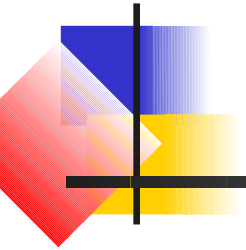
Programming with Java

Streams and Serialisation

P. Bakowski

bako@ieee.org



The streams

The data stream is the interface through which Java handles the communication with the user (standard input/output), with the file system and the network.

In a stream, the data is written and read sequentially.

The stream can be accessed by units (blocks) of arbitrary size: byte, word, line, etc..

To improve the performance of communication the streams can use a buffer class (`Buffer`).

The classes related to streams are grouped in the package `java.io`.

The Streams

The abstract class Reader contains methods **`read()`** for reading characters and arrays of characters:

```
int read()

int read(char cbuf[])

int read(char cbuf[], int offset, int length)
```

The abstract class Writer contains methods **`write()`** for writing characters and arrays of characters:

```
int write(int c)

int write(char cbuf[])

int write(char cbuf[], int offset, int length)
```

The streams : InputStreamReader

```
import java.io.* ;
public class MyReadStream {
  public static void main(String[] args) throws IOException{
  int nChread;
  Reader stdIn = new InputStreamReader(System.in);
  char[] userInput = new char[256];
  while((nChread=stdIn.read(userInput))!=-1)
        System.out.println(userInput);
  }
}
```

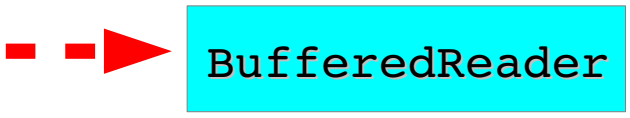
Unbuffered character reading

userInput[256]

The streams : BufferedReader

```
import java.io.* ;
public class MyReadBuffered{
  public static void main(String[] args) throws IOException{
  BufferedReader stdIn = new BufferedReader(
                         new InputStreamReader(System.in));
  String userInput;
  while((userInput=stdIn.readLine())!=null)
        System.out.println(userInput);
  }
}
```

Buffered character reading



The streams: System.in & System.out

```
System.in. …  // method
int nor;
while((nor=System.in.read())!='.')..
```

```
System.out. …  // method

String userInput ;
System.out.println(userInput);
```

Streams of bytes

The abstract class `InputStream` contains

```
int read();
int read(byte cbuf[]);
int read(byte cbuf[], int offset, int length);
```

The abstract class `OutputStream` contains:

```
int write();
int write(byte cbuf[]);
int write(byte cbuf[], int offset, int length);
```

Strings (characters) and bytes

memory:

```
CharArrayReader
CharArrayWriter

StringReader
StringWriter

StringBufferInputStream
```

```
ByteArrayInputStream
ByteArrayOutputStream
```

files:

```
FileReader
FileWriter
```

```
FileInputStream
FileOutputStream
```

Memory : read/write

```
String str = "XYZ"; // a string to display on screen
StringReader reader = new StringReader(str);
int ncr;
while(( ncr=reader.read()) != -1)
// reading character by character
{
System.out.println((char)ncl);
// writing character by character
}
```

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
int nor;
while((nor = System.in.read()) !=-1) { out.write(nor);}
// the size of buffer out inreases automatically
byte[] btab = out.toByteArray();  // conversion
out.reset(); // erasing the buffer
```

Files : read/write

Stream classes on files are: **`FileInputStream`**, **`FileOutputStream`**, **`FileReader`**, **`FileWriter`** and **`RandomAccessFile`**.

`FileInputStream` and `FileOutputStream` classes allow you to create byte stream.

`FileReader` and `FileWriter` classes are derived classes (for Files) of `InputStreamReader` and `OutputStreamWriter`.

The `RandomAccessFile` class to move the file pointer to the required position.

Constructors of file streams take as parameters the file name or the name of an object of the class **`File`**.

