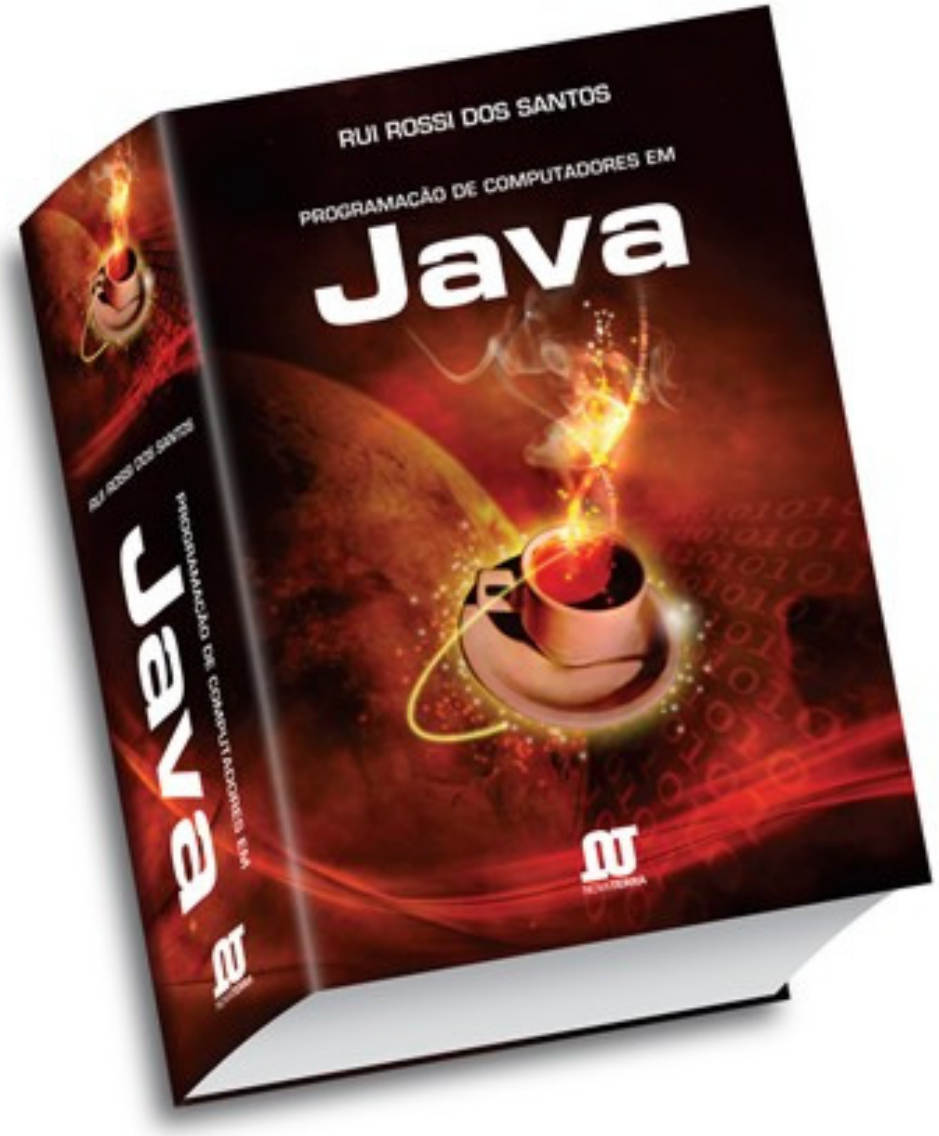


Capítulo 34

Arquivos e Fluxos de Dados

Objetivos do Capítulo

- Analisar os conceitos de fluxo de entrada e de fluxo de saída de dados.
- Apresentar os recursos fundamentais da API do Java destinados ao controle de fluxos de diferentes tipos dados.
- Indicar os recursos que podem ser empregados para a criação e exclusão de diretórios e de arquivos e como consultar e alterar suas propriedades.
- Explorar os recursos necessários para gravar diferentes tipos de dados em arquivos e também para recuperá-los.

Introdução

- **Conceitos:**
  - **Arquivo: forma comum de armazenamento de dados em disco.**
  - **Fluxo: qualquer sequencia de dados.**
    - ✓ **Fluxo de entrada: sequencia de dados lidos de uma origem qualquer.**
    - ✓ **Fluxo de saída: sequencia de dados escritos em qualquer destino.**
- **Origens e destinos mais comuns:**
  - **Arquivos em discos rígidos**
  - **Arquivos em dispositivos removíveis**
  - **Dispositivos de memória**
  - **Conexões de rede**

Diretórios e Arquivos

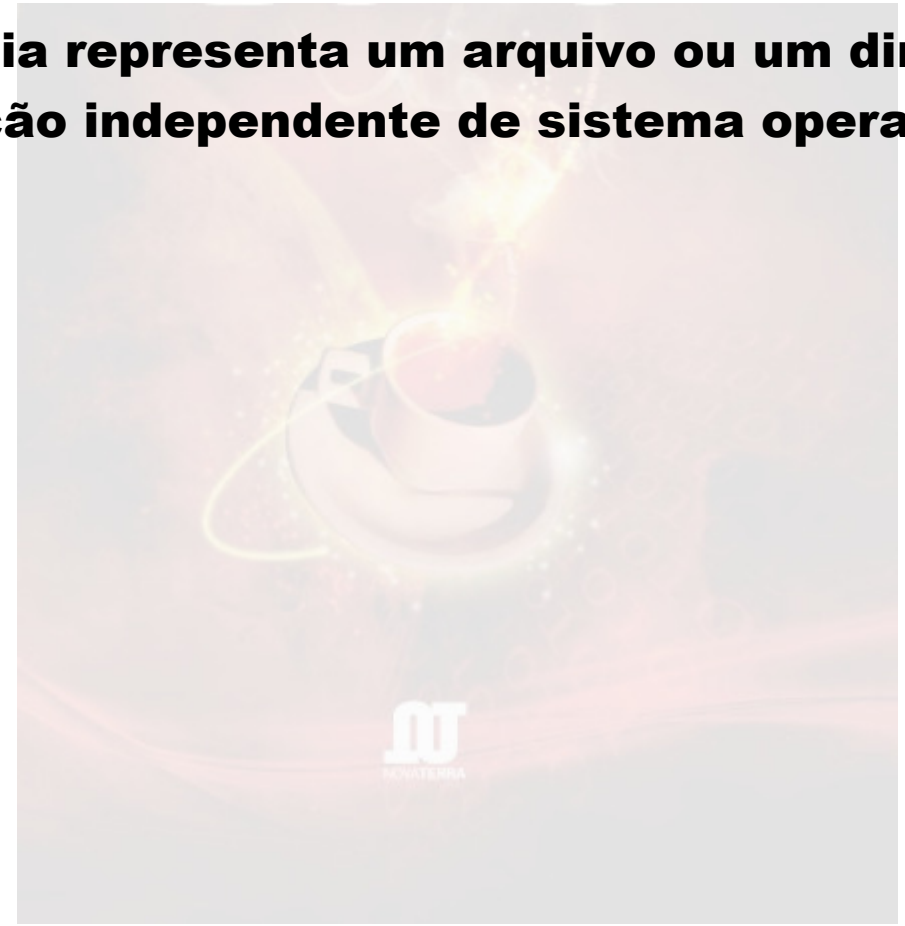
- **Algumas operações comuns:**
  - **Representar arquivos e diretórios**
  - **Criar arquivos e diretórios**
  - **Excluir arquivos e diretórios**
  - **Inspecionar as propriedades de arquivos e diretórios**
  - **Listar o conteúdo de diretórios**
  - **Ler o conteúdo de um arquivo**
  - **Escrever um conteúdo em um arquivo**

