

## Entrada / Salida



## Resumen

- Motivación
- Flujos de información
  - Características generales
  - Tipos
- E/S estándar
- Ficheros (texto y binarios) y directorios
- Línea de comandos
- Clase Scanner



## Objetivos

- Conocer y comprender qué es un flujo de datos
- Conocer y comprender distintos flujos de datos para poder interactuar con distintas fuentes y destinos
- Interactuar con la E/S estándar, ficheros (texto y binarios), directorios y la línea de comandos



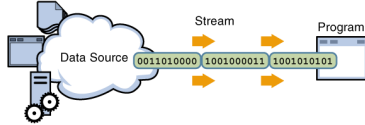
## Motivación

- ¿Programas aislados?
  - ¡No! → Comunicación con el exterior:
    - Recepción de datos de entrada
    - Emisión datos de salida
  - Intercambio de información a través de flujos de datos ó *streams*

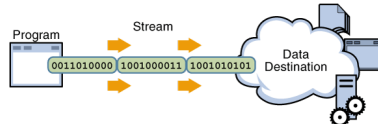


## Streams – Flujos. Definición

- Un stream representa un flujo de información:
  - Procedente de una fuente (teclado, fichero, memoria, red, ...)



- Dirigida a un destino (pantalla, fichero, ...)



- Comparten una misma interfaz para abstraer los detalles específicos de cada dispositivo de E/S
- Las clases de streams en el paquete java.io



Programación Orientada a Objetos



## Entrada y salida estándar I

- Entrada / salida estándar:
  - Clase java.lang.System
  - Flujos:
    - Entrada: System.in Teclado
    - Salida: System.out Pantalla
    - Error: System.err Pantalla



Programación Orientada a Objetos



## Entrada y salida estándar II

- System.in
  - Objeto de la clase java.io.InputStream
  - read(), read(byte[]), read(byte[],int,int)
  - Emite java.io.IOException
- System.out
  - Objeto de la clase java.io.PrintStream
  - print(<cualquier tipo>), println(<cualquier tipo>), flush()
- System.err
  - Objeto de la clase java.io.PrintStream
  - print(<cualquier tipo>), println(<cualquier tipo>), flush()



Programación Orientada a Objetos



## Streams – Flujos. Tipos

- El paquete java.io tiene dos tipos de flujos:
  - Streams de caracteres (caracteres Unicode de 16 bits)
  - Streams de bytes (8 bits)
- E/S puede estar basada:
  - En texto - streams de caracteres legibles
    - Ejemplo: el código fuente de un programa
  - En datos - streams de datos binarios
    - Ejemplo: imagen, sonido, ...
- Otra clasificación:
  - Sumideros de datos: reciben datos
  - Procesadores de datos: tratan/generan información según se lee o escribe



Programación Orientada a Objetos



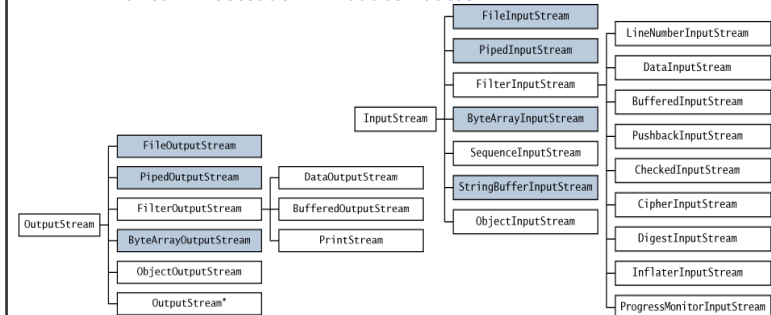
## Streams – Flujos. Tipos

- Uso:
  - Los *streams* de caracteres se utilizan en la E/S de texto
    - Se denominan lectores (*reader*) y escritores (*writer*)
  - Los *streams* de bytes se utilizan en la E/S de datos
    - Se denominan streams de entrada y streams de salida



