

PG II

Programação Orientada aos Objectos em Java

Java IO System

- *Streams* de bytes e *Streams* de caracteres
- Escrita e leitura de dados num ficheiro de texto
- Filtros
- PushBackReader
- System.in e System.out
- pg2.io.In e pg2.io.Out
- PrintStream
- Output na Consola
- Character-Stream <versus> Bytes-Stream
- Serialização

Streams de bytes e Streams de caracteres

- **Stream Java** – «sequencia, fluxo, ...» de bytes ou caracteres.
- Em **Java**, as classes que representam o conjunto de *streams* disponíveis, estão definidas no package **java.io**.
- Genericamente, as *streams* são distinguidas pelo **tipo de dados** e **device** (destino/fonte) que lhe estão, ou não, associadas:



TxtEditor - Escrita e leitura de dados num ficheiro de texto



- o class java.awt.[MenuComponent](#) (implements java
- o class java.awt.[MenuBar](#) (implements javax
- o class java.awt.[MenuItem](#) (implements javax
- o class java.awt.[CheckboxMenuItem](#)
- o class java.awt.[Menu](#) (implements jav
- o class java.awt.[PopupMenu](#)
- o class java.awt.[MenuShortcut](#) (implements java.io.

```

TxtEditor.java*

package pg2.aula08;

import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class TxtEditor extends Frame {
    /***** Variáveis de Instancia *****/
    TextArea ta;

    /***** Construtor *****/
    public TxtEditor() {
        super("TxtEditor");
        ta = new TextArea ();
        this.add(ta);

        /**** MenuBar ***/
        MenuBar mb = new MenuBar();
        Menu mn = new Menu("File");
        MenuItem mi_save = new MenuItem ("Save", new MenuShortcut (KeyEvent.VK_S));
        MenuItem mi_load = new MenuItem ("Load", new MenuShortcut (KeyEvent.VK_L));
        mi_save.addActionListener(new SaveFile());
        mi_load.addActionListener(new LoadFile());
        mn.add(mi_save);
        mn.add(mi_load);
        mb.add(mn);
        this.setMenuBar (mb);

        this.addWindowListener( new WindowAdapter() {
            public void windowClosing (WindowEvent e){
                System.exit(0);
            }
        });
        this.pack();
        this.show();
    }
}
    
```

Estas classes são Listeners cuja definição é apresentada nas páginas seguintes

TxtEditor - Escrita e leitura de dados num ficheiro de texto ...

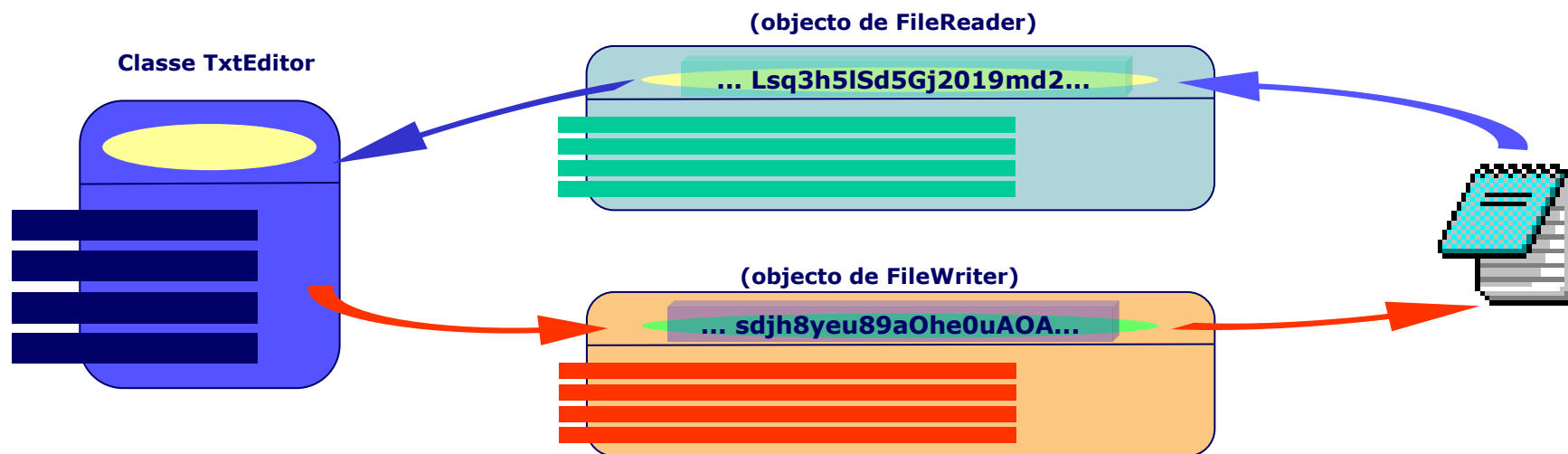
Para ler e escrever para um ficheiro de texto teremos que usar duas *streams* distintas: uma de saída e outra para entrada de dados. A escolha do tipo de *streams* é feito em função do:

- 1º) Tipo de dados;
- 2º) Destino/fonte dos dados.

A resposta a cada um dos critérios anteriores é respectivamente:

- 1º) Caracteres → Readers e Writers;
- 2º) File.

