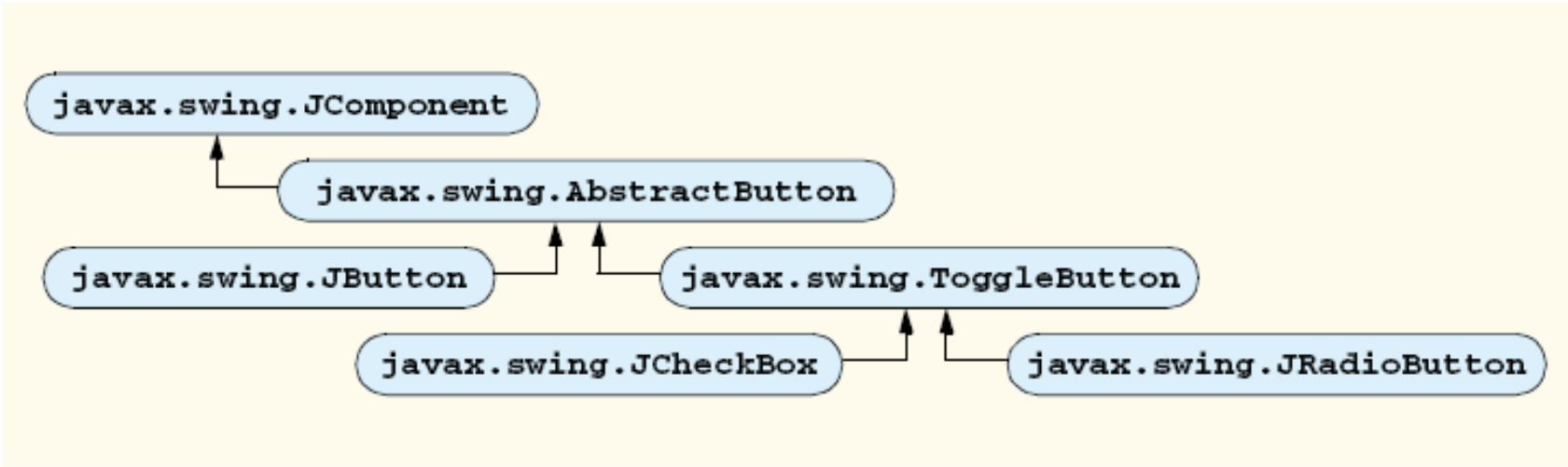
- Faça uma modificação no código do exemplo anterior para pintar a tela de verde quando o mouse estiver, horizontalmente, entre as coordenadas 0 e 150 e de amarelo a partir da coordenada 150:

Exemplo 4

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo04/

Exemplo 4

- Considerações:
  - Um botão é um componente em que o usuário clica para acionar uma ação específica;
  - Vejamos a hierarquia de botões no Java:



Exemplo 4

- Considerações:
  - Um **JButton** pode exibir um **Icon** bem como um **Icon rellover** - um **Icon** exibido quando o usuário posiciona o mouse sobre o botão;
  - **JOptionPane showMessageDialog** utiliza **ButtonFrame.this** em vez de **null** como primeiro argumento. Dessa forma, ele representa o componente GUI-pai do diálogo da mensagem e permite que o diálogo seja centralizado.
  - **ButtonFrame.this** representa a referência **this** do objeto de classe de primeiro nível **ButtonFrame**;

Exemplo 5

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo05/

Exemplo 5

- Considerações:
  - O **JTextField** utiliza o método **setFont** para configurar a fonte do **JTextField** como um novo objeto de classe **Font** (pacote **java.awt**);
  - A nova Fonte é inicializada com "Serif" (nome de fonte para representar uma fonte genérica como Times), estilo **Font.PLAIN** e corpo 14;
  - Quando o usuário clica em uma **JCheckBox**, um **ItemEvent** ocorre. Esse evento pode ser tratado por um objeto **ItemListener**, que deve implementar o método **itemStateChanged**;