Diretórios e Arquivos

- **Sistema de arquivos:**
  - Representados nos sistemas operacionais como árvores hierárquicas.
  - Topo da árvore: o nó raiz
  - Divisões: diretórios e subdiretórios.
- **Linux:**
  - Um único nó na raiz
  - Representado por uma barra: “/”
- **Windows:**
  - Múltiplos nós na raiz: unidades ou volumes
  - Representação: “C:\”, “D:\” etc.

Diretórios e Arquivos

- **Classe java.io.File**
  - **Uma instância representa um arquivo ou um diretório.**
  - **Representação independente de sistema operacional.**



Diretórios e Arquivos

- **Classe java.io.File**
  - **Representação de um diretório:**

```
File fl = new File("/home/rui/PCJ/Cap34");
File fl = new File("C:\\PCJ\\Cap34");
```

  - **Criação de diretórios:**

```
if (!fl.exists( )) fl.mkdir();
if (!fl.exists( )) fl.mkdirs();
```

  - **Representação de um arquivo:**

```
File fl = new File("/home/rui/PCJ/Cap34/Edipo.txt");
File fl = new File("C:\\PCJ\\Cap34\\Edipo.txt");
```

Diretórios e Arquivos

- **Classe java.io.File**
  - **Métodos:**

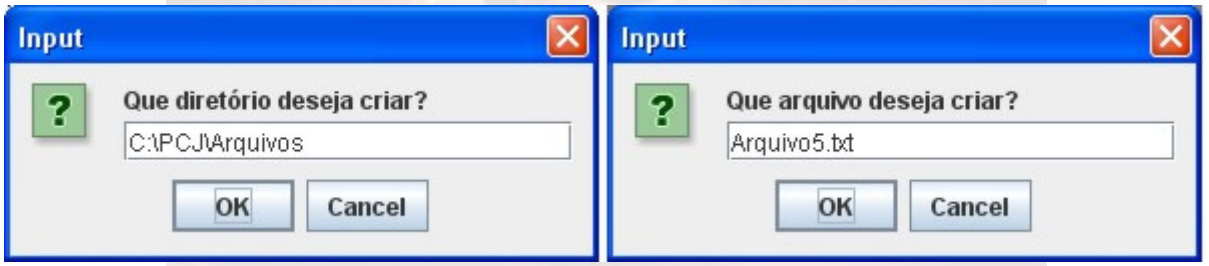
    boolean canExecute( ): testa se o arquivo/diretório pode ser executado.
    boolean canRead( ): testa se o arquivo/diretório pode ser lido.
    boolean canWrite( ): testa se o arquivo/diretório pode ser modificado.
    void createNewFile( ): cria o arquivo (se ele ainda não existir).
    boolean delete( ): exclui o diretório ou arquivo.
    boolean exists( ): testa se o diretório ou arquivo existe.
    String getPath( ): recupera o caminho do diretório ou arquivo.
    boolean isDirectory( ): testa se é um diretório.
    boolean isFile( ): testa se é um arquivo.
    boolean isHidden( ): testa se é um diretório/arquivo oculto.
    long lastModified( ): retorna a data da última modificação.
    String[] list( ): retorna o conteúdo do diretório representado.
  - **Atributos:**

    File.separator: representa o separador padrão do S.O.

Diretórios e Arquivos

- **Exemplo: uso da classe java.io.File para a criação de diretórios e de arquivos.**

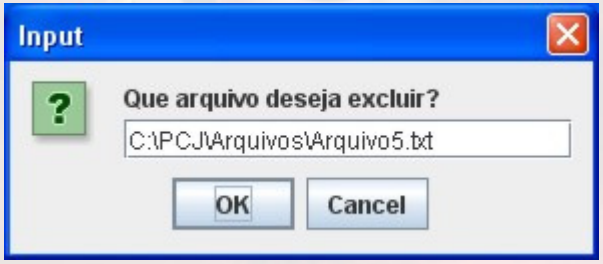
- **Código 34.1 – CriacaoArquivo.java**



Diretórios e Arquivos

- **Exemplo: exclusão de arquivos e de diretórios.**

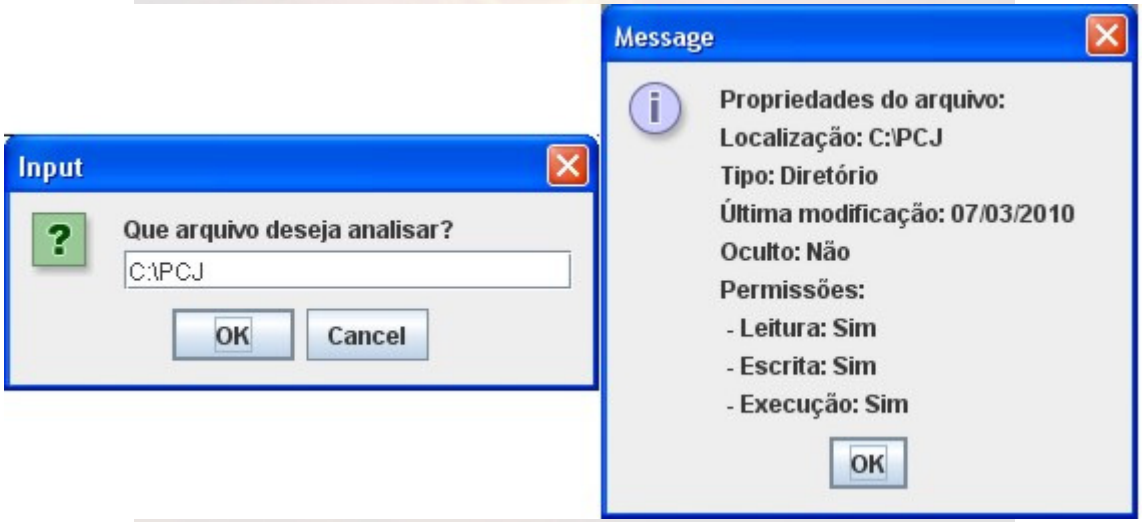
- **Código 34.2 – ExclusaoArquivo.java**