Logo, as *streams* a usar para **escrita** e **leitura** de dados na aplicação **TxtEditor**, serão respectivamente: **FileWriter** e **FileReader**:



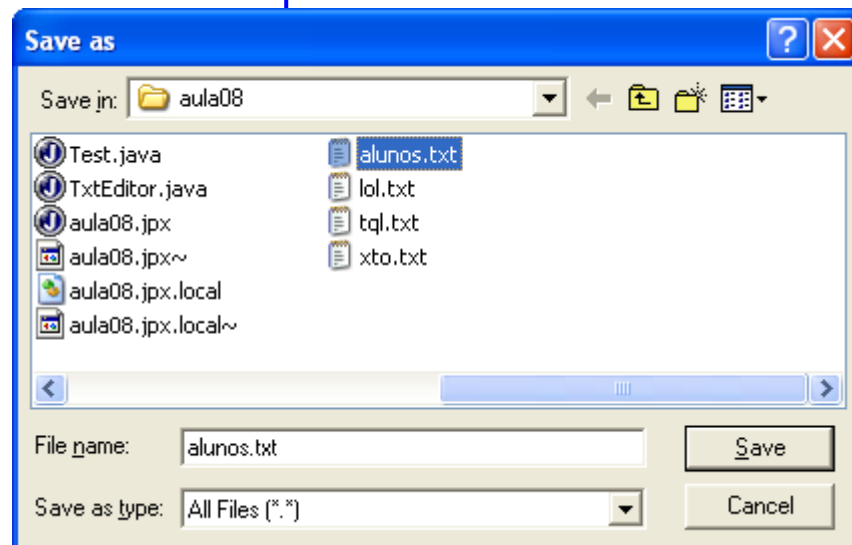
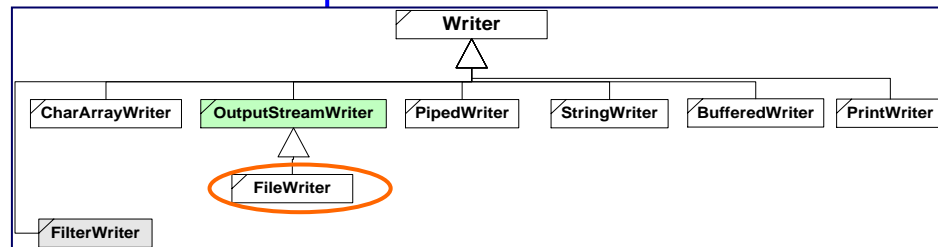
TxtEditor - Escrita e leitura de dados num ficheiro de texto ...

```

TxtEditor.java

/***** Inner Class SaveFile *****/
class SaveFile implements ActionListener {
    public void actionPerformed (ActionEvent e){
        FileWriter out = null;
        FileDialog fd = new FileDialog (TxtEditor.this, "Save as", FileDialog.SAVE);
        fd.setVisible(true);
        StringBuffer f = new StringBuffer();
        if (fd.getDirectory() != null) f.append(fd.getDirectory());
        if (fd.getFile() != null) f.append(fd.getFile());
        try {
            out = new FileWriter (f.toString());
            out.write(ta.getText());
            out.flush();
        }
        catch (IOException ev) {
            new MsgBox (TxtEditor.this, "Error", true, "Invalid file name " + f + " - File not saved");
        }
    }
}

/***** Inner Class Message Box (classe auxiliar) *****/
class MsgBox extends Dialog {
    public MsgBox(Frame owner, String title, boolean modal, String msg){
        super(owner, title, modal);
        Label l = new Label(msg);
        l.setFont(new Font("Arial", Font.BOLD, 14));
        add(l);
        addWindowListener( new WindowAdapter() {
            public void windowClosing (WindowEvent e){
                MsgBox.this.dispose();
            }
        });
        this.setLocation(owner.getX() + owner.getWidth()/2,
                        owner.getY() + owner.getHeight()/2);
        pack();
        show();
    }
}
    
```



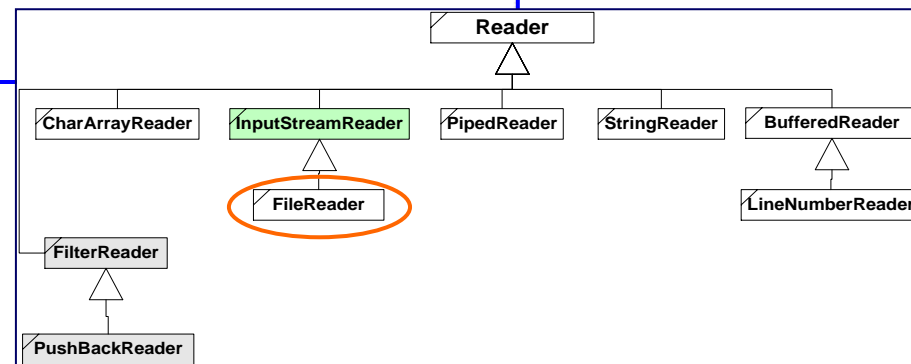
TxtEditor - Escrita e leitura de dados num ficheiro de texto ...

