



Java Avanzado

Entrada/Salida

Copyright

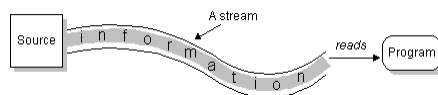
- Copyright (c) 2004
José M. Ordax
- Este documento puede ser distribuido solo bajo los términos y condiciones de la Licencia de Documentación de javaHispano v1.0 o posterior.
- La última versión se encuentra en
<http://www.javahispano.org/licencias/>

Streams

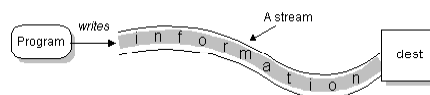
- La entrada/salida en Java se gestiona mediante los streams.
- Un stream tiene un origen y un destino, y por él fluye la información.
- Se lee de un origen.
- Se escribe en un destino.
- Se manejan independientemente del origen o del destino.
- Se encuentran en el paquete `java.io.*`;

Streams (cont.)

- Streams de lectura:



- Streams de escritura:

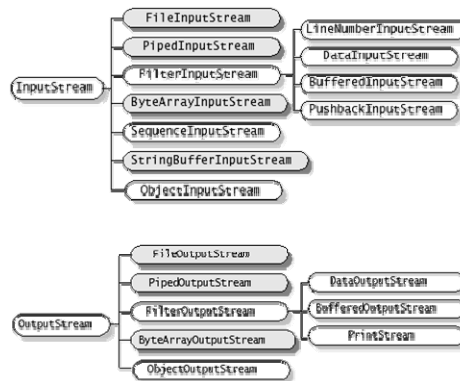


Jerarquía



Se encuentran divididos en dos jerarquías dependiendo de si tratan con bytes o caracteres.

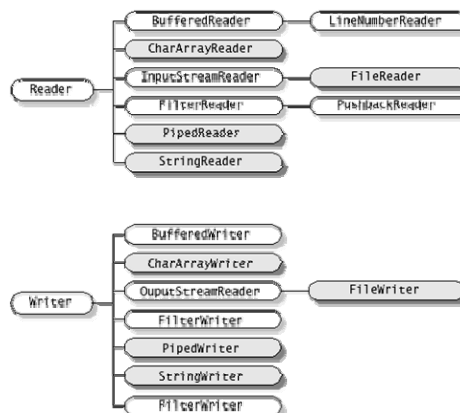
Bytes:



Jerarquía (cont).



Caracteres:




Byte Streams

- Leen o escriben bytes (8 bits).
- Para leer hay que usar la clase `java.io.InputStream` o cualquiera de las clases que heredan de ella.
- Para escribir hay que usar la clase `java.io.OutputStream` o cualquiera de las clases que heredan de ella.
- Pero ambas clases no pueden ser instanciadas puesto que son abstractas. Se reciben como resultado de la ejecución de algún método.
- Suelen usarse para manejar imágenes, sonidos,...




Character Streams

- Leen o escriben caracteres (16 bits).
- Para leer hay que usar la clase `java.io.Reader` o cualquiera de las clases que heredan de ella.
- Para escribir hay que usar la clase `java.io.Writer` o cualquiera de las clases que heredan de ella.
- Pero ambas clases no pueden ser instanciadas puesto que son abstractas. Se reciben como resultado de la ejecución de algún método.
- Suelen usarse para manejar textos,...




InputStream y Reader

 Ambas tienen métodos parecidos pero con distintos parámetros.


 `java.io.InputStream:`

 `int read();`
 `int read(byte[] b);`
 `int read(byte[] b, int off, int len);`




 `java.io.Reader:`

 `int read();`
 `int read(char[] c);`
 `int read(char[] c, int off, int len);`




OutputStream y Writer

 Ambas tienen métodos parecidos pero con distintos parámetros.

