Material de Apoio
Aula 3

Manipulação de Streams

Em Java, assim como em C e C++, a entrada/saida (input/output) é realizada através de streams. Streams consistem em uma abstração criada para representar locais reais (teclado, disco, monitor, rede de comunicação, etc) de onde dados devem ser lidos ou escritos. Streams representam, portanto a fonte de um fluxo de dados em entrada ou em saída, observe que estes fluxos são unidirecionais: um stream de entrada pode ser lido e um stream de saída pode ser escrito. O interesse maior desta abstração é prover o mesmo conjunto de serviços para manipular diferentes tipos de dispositivos e arquivos. A Figura 1 ilustra streams de entrada e saída. Observe que canais de comunicação e arquivos em disco podem ser abertos em entrada ou em saída. Teclado e monitor permitem apenas uma direção.

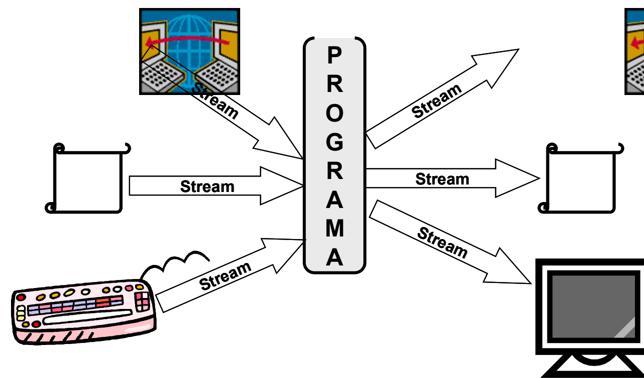

Figura 1. Streams representam fluxos unidirecionais de dados, seja em entrada ou saída.

Curiosidade: a palavra stream deriva de uma analogia a expressão em inglês *stream of water*, significando fluxo de água. Um stream de entrada parece-se com uma torneira, a qual abrimos para permitir a entrada de água e um stream de saída é semelhante a um ralo onde, com ajuda de uma rolha, o fluxo de saída pode ser regulado.

Existem dois mecanismos básicos para manipular streams: bufferizados e não bufferizados. Estes últimos acessam diretamente os arquivos ou dispositivos indicados como de entrada ou saída de dados. Mecanismos bufferizados empregam uma zona de memória própria ao mecanismo de entrada e saída para armazenar temporariamente os dados manipulados. Utilizando a analogia com fluxo de água, mecanismos bufferizados utilizam uma "caixa d´água" para reter uma quantidade de água próxima ao consumidor (o programa em execução), quando esta caixa estiver cheia, o fluxo de água para dentro dela é interrompido até que ela seja esvaziada um pouco. De forma semelhante, um tanque de saída armazena água até que esteja cheio – quando isto ocorre, todo um "conjunto de água" é despejado.

Em Java são definidos dois tipos de streams: streams de caracteres e streams de bytes. O primeiro é dito modo texto, cujo conteúdo é compreensível na linguagem humana. O segundo consiste na representação da informação em termos de bytes, a mesma representação utilizada pelos computadores. A vantagem do primeiro tipo é que os arquivos podem ser manipulados diretamente por humanos. A vantagem do segundo é que a manipulação em programas é mais eficiente, uma vez que não é necessário traduzir "texto" em informação manipulável pelo programa.

Recursos em Java

O pacote `java.io` disponibiliza um grande conjunto de classes para manipular streams. Para poder utilizar este pacote é necessário informar explicitamente sua necessidade no início de seu arquivo .java com a seguinte diretiva:

```
import java.io.*;
```

Com esta diretiva esta sendo indicado que todas as classes do referido pacote poderão ser utilizadas no programa. Neste pacote existem classes para manipular tanto streams bufferizados como não bufferizados, em modo texto e em modo byte. Algumas destas classes e seus serviços serão objetos de estudo neste material.

`InputStream` e `OutputStream`

Estas classes são as classes mais genéricas para entrada e saída de stream de bytes. Esta classe define a estrutura para entrada e saída a qual é especializada por outras classes, conforme mostra a Figura 2. A Figura 3 descreve alguns dos serviços disponíveis na classe InputStream e a Figura 4 serviços da classe OutputStream.

