Tema 6
Entrada/Salida

Programación II

Alicia Garrido Alenda

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Entrada/salida estándar

- Normalmente:
  - Salida estándar: pantalla
  - Salida de error estándar: pantalla si no se redirige
  - Entrada estándar: teclado
- En Java se realizan mediante tres objetos que puede usar el programador directamente (paquete `java.io`):
  - Salida estándar: `System.out`
  - Salida de error estándar: `System.err`
  - Entrada estándar: `System.in`

Salida estándar: Objeto `System.out`

Métodos más frecuentes:

- `print(variable)`: Imprime la variable por pantalla.
- `println(variable)`: Imprime la variable por pantalla y termina con un retorno de carro.

- Por ejemplo:

```
int i=12;
System.out.print(314);
System.out.print(false);
System.out.print(" -> sale esto");
System.out.println();
System.out.println("Podemos concatenar "+i);
```

- Mostraría:

```
314false -> sale esto
Podemos concatenar 12
```


Salida de error estándar: Objeto `System.err`

- Los métodos son iguales a los de `System.out` pero usan la salida de error estándar.
- Por ejemplo:

```
int i=12;
System.out.print(314);
System.err.print(false);
System.out.print(" -> sale esto");
System.out.println();
System.err.println("Podemos concatenar "+i);
```

- Si lo ejecutamos redirigiendo la salida de error a un fichero:

```
java Salida_std 2>errores
```

- Mostraría por pantalla únicamente:

```
314 -> sale esto
```

Entrada estándar: Objeto `System.in`

Método más frecuente:

`int read()`: Lee un carácter del teclado y lo devuelve como entero (código ascii) cuando se pulsa el retorno de carro o fin de fichero.

- El proceso de lectura tiene que estar dentro de una instrucción `try...catch`

```
try{
 System.in.read();
} catch (Exception e) {
 System.err.println("Error");
}
```


`System.in.read()`

- Este método se puede usar para leer hasta fin de fichero (ctrl+d):

```
int i=0;
try{
 while(i!=-1) {   //CARACTER FIN DE FICHERO DE TEXTO
  i=System.in.read();
  if (((char)i!='\n')&&(i!=-1))
   System.out.println("Leido: "+(char)i);
 }
}catch(Exception e) {System.err.println("Error");}
```

- Podemos leer de teclado o bien un fichero de texto redirigiendo la entrada del programa desde línea de comando:

```
java Entrada_Std < fichero.txt
```


`System.in.read()`

- Podemos leer un conjunto de caracteres hasta pulsar retorno de carro:

```
StringBuffer cad = new StringBuffer();
char c=0;
try{
 while(c!='\n') {
  c=(char)System.in.read();
  if (c!='\n')
   cad.append(c);
 }
 System.out.println("Leido: "+cad);
}catch(Exception e) {System.err.println("Error");}
```

- La clase `StringBuffer` permite crear strings de forma incremental.
  - Método `append(char)`: Añade el carácter `c` al final del buffer.
  - Método `toString()`: Convierte StringBuffer a String.

Abrir ficheros

- Apertura: La apertura de un fichero siempre se realiza dentro de una instrucción

```
try{
 ...
}catch(IOException ex){System.err.println(ex);}
```

- Esto se hace para capturar las posibles excepciones (no existe el fichero, no se puede abrir, ...).
- En Java un fichero se abre cuando creamos un objeto de alguna de las clases de *streams* (diferentes tipos de ficheros).
- En cualquier caso siempre devuelven el descriptor del fichero.

Cerrar ficheros

- Clausura: Cuando terminamos de escribir, leer, etc. un fichero tenemos que *cerrar* el fichero.
- Esto se realiza habitualmente en un bloque

```
finally{
...
}
```

- Todas las clases de fichero tienen un método `close()` que cierra el descriptor de fichero.
- Si no se cierra un fichero en el que se han escrito datos, dichos datos es posible que se pierdan, ya que los datos no se guardan hasta que se cierra el fichero o se llena el *buffer* intermedio.