## Streams – Flujos. Orientados a byte

- Gris - Sumidero de datos – Reciben datos
- Blanco - Procesador – Producen datos



\* In a different package



## InputStream. Subclases

- Representa clases que producen una entrada desde diferentes fuentes: array de bytes, *String*, ficheros, tuberías, una secuencia de varios streams, ...
- Subclases:
  - *FileInputStream*: Lectura de ficheros byte a byte
  - *ObjectInputStream*: Lectura de ficheros como objetos
  - *FilterInputStream* (clase abstracta):
    - *BufferedInputStream*: Lectura con un buffer de datos (+ eficiente)
    - *DataInputStream*: Lectura de tipos de datos primitivos (int, double, etc.)



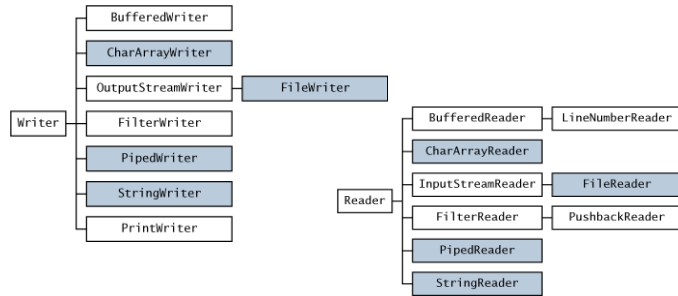
## OutputStream. Subclases

- Representan clases que indican donde irá la salida: array de bytes, ficheros, o tuberías
- *FileOutputStream*: Escritura de ficheros byte a byte
- *ObjectOutputStream*: Escritura objetos en ficheros
- *FilterOutputStream* (clase abstracta):
  - *BufferedOutputStream*:
    - Escritura con un buffer de datos (+ eficiente)
    - Uso del método *flush()* para volcar los datos del buffer
  - *DataOutputStream*: Escritura de tipos de datos primitivos (int, double, ...)



## Streams – Flujos. Orientados a carácter

- Soportan UNICODE (16 bits para un char)
- Gris - Sumidero de datos – Reciben datos
- Blanco - Procesador – Producen datos



Programación Orientada a Objetos



## Subclases de Reader

- `InputStreamReader`: Convierte un stream de bytes en un stream de chars
  - `FileReader`: Se asocia a files de caracteres para leerlos
- `BufferedReader`: Proporciona una entrada de caracteres a través de un buffer (+ eficiente)



Programación Orientada a Objetos



## Entrada estándar - Ejemplo

- Programa que lee una línea de la entrada estándar usando *streams* y la muestra por pantalla

```
import java.io.*;
public class Lector {
    public static void main(String[] args) throws IOException {
        BufferedReader lee;
        lee = new BufferedReader(new InputStreamReader(System.in));
        String st = lee.readLine();
        System.out.println(st);
    }
}
```



Programación Orientada a Objetos



## Lectura de un fichero de texto - Ejemplo

- Programa que lee una línea de un fichero de texto y lo muestra por pantalla utilizando *streams*

```
try
{
    BufferedReader lec;
    lec = new BufferedReader (new FileReader ("sal.txt"));
    String st = lec.readLine();
    System.out.println(st);
} catch (Exception e){}
```



Programación Orientada a Objetos



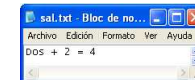
## Subclases de Writer

- `OutputStreamWriter`: convierte un stream de bytes en un stream de chars.
  - `FileWriter`: se asocia a files de chars para modificarlos.
- `BufferedWriter`: proporciona salida de caracteres a través de un buffer (más eficiencia).
- `PrintWriter`: métodos `print()` y `println()` para distintos tipos de datos.