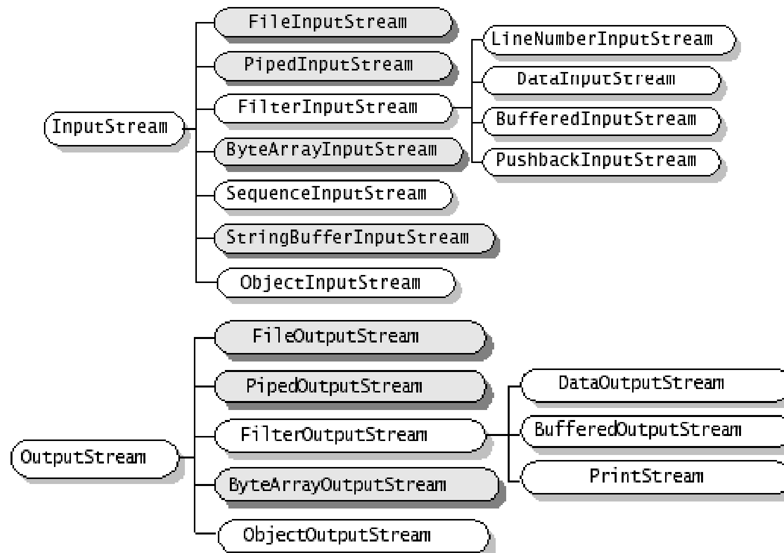

Figura 2. Hierarquia de classes para as classes `InputStream` e `OutputStream`.

int read ( )	Lê um byte e retorna o seu valor em um inteiro. Retorna –1 se chegou ao fim do arquivo.
int read (byte b[ ])	Escreve os bytes lidos na array de bytes b. Será lido, no máximo, b.length bytes. Retorna o número de bytes lidos.
int read (byte b[], int off, int length)	Escreve length bytes lidos na array de bytes passada como parâmetro. O primeiro byte lido é armazenado na posição off da array. Retorna o número de bytes lidos
void close ( )	Fecha a stream. Se existir uma pilha de stream, fechar a stream do topo da pilha, irá fechar todas as outras streams.
int available ( )	Retorna o número de bytes disponíveis para leitura.
long skip (long nroBytes)	Este método é usado para movimentar o ponteiro do arquivo. Ele descarta nroBytes bytes da stream. Retorna o número de bytes descartados.
boolean void markSupported ( )	retorna true se os métodos mark() and reset() são suportados pela stream
void mark (int posicao)	marca um determinado byte no arquivo
void reset( )	volta ao ponto marcado

Figura 3. Serviços para fluxos de entrada com `InputStream` e suas classes especializadas.

void write (int)	Grava um byte na stream.
void write (byte b [])	Grava os bytes contidos na array b na stream.
void write (byte b[], int off, int length)	Grava length bytes da array para a stream. O byte b[off] é o primeiro a ser gravado.
void flush()	Algumas vezes a stream de saída pode acumular os bytes antes de gravá-los. O método flush ( ) força a gravação dos bytes.
void close ( )	Fecha a stream.

Figura 4. Serviços para fluxos de saída com `OutputStream` e suas classes especializadas.

O exemplo de código da Figura 5 apresenta um trecho de código para experimentar a entrada de dados a partir da console. Neste exemplo, `System.in` representa o dispositivo de entrada "teclado".

```
import java.io.*;
public class ConsoleInputStream {

  public static void main( String args[] ) {
    String str = "";
    char c = ' ';

    System.out.print( "Digite alguma coisa: " );
    for( ; ; ) {
      try {
        c = (char)System.in.read();
      }
      catch(IOException e) {
        System.out.println ("Ocorreu erro");
        c = -1;
        str = "";
      }
      if( (c == -1) || (c=='\n') ) // Quando chegar no fim ou <ENTER>
        break;                       // interrompe o loop de leitura
      str += c;
    }
    System.out.print( "Foi Digitado: " + str );
  }
}
```

Figura 5. Exemplo de programa para ler um string do teclado, caracter a caracter.

