Programación Interactiva
Archivos

Escuela de Ingeniería de Sistemas y Computación
Facultad de Ingeniería
Universidad del Valle

- Los programas necesitan comunicarse con su entorno, tanto para recoger datos e información que deben procesar, como para devolver los resultados obtenidos.

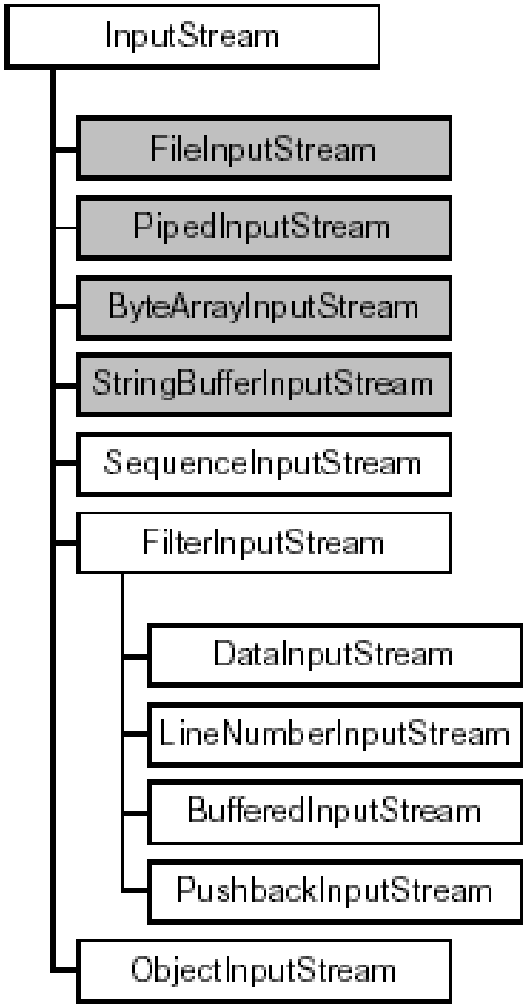
Streams

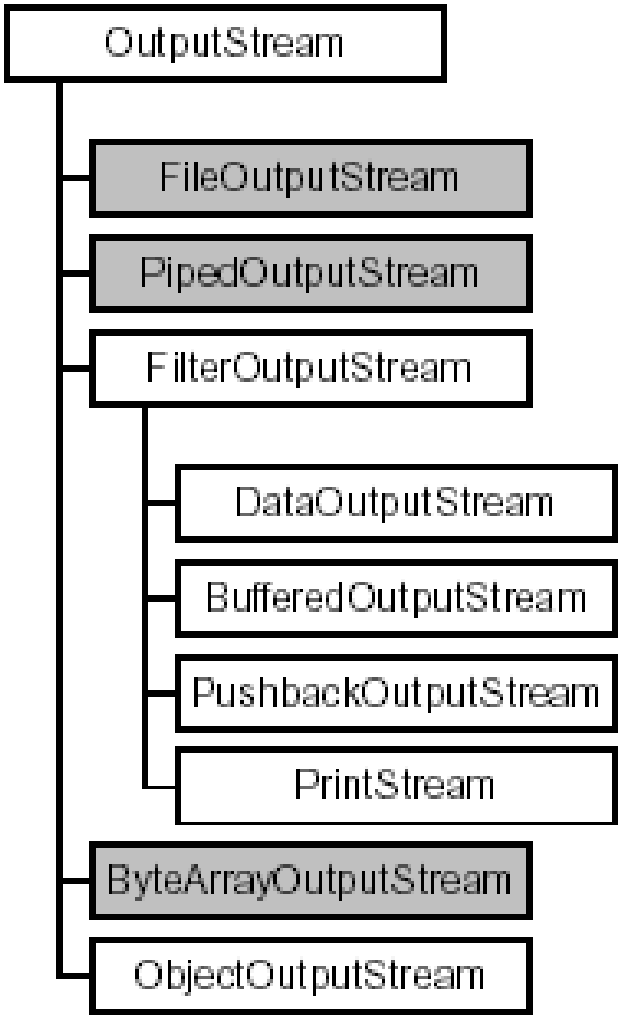
- Un stream es una conexión entre el programa y la fuente o destino de los datos. La información se traslada en serie (un carácter a continuación de otro) a través de esta conexión.
- Esto da lugar a una forma general de representar muchos tipos de comunicaciones.

