Jogos em Rede

Revisão – Streams e Arquivos

Streams

- Stream é o nome que se dá a seqüências de bytes associadas a um fluxo de origem ou destino.

- Em operações de entrada de dados podemos dizer que os bytes fluem da origem (teclado, rede, disco, etc) para a memória. No Java, as operações de entrada de dados são controladas por classes que herdam características da classe InputStream.

- Em operações de saída de dados podemos dizer que os bytes fluem da memória para o destino (monitor, impressora, disco, rede, etc). No Java, as operações de saída de dados são controladas por classes que herdam características da classe OutputStream.

- No Java foram criadas classes para controlar o fluxo de diferentes tipos de dados desde os tipos mais básicos até objetos mais complexos.

Streams – Classes Principais (java.io)

- **InputStream:** classe abstrata que define as operações de manipulação de streams a partir de uma fonte qualquer. Define as maneiras pelas quais o destino recebe um stream de bytes de uma dada origem. O método mais importante da classe InputStream é o método ***read()***.

- O método **read()** tem como objetivo realizar a leitura dos dados da origem para o destino. O método read() é bloqueante, ou seja, fica em modo bloqueado (aguardando) até que toda a entrada esteja disponível para leitura.

Streams – Classes Principais (java.io)

- **OutputStream:** classe abstrata que define as operações de manipulação de streams que serão realizadas em uma saída qualquer. Define as maneiras pelas quais uma origem escreve um stream de bytes para algum destino. O método mais importante da classe OutputStream é o método ***write()***.

- O método **write()** tem como objetivo realizar a escrita dos dados da origem para o destino.

- A origem e o destino podem ser diversos como, por exemplo, ByteArray em memória, Arquivo, Vídeo, Rede, etc. Um programa pode ler um arquivo do disco usando o método read() da classe InputStream e escrevê-lo em outro computador por meio da rede usando o método write() da classe OutputStream.

- Para usar as classes InputStream e OutputStream é necessário importar a biblioteca "java.io.*"

Streams – Algumas classes associadas

- **Exemplos de classe que herdam da classe InputStream:**
  - *FileInputStream*
  - *DataInputStream*
  - *ObjectInputStream*

- **Exemplos de classe que herdam da classe OutputStream:**
  - *FileOutputStream*
  - *DataOutputStream*
  - *ObjectOutputStream*

- *Todas as classes derivadas de InputStream possuem o método **read()** e todas as classes derivadas de OutputStream possuem o método **write().***

- *As classes podem ser usadas em conjunto de forma a tratar os dados de modo mais específico.*

Arquivos em Java

- Para a linguagem Java, arquivos são streams de dados.
- Para manipular arquivos em Java devemos usar as classes FileInputStream e FileOutputStream que associam um arquivo a um stream de entrada e saída, respectivamente.
- As quatro operações básicas em arquivos são:
  - **Abertura do arquivo:** realizada na criação de objetos FileInputStream e FileOutputStream.
  - **Leitura dos dados do arquivo:** realizada pelo método read() da classe FileInputStream.
  - **Escrita dos dados do arquivo:** realizada pelo método write() da classe FileOutputStream.
  - **Fechamento do arquivo:** realizado pelo método close() das classes FileInputStream e FileOutputStream.

FileInputStream e FileOutputStream

- ***FileInputStream:*** Mapeia um arquivo a um stream de entrada. *Pelo método read() é possível ler um conjunto de bytes do arquivo de origem.*

- ***FileOutputStream:*** Mapeia um arquivo a um stream de saída. Pelo método write() é possível escrever um conjunto de bytes no arquivo de destino.

- Alguns métodos:
  - read(), write(int b)
  - read(byte[] b), write(byte[] b)
  - read(byte[] b, int off, int len), write(byte[] b, int off, int len)
  - close(), close()

FileInputStream e FileOutputStream - Exemplo

- Criando um arquivo texto, escrevendo uma string no arquivo e lendo seu conteúdo.

```
import java.io.*;

public class FileStreamTest {
	public static void main( String args[] ) {
		String msgOut	= "Mensagem";
		byte[] msgIn	= new byte[100];

		try {
			OutputStream out = new FileOutputStream("teste.txt");  /*Cria um arquivo de nome teste.txt*/
			out.write(msgOut.getBytes());  /*escreve a mensagem contida em msgOut no arquivo*/
			out.close();  /*fecha o arquivo após seu uso*/


			InputStream in = new FileInputStream("teste.txt");    /*Abre o arquivo teste.txt para leitura*/
			in.read(msgIn);  /*le 100 bytes do arquivo e armazena no array de bytes msgIn*/
			System.out.println("Conteudo do arquivo: " + new String(msgIn)); /*Mostra os bytes lidos*/
			in.close();  /*fecha o arquivo após seu uso*/
		} catch (Exception e){  //FileNotFoundException, IOException
			System.err.println(e);
		}
	}
}
```