Diretórios e Arquivos

- **Exemplo: inspeção de propriedades de diretórios e de arquivos.**

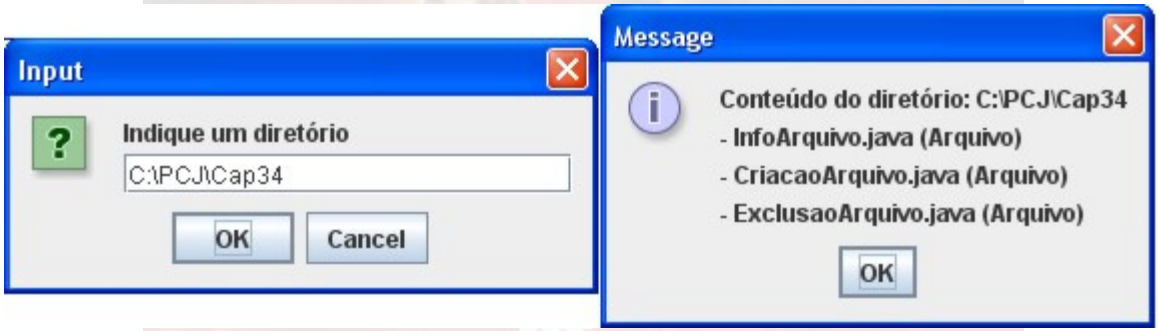
- **Código 34.3 – InfoArquivo.java**



Diretórios e Arquivos

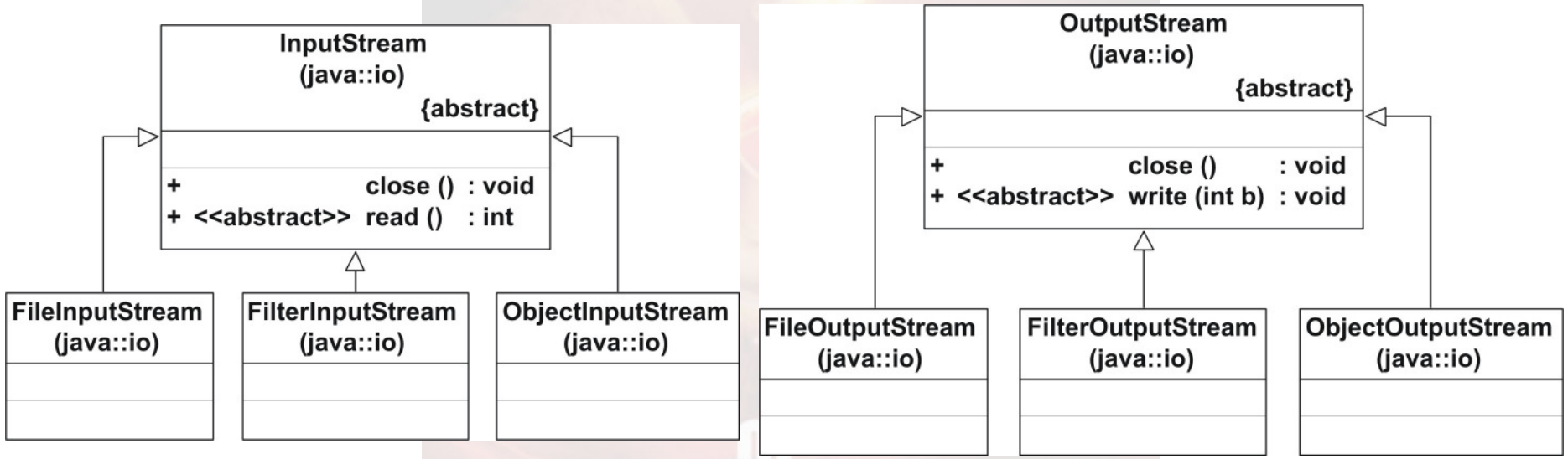
- **Exemplo: inspeção do conteúdo de um diretório.**

- **Código 34.4 – Explorador.java**



Fluxo de Bytes

- **Finalidade:**
  - **Leitura de arquivos: fluxo de entrada de bytes**
  - **Escrita de arquivos: fluxo de saída de bytes**



Fluxo de Bytes

- **public abstract class java.io.InputStream**
  - **Empregada para manipular fluxos de entrada de bytes.**
  - **Método read( ): lê o próximo byte de um fluxo de entrada.**
    - ✓ **Retorna o byte lido ou -1 se o final do fluxo for encontrado.**
  - **Método close( ): fecha o fluxo de entrada e libera os recursos associados a ele.**
  - **Assinatura dos métodos:**
    - ✓ **public abstract int read( ) throws IOException**
    - ✓ **public void close( ) throws IOException**

Fluxo de Bytes

- **public abstract class java.io.OutputStream**
  - **Empregada para manipular fluxos de saída de bytes.**
  - **Método write( ): escreve o byte especificado no fluxo de saída.**
    - ✓ **O parâmetro representa o byte a ser escrito.**
  - **Método close( ): fecha o fluxo de saída e libera os recursos associados a ele.**
  - **Assinatura dos métodos:**
    - ✓ **public abstract void write(int b) throws IOException**
    - ✓ **public void close( ) throws IOException**