 `java.io.OutputStream:`

 `void write(int b);`
 `void write(byte[] b);`
 `void write(byte[] b, int off, int len);`

 `java.io.Writer:`

 `void write(int c);`
 `void write(char[] b);`
 `void write(char[] b, int off, int len);`

Mas sobre Streams

Habr  ocasiones en las que tengamos que pasar de streams tipo byte a tipo car cter. En esos casos, usaremos estas dos clases:

`java.io.InputStreamReader`

Lee bytes y nos devuelve caracteres.

`java.io.OutputStreamWriter`

Le damos caracteres y escribe bytes.

Los streams se abren impl citamente en su creaci n.

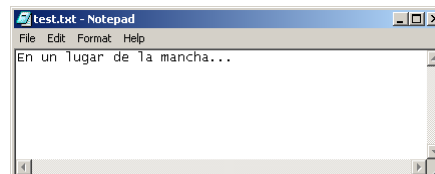
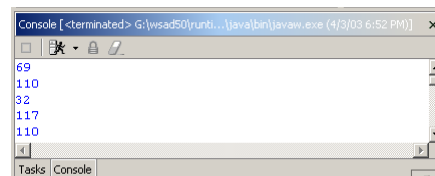
Los streams se cierran expl citamente mediante el m todo: `public void close() throws IOException;`

Ejemplo

```
import java.io.*;

public class EntradaTest
{
    public static void main(String[] args)
    {
        FileInputStream fis = null;
        int aux = 0;

        try
        {
            fis = new FileInputStream("c:\\test.txt");
            while((aux = fis.read()) != -1)
                System.out.println(aux);
        }
        catch(FileNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            try
            {
                fis.close();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```



```

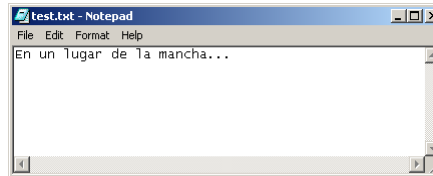
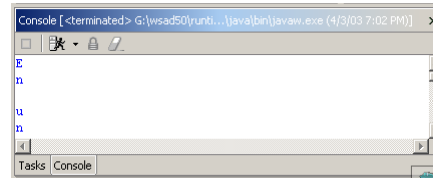
import java.io.*;

public class EntradaTest2
{
    public static void main(String[] args)
    {
        FileReader fr = null;
        int aux = 0;

        try
        {
            fr = new FileReader("c:\\test.txt");
            while((aux = fr.read()) != -1)
                System.out.println((char)aux);
        }
        catch(FileNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            try
            {
                fr.close();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

Ejemplo



```

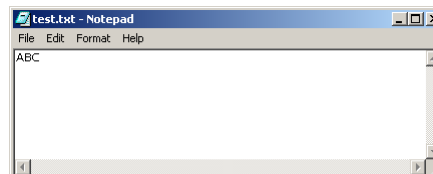
import java.io.*;

public class SalidaTest
{
    public static void main(String[] args)
    {
        FileOutputStream fos = null;

        try
        {
            fos = new FileOutputStream("c:\\test.txt");
            fos.write(65);
            fos.write(66);
            fos.write(67);
        }
        catch(FileNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            try
            {
                fos.close();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

Ejemplo

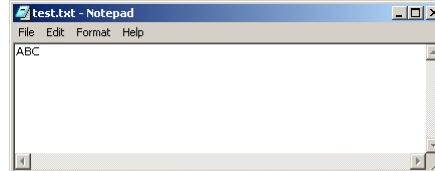


Ejemplo

```
import java.io.*;

public class SalidaTest2
{
    public static void main(String[] args)
    {
        FileWriter fw = null;

        try
        {
            fw = new FileWriter("c:\\test.txt");
            fw.write('A');
            fw.write('B');
            fw.write('C');
        }
        catch(FileNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            try
            {
                fw.close();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```



Filtros



En el paquete java.io.* tenemos una serie de clases abstractas que definen y medio implementan como filtrar el contenido que viene o va a un stream.



Son las siguientes:



java.io.FilterInputStream



java.io.FilterOutputStream



java.io.FilterReader



java.io.FilterWriter



Se suelen usar las clases que heredan de ellas.

