PRÁTICA 01 – STREAMS EM JAVA

Tales Bitelo Viegas <talesbv@ulbra.edu.br>

Universidade Luterana do Brasil (Ulbra) – Curso de Ciência da Computação – Câmpus Gravataí
Av. Itacolomi, 3.600 – Bairro São Vicente – CEP 94170-240 – Gravataí - RS

1 STREAMS

Uma stream é uma abstração que representa uma fonte genérica de entrada de dados ou um destino genérico para escrita de dados, que é definida independentemente do dispositivo físico concreto. Todas as classes que implementam streams em Java são subclasses das classes abstratas **InputStream** e **OutputStream** para streams de bytes e das classes de abstratas **Reader** e **Writer** para streams de caracteres (texto).

A hierarquia de algumas destas classes é a seguinte:

Tabela 1 – Hierarquia de Classes de Sreams, Writer e Reader

OutputStream |_ ByteArrayOutputStream |_ **FileOutputStream** |_ FilterOutputStream | |_ BufferedOutputStream | |_ **DataOutputStream** |_ PipedOutputStream |_ **ObjectOutputStream**	**InputStream** |_ ByteArrayInputStream |_ **FileInputStream** |_ FilterInputStream | |_ BufferedInputStream | |_ **DataInputStream** |_ PipedInputStream |_ **ObjectInputStream**
Writer |_ **BufferedWriter** | |_ LineNumberWriter |_ **PrintWriter** |_ OutputStreamWriter | |_ **FileWriter** |_ PipedWriter |_ StringWriter |_ CharArrayWriter	**Reader** |_ **BufferedReader** | |_ LineNumberReader |_ InputStreamReader | |_ **FileReader** |_ PipedReader |_ StringReader |_ CharArrayReader

2 STREAMS DE CARACTERES

As subclasses de Writer tem que implementar os métodos definidos nesta classe abstrata, bem como as subclasses de Reader tem de implementar os definidos nela.

2.1 AS CLASSES FILEREADER E FILEWRITER

Construtores:

FileReader(File file); FileReader(String filename); ...

FileWriter(File file); FileWriter(String filename); ...

As classes FileReader e FileWriter permitem-nos, respectivamente, ler e escrever caracteres em objetos do tipo File.

No entanto, a leitura e a escrita de um caracter de cada vez geralmente não é a forma mais eficiente de manipular arquivos de texto. As classes BufferedReader e BufferedWriter possuem métodos para leitura e escrita linha-a-linha.

2.2 AS CLASSES BUFFEREDREADER E BUFFEREDWRITER

Construtores:

BufferedReader(Reader in)

BufferedWriter(Writer out)

Teste a classe abaixo:

```
import java.io.*;

public class ClasseTesteStreams01 {

	public static void main(String[] args){
		BufferedWriter bw;
		try{
			bw = new BufferedWriter(new FileWriter("teste1.txt"));
			bw.write("1");
			bw.newLine();
			bw.write("2");
			bw.flush();
			bw.close();
		} catch (IOException e) {
			System.out.println(e.getMessage());
		}

	}
}
```

Figura 1 – Classe de Teste de Stream

A principal vantagem da classe BufferedWriter é que esta realiza escritas otimizadas sobre streams (de caracteres) através de um mecanismo de "buffering". Os dados vão sendo armazenados em um buffer intermediário, sendo a escrita no destino apenas efetuada quando se atinge o máximo da capacidade do buffer.

Como vimos acima, o construtor da classe BufferedWriter recebe como argumento um objeto da classe Writer, o que significa que uma instância da classe BufferedWriter pode ser definida sobre qualquer subclasse da classe Writer, sempre que for necessário otimizar operações de escrita pouco eficientes.

Simetricamente, uma classe BufferedReader pode ser definida sobre qualquer subclasse da classe Reader.

ATIVIDADE: CRIE UMA CLASSE QUE LEIA O ARQUIVO teste1.txt

2.3 A CLASSE PRINTWRITER

Construtores:

PrintWriter(OutputStream out); PrintWriter(OutputStream out, boolean autoFlush);

PrintWriter(Writer out); PrintWriter(Writer out, boolean autoFlush);

As instâncias de PrintWriter podem ser criadas sobre qualquer classe de Writer e também sobre uma stream qualquer de bytes (subclasses de OutputStream).

Esta classe define os métodos print() e println(), que recebem como parâmetro um valor de **qualquer** tipo simples.