Exemplo 6

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo06/

Exemplo 6

- Considerações:
  - Os botões de opção têm apenas dois estados: selecionado e não selecionado, entretanto, em um grupo de botões opções apenas um pode ser selecionado por vez;
  - O relacionamento lógico entre botões de opção é mantido por um objeto **`ButtonGroup`** (pacote **`javax.swing`**);
  - Quando o usuário clica em um **`JRadioButton`**, **`radioGroup`** desliga o **`JRadioButton`** anteriormente selecionado e o método **`itemStateChanged`** configura o **`JTextField`** como a **`Font`** armazenada no objeto instância **`textField`** da classe de primeiro nível para configurar a fonte;

Exemplo 7

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo07/

Exemplo 7

- Considerações:
  - O **Icon** é selecionado a partir do array icons, determinando o índice de um **JComboBox**, a seleção do primeiro item é removida;
  - Dessa forma, dois **ItemEvents** ocorrem quando um item é selecionado;
  - Por essa razão, o método **ItemEvent getStateChange** retorna **ItemEvent.SELECTED**;

Exemplo 8

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo08/

Exemplo 8

- Considerações:
  - A classe **keyDemoFrame** implementa a interface **KeyListener** e herda da classe **JFrame**;
  - O construtor registra o aplicativo para tratar seus próprios eventos de teclado utilizando o método **addKeyListener**;
  - A interface **KeyListener** serve para tratar eventos de teclado, gerados quando as teclas no teclado são pressionadas e liberadas;

Exemplo 8

- Considerações:
  - Uma classe que implementa **KeyListener** deve fornecer declarações para métodos **keyPressed**, **keyReleased** e **keyTyped**, cada um dos quais recebe um **KeyEvent** como argumento;
    - **keyPressed**: chamado em resposta ao pressionamento de qualquer tecla;
    - **keyTyped**: chamado em resposta ao pressionamente de qualquer tecla de ação (Home, End, Page Up, qualquer tecla de função, etc);
    - **keyReleased**: chamado quando a tecla é liberada depois de qualquer evento keyPressed ou keyTyped;

Exemplo 9

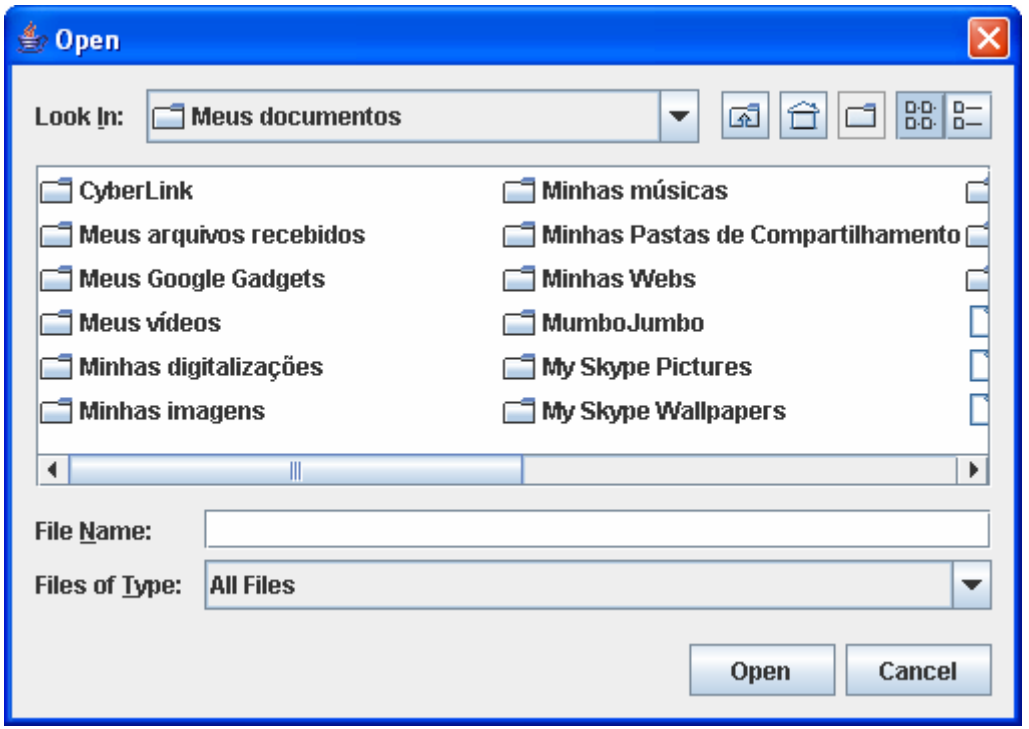
- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo09/