Filtros (cont.)

Los filtros se construyen pasando como parámetro otro stream (o filtro) al que van a complementar.

Algunos filtros interesantes son:

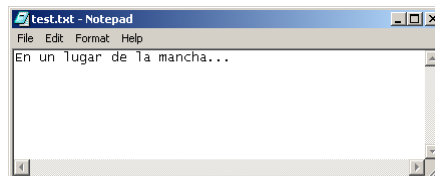
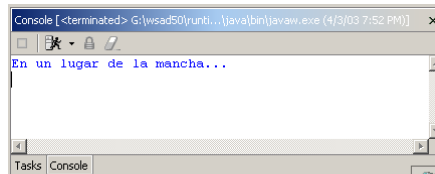
- java.io.BufferedReader y java.io.BufferedWriter
- java.io.BufferedOutputStream y java.io.BufferedWriter
- java.io.DataInputStream
- java.io.DataOutputStream

Ejemplo

```
import java.io.*;

public class FilterTest
{
    public static void main(String[] args)
    {
        DataInputStream dis = null;
        String aux = null;

        try
        {
            dis = new DataInputStream(new BufferedInputStream(new FileInputStream("c:\\test.txt")));
            while((aux = dis.readLine()) != null)
                System.out.println(aux);
        }
        catch(FileNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            try
            {
                dis.close();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```



Todos los streams

Tipo de I/O	Clase	Descripción
Memoria	ByteArrayInputStream ByteArrayOutputStream CharArrayReader CharArrayWriter	Leer de y escribir a memoria. Se crean a partir de una array.
	StringBufferInputStream StringReader StringWriter	Leer de un String en memoria. Escribir a un String en memoria.
Pipe	PipedInputStream PipedOutputStream PipedReader Pipewriter	Encadenar la salida o entrada de un thread con la entrada o salida de otro.

Todos los streams

Tipo de I/O	Clase	Descripción
Fichero	FileInputStream FileOutputStream FileReader FileWriter	Leer de y escribir a fichero.
Concatenación	SequenceInputStream	Concatena múltiples streams de entrada en uno solo.
Serialización de Objetos	ObjectInputStream ObjectOutputStream	Serializan/deserializan objetos.
Conversión de datos	DataInputStream DataOutputStream	Leer o escribir tipos primitivos en vez de bytes.

Todos los streams

Tipo de I/O	Clase	Descripción
Cuenta	LineNumberReader LineNumberInputStream	Lleva la cuenta de las líneas leídas o escritas.
Con retroceso	PushbackReader PushbackInputStream	Permite deshacer la lectura.
Impresión	PrintWriter PrintStream	Tienen métodos potentes de impresión.
Conversión de stream de byte a stream de carácter.	InputStreamReader OutputStreamReader	Puente entre ambos tipos de streams.

Todos los streams

Tipo de I/O	Clase	Descripción
Buffer	BufferedInputStream BufferedOutputStream BufferedReader BufferedWriter	Mantienen en buffer las lecturas y escrituras para mejorar el rendimiento.
Filtros	FilterInputStream FilterOutputStream FilterReader FilterWriter	Clases abstractas, base para los streams que filtran.

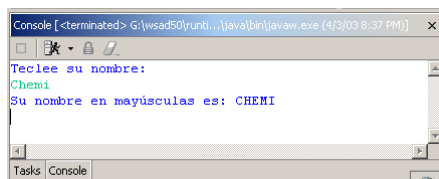
Entrada y salida estándar

- La clase `java.lang.System` tiene los siguientes atributos estáticos:
- System.in:**
 - Es una instancia de la clase `java.io.InputStream`
 - Representa la entrada estándar (por defecto el teclado).
- System.out:**
 - Es una instancia de la clase `java.io.PrintStream`
 - Representa la salida estándar (por defecto la pantalla).
- System.err:**
 - Es una instancia de la clase `java.io.PrintStream`
 - Representa la salida de errores (por defecto la pantalla).

Ejemplo

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class IOStandarTest
{
    public static void main(String[] args)
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Teclee su nombre: ");
        try
        {
            String aux = br.readLine();
            System.out.println("Su nombre en mayúsculas es: " + aux.toUpperCase());
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
    }
}
```