TxtEditor.java*

```

/***** Inner Class LoadFile *****/
class LoadFile implements ActionListener {
    public void actionPerformed (ActionEvent e){
        FileReader in = null;
        File fl = null;
        FileDialog fd = new FileDialog (TxtEditor.this, "Load file", FileDialog.LOAD);
        fd.setVisible(true);
        StringBuffer f = new StringBuffer();
        if (fd.getDirectory() != null) f.append(fd.getDirectory());
        if (fd.getFile() != null) f.append(fd.getFile());
        try {
            fl = new File(f.toString());
            in = new FileReader (fl);
            char [] aux = new char[(int)fl.length()];
            in.read(aux);
            ta.setText (String.valueOf(aux));
        }
        catch(IOException ev) {
            new MsgBox (TxtEditor.this,"Error",true, "Invalid file name " + f + " - File not loaded");
            return;
        }
    }
}

```



O **3º Critério** na escolha das *streams*, será a forma ou modo, como queremos ler/escrever os dados.

No exemplo anterior, os dados são lidos do ficheiro numa única instrução:

- `in.read(aux) ;`

(sendo aux o array de caracteres destino)

Se existisse a necessidade de ler o conteúdo do ficheiro linha a linha, o método anteriormente usado não seria o indicado, mas sim, um método que implementasse uma funcionalidade semelhante a um `readLine()`.

Contudo, embora um objecto da classe **FileReader** sirva os nossos interesses em termos do **tipo de dados transmitidos** e do **device manipulado**, já não satisfaz os objectivos no que diz respeito ao comportamento, porque não disponibiliza nenhum método com as características de um `readLine()`.

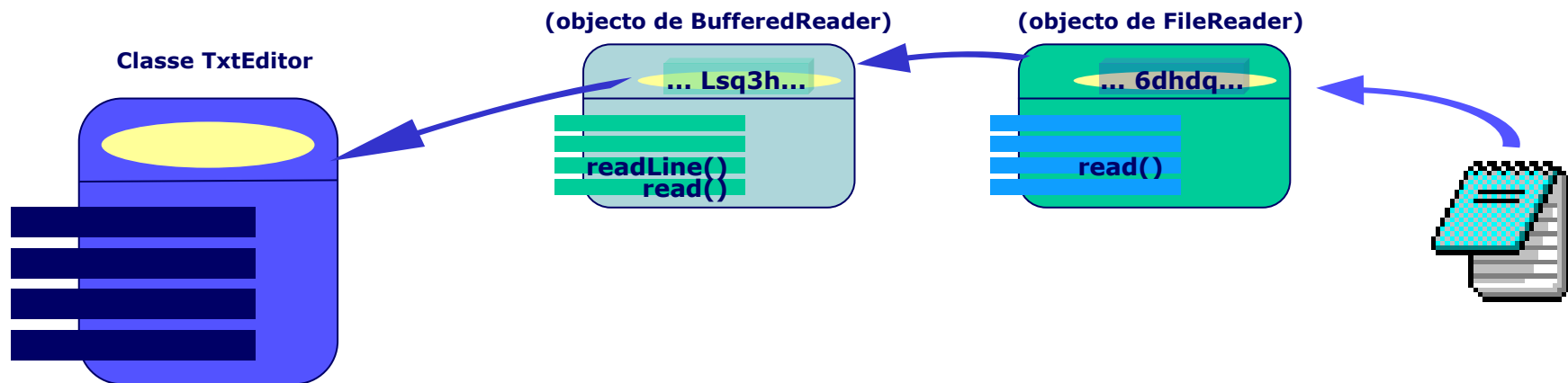
➤ A única Classe derivada de Reader, com este método é **BufferedReader**.

Analisando o seu construtor (`BufferedReader(Reader in)`), verificamos que não conseguimos instanciar esta *stream* associando-lhe um *device* (file, pipe, etc...) para leitura dos dados, mas apenas um outro **Reader**.

Assim, para concretizar o objectivo de ler **caracteres**, de um **ficheiro, linha a linha**, a solução passa por juntar a funcionalidade de um **BufferedReader** e de um **FileReader**.

Este mecanismo é vulgarmente designado por “**aninhamento**” de *streams* e no caso anterior é implementado da seguinte forma:

```
...  
String;  
BufferedReader bf = new BufferedReader (new FileReader (f1));  
    while( (String s = bf.readLine()) != null)  
        ...;
```



(Outra perspectiva, é visualizar as streams como peças de lego)

PushBackReader

O PushBackReader também é um filtro que pode ser aplicado juntamente com outro Reader, acrescentando os métodos:

- `unread(char[] cbuf)`
- `void unread(char[] cbuf, int off, int len)`
- `void unread(int c)`

No caso da classe `pg2.io.In` o método `unread()` é utilizado para a implementação do `eof()`:

```
public boolean eof() throws InException {
    try {
        if (!eofFlag) {
            int c = read();
            if (c != EOF) { unread(c); return false; }
        }
        return true;
    }
    catch (IOException e) {
        throw new InException("Error: read char.");
    }
}
```

System.in e System.out



Java 2 Platform SE v1.3.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address D:\Java\docs\api\index.html Go Links

Field Detail

in

public static final [InputStream](#) in

The "standard" input stream. This stream is already open and ready to supply input data. Typically this stream corresponds to keyboard input or another input source specified by the host environment or user.