Exemplo 9

- Considerações:
  - A classe **JMenuBar** (subclasse de **JComponent**) contém os métodos necessários para gerenciar uma barra de menus que é um contêiner de menus;
  - A classe **JMenuItem** (subclasse de **javax.swing.AbstractButton**) contém os métodos necessários para gerenciar itens de menu;
  - O aplicativo demonstra também como especificar caracteres especiais chamados de mnimônicos que podem fornecer acesso rápido a um menu ou ao item do menu do teclado;
  - Um ActionListener é criado como classe interna anônima para processar o evento de ação de aboutItem;

Exemplo 10

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo10/



Exemplo 10

- Link: http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/POO_2007_2/Codigos_Java/Exemplo10/

Lendo dados a partir de um arquivo de texto de acesso aleatório

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.lang.IllegalStateException;
4  import java.util.NoSuchElementException;
5  import java.util.Scanner;
6
7  public class ReadTextFile
8  {
9     private Scanner input;
10
11    // permite ao usuário abrir o arquivo
12    public void openFile()
13    {
14       try
15       {
16          input = new Scanner( new File( "clients.txt" ) );
17       } // fim do try
18       catch ( FileNotFoundException fileNotFoundException )
19       {
20          System.err.println( "Error opening file." );
21          System.exit( 1 );
22       } // fim do catch
23    } // fim do método openFile
24
```

Lendo dados a partir de um arquivo de texto de acesso aleatório

```
25     // lê o registro no arquivo
26     public void readRecords()
27     {
28        // objeto a ser gravado na tela
29        AccountRecord record = new AccountRecord();
30
31        System.out.printf( "%-10s%-12s%-12s%10s\n", "Account",
32           "First Name", "Last Name", "Balance" );
33
34        try // lê os registros no arquivo utilizando o objeto Scanner
35        {
36           while (input.hasNext())
37           {
38              record.setAccount( input.nextInt() ); // lê o número de conta
39              record.setFirstName( input.next() ); // lê o primeiro nome
40              record.setLastName( input.next() ); // lê o sobrenome
41              record.setBalance( input.nextDouble() ); // lê o saldo
42
43              // exibe o conteúdo de registro
44              System.out.printf( "%-10d%-12s%-12s%10.2f\n",
45                 record.getAccount(), record.getFirstName(),
46                 record.getLastName(), record.getBalance() );
47           } // fim do while
48        } // fim do try
```

Lendo dados a partir de um arquivo de texto de acesso aleatório

```
46                 record.getLastName(), record.getBalance() );
47              } // fim do while
48           } // fim do try
49           catch ( NoSuchElementException elementException )
50           {
51              System.err.println( "File improperly formed." );
52              input.close();
53              System.exit( 1 );
54           } // fim do catch
55           catch ( IllegalStateException stateException )
56           {
57              System.err.println( "Error reading from file." );
58              System.exit( 1 );
59           } // fim do catch
60        } // fim do método readRecords
61
62        // fecha o arquivo e termina o aplicativo
63        public void closeFile()
64        {
65           if ( input != null )
66              input.close(); // fecha o arquivo
67        } // fim do método closeFile
68     } // fim da classe ReadTextFile
69
```

Lendo dados a partir de um arquivo de texto de acesso aleatório

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

Bibliografia

- Deitel, H. M. & Deitel, P. J. Java*: como programar,* Editora Bookman. 6ª ed. São Paulo: 2005.