FileInputStream e FileOutputStream - Exemplo

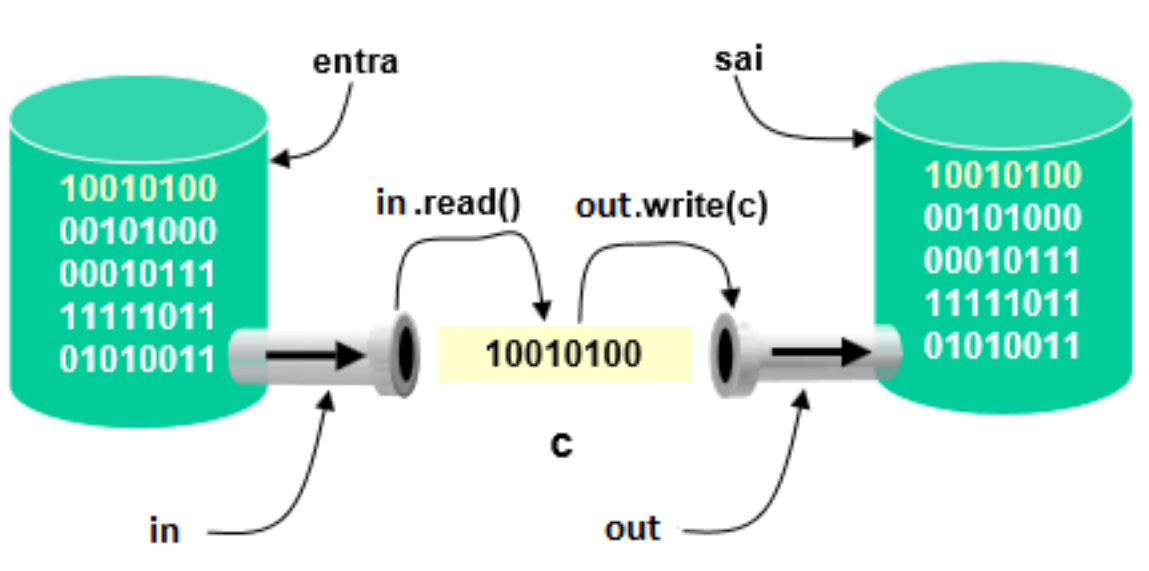

Imagem Fonte: Frederico Costa Guedes Pereira: Programação Orientada a Objetos - Streams, 2004

BufferedInputStream e BufferedOutputStream

- **BufferedInputStream e BufferedOutputStream:** Streams que são associados (concatenados) a outros Streams e fornecem um buffer para aumentar a performance das operações de E/S. Geralmente são associados a FileInputStream e FileOutputStream.

Concatenando Streams

- **Os Streams podem ser concatenados a outros Streams, para melhoria de performance ou para facilitar a manipulação de dados.**