out

public static final [PrintStream](#) out

The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

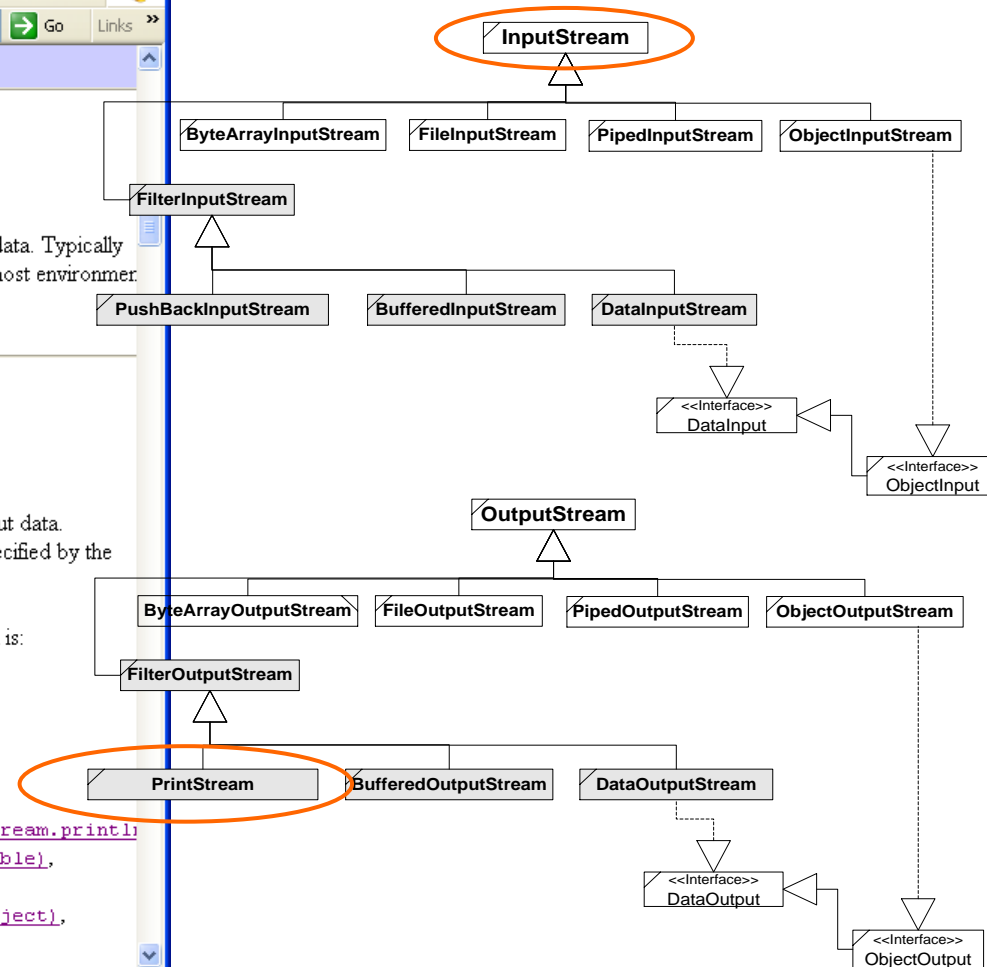
For simple stand-alone Java applications, a typical way to write a line of output data is:

```
System.out.println(data)
```

See the `println` methods in class `PrintStream`.

See Also:

- `PrintStream.println()`, `PrintStream.println(boolean)`, `PrintStream.println(char)`, `PrintStream.println(char[])`, `PrintStream.println(double)`, `PrintStream.println(float)`, `PrintStream.println(int)`, `PrintStream.println(long)`, `PrintStream.println(java.lang.Object)`, `PrintStream.println(java.lang.String)`



PrintStream

A classe `PrintStream` acrescenta ainda os seguintes métodos:

- `void print(boolean b)`
- `void print(char c)`
- `void print(char[] s)`
- `void print(double d)`
- `void print(float f)`
- `void print(int i)`
- `void print(long l)`
- `void print(Object obj)`
- `void print(String s)`
- `void println()`
- `void println(boolean x)`
- `void println(char x)`
- `void println(char[] x)`
- `void println(double x)`
- `void println(float x)`
- `void println(int x)`
- `void println(long x)`
- `void println(Object x)`
- `void println(String x)`