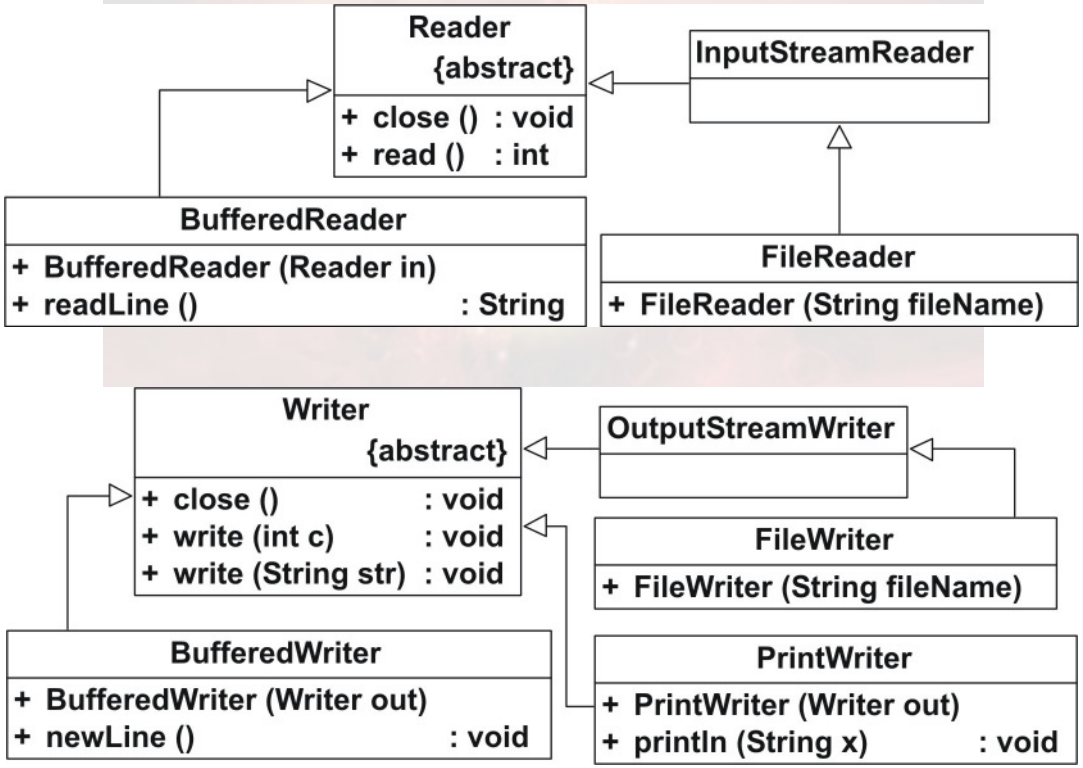
Fluxo de Bytes

- **Exemplo:**
  - **Leitura de um arquivo de texto utilizando um fluxo de entrada de bytes.**
  - **Criação de uma cópia do arquivo de texto lido utilizando um fluxo de saída de bytes.**

- **Composição do exemplo:**
  - **Código 34.5 – Edipo.txt**
  - **Código 34.6 – FluxoBytes.java**

Fluxo de Caracteres

- **Finalidade:**
  - **Leitura de arquivos: fluxo de entrada de caracteres**
  - **Escrita de arquivos: fluxo de saída de caracteres**



Fluxo de Caracteres

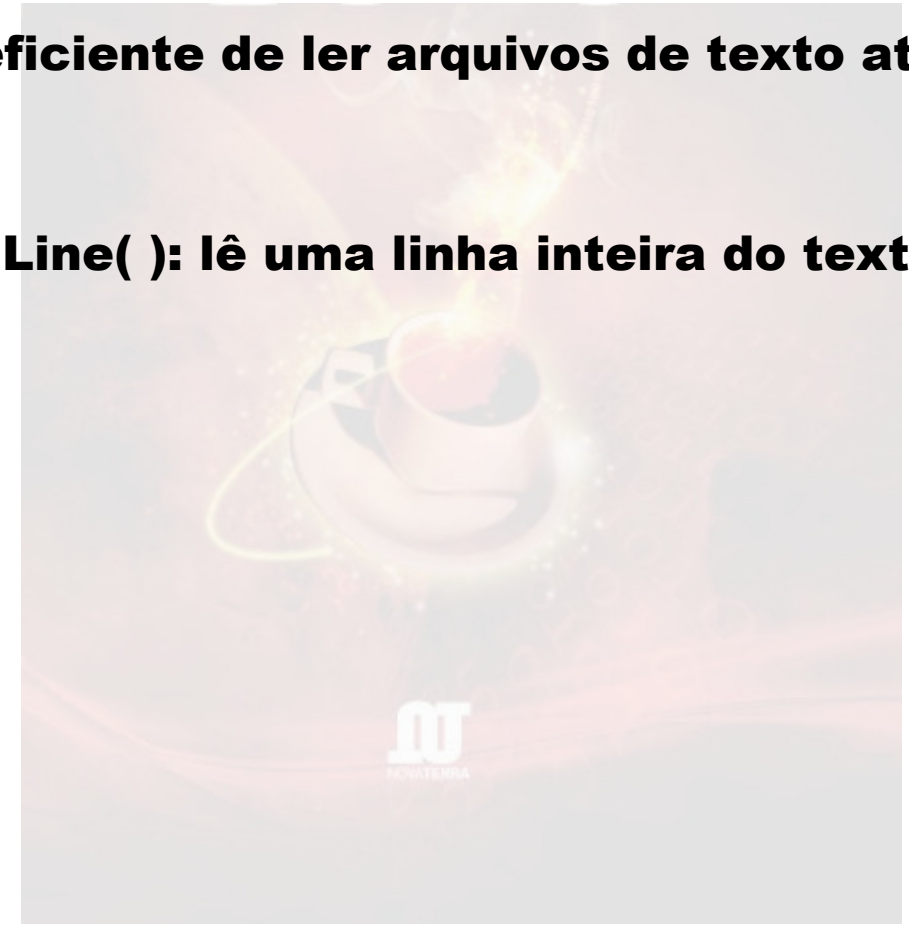
- **public abstract class java.io.Reader**
  - **Empregada para manipular fluxos de entrada de caracteres.**
  - **Método read( ): lê o próximo caractere de um fluxo de entrada.**
    - ✓ **Retorna o caractere lido ou -1 se o final do fluxo for encontrado.**
  - **Método close( ): fecha o fluxo de entrada e libera os recursos associados a ele.**
  - **Assinatura dos métodos:**
    - ✓ **public int read( ) throws IOException**
    - ✓ **public abstract void close( ) throws IOException**

Fluxo de Caracteres

- **public abstract class java.io.FileReader**
  - **Utilizada para a leitura de arquivos de texto através de fluxos de caracteres.**
  - **FileReader(String fileName): cria uma instância para ler o arquivo especificado.**

