

## Streams

Uma “stream” é uma abstracção 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 abstractas

**InputStream** e **OutputStream** para ler/ escrever bytes e das classes abstractas

**Reader** e **Writer** para ler /escrever caracteres (texto).

Subclasses de **InputStream** e de **Reader** são fontes de dados

Subclasses de **OutputStream** e de **Writer** são destinos de dados

*Uma stream é uma sequência de itens de dados, geralmente bytes de 8 bits.*

**Hierarquia de classes →**

<http://www.di.ubi.pt/~pprata>

## **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**

## Input / Output de baixo nível

Para ler ou escrever num ficheiro temos de criar um objecto do tipo “stream” que depois é associado a um objecto do tipo File.

As classes **FileInputStream** e **FileOutputStream** definem objectos do tipo stream que nos permitem ler / escrever sequências de bytes em ficheiros.

### Exemplo:

```
import java.io.*;  
  
public class Um {  
  
    public static void main(String []args){  
  
        // criar o objecto do tipo File, ficheiro:  
  
        File ficheiro = new File( "teste.dat" );  
  
        try {  
  
            // associar um objecto do tipo FileOutputStream a um ficheiro  
  
            FileOutputStream os =  
  
                new FileOutputStream ( ficheiro);
```

<http://www.di.ubi.pt/~pprata>

```
// try {  
...  
    byte [] arrayBytes = { 10, 20, 30, 40, 50, 60, 70, 80 };  
    // escrever um array de bytes no ficheiro  
    os.write( arrayBytes);  
    // depois de concluir a escrita, devemos fechar a stream *  
    os.close();  
}  
  
// os métodos de acesso a ficheiros lançam exceções do  
// tipo IOException  
catch (IOException e){  
    System.out.println (e.getMessage());  
}  
}}
```

\* Os dados são previamente escritos em memória principal, num buffer de memória temporário (cache).

Quando o buffer fica cheio, são copiados para o disco. Se quando o programa termina o buffer não foi fechado, podem perder-se dados que ainda não foram copiados para disco.

<http://www.di.ubi.pt/~pprata>

Para ler os dados do ficheiro usamos o método **read** da classe **FileInputStream**

```
import java.io.*;

public class dois {

    public static void main(String []args){

        File ficheiro = new File( "teste.dat" );

        // podemos saber o tamanho do ficheiro

        int tamanhoFicheiro = (int)ficheiro.length();

        byte [] arrayBytes = new byte [tamanhoFicheiro];

        try {

            // criamos um objecto do tipo FileInputStream que
            associamos ao ficheiro

            FileInputStream is = new FileInputStream ( ficheiro );

            // ler os dados

            arrayBytes = is.read ();

            for (int i=0; i<tamanhoFicheiro; i++){

                System.out.println( arrayBytes[i]);

            } is.close() ...
```

<http://www.di.ubi.pt/~pprata>

## Input / Output de alto nível

A classe **DataOutputStream** permite-nos fazer o output de tipos primitivos de dados, convertendo-os em sequências de bytes.

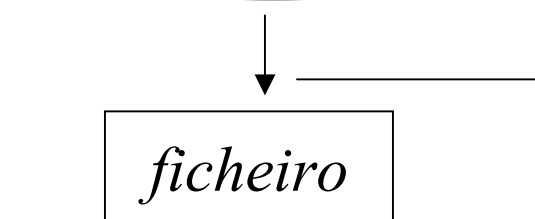
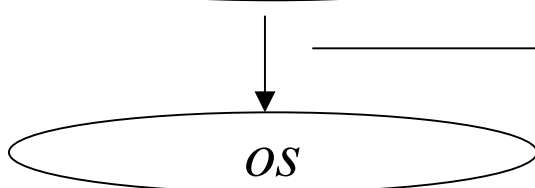
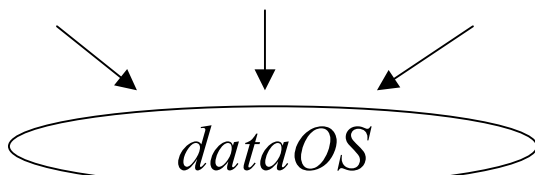
Esta classe fornece um acesso de mais alto nível para aceder a um ficheiro. Não está ligada directamente a um objecto do tipo file mas sim a um objecto do tipo **FileOutputStream**.

```
File ficheiro = new File("teste2.dat");
```

```
FileOutputStream os = new FileOutputStream( ficheiro);
```

```
DataOutputStream dataOS = new DataOutputStream( os );
```