Teste a classe abaixo:

```
import java.io.*;

public class ClasseTestePrintWriter {

	public static void main(String[] args) {

		PrintWriter pw;
		try {
			pw = new PrintWriter(new FileWriter("teste2.txt"));
			pw.println(2.31);
			pw.println(false);
			pw.print("X");
			pw.flush();
			pw.close();
		} catch (IOException e){
			System.out.println(e.getMessage());
		}

	}

}
```

Figura 2 – Classe de Teste de PrintWriter

ATIVIDADE: CRIE UMA CLASSE QUE LEIA O ARQUIVO teste2.txt

3 STREAM DE BYTES

As subclasses de OutputStream e InputStream implementam stream de bytes. Os métodos abstratos definidos na classe OutputStream são idênticos aos de Writer com a diferença de que, em vez de caracteres, aceitam bytes.

3.1 AS CLASSES DATAOUTPUTSTREAM E DATAINPUTSTREAM

Estas classes são úteis para escrever/ler tipos primitivos de dados. Possuem métodos de leitura e escrita para cada um dos tipos primitivos de dados.

Construtores:

DataOutputStream(OutputStream out);

DataInputStream(InputStream in);

Teste a classe abaixo:

```
import java.io.*;

public class ClasseTesteOutputStream {

	public static void main(String[] args) {
		DataOutputStream os;
		try {
			os = new DataOutputStream(new FileOutputStream("teste3.dat"));
			os.writeDouble(2.335);
			os.writeInt(33);
			os.writeUTF("XPTO"); // Unicode Text Format
			os.flush();
			os.close();
		} catch (IOException e){
			System.out.println(e.getMessage());
		}
	}
}
```

Figura 3 – Classe de Teste Output Stream

ATIVIDADE: CRIE UMA CLASSE QUE LEIA O ARQUIVO teste3.dat

3.2 AS CLASSES OBJECTINPUTSTREAM E OBJECTOUTPUTSTREAM

Construtores:

ObjectInputStream(InputStream in);

ObjectOutputStream(OutputStream out);

Um ObjectOutputStream permite armazenar objetos através do método writeObject(), que implementa um algoritmo de serialização que garante que todas as referencias cruzadas existentes entre instâncias de diferentes classes serão repostas quando do processo de leitura destas mesmas instâncias.

Para que se possa gravar instâncias de uma determinada classe em um ObjectOutputStream é necessário que a classe implemente a interface *Serializable*. Além disso, todos os atributos desta classe também deverão ser serializáveis. Isto significa que todos os atributos devem, por sua vez, pertencer a classes serializáveis. Os tipos primitivos são, por definição, serializáveis, assim como arrays e as instâncias da classe String, Vector ou ArrayList.

ATIVIDADE:

Construa uma classe a sua escolha, definindo os atributos e construindo os getters e setters. Após, construa um programa de teste que instancie vários objetos desta classe e escreva estes objetos em um arquivo utilizando ObjectStreams. Após, crie um outro programa para ler os dados escritos no arquivo.

4 A CLASSE INETADDRESS

O programa abaixo permite obter o nome da máquina onde ele está sendo executado.

```
import java.net.*;

public class GetName {

	public static void main(String[] args) {
		InetAddress host = null;
		try {
			host = InetAddress.getLocalHost();
			System.out.println(host.getHostName());
		} catch (UnknownHostException e) {
			e.printStackTrace();
		}
	}

}
```

Figura 4 – Obter o nome da máquina local

O programa abaixo permite obter o endereço IP da máquina onde ele está sendo executado.

```
import java.net.*;

public class GetIP {

	public static void main(String[] args) {
		InetAddress host = null;
		try {
			host = InetAddress.getLocalHost();
			byte ip[] = host.getAddress();
			for (int i = 0; i < ip.length; i++){
				if (i > 0){
					System.out.print(".");
				}
				System.out.print(ip[i] & 0xff);
			}
			System.out.println();
		} catch (UnknownHostException e) {
			e.printStackTrace();
		}
	}

}
```

Figura 5 – Obter o IP da máquina local

Pretende-se, com o programa abaixo, que o usuário informe um determinado IP e, a partir dele, nos

seja informado o nome da máquina. **ATIVIDADE: COMPLETE O EXERCÍCIO**

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

public class IPToName {

        public static void main(String[] args) {

                String s = " ";
                char c;
                System.out.print("Informe um endereco IP:");
                try {
                        while ((c = (char)System.in.read()) != 10){
                                s += c;
                        }
                        s = s.trim();
                        InetAddress address = null;
                        <exercício>
                } catch (IOException e) {
                        System.out.println(e.getMessage());
                } catch (UnknownHostException e) {
                        System.out.println(e.getMessage());
                }

        }

}
```

Figura 6 – Obter o nome da máquina através de um IP

ATIVIDADE: CONSTRUA UM PROGRAMA QUE, DADO O NOME DE UMA MÁQUINA, NOS DIGA QUAL O IP CORRESPONDENTE