Stream de Arquivos

Prof. Bruno Gomes

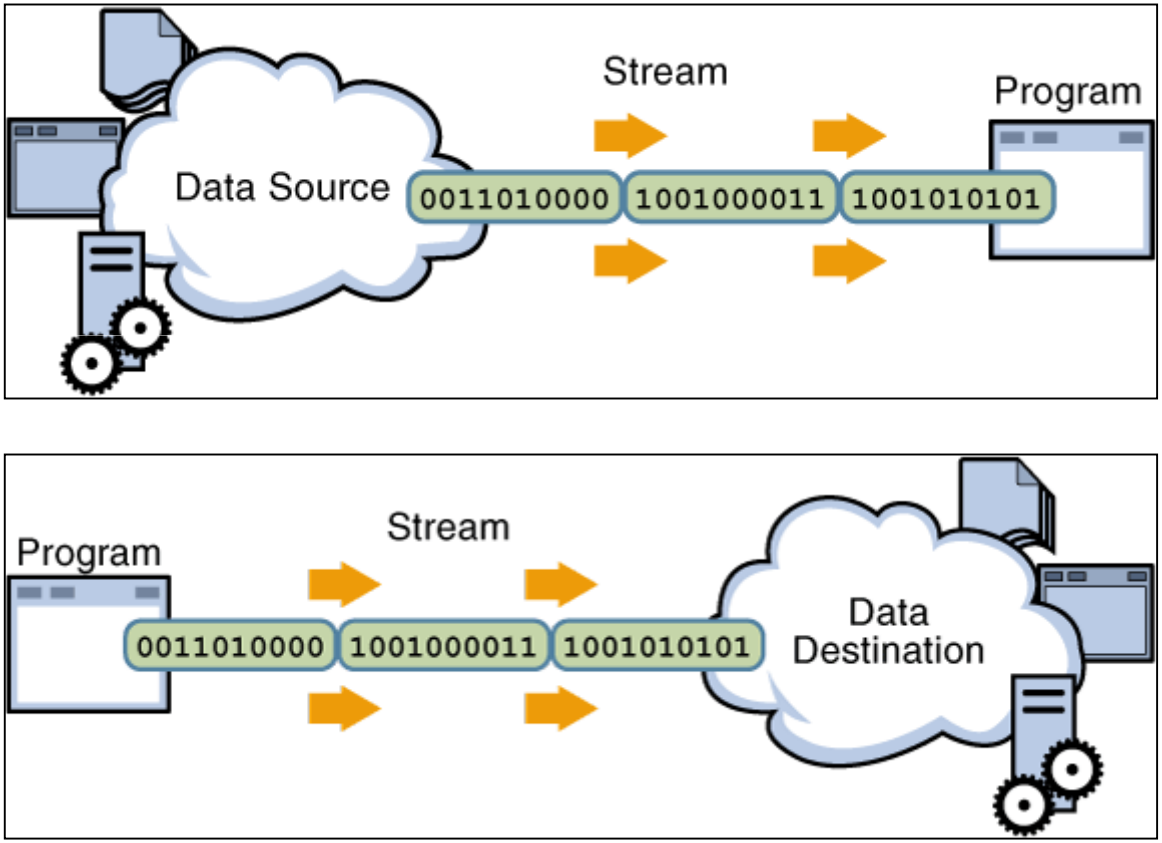
bruno.gomes@ifrn.edu.br

Programação Orientada a Objetos

Introdução

- Arquivos são utilizados para armazenamento a longo prazo de dados, mesmo finalizando o programa que criou os dados
- Os dados armazenados em arquivos são chamados de dados persistentes
- Java reconhece cada arquivo como um fluxo sequencial de *bytes* – *Array* de *Bytes* (*Stream*)
- Java abre um arquivo através da criação de um objeto e a associação de um fluxo de *bytes* a este objeto