writeFloat writeInt writeDouble <= métodos da classe  
DataOutputStream



os tipos primitivos são convertidos em sequências de bytes

os bytes são escritos no ficheiro

<http://www.di.ubi.pt/~pprata>

Exemplo:

```
File ficheiro = new File("teste2.dat");
try {
    FileOutputStream os = new FileOutputStream( ficheiro);
    DataOutputStream dataOS = new DataOutputStream( os );
    dataOS.writeInt ( 123456789);
    dataOS.writeLong(123456L);
    dataOS.writeFloat(777.3F);
    dataOS.writeDouble(88888888.9D);
    dataOS.writeChar('A');
    dataOS.writeBoolean (false);
    dataOS.close();
}
catch (IOException e){
    System.out.println(e.getMessage());
}
```

<http://www.di.ubi.pt/~pprata>

Para ler,

revertemos o processo, usando as classes **FileInputStream** e **DataInputStream**:

Os dados terão que ser lidos pela mesma ordem porque foram escritos.

```
File ficheiro = new File("teste2.dat");
```

```
FileInputStream is = new FileInputStream( ficheiro);
```

```
DataInputStream dataIS = new DataInputStream( is );
```

readFloat readInt readDouble <= métodos da classe  
DataInputStream

*dataIS*

*iss*

*ficheiro*

a sequências de bytes é convertida  
em valores de tipos primitivos

os bytes são lidos do ficheiro



<http://www.di.ubi.pt/~pprata>

...

```
File ficheiro = new File("teste2.dat");  
  
try {  
    FileInputStream is = new FileInputStream( ficheiro);  
    DataInputStream dataIS = new DataInputStream( is );  
    System.out.println(dataIS.readInt ());  
    System.out.println(dataIS.readLong());  
    System.out.println(dataIS.readFloat());  
    System.out.println(dataIS.readDouble());  
    System.out.println(dataIS.readChar());  
    System.out.println(dataIS.readBoolean ());  
    dataIS.close();  
}  
  
catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```

- a ordem de leitura tem de corresponder à ordem de escrita

<http://www.di.ubi.pt/~pprata>

## Ficheiros de Texto

Os dados podem ser armazenados em formato ASCII

Para gerar um ficheiro de texto podemos usar um objecto da classe **PrintWriter**

A classe tem dois métodos de output,

**void print ( ...)**

**void println (...)**

podendo ser o argumento um valor de qualquer um dos tipos primitivos.

Os métodos convertem o valor do argumento numa string e fazem o seu output. O construtor da classe `PrintWriter` recebe uma stream de output.

### **Exemplo:**

```
File ficheiro = new File("teste3.dat");
```

```
try {
```

```
FileOutputStream os = new FileOutputStream( ficheiro);
```

```
PrintWriter pw = new PrintWriter( os );
```

<http://www.di.ubi.pt/~pprata>

```
// escrever um int
    pw.println ( 123456789);
pw.println(123456L); um long
pw.println(777.3F); // um float
pw.println(88888888.9D); // um double
pw.println('A'); // um char
pw.println (false); // um boolean
pw.close();
} ..
```

## Ler um ficheiro de texto

Usamos as classes **FileReader** e **BufferedReader**

*Exemplo:*

```
File ficheiro = new File("teste3.dat");
try {
    FileReader fr = new FileReader( ficheiro);
    BufferedReader br = new BufferedReader(fr);
```

<http://www.di.ubi.pt/~pprata>

...