Streams

- Desde Java 1.0, la entrada y salida de datos del programa se podía hacer con clases derivadas de InputStream (para lectura) y OutputStream (para escritura).
- Estas clases tienen los métodos básicos read() y write() que manejan bytes y que no se suelen utilizar directamente.

Clases InputStream y OutputStream



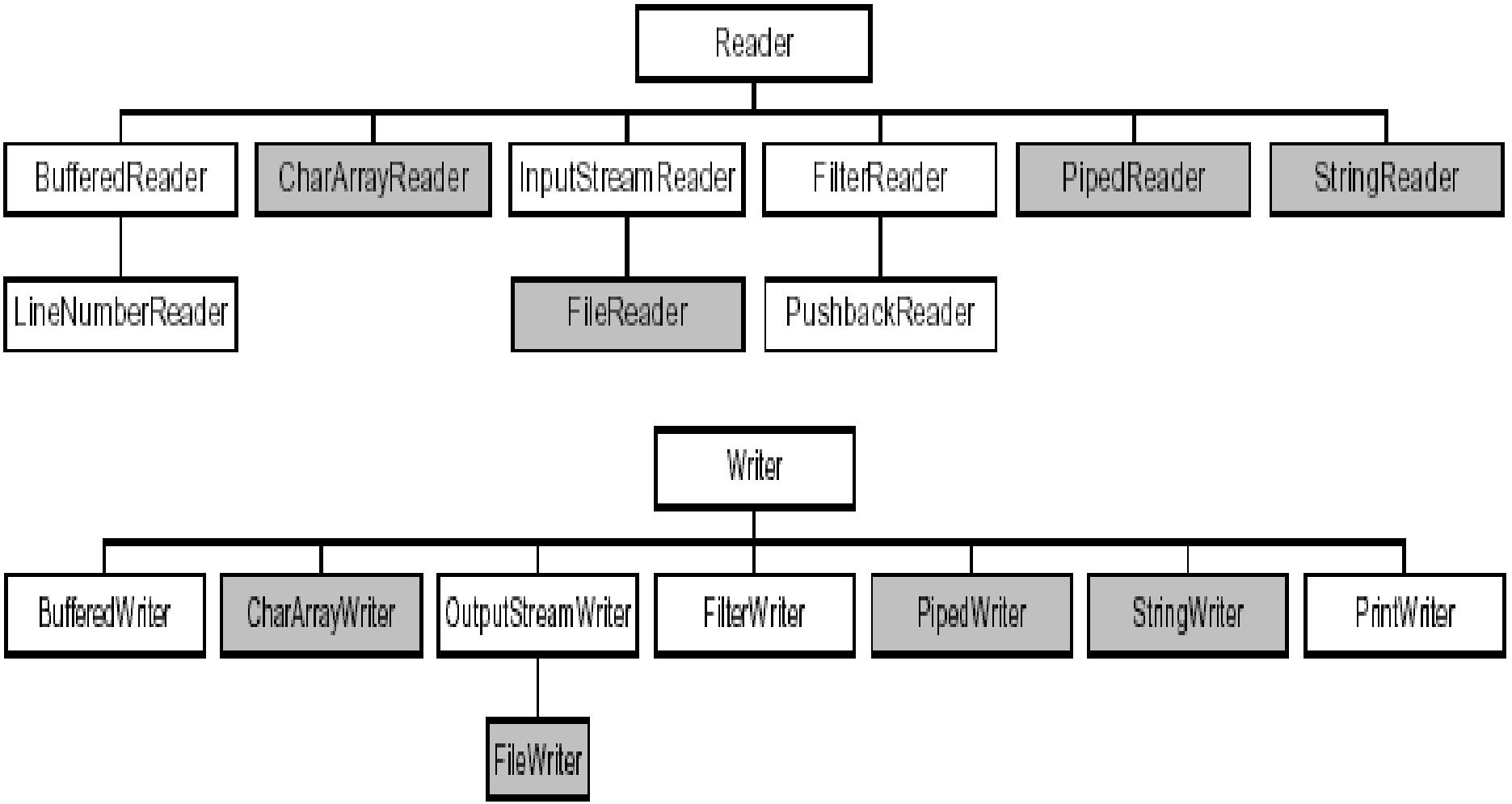


Clases Reader y Writer

- En Java 1.1 aparecieron dos nuevas familias de clases, derivadas de Reader y Writer, que manejan caracteres en vez de bytes.

- Estas clases resultan más prácticas para las aplicaciones en las que se maneja texto.

