Manipulando Arquivos em Java

Arquivos em Java

- O Java trata qualquer entrada e saída de dados como fluxos (Stream), pertencentes ao pacote **java.io**;
- Pode-se tratar um fluxo de dados de maneira que, independente se ele é proveniente de uma comunicação em rede, de arquivos ou do teclado, a leitura e o controle sejam realizados da mesma forma.

File

- A classe File representa um ponteiro para um caminho. Isso não significa que o arquivo exista ou seja criado.
- Exemplos:
  - File arquivoNaPasta = new File(“arquivo.txt”);
  - File arquivoNaPastaPai = new File(“../arquivo.txt”);
- Em Java, não há como mudar a pasta atual, mas pode-se usar caminhos relativos a outros arquivos:
  - File diretorioRaiz = new File("/");
  - File arquivo1 = new File(diretorioRaiz,"autoexec.bat");
  - File arquivo2 = new File(diretorioRaiz,"config.sys");
  - File diretorioWindows = new File(diretorioRaiz,"windows");
  - File diretorioWindows2 = new File("/windows/");
  - File diretorioWindows3 = new File("/windows");
  - File diretorioWindows4 = new File("c:\\windows");

File

- Alguns métodos importantes:
  - getName() - Retorna o nome do arquivo.
  - renameTo(File) - Renomeia o arquivo.
  - exists() - Verifica se o arquivo existe.
  - canWrite() - Verifica se é possível escrever no arquivo.
  - canRead() - Verifica se um arquivo pode ser lido.
  - isFile() - Verifica se o caminho definido é um arquivo.
  - lastModified() - Recupera a data da última alteração do arquivo.
  - length() - Tamanho do arquivo.
  - delete() - Exclui o arquivo.
  - getPath() - Retorna o nome do diretório.
  - getAbsolutePath() - Nome completo do diretório.
  - getParent() - Retorna os diretórios acima do arquivo.
  - isDirectory() - Verifica se o caminho definido é um diretório.
  - isAbsolute() - Verifica se o caminho é absoluto.
  - mkDir() - Cria um diretório.
  - List() -Lista arquivos no diretório.

Streams

- Da mesma forma que na comunicação em rede, a transferência de dados é realizada por objetos da classe Stream:
  - **Reader** – Streams de entrada de caracteres.
  - **Writer** – Streams de saída de caracteres.
  - **InputStream** – Streams de entrada de bytes.
  - **OutputStream** – Streams de saída de bytes.
  - **FileReader** – Leitura de caracteres de um arquivo.
  - **FileWriter** – Escrita de caracteres em arquivo.
  - **FileInputStream** – Leitura de bytes de um arquivo.
  - **FileOutputStream** – Escrita de bytes em um arquivo.

Streams

- Para auxiliar a leitura e escrita em Streams, utilizam-se métodos de subclasses que gerenciam seus conteúdos em memória:
  - **BufferedReader** – permite a leitura de caracteres, linhas e texto de maneira mais eficiente;
  - **PrintWriter** – permite a escrita de linha de texto de maneira eficiente.
- Dessa forma, a leitura e escrita de textos resume-se aos métodos:
  - String s = BufferedReader.read();
  - String s = BufferedReader.readLine();
  - PrintWriter.print(String s);
  - PrintWriter.println(String s);

JFileChooser

- Para a seleção de um arquivo pelo usuário, pode-se usar as caixas de diálogo da clase JFileChooser:
  - showOpenDialog();
  - showSaveDialog();
- Estes métodos retornam um valor inteiro, que pode ser JFileChooser.APPROVE_OPTION ou JFileChooser.CANCEL_OPTION;
- O arquivo selecionado pode ser recuperado pelo método **getSelectedFile()**;

JFileChooser

- Exemplo:

```
JFileChooser abrir = new JFileChooser();
int res = abrir.showOpenDialog(this);
if (res == JFileChooser.APPROVE_OPTION) {
    JOptionPane.showMessageDialog(null,"Voce selecionou o arquivo
" + abrir.getSelectedFile().getName());
}
```