Fluxo de Caracteres

- **public class java.io.BufferedReader**
  - **Modo mais eficiente de ler arquivos de texto através de fluxos de caracteres.**
  - **Método readLine( ): lê uma linha inteira do texto.**



Fluxo de Caracteres

- **public abstract class java.io.Writer**
  - **Empregada para manipular fluxos de saída de caracteres.**
  - **Métodos:**
    - ✓ **void write(int c): escreve o caractere especificado no fluxo de saída.**
    - ✓ **void write(String str): escreve a string no fluxo de saída.**
  - **Método close( ): fecha o fluxo de saída e libera os recursos associados a ele.**
  - **Assinatura dos métodos:**
    - ✓ **public void write(int c) throws IOException**
    - ✓ **public void write(String str) throws IOException**
    - ✓ **public abstract void close( ) throws IOException**

Fluxo de Caracteres

- **public abstract class java.io.FileWriter**
  - **Utilizada para a escrita de arquivos de texto através de fluxos de caracteres.**
  - **FileWriter(String fileName): cria uma instância para escrever no arquivo especificado.**

Fluxo de Caracteres

- **public class java.io.BufferedWriter**
  - **Modo mais eficiente de escrever textos através de fluxos de caracteres.**
  - **Método newLine( ): imprime um separador de linha de forma independente de sistema operacional.**
    - ✓ **Alternativa: System.getProperty("line.separator")**

Fluxo de Caracteres

- **public class java.io.PrintWriter**
  - **Oferece um meio de tratar o separador de linha de modo independente do sistema operacional.**
  - **Método println( ): imprime o dado especificado em um fluxo de saída de caracteres e encerra a linha.**
    - ✓ **Há diferentes implementações desse método.**
  - **Assinaturas dos métodos:**
    - ✓ **public void println(String x)**
    - ✓ **public void println(Object x)**
    - ✓ **public void println(double x)**
    - ✓ **public void println(int x)**
    - ✓ **public void println(char x)**

Fluxo de Caracteres

- **Exemplo:**
  - **Uso da classe java.io.FileReader para a leitura de um arquivo de texto.**
  - **Uso da classe java.io.FileWriter para criar uma cópia do arquivo de texto lido.**

- **Composição do exemplo:**
  - **Código 34.5 – Edipo.txt**
  - **Código 34.7 – FluxoCaracteres.java**

Leitura e Escrita Bufferizadas

- **Exemplo:**
  - **Uso da classe java.io.BufferedReader para a leitura de um arquivo de texto.**
  - **Uso da classe java.io.BufferedWriter para criar uma cópia do arquivo de texto lido.**

- **Composição do exemplo:**
  - **Código 34.5 – Edipo.txt**
  - **Código 34.8 – FluxoBufferizado.java**

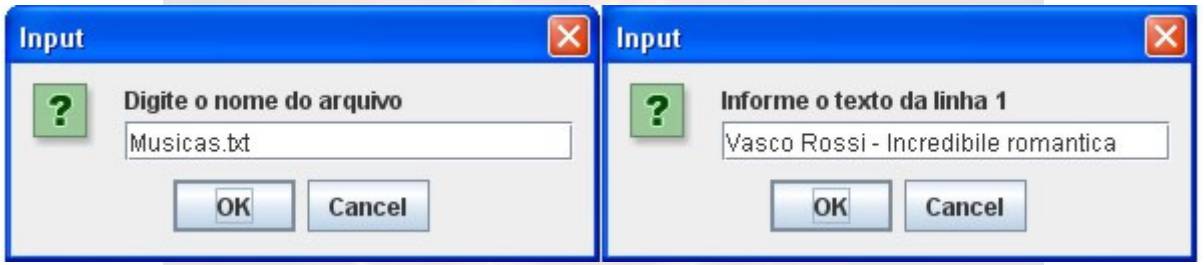
Entrada e Saída de Linhas

- **Exemplo:**
  - **Uso da classe java.io.BufferedReader para a leitura de um arquivo de texto.**
  - **Uso da classe java.io.PrintWriter para criar uma cópia do arquivo de texto lido.**

- **Composição do exemplo:**
  - **Código 34.5 – Edipo.txt**
  - **Código 34.9 – FluxoLinhas.java**

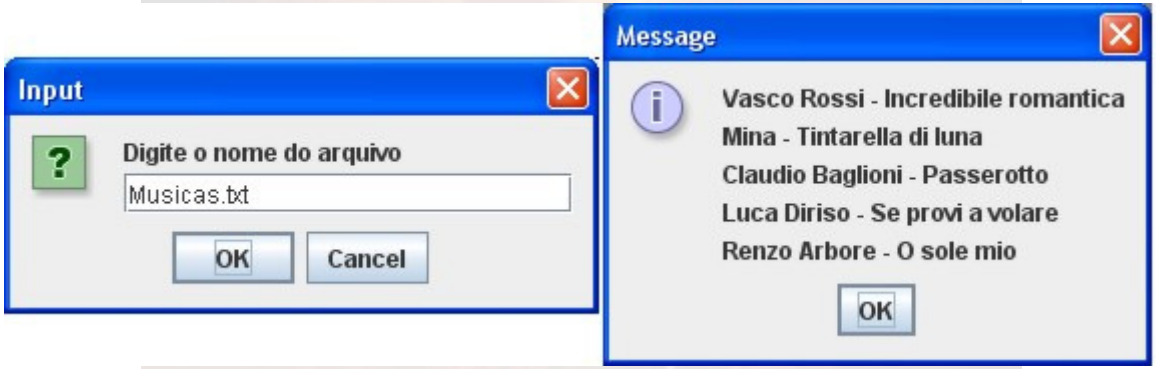
Estudo de Caso: Escritor de Arquivos

- **Código 34.10 – EscritorArquivo.java**
  - **Demonstra o uso da classe java.io.PrintWriter para criar um arquivo de texto e escrever qualquer número de linhas desejado no mesmo.**