Clases Reader y Writer



Clases Reader y Writer

```
BufferedReader in = new BufferedReader(new
        FileReader("autoexec.bat"));
```

- Con esta línea se ha creado un stream que permite leer del archivo autoexec.bat. Además, se ha creado a partir de él un objeto BufferedReader (que aporta la característica de utilizar buffer).
- Los caracteres que lleguen a través del FileReader pasarán a través del BufferedReader, es decir utilizarán el buffer.

Buffer

- Un buffer es un espacio de memoria intermedia que actúa de “colchón” de datos. Cuando se necesita un dato del disco se trae a memoria ese dato y sus datos contiguos, de modo que la siguiente vez que se necesite algo del disco la probabilidad de que esté ya en memoria sea muy alta.
- Algo similar se hace para escritura, intentando realizar en una sola operación de escritura física varias sentencias individuales de escritura.

Clases de java.io.

Palabra	Significado
InputStream, OutputStream	Lectura/Escritura de bytes
Reader, Writer	Lectura/Escritura de caracteres
File	Archivos
String, CharArray, ByteArray, StringBuffer	Memoria (a través del tipo primitivo indicado)
Piped	Tubo de datos
Buffered	Buffer
Filter	Filtro
Data	Intercambio de datos en formato propio de Java
Object	Persistencia de objetos
Print	Imprimir

Clases de java.io.

- Clases que leen y escriben en archivos de disco.
  - FileReader,
  - FileWriter,
  - FileInputStream y
  - FileOutputStream

Clases de java.io.

- Estas clases tienen en común que se comunican con la memoria del ordenador. En vez de acceder del modo habitual al contenido de un String, por ejemplo, lo leen como si llegara carácter a carácter. Son útiles cuando se busca un modo general e idéntico de tratar con todos los dispositivos que maneja un programa.

- StringReader,
- StringWriter,
- CharArrayReader,
- CharArrayWriter,

- ByteArrayInputStream,
- ByteArrayOutputStream,
- StringBufferInputStream

Clases de java.io.

- Se utilizan como un “tubo” o conexión bilateral para transmisión de datos. Por ejemplo, en un programa con dos threads pueden permitir la comunicación entre ellos. Un thread tiene el objeto PipedReader y el otro el PipedWriter.
- Si los streams están conectados, lo que se escriba en el PipedWriter queda disponible para que se lea del PipedReader. Tambíen puede comunicar a dos programas distintos.
  - PipedReader,
  - PipedWriter,
  - PipedInputStream,
  - PipedOutputStream

Clases de java.io.

- Como ya se ha dicho, añaden un buffer al manejo de los datos. Es decir, se reducen las operaciones directas sobre el dispositivo (lecturas de disco, comunicaciones por red), para hacer más eficiente su uso.
- BufferedReader por ejemplo tiene el método readLine() que lee una línea y la devuelve como un String.
  - BufferedReader,
  - BufferedWriter,
  - BufferedInputStream,
  - BufferedOutputStream

Clases de java.io.

- Son clases puente que permiten convertir streams que utilizan bytes en otros que manejan caracteres. Son la única relación entre ambas jerarquías y no existen clases que realicen la transformación inversa.
  - InputStreamReader,
  - OutputStreamWriter

Clases de java.io.

- Pertenecen al mecanismo de la serialización y se explicarán más adelante.
  - ObjectInputStream,
  - ObjectOutputStream

Clases de java.io.

- Son clases base para aplicar diversos filtros o procesos al stream de datos. También se podrían extender para conseguir comportamientos a medida.
  - FilterReader,
  - FilterWriter,
  - FilterInputStream,
  - FilterOutputStream

Clases de java.io.

- Se utilizan para escribir y leer datos directamente en los formatos propios de Java. Los convierten en algo ilegible , pero independiente de plataforma y se usan por tanto para almacenaje o para transmisiones entre ordenadores de distinto funcionamiento.
  - DataInputStream,
  - DataOutputStream

Clases de java.io.