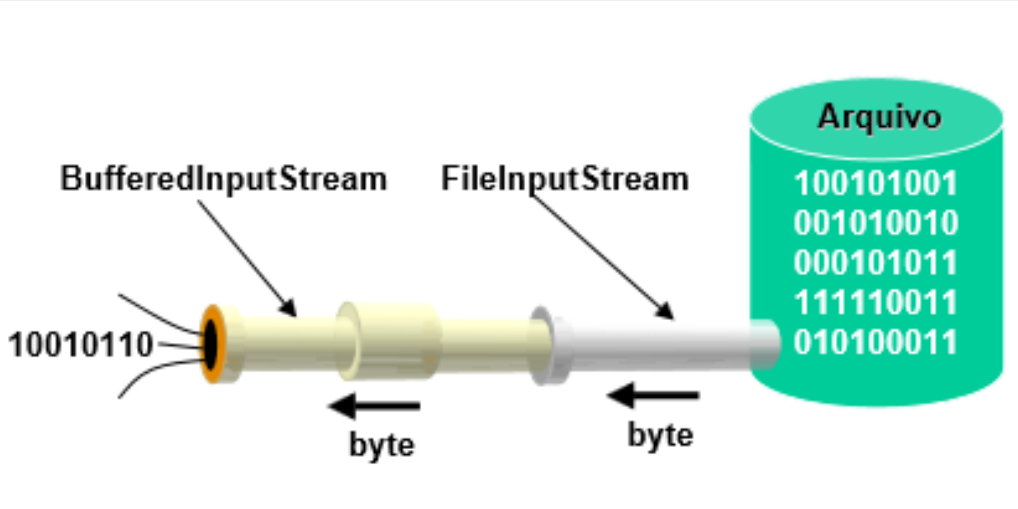

Imagem Fonte: Frederico Costa Guedes Pereira: Programação Orientada a Objetos - Streams, 2004

DataInputStream e DataOutputStream

- ***DataInputStream:*** Encapsula um stream de entrada em um stream que manipula dados. Permite a leitura de tipos básicos do java como byte, int, long, double, boolean a partir de um stream de entrada.
- ***DataOutputStream:*** Encapsula um stream de saída em um stream que manipula dados. Permite a escrita de tipos básicos do java como byte, int, long, double, boolean em um stream de saída.
- Alguns métodos:
  - readBoolean(), writeBoolean(boolean v)
  - readByte(), writeByte(int v)
  - readChar(), writeChar(int v)
  - readDouble(), writeDouble(double v)
  - readFloat(), writeFloat(float v)
  - readInt(), writeInt(int v)
  - readLong(), writeLong(long v)
  - readShort() , writeShort(int v)

DataInputStream e DataOutputStream - Exemplo

- Criando um arquivo, escrevendo um double no arquivo e lendo seu conteúdo.

```
import java.io.*;

public class DataStreamTest {
	public static void main( String args[] ) {
		double dataOut  = 2.589;
		double dataIn   = 0;

		try {
			OutputStream out = new FileOutputStream("data"); /*Cria um arquivo de nome teste.txt*/
			DataOutputStream o = new DataOutputStream(out); /*Associa o stream de saída com DataOutputStream*/
			o.writeDouble(dataOut); /*escreve o conteúdo de dataOut no arquivo*/
			o.close(); /*fecha o arquivo após seu uso*/

			InputStream in = new FileInputStream("data");          /*Abre o arquivo data para leitura*/
			DataInputStream i = new DataInputStream(in); /*Associa o stream de entrada com o tipo DataInputStream*/
			dataIn = i.readDouble(); /*lê o double contido no arquivo e armazena em dataIn*/
			System.out.println("Conteudo do arquivo: " + String.valueOf(dataIn)); /*Mostra os bytes lidos*/
			i.close();   /*fecha o arquivo após seu uso*/
		} catch (Exception e){  //FileNotFoundException, IOException
			System.err.println(e);
		}
	}
}
```

ObjectInputStream e ObjectOutputStream

- ***ObjectInputStream:*** Encapsula um stream de entrada em um stream que manipula objetos. Permite a leitura de objetos Java complexos a partir de um stream de entrada.

- ***ObjectOutputStream:***Encapsula um stream de saída em um stream que manipula objetos. Permite a escrita de objetos Java complexos em um stream de saída. Alguns métodos:
  - readBoolean(), writeBoolean(boolean v)
  - readByte(), writeByte(int v)
  - readChar(), writeChar(int v)
  - readDouble(), writeDouble(double v)
  - readFloat(), writeFloat(float v)
  - readInt(), writeInt(int v)
  - readLong(), writeLong(long v)
  - readShort() , writeShort(int v)
  - **readObject(), writeObject(Object obj)**

- **Obs: Todas as classes da hierarquia que definem o objeto escrito/lido precisam implementar a interface serializable.**

ObjectInputStream e ObjectOutputStream

- Para usar as classes ObjectInputStream e ObjectOutputStream todos os objetos a serem escritos ou lidos do stream de dados devem implementar a interface Serializable, ou seja, devem ser objetos serializáveis.
- Um objeto serializado é um objeto representado como uma sequência de bytes que incluem os dados do objeto assim como os tipos dos objetos e dados armazenados no objeto. Após o objeto serializado ser escrito no arquivo, ele poderá ser lido do arquivo e desserializado retornando sua forma original.
- No exemplo a seguir a classe Livro implementa a interface Serializable para que o objeto Livro torne-se serializável.
- Caso a classe livro fizesse referências a outras classes, essas outras classes também precisariam implementar a interface Serializable.
- Os tipos básicos Java como int, boolean, double e outros já implementam a interface Serializable.