## Escritura en un fichero de texto - Ejemplo

```
try
{
    PrintStream flujo;
    flujo = new PrintStream (new FileOutputStream("sal.txt"));
    flujo.print("Dos + " + 2);
    flujo.println(" = " + (2+2));
}catch (Exception e){}
```



## Streams – Flujos. 'Parsing' de tipos básicos

- Prototipo:
  - `public static T parseT(String s) throws NumberFormatException;`
- Métodos: `parseInt`, `parseLong`, `parseFloat`, `parseDouble`
- Entrada: El string que será convertido al tipo solicitado
- Salida: Valor numérico (del tipo solicitado) del string de entrada
- Ejemplo:

```
BufferedReader stdin = new BufferedReader( new
InputStreamReader(System.in));
System.out.print("Introduzca una línea:");
String linea = stdin.readLine();
int a = Integer.parseInt(linea);
System.out.println(a);
```



## Streams – Flujos. Usos comunes. Lectura de líneas

- Lectura de un fichero línea a línea

```
Maria Martin#123456
Pepe Lopez#234567
Juan Sanchez#89230

public static void main(String[] args) throws IOException
{
    BufferedReader in = new BufferedReader( new
FileReader("miagenda.txt"));
    String s, s2 = new String();
    while((s = in.readLine())!= null)
        s2 += s + "\n";
    in.close();
}
```



## Streams – Flujos. Usos comunes. Lectura de líneas

- Lectura de líneas a través de teclado

```
public static void main(String[] args) throws IOException
{
    BufferedReader stdin = new BufferedReader( new
        InputStreamReader(System.in));
    System.out.print("Introduce una línea:");
    System.out.println(stdin.readLine());
}
```

```
Introduce una línea:
Esta es una de las líneas que introduzco
Esta es una de las líneas que introduzco
```



## Streams – Flujos. Usos comunes. Escritura por líneas

- Ejemplo:

```
String[] s = {"hola", "que", "tal"}; // Inicialización
PrintWriter out1 = new PrintWriter(new
    BufferedWriter(new FileWriter("fsalida.out")));
int cont = 1;
for (int i=0; i< s.length; i++)
    out1.println(cont++ + ": " + s[i]);
out1.close();
```

- Obs: Puede lanzar excepciones de IO (throws IOException)



## Otras clases de java.io - File

- Operaciones habituales con ficheros y directorios
- Constructor:
  - File (String ruta fichero)
  - System.getProperty("file.separator")
- Métodos:
  - exists(): Comprueba si el fichero o directorio existe
  - getName(): Devuelve el nombre del fichero o del directorio
  - isDirectory() / isFile(): Comprueba si es un directorio o fichero
  - mkdir() / createNewFile(): Creación de un directorio o fichero
  - listFiles(): Devuelve una lista de objetos 'File' de un directorio
  - delete(): Borra un fichero/directorio (los directorios deben estar vacíos)



## Ejercicio – Otras clases de java.io - File

- Ficheros - Realizar un programa que:
  - Cree un nuevo fichero llamado 'ficherito.txt' en la unidad "D:" de nuestro disco duro, si no existe ya
  - En el caso que este fichero exista, el programa deberá mostrar por pantalla su nombre y comprobar si es un fichero o por el contrario es un directorio



### Ejercicio – Otras clases de java.io – File - Solución

```
String base = "d:" + System.getProperty("file.separator");
String fichero = base + "ficherito.txt";
File f = new File(fichero);

if (f.exists())
{
    System.out.println("El fichero " + f.getName() + " existe");

    if (f.isFile()) System.out.println("Es un fichero");
    else System.out.println("Es un directorio");
}

else
{
    System.out.print("El fichero 'ficherito.txt' NO existe. Creando... ");
    if (f.createNewFile()) System.out.println("correctamente");
    else System.out.println("malamente");
}
```



Programación Orientada a Objetos



### Ejercicio – Otras clases de java.io - File

- Directorios - Realizar un programa que:
  - Cree un nuevo directorio llamado 'nuevo' en la unidad "D:" de nuestro disco duro, si no existe ya. Además se deberán crear tres ficheros llamados f0.txt, f1.txt y f2.txt dentro del directorio recién creado
  - En el caso que este directorio exista, se borrarán tanto los ficheros que contiene el directorio como el propio directorio