- Tienen métodos adaptados para imprimir las variables de Java con la apariencia normal. A partir de un boolean escriben “true” o “false”, colocan la coma de un número decimal, etc.
  - PrintWriter,
  - PrintStream

Entrada y Salida Estandar

- En Java, la entrada desde teclado y la salida a pantalla están reguladas a traves de la clase System.
- Esta clase pertenece al package java.lang y agrupa diversos métodos y objetos que tienen relación con el sistema local. Contiene, entre otros, tres objetos static que son:

  System.in: Objeto de la clase InputStream (read())

  System.out: Objeto de la clase PrintStream

  System.err: Objeto de la clase PrintStream. (print() y println())

Entrada y Salida Estandar

- Existen tres métodos de System que permiten sustituir la entrada y salida estándar. Por ejemplo, se utiliza para hacer que el programa lea de un archivo y no del teclado.
  - System.setIn(InputStream is); (FileInputStream)
  - System.setOut(PrintStream ps); (OutputStream)
  - System.setErr(PrintStream ps); (OutputStream)

Entrada y Salida Estandar

```
char c;
String frase = new String("");
      // StringBuffer frase=new StringBuffer("");
try {
   while((c=System.in.read()) != '\n')
   frase = frase + c; // frase.append(c);
}
catch(java.io.IOException ioex) {}
```

Entrada y Salida Estandar

```
InputStreamReader isr = new
    InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
// BufferedReader br2 = new BufferedReader(new
    InputStreamReader(System.in));
String frase = br2.readLine();
// Se lee la línea con una llamada
```

Entrada y Salida Estandar

- La clase java.util.StringTokenizer da la posibilidad de separar una cadena de carácteres en las “palabras” (tokens) que la forman (por defecto, conjuntos de caracteres separados por un espacio, '\t', '\r', o por '\n').

- Cuando sea preciso se pueden convertir las “palabras” en números.

Entrada y Salida Estandar

Métodos	Función que realizan
StringTokenizer(String)	Constructor a partir de la cadena que hay que separar
boolean hasMoreTokens()	¿Hay más palabras disponibles en la cadena?
String nextToken()	Devuelve el siguiente token de la cadena
int countTokens()	Devuelve el número de tokens que se pueden extraer de la frase

Entrada y Salida en Archivos

- Existen las clases FileInputStream y FileOutputStream (extendiendo InputStream y OutputStream) que permiten leer y escribir bytes en archivos.
- Para archivos de texto son preferibles FileReader (desciende de Reader) y FileWriter (desciende de Writer), que realizan las mismas funciones.

Entrada y Salida en Archivos

```
FileReader fr1 = new
  FileReader("archivo.txt");
```

es equivalente a:

```
File f = new File("archivo.txt");
FileReader fr2 = new FileReader(f);
```

Clase File y FileDialog

Un objeto de la clase ***File*** puede representar un ***archivo*** o un ***directorio***. Tiene los siguientes constructores:

```
File(String name)
File(String dir, String name)
File(File dir, String name).
```

```
File f1 = new File("c:\\windows\\notepad.exe"); // La
  barra '\' se escribe '\\'
File f2 = new File("c:\\windows"); // Un directorio
File f3 = new File(f2, "notepad.exe"); // Es igual a f1
```

Métodos de la Clase File (Archivos)

Métodos	Función que realizan
boolean isFile()	true si el archivo existe
long length()	tamaño del archivo en bytes
long lastModified()	fecha de la última modificación
boolean canRead()	true si se puede leer
boolean canWrite()	true si se puede escribir
delete()	borrar el archivo
RenameTo(File)	cambiar el nombre

Métodos de la Clase File (Directorios)

Métodos	Función que realizan
boolean isDirectory()	true si existe el directorio
mkdir()	crear el directorio
delete()	borrar el directorio
String[] list()	devuelve los archivos que se encuentran en el directorio

Métodos de la Clase File (Paths)

Métodos	Función que realizan
String getPath()	Devuelve el path que contiene el objeto File
String getName()	Devuelve el nombre del archivo
String getAbsolutePat	Devuelve el path absoluto (juntando el relativo al actual)
String getParent()	Devuelve el directorio padre

Clase FileDialog

- La clase ***java.awt.FileDialog*** presenta el diálogo típico de cada sistema operativo para guardar o abrir ficheros. Sus constructores son:
  - `FileDialog(Frame fr)`
  - `FileDialog(Frame fr, String title)`
  - `FileDialog(Frame fr, String title, int type)`

- donde ***type*** puede ser ***FileDialog.LOAD*** o ***FileDialog.SAVE*** según la operación que se desee realizar.

Clase FileDialog

- Es muy fácil conectar este diálogo con un ***File***, utilizando los métodos ***String getFile()*** y ***String getDirectory()***. Por ejemplo:

```
FileDialog fd = new FileDialog(f, "Elija
  un archivo");
fd.show();
File f = new File(fd.getDirectory(),
  fd.getFile());
```

Clase RandomAccessFile

- A menudo, no se desea leer un fichero de principio a fin; sino acceder al fichero como una base de datos, donde se salta de un registro a otro; cada uno en diferentes partes del fichero.
- Java proporciona una clase RandomAccessFile para este tipo de entrada/salida.