ObjectInputStream e ObjectOutputStream - Exemplo

- A classe do objeto a ser escrito e lido do stream de dados.

```
import java.io.Serializable;          /*Importante*/

public class Livro implements Serializable{
        private String titulo;
        private String autor;
        private String editora;
        private int codigo;

        Livro(String titulo, String autor, String editora, int codigo) {
                this.titulo = titulo;
                this.autor = autor;
                this.editora = editora;
                this.codigo = codigo;
        }

        String getTitulo () {
                return titulo;
        }
        String getAutor () {
                return autor;
        }
        String getEditora () {
                return editora;
        }
        int getCodigo () {
                return codigo;
        }
}
```

ObjectInputStream e ObjectOutputStream - Exemplo

- Criando um arquivo, escrevendo um objeto Livro no arquivo e lendo o objeto Livro escrito no arquivo.

```
import java.io.*;

public class ObjectStreamTest {
      public static void main( String args[] ) {
            Livro objectOut    = new Livro("Minha vida", "Joao", "Nova", 1234) ;
            Livro objectIn     = null;

            try {
                  OutputStream out = new FileOutputStream("data"); /*Cria um arquivo de nome teste.txt*/
                  ObjectOutputStream o = new ObjectOutputStream(out); /*Associa o stream de saída com DataOutputStream*/
                  o.writeObject(objectOut); /*escreve o objeto no arquivo*/
                  o.close(); /*fecha o arquivo após seu uso*/

                  InputStream in = new FileInputStream("data");         /*Abre o arquivo data para leitura*/
                  ObjectInputStream i = new ObjectInputStream(in); /*Associa o stream de entrada com o tipo DataInputStream*/
                  objectIn = (Livro) i.readObject(); /*le o objeto do arquivo e armazena em objectIn*/
                  System.out.println("Titulo: " + objectIn.getTitulo() + ", Autor: " + objectIn.getAutor()); /*Mostra os bytes lidos*/
                  i.close();    /*fecha o arquivo após seu uso*/
            } catch (Exception e){  //FileNotFoundException, IOException
                  System.err.println(e);
            }
      }
}
```

Arquivos - Exercícios

1. (Exercício escrito II - ADO) – Explique qual a vantagem de usar um ObjectOutputStream no lugar de um OutputStream, quando precisamos escrever atributos de uma classe.

2. Faça um programa que escreva a mensagem "Novo recorde: Jogador 1" em um arquivo de nome "teste.txt".

3. Crie um arquivo no notepad que contenha uma mensagem qualquer menor que 100 bytes e faça um programa que leia o arquivo e mostre seu conteúdo na tela.

4. Faça um programa que leia um arquivo de tamanho variável e mostre seu conteúdo na tela. Use o comando System.out.print((char)b); para imprimir o byte lido na tela. Use o método read() para ler o arquivo byte a byte:

int read() : Lê um byte do stream de entrada e retorna -1 se o fim do arquivo foi encontrado.

Arquivos - Exercícios

5. Faça um programa que cria um arquivo de nome “configuracao” e grave nesse arquivo o valor de uma variável inteira e uma varíavel double. Faça um segundo programa que leia o arquivo e mostre seu conteúdo na tela.

6. Crie uma classe Jogador que contenha três propriedades: Nome (String), Pontos (int) e Fase (int). Faça um programa que crie uma instância do objeto jogador e salve o objeto em um arquivo chamado “SavedGame”. Crie outro programa que leia os dados escritos no arquivo e apresente-os na tela.

7. Altere o programa anterior de forma a salvar o status de até 10 jogadores.Faça um programa que escreva os dados dos jogadores em um arquivo chamado “SavedGame” e outro programa que leia os dados dos jogadorese apresente-os na tela.

8. (Exercício escrito III - ADO) - Faça um programa que leia um arquivo e uma string fornecida pelo usuário e retorne uma mensagem na tela dizendo se encontrou ou não a string fornecida no arquivo. Simule em papel.

Referências

- DEITEL, P. J., DEITEL, H. M., FURMANKIEWICZ, E., Java: como programar. 3 ed. - Porto Alegre: Bookman, 2001.
- Frederico Costa Guedes Pereira: Programação Orientada a Objetos - Streams, 2004