Exercício

1. Digite o programa acima, compile-o e observe o resultado. Documente, linha a linha, o programa fonte.

Manipulação de arquivos

A manipulação de arquivos em Java pode ser realizada com as classes `FileInputStream` e `FileOutputStream`. Ao contrário do teclado e do monitor, arquivos não se encontram abertos para leitura ou escrita. O desejo destas operações deve ser indicado de forma explicita. Assim como o arquivo é aberto, ele deve ser fechado após seu uso. Os serviços básicos destas classes são apresentados na Figura 6. Informações completas em http://java.sun.com/j2se/1.3/docs/api/java/io/FileInputStream.html e .../FileOutputStream.html.

<table>
<tr><th colspan="2">FileInputStream</th><th colspan="2">FileOutputStream</th></tr>
<tr><td colspan="4">Construtores</td></tr>
<tr><td rowspan="2">FileInputStream(String n)</td><td rowspan="2">Abre um arquivo para leitura que possui como nome um string n</td><td>FileOutputStream(String n)</td><td>Abre um arquivo para escrita que possui como nome um string n. Deleta o arquivo com mesmo nome se ele existir.</td></tr>
<tr><td>FileOutputStream(String n, boolean a)</td><td>Abre um arquivo para escrita que possui como nome um string n, se o booleano a for true, adiciona o que for escrito no final do arquivo já existente. Caso seja false, deleta o arquivo existente.</td></tr>
<tr><td>int available()</td><td>Retorna o número de bytes disponíveis para serem serem lidos do stream.</td><td rowspan="2">void write(int b)</td><td rowspan="2">Grava o byte b no stream de saída.</td></tr>
<tr><td>void close()</td><td>Fecha o stream de entrada.</td></tr>
<tr><td>int read()</td><td>Lê um byte de dados.</td><td rowspan="2">void write(byte[] b)</td><td rowspan="2">Grava b.length bytes de b no stream de saída.</td></tr>
<tr><td>int read(byte[] b)</td><td>Lê b.length bytes de dados e coloca no array b.</td></tr>
<tr><td>long skip(long n)</td><td>Descarta n bytes do stream</td><td></td><td></td></tr>
</table>

Figura 5. Serviços de manipulação de fluxos.

Atenção: se o valor de leitura com o `read` for -1, significa que o final do arquivo foi atingido.

Exercícios

2. Escreva um programa que opere como o comando `type` do MS-DOS leia um arquivo cujo nome foi passado como parâmetro e o imprima na tela. Antes de abrir o arquivo, teste com a classe File se o arquivo existe, e avise o usuário caso ele não exista.

   Dica:

Linha de comando:	Dentro do programa:

`java meuprograma meuprograma.java`	`public static void main( String[] args ) throws IOException {`
	` FileInputStream in = new FileInputStream(args[1]);`
	` …`
	`}`

3. Escreva um programa que opere como o comando `copy` do MS-DOS, o qual deve receber como parâmetro dois nomes de arquivo, um de entrada e outro de saída. O programa deve copiar o arquivo de entrada no novo arquivo de saída.

4. Implemente e analise o resultado do programa abaixo. Comente o programa linha a linha e observe que este programa inclui o operador ternário.

```
import java.io.*;
import java.text.*;  //DateFormat, SimpleDateFormat
import java.util.Date;

class TestaArquivo {
  public static void main (String args []) {
    File f = new File("TestaArquivo.java");
    System.out.println("Nome do arquivo: " + f.getName());
    System.out.println("Caminho: " + f.getPath()); // retorna diretorio denotado pelo nome
    System.out.println("Caminho Absoluto: " + f.getAbsolutePath ()); // consulta SO
    System.out.println("DiretÃ³rio pai: " + f.getParent ());
    System.out.println(f.exists() ? "Existe" : "Nao existe");
    System.out.println(f.canWrite() ? "Pode ser gravado":"Nao pode ser gravado");
    System.out.println(f.canRead() ? "Pode ser lido" : "Nao pode ser lido");
    System.out.println(f.isDirectory () ? "Eh diretorio":"Nao eh diretorio");
    DateFormat df = new SimpleDateFormat( "dd/mm/yyyy" );
    Date data = new Date( f.lastModified() );
    System.out.println("Ultima modificacao do arquivo: " + df.format (data));
    System.out.println("Tamanho do arquivo: " + f.length() + " bytes.");
  }
} // da classe TestaArquivo
```