Clase RandomAccessFile

- Hay dos posibilidades para abrir un fichero de acceso aleatorio:

  miRAFile = new RandomAccessFile(String nombre, String modo) ;

  miRAFile = new RandomAccessFile(File fichero,String modo);

- El argumento modo determina si se tiene acceso de sólo lectura (r) o de lectura/escritura (r/w).

Clase RandomAccessFile

- read()
- write()
- long getFilePointer();
- long length();
- void seek(long pos); 0... Lenght()

Clase RandomAccessFile

```
class Log {
    public static void main( String args[] ) throws IOException {
        RandomAccessFile miRAFile;
        String s = "Informacion a incorporar\nTutorial de Java\n";
        // Abrimos el fichero de acceso aleatorio
        miRAFile = new RandomAccessFile( "/tmp/java.log","rw" );
        // Nos vamos al final del fichero
        miRAFile.seek( miRAFile.length() );
        // Incorporamos la cadena al fichero
        miRAFile.writeBytes( s );
        // Cerramos el fichero
        miRAFile.close();
        }
    }
```

Escritura de Archivos

```
try {
   FileWriter fw = new FileWriter("escribeme.txt");
   BufferedWriter bw = new BufferedWriter(fw);
   PrintWriter salida = new PrintWriter(bw);
   salida.println("Hola, soy la primera línea");
   salida.close();
   bw = new BufferedWriter(new FileWriter("escribeme.txt",
   true));
   salida = new PrintWriter(bw);
   salida.print("Y yo soy la segunda. ");
   double b = 123.45;
   salida.println(b);
   salida.close();
}
cacth(java.io.IOException ioex) { }
```

Archivos (no Texto)

```
// Escritura de una variable double
DataOutputStream dos = new DataOutputStream(
   new BufferedOutputStream(
      new FileOutputStream("prueba.dat")));
double d1 = 17/7;
dos.writeDouble(d);
dos.close();
// Lectura de la variable double
DataInputStream dis = new DataInputStream(
   new BufferedInputStream(
      new FileInputStream("prueba.dat")));
double d2 = dis.readDouble();
```

Archivos en Internet

```
//Lectura del archivo (texto HTML)
URL direccion = new
   URL("http://www1.ceit.es/subdir/MiPagina.htm");
String s = new String();
String html = new String();
try {
   BufferedReader br = new BufferedReader(
   new InputStreamReader(direccion.openStream()));
   while((s = br.readLine()) != null)
      html += s + '\n';
   br.close();
}
catch(Exception e) { System.err.println(e); }
```

Serialización

- La ***serialización*** es un proceso por el que un objeto cualquiera se puede convertir en una ***secuencia de bytes*** con la que más tarde se podrá reconstruir dicho objeto manteniendo el valor de sus variables. Esto permite guardar un objeto en un archivo o mandarlo por la red.
- Para que una clase pueda utilizar la serialización, debe implementar la interface ***Serializable***, que no define ningún método. Casi todas las clases estándar de ***Java*** son serializables.

Serialización

- La clase MiClase se podría serializar declarándola como:

```
public class MiClase implements
Serializable { }
```

- Para escribir y leer objetos se utilizan las clases ***ObjectInputStream*** y ***ObjectOutputStream***, que cuentan con los métodos ***writeObject()*** y ***readObject()***.

Serialización

```
ObjectOutputStream objout = new
  ObjectOutputStream(
    new FileOutputStream("archivo.x"));
String s = new String("Me van a
  serializar");
objout.writeObject(s);
ObjectInputStream objin = new
  ObjectInputStream(new
  FileInputStream("archivo.x"));
String s2 = (String)objin.readObject();
```

Serialización

- La palabra clave ***transient*** permite indicar que un objeto o varible miembro no sea serializado con el resto del objeto. Al recuperarlo, lo que esté marcado como ***transient*** será ***0***, ***null*** o ***false*** (en esta operación no se llama a ningún constructor) hasta que se le dé un nuevo valor. Podría ser el caso de un ***password*** que no se quiere guardar por seguridad.
- Las variables y objetos ***static*** no son serializados. Si se quieren incluir hay que escribir el código que lo haga.

Externalización

- La interface ***Externalizable*** extiende ***Serializable***. Tiene el mismo objetivo que ésta, pero no tiene ningún comportamiento automático, todo se deja en manos del programador.