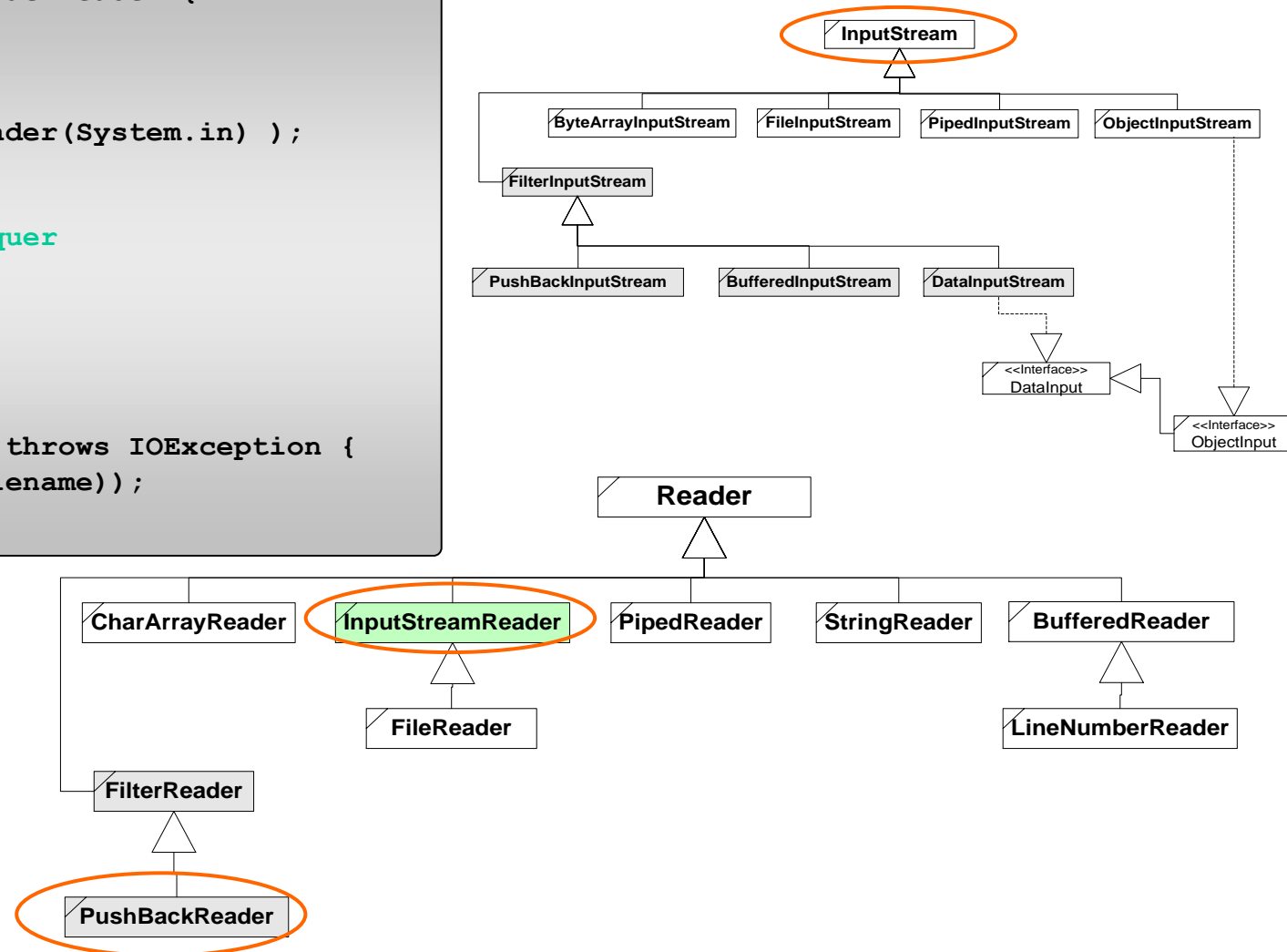
```

public class In extends PushbackReader {
    ...
    //Leitura da Consola
    public In() {
        super( new InputStreamReader(System.in) );
    }

    //Leitura de um reader qualquer
    public In(Reader r) {
        super(r);
    }

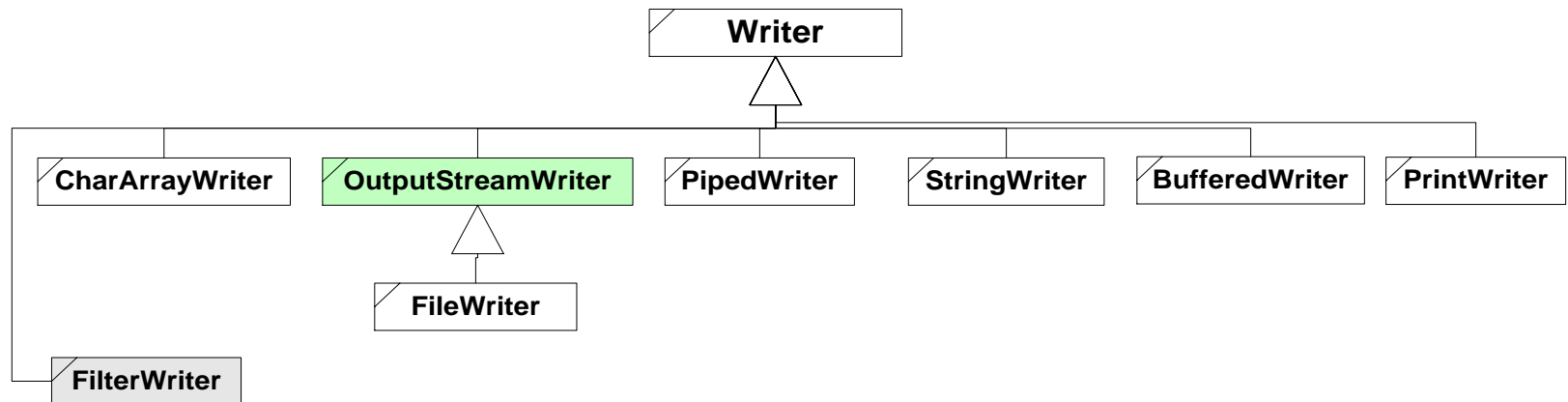
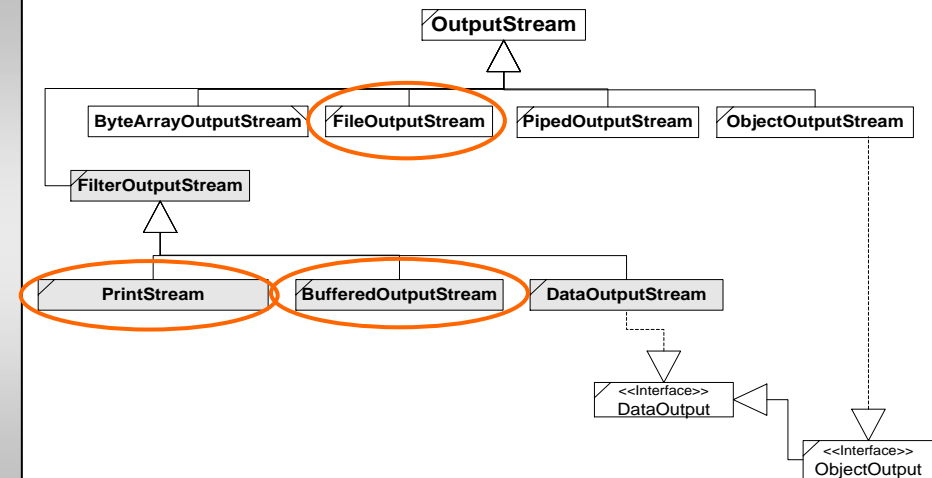
    //Leitura de um Ficheiro
    public In(String filename) throws IOException {
        super(new FileReader(filename));
    }
}