5. Implemente o comando `cat` do Unix. A linha de comando do programa deve ser:

```
java Cat nomeArquivo1 nomeArquivo2 ... nomeArquivon
```

Como resultado, o conteúdo de todos os arquivo que existam devem ser apresentados na tela. Caso algum arquivo não exista, o comando não deve ser realizado.

A classe File

Esta classe é uma abstração para manipulação de arquivos em Java. Alguns de seus serviços encontram-se listados na Figura 6. Note que estes serviços não são para ler ou gravar nos arquivos, e sim manipulá-los. Mais informações em: http://java.sun.com/j2se/1.3/docs/api/java/io/File.html.

`File( String n )`	Cria um novo objeto para referenciar o arquivo especificado.
`boolean canRead()`	Retorna true ou false, indicando se a aplicação pode ou não ler o dito arquivo.
`boolean canWrite()`	Retorna true ou false, indicando se a aplicação pode ou não escrever no dito arquivo.
`boolean compareTo( Object ob )`	Compara o nome do arquivo corrente com o arquivo passado em parâmetro.
`String getName()`	Retorna o nome do arquivo.
`String getPath()`	Retorna o nome do caminho do arquivo.
`String isFile()`	Retorna true se o objeto se refere a um arquivo.
`String isDirectory()`	Retorna true se o objeto se refere a um diretorio.
`String[] list()`	Retorna um array de strings, contendo todos os arquivos que constam no diretório.

Figura 6. Serviços de manipulação de arquivos.

Exercício

6. Escreva um programa que receba como parâmetro o nome de um diretório. O programa deve listar todos os arquivos deste diretório. Caso não seja um diretório, deve ser retornada uma mensagem adequada.

Manipulação de arquivos não bufferizada

A manipulação de arquivos se dá através das classes `FileReader` e `FileWriter`, cuja hierarquia de classes é apresentada na Figura 7. A Figura 8 apresenta alguns dos serviços disponíveis nesta classe. Estas operações acessam diretamente os arquivos especificados a cada operação.

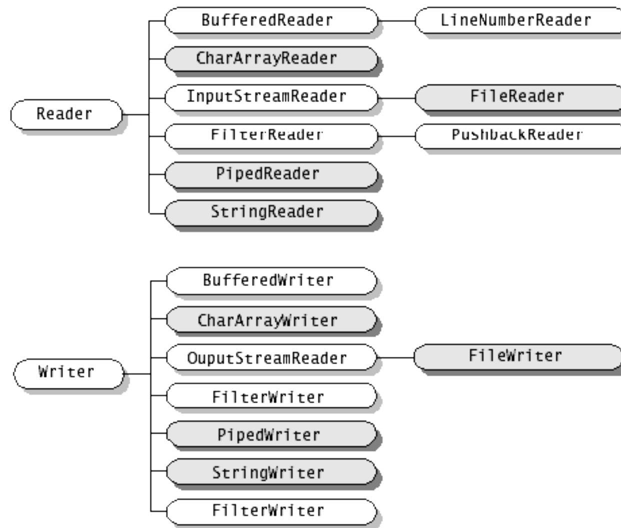

Figura 7. Hierarquia de classes para manipulação de arquivos.

FileReader		FileWriter	
`FileReader(File f)`	Cria um objeto para manipular a leitura de um arquivo especificado pelo objeto File f	`FileWriter(File f)`	Cria um objeto para manipular a escrita em um arquivo especificado pelo objeto File f
`FileReader(String n)`	Cria um objeto para manipular a leitura um arquivo especificado pelo nome n	`FileWriter(String n)`	Cria um objeto para manipular a escrita de um arquivo especificado pelo nome n
`close, getEncoding, read, read, ready`	Métodos herdados de `InputStreamReader`	`FileWriter(String n, append a)`	Cria um objeto para manipular a escrita de um arquivo especificado pelo nome n, se o parâmetro a for true, escreve no final do arquivo.
		`close, flush, getEncoding, write, write, write`	Métodos herdados de `InputStreamWriter`

Figura 8. Serviços para manipulação de streams em arquivos com `FileReader` e `FileWriter`.