```
FileReader in = new FileReader("fichier.txt");

FileReader in = new FileReader(new File("fichier.txt"));
```

Text files : CopyTextFile

```
import java.io.*;
public class CopyTextFile {
public static void main(String[] args) throws IOException
{
File inputFile  = new File(args[0]);
File outputFile = new File(args[1]);
FileReader in  = new FileReader(inputFile);
FileWriter out = new FileWriter(outputFile);
int c;
while ((c = in.read()) != -1) out.write(c);
in.close(); out.close(); }
}
```

Program parameters are : `args[0]` et `args[1]`: `toto.txt tata.txt`
Attention: if the program is running in Eclipse, the `toto.txt` file must be available in the project directory.
Otherwise directly with java interpreter you write:

```
pi@raspberrypi ~/java/MyClasses
$ java CopyTextFile CopyTextFile.java TextFile.java
```

Binary file: CopyByteFile

```
import java.io.*;
public class CopyByteFile {
public static void main(String[] args) throws IOException
{
FileInputStream  in  = new FileInputStream(args[0]);
FileOutputStream out = new FileOutputStream(args[1]);
byte[] buffer  = new byte[1024] ;
int bl;
while ((bl=in.read(buffer))!=-1) out.write(buffer,0,bl);
in.close();
out.close(); }
}
```

```
args[0] args[1]  -  keyboard1.png keyboard2.png
```

Attention: if the program is running in Eclipse, the toto.txt file must be available in the project directory.

Otherwise directly with java interpreter you write:

```
pi@raspberrypi ~/java/MyClasses
$ java CopyByteFile CopyByteFile.class File.class
```

Buffers and filters

When a program needs the data, it starts by looking in the stream's buffer before returning to the original source of the stream.
The buffer is implemented by classes `BufferedInputStream` and `BufferedOutputStream`.

The `InputStream` can exploit two constructors:
`BufferedInputStream` (`InputStream`) and
`BufferedInputStream(InputStream,int)`. The simplest method `read()` has no arguments, it returns a byte - a value between 0 and 255.
At the end of the stream method returns `-1`.

The `OutputStream` can also use two constructors:
`BufferedOutputStream(InputStream)` and
`BufferedOutputStream(InputStream,int)`.

The `int` specifies the buffer size. The method `write(int)` (0-255) sends a byte stream registered in the output buffer.

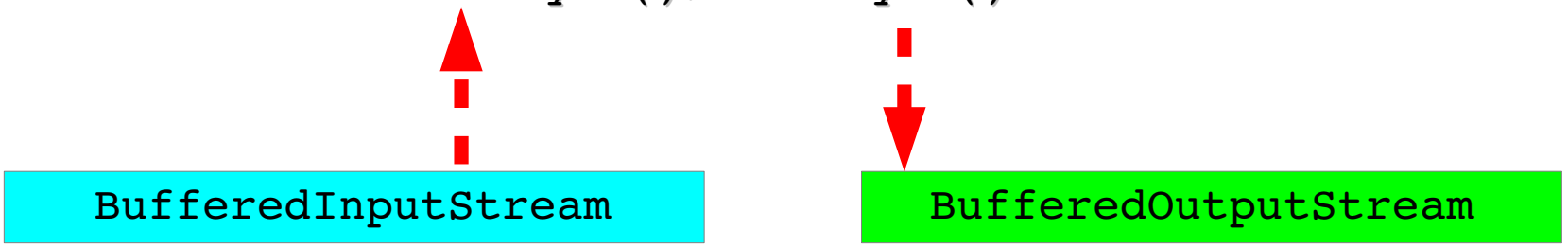
Buffers and filters