Algunos orígenes de streams



`java.io.FileInputStream`

```
FileInputStream fis = new FileInputStream("c:\\test.txt");
```



`java.net.URL`

```
URL miUrl = new URL("http://www.javahispano.org/a.gif");  
InputStream is = miUrl.openStream();
```



`javax.comm.CommPort`

```
Enumeration portList = CommPortIdentifier.getPortIdentifiers();  
CommPortIdentifier portId = portList.nextElement();  
SerialPort serial = null;  
if(portId.getPortType == CommPortIdentifier.PORT_SERIAL)  
    serial = (SerialPort)portId.open("Test",2000);  
InputStream is = serialPort.getInputStream();
```

Otras clases java.io



`java.io.File`



Representa un fichero o directorio del sistema. Tiene métodos para conocer el path, nombre, atributos,....



`java.io.StreamTokenizer`



Divide el contenido de un stream en trozos según un separador.



`java.util.jar.JarInputStream` y `JarOutputStream`



Manejan los ficheros JAR.



`java.util.zip.ZipInputStream` y `ZipOutputStream`



Manejan los ficheros ZIP.

Serialización

- Existen dos streams que permiten serializar y deserializar un objeto.
- Serializar significa transformar un objeto en una secuencia de bytes para escribirlo en un stream.
- Deserializar significa transformar una secuencia de bytes leída de un stream en un objeto.
- Esta posibilidad es muy útil para:
 - Persistir objetos (instancias).
 - Transmitir objetos (instancias).
- Son `ObjectInputStream` y `ObjectOutputStream`.

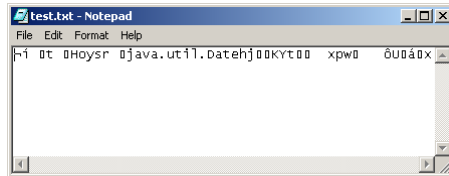
Serialización (cont.)

- Para que un objeto sea serializable debe implementar el interface `java.io.Serializable`
- Este interfaz no tiene definido ningún método, es simplemente como una marca.
- Existe una *keyword* de Java para marcar aquellos atributos que no queremos que sean serializados.
`transient`
- Al deserializar todos los atributos *transient* serán inicializados con sus valores por defecto.
- Al deserializar hay que tener en cuenta un casting.

Ejemplo

```
import java.io.*;
import java.util.Date;

public class SerializarTest
{
    public static void main(String[] args)
    {
        try
        {
            FileOutputStream fos = new FileOutputStream("c:\\test.txt");
            ObjectOutputStream sos = new ObjectOutputStream(fos);
            sos.writeObject("Hoy");
            sos.writeObject(new Date());
            sos.close();
        }
        catch (FileNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
}
```



Ejemplo

```
import java.io.*;
import java.util.Date;

public class DeserializarTest
{
    public static void main(String[] args)
    {
        try
        {
            FileInputStream fis = new FileInputStream("c:\\test.txt");
            ObjectInputStream sis = new ObjectInputStream(fis);
            System.out.println((String)sis.readObject());
            System.out.println((Date)sis.readObject());
            sis.close();
        }
        catch (FileNotFoundException ex)
        {
            ex.printStackTrace();
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
        catch (ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }
    }
}
```



Bibliografía



Java I/O

Elliott Rusty Harold.
O'Reilly.



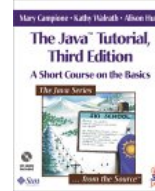
Java NIO

Ron Hitchens.
O'Reilly.



The Java tutorial (3rd edition)

Mary Campione, Kathy Walrath y Alison Huml.
Addison-Wesley.



The Java tutorial (on-line)

<http://java.sun.com/docs/books/tutorial/essential/io>