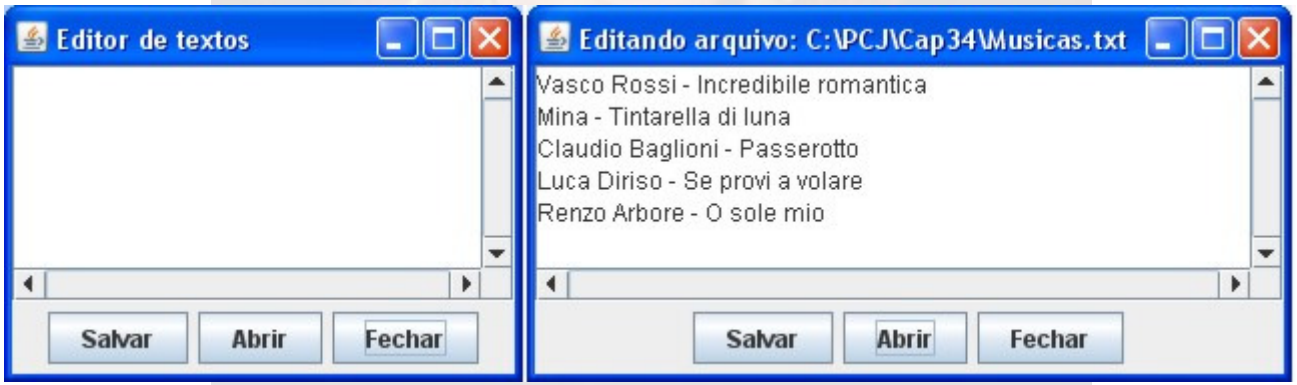
Estudo de Caso: Leitor de Arquivos

- **Código 34.11 – LeitorArquivo.java**
  - **Demonstra como ler um arquivo de texto utilizando as classes java.io.FileReader e java.io.BufferedReader.**



Estudo de Caso: Editor de Textos

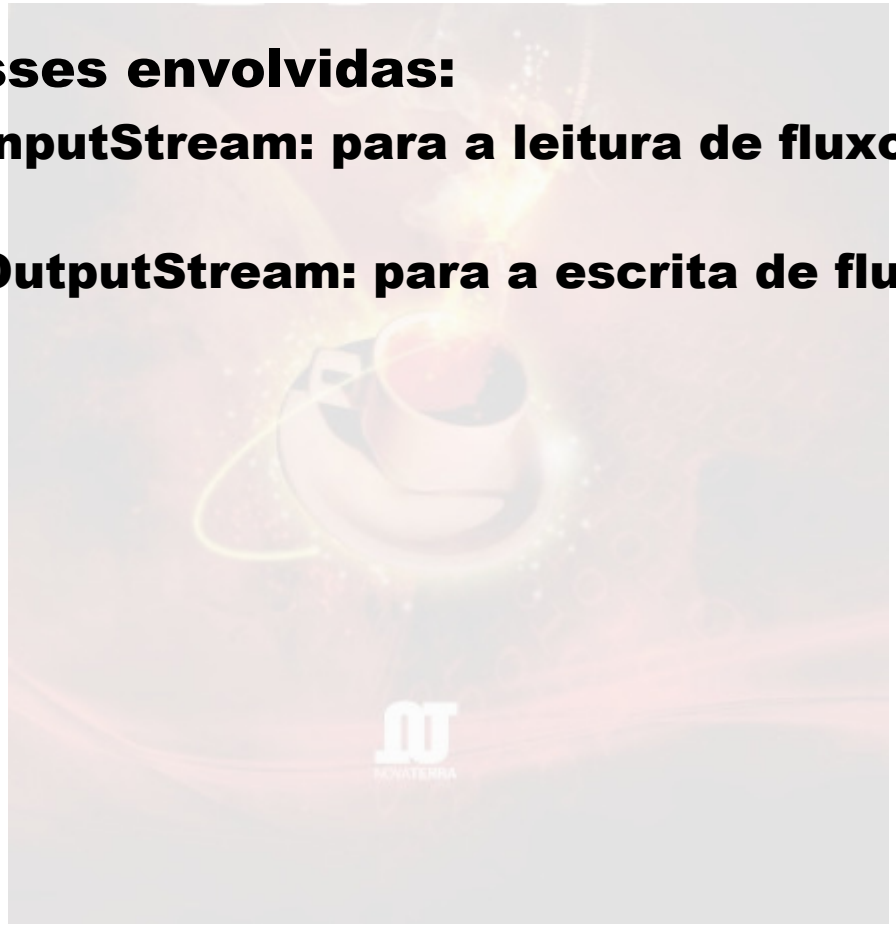
- **Código 34.12 – EditorTexto.java**
  - **Demonstra o uso de diversos recursos já apresentados na construção de um pequeno editor de textos com uma interface gráfica e que permite criar novos arquivos, salvá-los em disco, abrir arquivos existentes, alterá-los e salvar as alterações.**
  - **Demonstra o uso da classe javax.swing.JFileChooser para gerar diálogos para abrir e salvar arquivos.**



Fluxos de Dados

- Operações de entrada e saída em formato binário.

- Principais classes envolvidas:
  - java.io.DataInputStream: para a leitura de fluxos de entrada de dados.
  - java.io.DataOutputStream: para a escrita de fluxos de saída de dados.



Fluxos de Dados

Classe java.io.DataInputStream

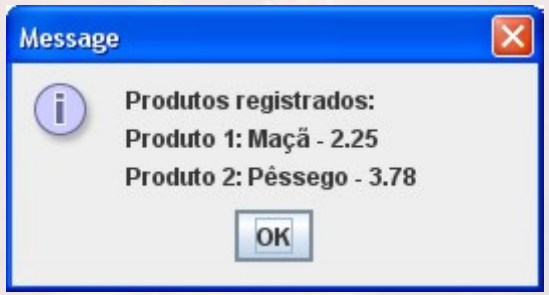
- public DataInputStream(InputStream in)
- public final boolean readBoolean( ) throws IOException
- public final byte readByte( ) throws IOException
- public final short readShort( ) throws IOException
- public final int readInt( ) throws IOException
- public final long readLong( ) throws IOException
- public final float readFloat( ) throws IOException
- public final double readDouble( ) throws IOException
- public final char readChar( ) throws IOException
- public final String readUTF( ) throws IOException

Fluxos de Dados

- **Classe java.io.DataOutputStream:**
  - **public DataOutputStream(OutputStream out)**
  - **public final void writeBoolean(boolean v) throws IOException**
  - **public final void writeByte(int v) throws IOException**
  - **public final void writeShort(int v) throws IOException**
  - **public final void writeInt(int v) throws IOException**
  - **public final void writeLong(long v) throws IOException**
  - **public final void writeFloat(float v) throws IOException**
  - **public final void writeDouble(double v) throws IOException**
  - **public final void writeChar(int v) throws IOException**
  - **public final void writeUTF(String str) throws IOException**

Fluxos de Dados

- **Código 34.13 – FluxoDados.java**
  - **Demonstra como utilizar fluxos de dados para gravar dados em arquivos e também para realizar sua leitura.**