```
import java.io.*;
public class BufCopyFile {
public static void main(String[] args) throws IOException
{
FileInputStream  fin  = new FileInputStream(args[0]);
BufferedInputStream bufferIn =
                          new BufferedInputStream(fin);
FileOutputStream fout = new FileOutputStream(args[1]);
BufferedOutputStream bufferOut =
                          new BufferedOutputStream(fout);
int bn;
while ((bn=bufferIn.read())!=-1)
                          bufferOut.write(bn);
bufferIn.close();
bufferOut.close(); }
}
```

Buffers and filters

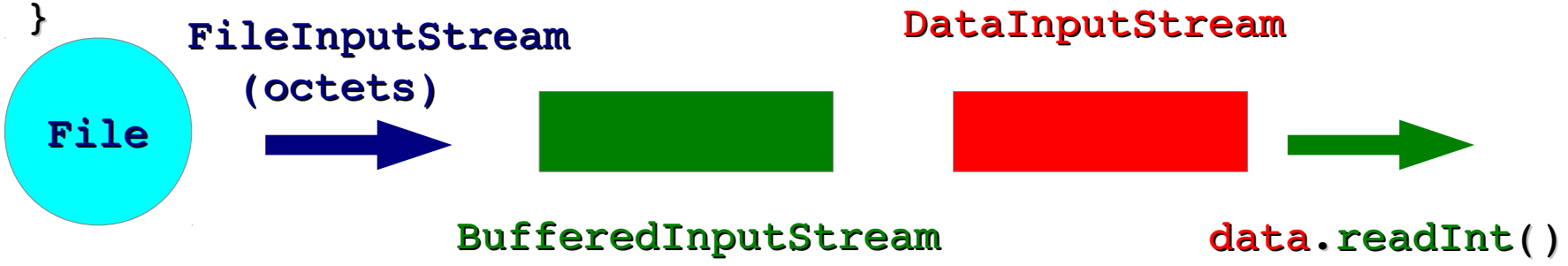
The read/write data streams can be carried out by the following methods (filters):

```
readInt()/writeInt()
readFloat()/writeFloat()
readLong()/writeLong()
readBoolean()/writeBoolean()
readDouble()/writeDouble()
readByte()/writeByte()
```



Buffers and filters

```
import java.io.*;
public class ReadIntFile {
public static void main(String[] args) throws IOException {
FileInputStream  in  = new FileInputStream(args[0]);
BufferedInputStream buffer =
                          new BufferedInputStream(in);
DataInputStream  data = new DataInputStream(buffer) ;
try {
while (true) {
  int val = data.readInt();
  System.out.print(val + « »);}
  }
  catch(EOFException eof) {buffer.close();}
  }
}
```



Buffers and filters