```



```
public class Out extends PrintStream {
    ...
    //Escrita na Consola
    public Out() {
        super (new BufferedOutputStream(System.out));
    }

    //Escrita num Ficheiro
    public Out(String n) throws FileNotFoundException {
        super (new BufferedOutputStream(
            new FileOutputStream(n)));
    }
}
```



Output na Consola

Porque é que quando enviamos uma String com caracteres portugueses para a consola, estes não são correctamente apresentados?

Analisemos o seguinte código:

```

COut.java*
package pg2.aula08;

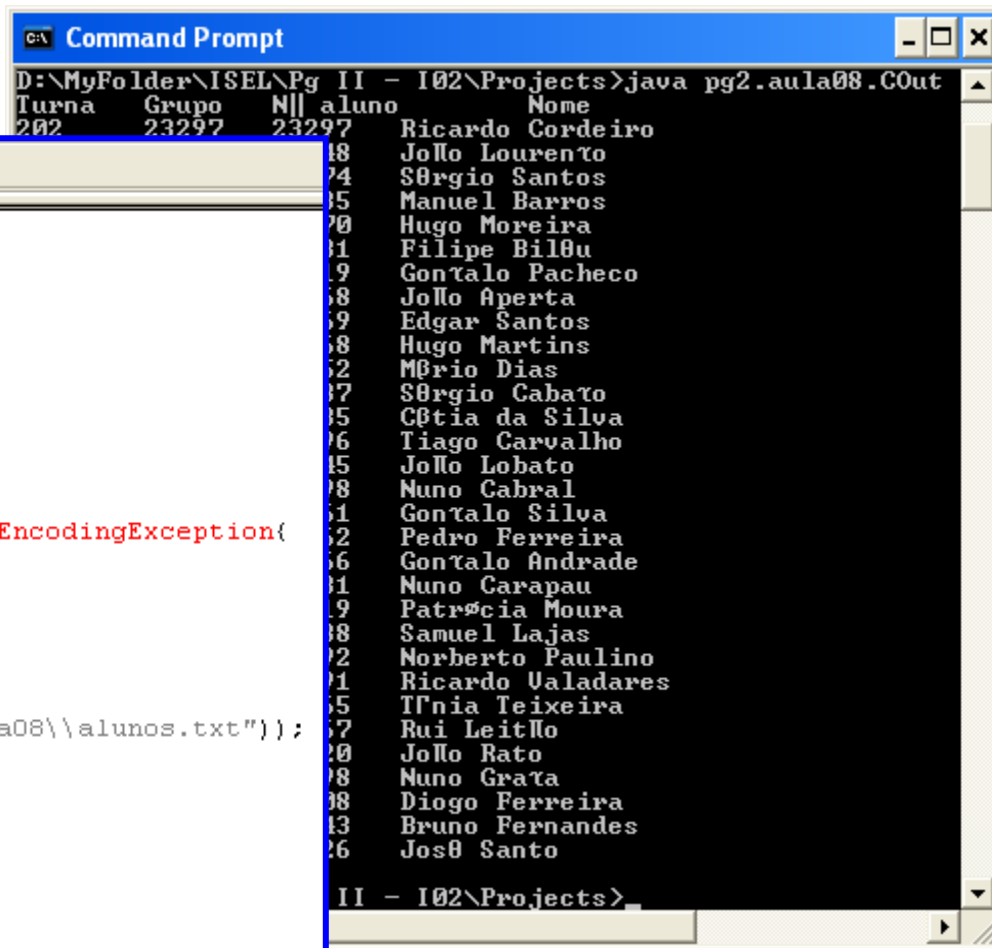
import java.io.*;

public class COut extends PrintWriter {
    public COut() throws UnsupportedEncodingException{
        super (new OutputStreamWriter (System.out), true);
    }

    public static void main (String args [])throws UnsupportedEncodingException{
        COut out = new COut ();

        try{
            BufferedReader in = new BufferedReader (
                new FileReader(
                    "D:\\MyFolder\\ISEL\\Pg II - I02\\Projects\\pg2\\aula08\\alunos.txt"));
            String s = null;
            while( (s=in.readLine()) != null)
                out.println(s);
        }
        catch (IOException e) {
            out.println(e);
        }
    }
}