Introdução



Introdução

- Processamento de arquivos é feito em programas java utilizando as classes do pacote *java.io:*
  - *FileInputStream* (entrada baseada em *bytes*)
  - *FileOutPutStream* (saída baseada em *bytes*)
  - *FileReader* (Entrada baseada em caracteres)
  - *FileWriter* (Saída baseada em caracteres)
- Arquivos são abertos criando-se objetos dessas classes de fluxo, que herdam de *InputStream*, *OutPutStream*, *Reader* e *Writer*

Classe File

- Representam caminhos (paths) para possíveis Arquivos ou Diretórios no SO
- Arquivos e diretórios
  - Diretórios são arquivos especiais cujo conteúdo são outros arquivos

Classe File

- Métodos
  - int length() - tamanho do arquivo em bytes
  - boolean isFile() - verdadeiro se é arquivo
  - boolean isDirectory() - verdadeiro se é diretório
  - boolean canRead() - verdadeiro se tem permissão de leitura
  - boolean canWrite() - verdadeiro se tem permissão de escrita
  - String[] list()
    - Se o objeto for diretório retorna *array* de Strings como os nomes dos arquivos desse diretório
    - Se for um arquivo retorna *null*

Classe File

- Exemplo:

```
File file = new File("c:\\arquivos\\documento.txt");
System.out.println(file.getAbsolutePath());
System.out.println(file.getName());
System.out.println(file.getParent());
System.out.println(file.getPath());
System.out.println(file.getTotalSpace());
System.out.println(file.getUsableSpace());
System.out.println(file.lastModified());
System.out.println(file.isFile());
System.out.println(file.isDirectory());
System.out.println(file.canExecute());
```

Classe File

- Exemplo:

```
File file;
String nomeArquivo = JOptionPane.showInputDialog(“Digite um arquivo ou
diretório do computador");
file = new File(nomeArquivo);
if (file.isDirectory()){
  System.out.println("É um diretório");
  String [] arquivos = file.list();
  for(int i = 0 ; i < arquivos.length ; i++){
    System.out.println(arquivos[i]);
  }
} else {
  System.out.println("É um arquivo");
  System.out.println("Tamanho: "+file.length());
}
```

Byte Streams

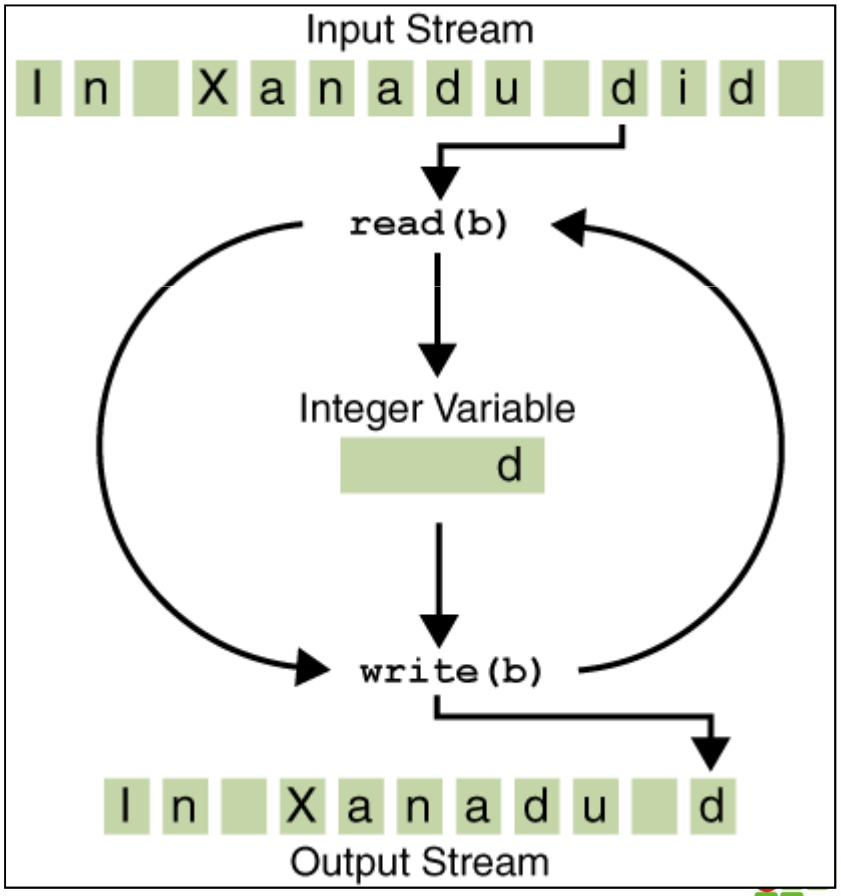
- Programas usam *streams* de *bytes* para manipular entradas e saídas de bytes de 8-bits.
- Todas as classes que manipulam *stream* de *bytes* herdam de *InputStream* e *OutputStream*.