Programación Orientada a Objetos



### Otras Clases de java.io –RandomAccessFile

- Permite acceder directamente a una posición de un fichero binario
- Lanza java.io.IOException (EOF o flujo cerrado)
- Abrir / cerrar un fichero:
  - **RandomAccessFile f = new RandomAccessFile("s.txt", "r");**
  - **f.close();**
  - **Modos → r y rw (rw crea el fichero si no existe)**
- Métodos:
  - **Lectura-escritura de datos primitivos:**
    - **readInt() → int, readBoolean() → boolean, ...**
    - **writeInt(int), writeBoolean(boolean),...**
  - **Devolución de la longitud del fichero: length()**
  - **Obtención de la posición actual del fichero: getFilePointer()**
  - **Posicionamiento en la posición anterior al byte especificado: seek(long). El final del fichero es f.seek(f.length())**



Programación Orientada a Objetos



### Ejercicio – Otras Clases de java.io – RandomAccessFile

- Realizar un programa que:
  - Cree un fichero binario llamado 'numeritos.dat' que contenga los números del 10 al 100 y cerrar el fichero
  - Abrir el fichero para modificar el cuarto entero a 101 sabiendo que un entero ocupa 4 bytes y cerrar el fichero
  - Abrir el fichero para mostrar por pantalla los datos almacenados tal y como se muestra a continuación

```
Numero: 1: 10
Numero: 2: 20
Numero: 3: 30
Numero: 4: 101
Numero: 5: 50
Numero: 6: 60
Numero: 7: 70
Numero: 8: 80
Numero: 9: 90
Numero: 10: 100
```



Programación Orientada a Objetos



### Otras Clases de java.io –ObjectXXStream

- Java permite guardar objetos en ficheros, siempre y cuando implementen la interfaz *Serializable*:
  - public class MySerializableClass implements Serializable { ... }
- ObjectOutputStream:
  - Escribe tipos primitivos y objetos
  - Puede lanzar una IOException
- ObjectInputStream:
  - Lee tipos primitivos y objetos de un fichero
  - Los datos han sido escritos utilizando ObjectOutputStream
  - Puede lanzar una IOException y ClassNotFoundException



### Otras Clases de java.io –ObjectXXStream

```
FileOutputStream out = new FileOutputStream("fechas.dat");
ObjectOutputStream so= new ObjectOutputStream(out);
so.writeObject("Hoy");
GregorianCalendar c = new GregorianCalendar();
c.set(2009, 0, 1); // Año, mes (empieza en 0) día
so.writeObject(c);
so.close();

FileInputStream in = new FileInputStream("fechas.dat");
ObjectInputStream si = new ObjectInputStream(in);
String hoy = (String)si.readObject();
GregorianCalendar fecha = (GregorianCalendar)si.readObject();
si.close();

                Fecha: Hoy
System.out.println("Fecha: " + hoy);          Fecha: Thu Jan 01 17:09:20 CET 2009
System.out.println("Fecha: " + fecha.getTime().toString());
```



### Otras clases de java.io - StreamTokenizer

- Permite partir un flujo en tokens
  - StreamTokenizer(Reader r)
  - public int nextToken():
    - El tipo del siguiente token se devuelve en el campo ttype
    - Información adicional en los campos:
      - nval si el tipo es numérico (TT\_NUMBER)
      - sval si el tipo es una cadena de caracteres (TT\_WORD)
    - Lanza una excepción IOException si ocurre un error de I/O
    - TT\_EOL: Fin de línea
    - TT\_EOF: Fin de fichero



### Otras clases de java.io - StreamTokenizer

```
FileReader rd = new FileReader("mifichero.txt");
StreamTokenizer st = new StreamTokenizer(rd);
int token = st.nextToken();
while (token != StreamTokenizer.TT_EOF)
{
    token = st.nextToken();
    switch (token)
    {
        case StreamTokenizer.TT_NUMBER:
            System.out.println("Numero: " + st.nval);
            break;
        case StreamTokenizer.TT_WORD:
            System.out.println("Cadena: " + st.sval);
            break;
        case StreamTokenizer.TT_EOL: // Fin de linea
            break;
        case StreamTokenizer.TT_EOF: // Fin de fichero
            break;
    }
}
rd.close();
```



## Entrada de datos

- Hay varias formas de suministrar datos a un programa:
  - Línea de comandos en la llamada
  - Entrada estándar - teclado (System.in)
  - Otros flujos de entrada como ficheros, otros dispositivos,...