```
java ReadIntFile ReadIntFile.java
1768779887 1920213098 1635148078 1768893994 990539893 1651272035 543
386721 1936924754 1700881481 1853113961 1818566779 175142242 1818845
984 1937006964 1768104054 1869177888 1835100526 676557938 1768843099
 1562403186 1735600416 1953002095 2004033609 1329952867 1701868649 1
869488251 172386668 1699311216 1970557812 1919246701 538995054 53898
3712 1852143392 1181314149 1231974517 1951626354 1700883752 16348875
39 1529896233 990528117 1717986674 1701071214 1886745683 1953654113
1830838901 1717986674 540876910 1702305858 1969645157 1919247433 185
2863860 1400140389 1634543721 1848195850 1147237473 1231974517 19516
26354 1700883744 543449460 1629502752 1852143392 1147237473 12319745
17 1951626354 1700883752 1651861094 1701980475 175403641 544934519 1
751739493 539522162 1969563936 2065697312 543780468 544629100 540876
900 1635017006 1919246692 1231975464 691735072 542341491 1952804142
1869968430 1886546286 1948808801 1814047520 583180322 691764490 5390
00074 538993505 1952671784 1162823237 2019779952 1953066862 54351856
6 689994594 1969645157 1915642732 1869833512 691764490 539000074
```

Give the interpretation of this result !

Streams and object filters

A flow of objects allows to read/write Java objects in a stream.
This operation, called serialization, can be achieved by `ObjectInputStream` and `ObjectOutputStream` filters.

The implementation of interfaces `DataInput` and `DataOutput` is necessary to safeguard the fields of primitive type.

When an object is stored in a stream for the first time, all objects that can be accessed from this object are also stored.
To save an object to a stream, the class of this object must implement the `Serializable` interface.
The `Serializable` interface has no method.
When saving a serializable object, we save the descriptor class, followed by the values of all non-static fields. The descriptor specifies the class name, version, and the types and names of its fields.
While reading a serialized object, this information allows the recovery of its class.
Field values stored in the stream are used to reconstruct an instance.

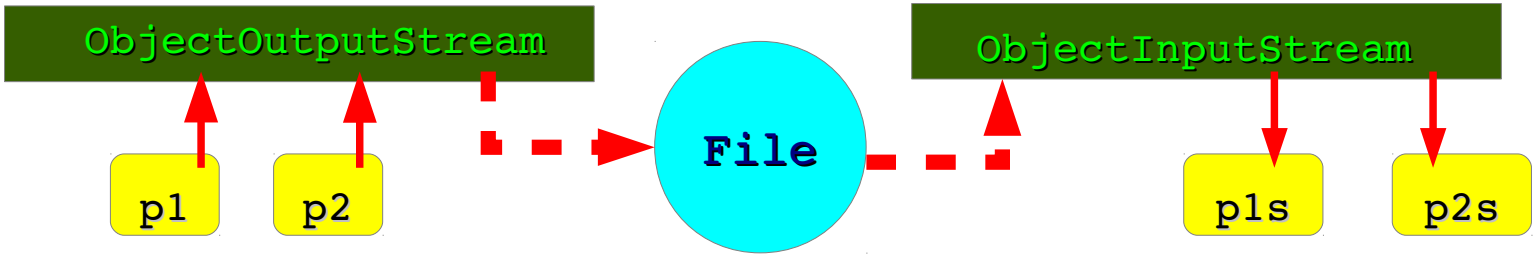
Streams and object filters

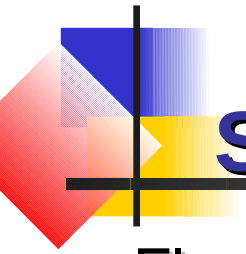
The object to serialize:

```
import java.io.*;
public class ReadSerPoint implements Serializable {
int x; int y;
public ReadSerPoint(int x, int y) {
this.x=x;
this.y=y;
}
public String toString() {
return "(" + x + "," + y + ")";
}
}
```

Streams and object filters (cont.)

```
public static void main(String[] args) throws Exception {
ReadSerPoint p1 = new ReadSerPoint(3,4);
ReadSerPoint p2 = new ReadSerPoint(5,6);
File strfile = File.createTempFile("tempFile",".tmp");
strfile.deleteOnExit(); // destroyed at the execution end
ObjectOutputStream ob_out = new ObjectOutputStream(new
FileOutputStream(strfile));
ob_out.writeObject(p1); ob_out.writeObject(p2); ob_out.close();
ObjectInputStream ob_in = new ObjectInputStream(new
FileInputStream(strfile));
ReadSerPoint p1s = (ReadSerPoint) ob_in.readObject();
ReadSerPoint p2s = (ReadSerPoint) ob_in.readObject();
ob_in.close();
System.out.println("p1="+p1);System.out.println("p2="+p2);
System.out.println("p1s="+p1s);System.out.println("p2s="+p2s);
System.out.println("p1 egale a p1s ? "+p1.equals(p1s));
System.out.println("p2 egale a p2s ? "+p2.equals(p2s)); }
}
```





Summary

Flows - Streams

`InputStreamReader` and `OutputStreamWriter`

Standard I/O: `System.in.` , `System.out.`

Devices: memory and file

Text files and binary files (bytes)

Filters and Buffers

Object serialization