Fluxos de Objetos

- **Finalidade:**
  - **Armazenamento de objetos em disco.**
  - **Transferência de objetos pela rede.**

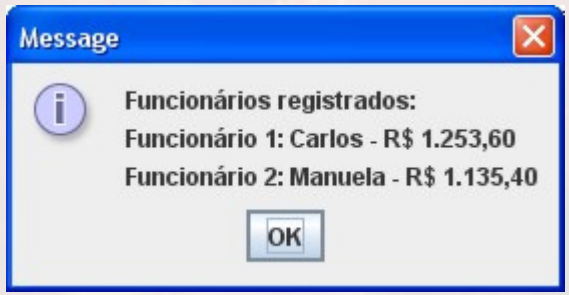
- **Só suporta objetos serializáveis**
  - **Objeto serializável: que pode ser convertido em série de bytes.**
  - **Sua classe deve realizar a interface java.io.Serializable.**

Fluxos de Objetos

- **Principais classes envolvidas:**
  - **java.io.ObjectInputStream: para a leitura de fluxos de entrada de objetos.**
    - ✓ **Object readObject( ): lê um objeto de um fluxo de entrada.**
  - **java.io.ObjectOutputStream: para a escrita de fluxos de saída de objetos.**
    - ✓ **void writeObject(Object obj): escreve o objeto especificado em um fluxo de saída.**

Fluxos de Objetos

- **Código 34.14 – FluxoObjetos.java**
  - **Demonstra como utilizar fluxos de objetos para gravar objetos em arquivos e também para realizar sua leitura.**



Exercícios

- Os exercícios que serão propostos formam uma seqüência de passos que devem ser dados para a construção de um aplicativo que utiliza diferentes tipos fluxos para gravação de dados em disco.
  - O primeiro exercício consiste na construção da janela principal deste aplicativo e na criação de sua a classe executável.
  - O segundo exercício consiste na implementação de três diferentes operações de cadastro.
  - O terceiro exercício implica em permitir a alteração e a gravação de duas configurações.
  - O quarto exercício consiste no registro do histórico de acesso ao aplicativo.
  - O quinto exercício requer apenas a geração de um diálogo com informações sobre o sistema.

Exercício 1

- **Crie uma nova janela, chamada JFPrincipal, de acordo com o modelo apresentado na figura abaixo.**

Exercício 1

- Esta é a janela principal do aplicativo e deve ter uma barra de menus e uma barra de ferramentas.
  - A barra de menus deve ter dois menus: “Cadastro” e “Sistema”.
  - O primeiro destes menus deve ter três itens: “Usuário”, “Professor” e “Aluno”.
  - O segundo menu também deve ter três itens: “Configurações”, “Histórico de Acesso” e “Sobre ...”.
  - A barra de ferramentas deve ter botões que permitam o acesso rápido às mesmas operações que são acessíveis através dos itens de menu.
  - A área de trabalho desta janela deve apresentar uma imagem de fundo.
- Crie outra classe, chamada Principal, que assuma o papel de classe principal do sistema.
  - Ela deverá ser a única classe executável deste aplicativo, ou seja, a única a ter o método main( ).
  - Ela é a responsável por criar e exibir a janela principal do sistema.

Exercício 2

- O cadastro de usuários deverá ser composto pelos seguintes dados: um código numérico, o nome completo do usuário, seu login e sua senha.
  - Crie uma classe, chamada, Usuario, para encapsular estes dados e utilize os métodos de escrita para validá-los.
- A janela de cadastro de Professores deverá ter campos para receber os seguintes dados: um código numérico, o nome completo do professor, seu salário, seu telefone e seu e-mail.
  - Além destes, acrescente campos para receber outros dados que você julgar que são pertinentes a este tipo de cadastro.
  - Crie uma classe, chamada, Professor, para encapsular estes dados e utilize os métodos de escrita para validá-los.
- A janela de cadastro de Alunos deverá ter campos para receber os seguintes dados: um código numérico, o nome completo do aluno, sua data de nascimento, seu telefone e seu e-mail.
  - Além destes, acrescente campos para receber outros dados que você julgar que são pertinentes a este tipo de cadastro.
  - Crie uma classe, chamada, Aluno, para encapsular estes dados e utilize os métodos de escrita para validá-los.

Exercício 2

- **Crie uma janela interna para cada um dos três cadastros supracitados.**
  - **A figura abaixo ilustra como os componentes destas janelas deverão ser organizados.**

Exercício 2

- Este modelo representa a janela de cadastro de usuários, mas você também pode empregá-lo para a construção das janelas de cadastro de professores e de alunos.
  - Note que estas janelas deverão dividir seus painéis em duas fichas.
  - A primeira ficha apresenta os registros existentes em uma grade.
  - A segunda ficha permite incluir novos registros, alterar registros existentes e excluí-los.
  - Utilize campos formatados sempre que isso for conveniente.

Exercício 2

- Cada usuário cadastrado deve ser representado como uma instância da classe Usuario, cada aluno cadastrado deve ser representado como uma instância da classe Aluno e cada professor cadastrado deve ser representado como uma instância da classe Professor.
  - Todos os cadastros devem ser gravados em arquivos mantidos no disco para que eles não sejam perdidos quando o aplicativo for encerrado.
  - Utilize um fluxo de dados para gravar os cadastros de usuários em um arquivo chamado “Usuario.dat”.
  - Utilize fluxos de objetos para gravar os cadastros de alunos e de professores em arquivos chamados “Aluno.obj” e “Professor.obj”, respectivamente.

Exercício 3

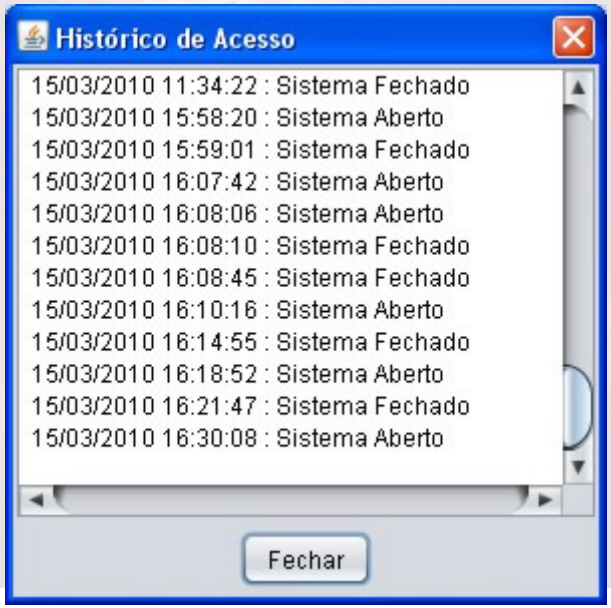
- **Sempre que o item de menu rotulado como “Configurações” for pressionado ou quando o botão correspondente da barra de ferramentas for acionado, o aplicativo deve apresentar um diálogo que permita ao usuário alterar as configurações de sua interface gráfica.**
  - **A figura abaixo ilustra como deve ser a aparência deste diálogo.**