```



```

C:\ Command Prompt
D:\MyFolder\ISEL\Pg II - I02\Projects>java pg2.aula08.COut
Turna Grupo N|| aluno Nome
202 23297 23297 Ricardo Cordeiro
18 Jollo Louren7o
24 S0rgio Santos
25 Manuel Barros
20 Hugo Moreira
21 Filipe Bil8u
19 Gon7alo Pacheco
18 Jollo Aperta
19 Edgar Santos
18 Hugo Martins
12 M8rio Dias
17 S0rgio Cabato
25 C8tia da Silva
26 Tiago Carvalho
15 Jollo Lobato
18 Nuno Cabral
21 Gon7alo Silva
12 Pedro Ferreira
16 Gon7alo Andrade
21 Nuno Carapau
19 Patr7cia Moura
18 Samuel Lajas
12 Norberto Paulino
21 Ricardo Valadares
15 T7nia Teixeira
17 Rui Leit7lo
20 Jollo Rato
18 Nuno Gra7a
18 Diogo Ferreira
13 Bruno Fernandes
26 Jos8 Santo
II - I02\Projects>

```

Output na Consola ...

Alterando explicitamente a norma de codificação dos caracteres para "MS-DOS Portuguese":

- `super (new OutputStreamWriter (System.out, "Cp860"), true);`

<http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html>

Extended Encoding Set (contained in lib/charsets.jar)

Supported by java.io and java.lang APIs

Canonical Name	Description
Big5	Big5, Traditional Chinese
Big5_HKSCS	Big5 with Hong Kong extensions, Traditional Chinese
Big5_Solaris	Big5 with seven additional Hanzi ideograph character mappings for the Solaris zh_TW.Big5 locale
Cp037	USA, Canada (Bilingual, French), Netherlands, Portugal, Brazil, Australia
Cp273	IBM Austria, Germany
Cp277	IBM Denmark, Norway
Cp278	IBM Finland, Sweden
Cp280	IBM Italy
Cp284	IBM Catalan/Spain, Spanish Latin America
Cp285	IBM United Kingdom, Ireland
Cp297	IBM France
Cp420	IBM Arabic
Cp424	IBM Hebrew
Cp437	MS-DOS United States, Australia, New Zealand, South Africa
Cp500	EBCDIC 500V1
Cp737	PC Greek
Cp775	PC Baltic
Cp838	IBM Thailand extended SBCS
Cp850	MS-DOS Latin-1
Cp852	MS-DOS Latin-2
Cp855	IBM Cyrillic
Cp856	IBM Hebrew
Cp857	IBM Turkish
Cp858	Variant of Cp850 with Euro character
Cp860	MS-DOS Portuguese
Cp861	MS-DOS Icelandic

```

C:\ Command Prompt
D:\MyFolder\ISEL\Pg II - I02\Projects>java pg2.aula08.COut
Turna Grupo Nº aluno Nome
202 23297 23297 Ricardo Cordeiro
202 23297 25948 João Lourenço
203 24035 24174 Sérgio Santos
202 24170 24035 Manuel Barros
202 24170 24170 Hugo Moreira
203 24969 24631 Filipe Biléu
202 25068 25219 Gonçalo Pacheco
202 25068 25168 João Aperta
202 25068 24969 Edgar Santos
202 25068 25068 Hugo Martins
202 25135 25252 Mário Dias
202 25135 25087 Sérgio Cabaço
202 25135 25135 Cátia da Silva
203 25245 26196 Tiago Carvalho
202 25261 25245 João Lobato
202 25261 25298 Nuno Cabral
202 25261 25261 Gonçalo Silva
202 25261 25262 Pedro Ferreira
202 25261 25366 Gonçalo Andrade
202 26131 26231 Nuno Carapau
202 26131 26419 Patrícia Moura
202 26131 26388 Samuel Lajas
203 26191 26492 Norberto Paulino
202 26298 26191 Ricardo Valadares
202 26298 26265 Tânia Teixeira
202 26298 26767 Rui Leitão
202 26298 26420 João Rato
202 26298 26298 Nuno Graça
203 26443 26608 Diogo Ferreira
203 26443 26443 Bruno Fernandes
203 26443 26726 José Santo

D:\MyFolder\ISEL\Pg II - I02\Projects>