Manipulação de arquivos bufferizada

A manipulação bufferizada de arquivos tem como principal beneficio o aumento de desempenho global nas operações de entrada e saída. Com este mecanismo, um conjunto de caracteres é armazenado em um espaço de dados temporário antes que seja necessário lê-lo pelo programa ou escrevê-lo efetivamente no arquivo. O uso deste recurso se dá como segue.

- Leitura bufferizada

```
FileReader fr = new FileReader( nome_do_rquivo ); //abre o arquivo para leitura
BufferedReader in = new BufferedReader(fr);       //cria um manipulador bufferizado
String str;
str = in.realine(); // le uma linha inteira do arquivo
```

- Escrita bufferizada

```
BufferedWriter out = new BufferedWriter(new FileWriter("CopiaArquivo.txt"));
out.write("Texto");
out.newLine(); // insere um separador de linha
```

As Figuras 9 e 10 apresentam exemplos de manipulação bufferizada de arquivos. Na Figura 9 é apresentado um exemplo de leitura a partir do teclado e a Figura 10 apresenta um exemplo de leitura de um arquivo.

```
import java.io.*;
class LeituraConsole{
      public static void main( String args[] ) {
        InputStream in = System.in;
        InputStreamReader is = new InputStreamReader(in);
        BufferedReader console  = new BufferedReader(is);
        System.out.print ("Qual é o seu nome: ");
        String name = null;
        try {
      name = console.readLine();
        } catch (IOException e) {
          name = "<" + e + ">";
        }
        System.out.println ("Hello "+name);
      }
}
```

Figura 9. Entrada de dados bufferizada a partir do teclado.

```
import java.io.*;
class LeArquivo {
       public static void main (String args []) {
         String filename = "Circle.java";
         try {
           FileReader fr = new FileReader(filename);
           BufferedReader in = new BufferedReader (fr);
           String line;
           while( (line=in.readLine())!= null )
         System.out.println (line);
              in.close ();
           }
           catch( IOException e ) {
             System.out.println ("Erro na leitura");
           }
       }
}
```

Figura 10. Entrada de dados bufferizada a partir do teclado.

Exercícios

7. As classes `Integer` e `Double` são classes que permitem o armazenamento de valores inteiros e em ponto flutuante – os mesmo valores que seus homônimos `int` e `double` como tipo de dados primitivos. Estas classes possuem os seguintes métodos estáticos:
   - `double Double.parseDouble( String aux )`, onde `aux` é um string que deve conter um número em ponto flutuante cujo valor é retornado;
   - `int Integer.parseInt( String aux )`, onde `aux` é um string que deve conter um número em inteiro cujo valor é retornado.

   Implementar uma classe de entrada de dados que responda a funcionalidade apresentada na documentação da classe `Entrada` do pacote `es.jar` em http://www.inf.unisinos.br/~gersonc/rep/es.

8. Escreva um programa que leia do teclado o nome e as notas de GA e GB de 5 alunos. O programa deve escrever os dados lidos em um arquivo, onde a informação de cada aluno é registrada em 3 linhas: a primeira com o nome, a segunda com o GA e a terceira com o GB.

9. Escreva um programa que receba como parâmetro um arquivo que contenha a informações sobre os alunos conforme especificado no exercício anterior. Para cada aluno deve ser calculada sua média. Como saída, deve ser gerado um novo arquivo, onde cada linha corresponde às informações de cada aluno no seguinte formato: nome, GA, GB, media.

10. Escreva um programa que contenha uma classe chamada `Listagem`. Esta classe recebe como parâmetro de construção um arquivo que deve respeitar o formato especificado no exercício anterior. Esta classe deve ler o arquivo e gerar um novo arquivo no mesmo formato com os alunos ordenados pelas suas notas – da maior para a menor.