## Entrada de datos – Línea de comandos

- `public static void main (String args[])`
  - `args` es una lista de todos los argumentos tecleados al llamar el programa desde línea de comandos
  - `args.length`: indica número de argumentos (0... )
  - `args[i]`: contiene (como String) el valor del i-ésimo argumento
- Paso de argumentos desde:
  - Línea de comandos:
    - `java NombrePrograma arg1 arg2 ... argN`
  - Netbeans:
    - Build → Set Main Project Configuration → Customize → Arguments



## Entrada de datos – Línea de comandos Ejemplo 1 – Cadenas de caracteres

```
public class Saluda
{
    public static void main(String[] args)
    {
        System.out.println("Hola " + args[0]);
        System.out.println("Tu email es: " + args[1]);
        System.out.println("Número de argumentos: " + args.length);
    }
}
```

```
Hola Estefania
Tu email es: estefania.martin@urjc.es
Número de argumentos: 2
```



## Entrada de datos – Línea de comandos Ejemplo 2 - Enteros

```
public class Suma
{
    public static void main (String args[])
    {
        int num1= Integer.parseInt(args[0]);
        int num2= Integer.parseInt(args[1]);
        System.out.println("Total: " + (num1 + num2));
    }
}
```



### Ejercicio – Entrada de datos – Línea de comandos

- Realizar un programa que calcule el área de un círculo o de un rectángulo, teniendo en cuenta que:
  - Los datos necesarios para el cálculo del área son numéricos que pueden contener decimales y que se reciben por línea de comandos
  - Si el programa recibe un argumento se trata del cálculo del área del círculo
  - Si el programa recibe dos argumentos deberá calcular el área de un rectángulo donde el primer argumento es la base y el segundo la altura



### Entrada de datos – Scanner

- Clase de utilidad para "leer" diferentes entradas: teclado, ficheros, ...
- Funcionalidad:
  - Entrada: Cadena de entrada y opcionalmente, un delimitador (por defecto, el espacio en blanco)
  - Salida: Tokens a los cuales se accede por métodos nextX()



### Entrada de datos – Scanner - Uso

- Importar la clase:

```
import java.util.Scanner;
```
- Construcción de un objeto de la clase Scanner con el flujo de entrada:
  - Teclado:

```
Scanner ent = new Scanner (System.in);
```
  - Fichero de texto:

```
Scanner ent = new Scanner (new File ("MiFichero.txt"));
```
- Uso de sus métodos nextX() para obtener los tokens:
  - nextInt(), nextLong(), nextDouble(), nextFloat(): Tipos primitivos
  - nextLine(): Lee lo que le queda de la línea actual
  - hasNextLine(): Devuelve true si quedan más líneas por leer



### Entrada de datos – Scanner - Ejemplo

```
import java.util.Scanner;
class InteraccionTeclado
{
    public static void main (String args[])
    {
        Scanner ent= new Scanner(System.in);
        System.out.print("Operador 1: ");
        int op1= ent.nextInt();
        System.out.print("Operador 2: ");
        int op2= ent.nextInt();
        System.out.println("Suma: " + (op1 + op2));
    }
}
```



## Ejercicio – Entrada de datos – Scanner

- Realizar un programa que lea de fichero los datos de una agenda de teléfonos y los muestre por pantalla, teniendo en cuenta que:
  - Cada línea del fichero tiene los datos de un contacto, que serán el nombre y su teléfono separados por el carácter #



- El programa deberá mostrar los datos de esta manera:

```
Nombre: Maria Martin      Telefono: 123456
Nombre: Pepe Lopez       Telefono: 234567
Nombre: Juan Sanchez     Telefono: 89230
```