```

Character-Stream <> Bytes-Stream



Character streams (Introduced in JDK(TM) 1.1) - Microsoft Internet Explorer		
File Edit View Favorites Tools Help		
Address D:\Java\docs\guide\io\io.html Go Links >>		
Character-stream class	Description	Corresponding byte class
Reader	Abstract class for character-input streams	InputStream
BufferedReader	Buffers input, parses lines	BufferedInputStream
LineNumberReader	Keeps track of line numbers	LineNumberInputStream
CharArrayReader	Reads from a character array	ByteArrayInputStream
InputStreamReader	Translates a byte stream into a character stream	(none)
FileReader	Translates bytes from a file into a character stream	FileInputStream
FilterReader	Abstract class for filtered character input	FilterInputStream
PushbackReader	Allows characters to be pushed back	PushbackInputStream
PipedReader	Reads from a PipedWriter	PipedInputStream
StringReader	Reads from a String	StringBufferInputStream
 Writer	 Abstract class for character-output streams	 OutputStream
BufferedWriter	Buffers output, uses platform's line separator	BufferedOutputStream
CharArrayWriter	Writes to a character array	ByteArrayOutputStream
FilterWriter	Abstract class for filtered character output	FilterOutputStream
OutputStreamWriter	Translates a character stream into a byte stream	(none)
FileWriter	Translates a character stream into a byte file	FileOutputStream
PrintWriter	Prints values and objects to a Writer	PrintStream
PipedWriter	Writes to a PipedReader	PipedOutputStream
StringWriter	Writes to a String	(none)

ObjectOutputStream é uma *stream* para armazenamento de objectos, arrays e valores de tipos primitivos, através do método `writeObject()`.

Este mecanismo é vulgarmente designado de serialização e para que seja possível executá-lo sobre uma determinada instancia, é necessário que a classe desse objecto seja serializável, isto é, que implemente a interface **Serializable**.

A interface **Serializable** é uma *marker interface*, ou seja, não impõe a definição de qualquer método à classe que a implementa. Como tal, para que uma classe possa implementar a interface **Serializable**, basta que as suas variáveis de instância sejam objectos de classes também serializáveis.

Os atributos **static** de uma classe não são serializados.

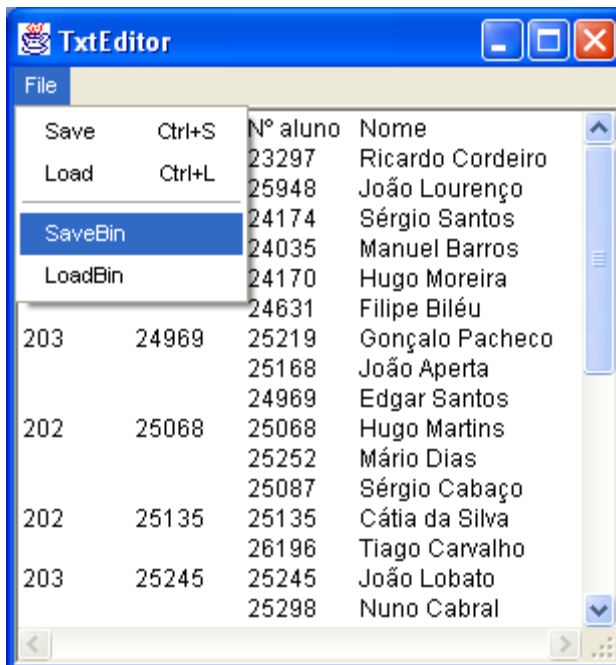
Além de **ObjectOutputStream** e **ObjectInputStream** para a serialização de objectos, existem ainda as seguintes *streams* para a serialização de dados:

- `DataOutputStream` `writeInt(int)`, `writeFloat(float)`, ...
- `DataInputStream` `int readInt()`, `float readFloat()`, ...

NOTA:

Uma única chamada a `writeObject()` pode escrever um grafo de objectos ligados por referências entre si.

Serialização no TxtEditor



```

/***** Inner Class SaveFileBin *****/
class SaveFileBin implements ActionListener{
    public void actionPerformed (ActionEvent e){
        FileDialog fd = new FileDialog (TxtEditor.this, "Save as", FileDialog.SAVE);
        fd.setVisible(true);
        StringBuffer f = new StringBuffer();
        if (fd.getDirectory() != null) f.append(fd.getDirectory());
        if (fd.getFile() != null) f.append(fd.getFile());
        try{
            ObjectOutputStream out = new ObjectOutputStream (new FileOutputStream (f.toString()));
            out.writeObject(ta.getText());
            out.flush();
        }
        catch(IOException ev) {
            new MsgBox (TxtEditor.this,"Error",true, "Invalid file name " + f + " - File not saved");
        }
    }
}

/***** Inner Class LoadFileBin *****/
class LoadFileBin implements ActionListener{
    public void actionPerformed (ActionEvent e){
        FileDialog fd = new FileDialog (TxtEditor.this, "Load File Bin", FileDialog.LOAD );
        fd.setVisible(true);
        StringBuffer f = new StringBuffer();
        if (fd.getDirectory() != null) f.append(fd.getDirectory());
        if (fd.getFile() != null) f.append(fd.getFile());
        try{
            ObjectInputStream in = new ObjectInputStream (new FileInputStream (f.toString()));
            ta.setText((String) in.readObject());
        }
        catch(IOException ev) {
            new MsgBox (TxtEditor.this,"Error",true, "Invalid file name " + f + " - File not saved");
        }
        catch(ClassNotFoundException ev) {
            new MsgBox (TxtEditor.this,"Error",true, ev + " - File not saved");
        }
    }
}

```