Exercício 3

- Este diálogo deve permitir a alteração do look and feel do sistema e definir se a barra de ferramentas deve estar visível ou oculta.
  - Sempre que for aberto, a primeira caixa de combinação deve carregar dinamicamente todos os look and feels disponíveis e o look and feel em uso deve ser selecionado automaticamente.
  - A segunda caixa de combinação também já deve estar indicando a configuração atual relativa à barra de ferramentas.
  - As configurações selecionadas pelo usuário do sistema devem ser gravadas em um arquivo de texto chamado “Config.txt”.
  - Este arquivo deve ser lido pela classe Principal e ela deve aplicar as configurações selecionadas pelo usuário antes de apresentar a janela principal do sistema.

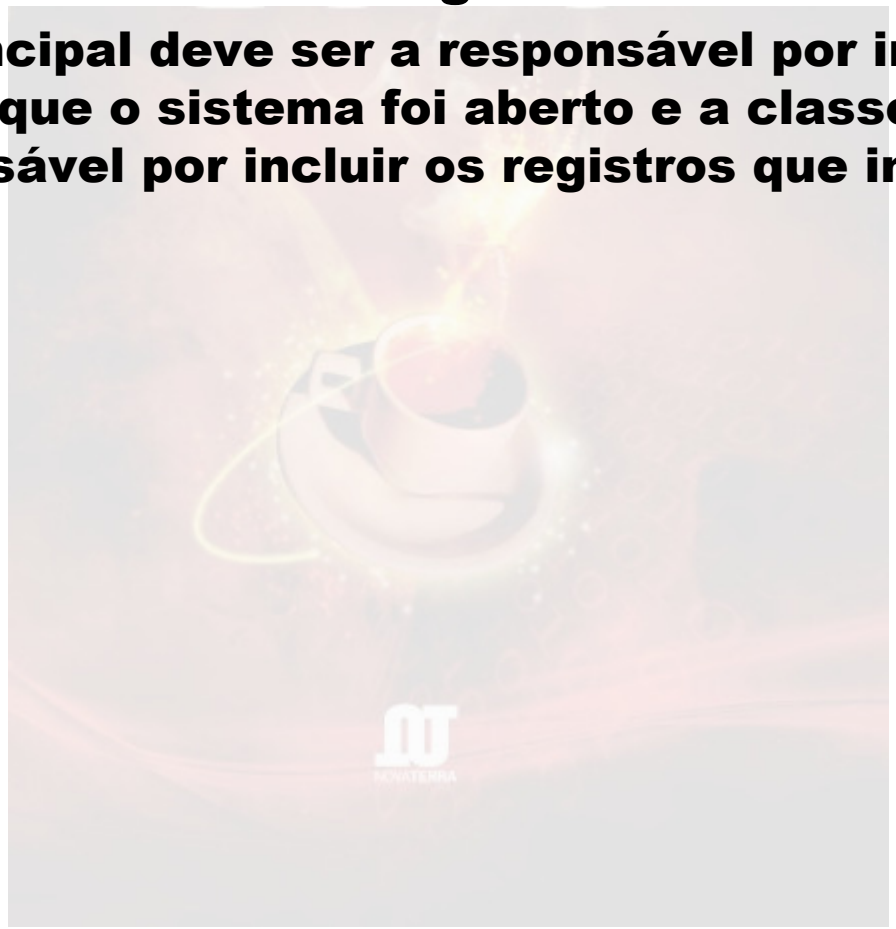
Exercício 4

- **Sempre que o item de menu rotulado como "Histórico de Acesso" for pressionado ou quando o botão correspondente da barra de ferramentas for acionado, o aplicativo deve exibir um diálogo que apresente uma lista com a data e o horário de todas as vezes em que o aplicativo foi aberto ou fechado.**
  - **A figura abaixo ilustra como deve ser a aparência deste diálogo.**



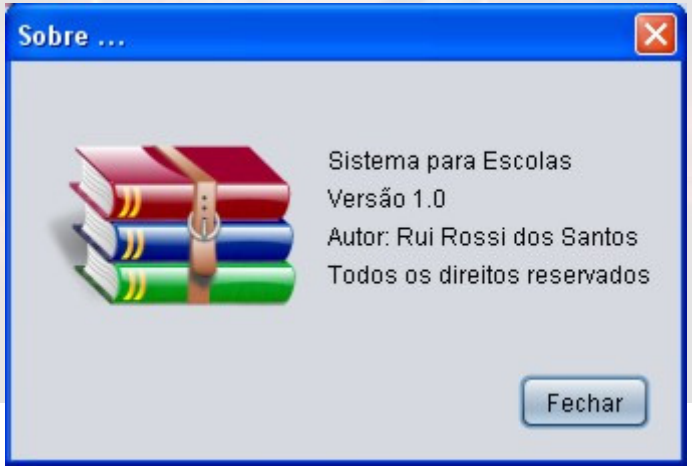
Exercício 4

- **Estes registros devem ser gravados pelo aplicativo em um arquivo de texto chamado “log.txt”.**
  - **A classe Principal deve ser a responsável por incluir os registros que indicam que o sistema foi aberto e a classe JFPrincipal deve ser a responsável por incluir os registros que indicam que ele foi fechado.**



Exercício 5

- Sempre que o item de menu rotulado como “Sobre ...” for pressionado ou quando o botão correspondente da barra de ferramentas for acionado, deve ser apresentado um diálogo contendo informações acerca do sistema.
  - A Figura 34.18 ilustra como deve ser a aparência deste diálogo.
- Este diálogo pode ser gerado através do método showOptionDialog( ) da classe javax.swing.JOptionPane.
  - Mas, se preferir, pode criar uma nova classe derivada da classe javax.swing.JDialog para representá-lo.



Contato

Com o autor:

Rui Rossi dos Santos
E-mail: livros@ruirossi.pro.br
Web Site: http://www.ruirossi.pro.br

Com a editora:

Editora NovaTerra
Telefone: (21) 2218-5314
Web Site: http://www.editoranovaterra.com.br