```
String linha;
```

```
linha = br.readLine();
```

```
int i = Integer.parseInt ( linha);
```

```
// se for só para escrever não é necessário converter
```

```
System.out.println(i);
```

```
linha = br.readLine();
```

```
long l = Long.parseLong (linha);
```

```
linha = br.readLine();
```

```
float f = Float.parseFloat ( linha);
```

```
linha = br.readLine();
```

```
double d = Double.parseDouble ( linha);
```

```
linha = br.readLine();
```

```
char c = linha.charAt(0);
```

```
linha = br.readLine();
```

```
boolean b = Boolean.getBoolean(linha);
```

```
br.close();
```

...

## Input / Output de objectos

Podemos também ler / escrever objectos de / num ficheiro, usando as classes **ObjectInputStream** e **ObjectOutputStream**

Uma **ObjectOutputStream** permite armazenar objectos através do método **writeObject()** que implementa um algoritmo de serialização que garante que todas as referências cruzadas existentes entre instâncias de diferentes classes serão repostas aquando do processo de leitura dessas mesmas instâncias.

Para que se possam gravar instâncias de uma determinada classe numa **ObjectOutputStream** é necessário que a classe implemente a interface *Serializable*.

Além disso, todas as variáveis dessa classe terão que ser também serializáveis. Isto significa que todas as variáveis de instância da classe devem por sua vez pertencer a classes serializáveis. Os tipos simples são por definição serializáveis, assim como o são os arrays e as instâncias das classes **String** e **Vector**.

<http://www.di.ubi.pt/~pprata>

Suponhamos a classe C7:

```
public class C7 implements Serializable {  
    private int n;  
    private String nome;  
    public C7(int n, String nome)  
    { ... }  
    public void setNumero (int i)  
    { ...}  
    public void setNome (String n)  
    {...}  
    public int getNumero()  
    {...}  
    public String getNome()  
    {...}  
    public String toString () {  
        return (n+" "+nome + "\n");  
    } }  
}
```

<http://www.di.ubi.pt/~pprata>

### **Para gravar objectos:**

- criamos uma `ObjectOutputStream`:

```
File f = new File ("teste4.dat");
```

```
FileOutputStream os = new FileOutputStream (f);
```

```
ObjectOutputStream oOS = new ObjectOutputStream(os);
```

### **Para escrever um objecto do tipo C7:**

- criamos o objecto

```
C7 o1 = new C7 (1, "XPTO");
```

- usamos o método `void writeObject(Object)`

```
oOS.writeObject( o1 );
```

### **Obs.**

Diferentes tipos de objectos podem ser escritos no mesmo ficheiro.

<http://www.di.ubi.pt/~pprata>

Exemplo:

...

```
int i;
```

```
C7 objectoC7;
```

```
File f = new File ("teste4.dat");
```

```
try {
```

```
    FileOutputStream os = new FileOutputStream (f);
```

```
    ObjectOutputStream oOS = new
```

```
        ObjectOutputStream(os);
```

```
    for (i=0; i<100; i++){
```

```
        objectoC7 = new C7( i, "XPTO da Silva");
```

```
        oOS.writeObject( objectoC7 );
```

```
    }
```

```
    oOS.close();
```

```
}
```

```
catch (IOException e){
```

```
    System.out.println(e.getMessage());
```

```
}
```



<http://www.di.ubi.pt/~pprata>

Para ler objectos, usamos as classes **FileInputStream** e **ObjectInputStream**

...

```
int i;
C7 objectoC7;
File f = new File ("teste4.dat");
try {
    FileInputStream is = new FileInputStream (f);
    ObjectInputStream oIS = new
        ObjectInputStream(is);
    for (i=0; i<100; i++){
        // lemos o objecto com o método
        //Object readObject(void);
        objectoC7 = (C7) oIS.readObject(); /*
        // * temos de converter para o tipo do objecto a ler
        System.out.println ( objectoC7.toString() );
    }
    oIS.close();
}
```

<http://www.di.ubi.pt/~pprata>

...

```
catch (IOException e){
```

```
    System.out.println(e.getMessage());
```

```
}
```

```
// o método readObject, além da exceção anterior, pode
```

```
// também gerar uma instância de ClassNotFoundException
```

```
catch (ClassNotFoundException e){
```

```
    System.out.println("Classe não existente - " +
```

```
                        e.getMessage());
```

```
}
```