Byte Streams – *FileInputStream e FileOutputStream*

```
public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
                in = new FileInputStream("documento1.txt");
                out = new FileOutputStream("documento2.txt");
                int c;
                while ((c = in.read()) != -1) {
                        out.write(c);
                }
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        } finally {
                if (in != null) {
                        in.close();
                }
                if (out != null) {
                        out.close();
                }
        }
}
```

Byte Streams

- Representação:



InputStream

- Declara métodos para ler bytes de uma determinada origem
- É a superclasse da maioria dos Streams de bytes de entrada em java.io
- Métodos:
  - int read(), int skip(byte[] buf), int avaliable(), void close();

InputStream

- Exemplo:

```
InputStream in = new
FileInputStream("C:\\arquivos\\documento.txt");
int total = 0;
while (in.read() != -1)
  total++;
System.out.println(total + " bytes");
```

Sempre feche o Stream

- Ao terminar de manipular um *stream* de *bytes*, sempre feche (close())
- Motivos:
  - Esta operação pode provocar uma exceção
  - Liberar recursos
- Portanto, escreva-a em um bloco *finally*

Por que usar o Byte Streams

- Devem ser usados apenas para as mais primitivas I/O
- Todos os outros tipos de fluxo são construídos sobre os fluxos de bytes

Character Streams

- Herdam de Reader e Writer
- Classes especializadas:
  - FileReader e FileWriter

Character Streams

```
public static void main(String[] args) throws IOException {
FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
            inputStream = new FileReader("documento1.txt");
            outputStream = new FileWriter("documento2.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
}
```

Leitura Linha a Linha

- Não é comum ler caractere a caractere
- Utiliza a leitura linha a linha
- Utilização de *Buffer*
- Classes *BufferedReader* e *PrintWriter*
- *A linha pode ser terminada por:*
  - "\r\n”
  - "\r”
  - "\n"

Leitura Linha a Linha

```
public static void main(String[] args) throws IOException {
BufferedReader inputStream = null;
      PrintWriter outputStream = null;
      try {
         inputStream = new BufferedReader(new FileReader("documento1.txt"));
         outputStream = new PrintWriter(new FileWriter("documento2.txt"));

         String l;
         while ((l = inputStream.readLine()) != null) {
            outputStream.println(l);
         }
      } finally {
         if (inputStream != null) {
            inputStream.close();
         }
         if (outputStream != null) {
            outputStream.close();
         }
      }
}
```

Utilizando Scanner

- Scanner são úteis para “quebrar” entradas em tokens e “traduzi-los” de forma individual
- Le *InputStreams*

Utilizando Scanner

- Exemplo:

```
public static void main(String[] args) throws IOException {
Scanner s = null;
      try {
         s = new Scanner(new BufferedReader(new FileReader("xanadu.txt")));

         while (s.hasNext()) {
            System.out.println(s.next());
         }
      } finally {
         if (s != null) {
            s.close();
         }
      }
}
```