Para utilizar cualquier clase de fichero es necesario importar los paquetes de entrada/salida de Java:

```
import java.io.*;
```


Leyendo por líneas

- Con `System.in` leemos sólo carácter a carácter.
- Para leer por líneas necesitamos:
  1. Crear un objeto de tipo *fichero de lectura* asociado a la entrada estándar ⇒ Clase `InputStreamReader`

     `InputStreamReader cin=new InputStreamReader(System.in);`

     Crea en `cin` un flujo de entrada de texto.
  2. Crear un buffer de entrada que permita leer líneas enteras ⇒ Clase `BufferedReader`

     ```
     BufferedReader entrada=new BufferedReader(cin);
     ```

  3. Hemos vinculado el objeto `cin`, que hace de puente entre la entrada estándar y los ficheros, con el objeto `entrada` que permite leer líneas enteras mediante el método `String readLine()`.

Ejemplo: Leer líneas de la entrada estándar

```
InputStreamReader cin=null;
BufferedReader mibuf=null;
try{
   // objeto tipo fichero
  cin=new InputStreamReader(System.in);
   // buffer para leer la linea
  mibuf=new BufferedReader(cin);
  String cadena=null;
  cadena=mibuf.readLine();
  while(cadena!=null){
     System.out.println("----"+cadena+"------");
     cadena=mibuf.readLine();
  }
}catch(IOException e){System.err.println("Error: "+e);}
finally{
 if (mibuf!=null)
  try{ mibuf.close(); }catch(IOException e){
      e.printStackTrace();}
 if (cin!=null)
  try{ cin.close(); }catch(IOException e){
      e.printStackTrace();}
}
```


Tipos de ficheros: texto

Fichero de texto

Contiene solamente caracteres imprimibles (aquellos cuyo código ascii es mayor o igual que 32).

- Los datos se convierten en caracteres para ser almacenados en un fichero de texto.
- Un fichero de texto es portable entre distintos sistemas operativos y entre distintos ordenadores.
- El acceso a los datos contenidos en un fichero de texto siempre será secuencial.

Tipos de ficheros: binarios

Fichero binario

Los datos se almacenan tal y como se almacenan en la memoria del ordenador.

- Los datos no se convierten para ser almacenados en un fichero binario.
- Un fichero binario puede no ser portable entre distintos sistemas operativos o entre distintos ordenadores.
- El acceso a los datos de un fichero binario puede ser secuencial o aleatorio (también llamado directo).

Ficheros de texto: Escritura

1. Abrir fichero para escritura
   - Clase `FileWriter`

```
FileWriter salida=null;
try{
  salida=new FileWriter("salida.txt");
}catch(IOException e){System.err.println(e);}
```

2. Métodos:
   - `void write(int c)`: Escribe el carácter correspondiente al código ascii que contiene `c`.
   - `void write(String s)`: Escribe la cadena contenida en `s`.

Ejemplo: Generar un fichero a partir de la entrada estándar

- Hacer un programa en java que lea de la entrada estándar y escriba en un fichero lo que ha leído cada vez que pulsemos *return* hasta encontrar el fin de fichero (`ctrl+d`).

```
InputStreamReader -> fichero asociado a entrada estándar
BufferedReader -> para lectura por líneas
FileWriter ->  fichero de escritura.
```


Ejemplo: Generar un fichero a partir de la entrada estándar

```
try{
  FileWriter descriptor=new FileWriter(args[0]);
  InputStreamReader cin=new InputStreamReader(System.in);
  BufferedReader mibuf=new BufferedReader(cin);
  String cadena=null;
  cadena=mibuf.readLine();
  while(cadena!=null){
     descriptor.write(cadena);
     descriptor.write('\n');
     cadena=mibuf.readLine();
    }
}catch(IOException ex){
 ex.printStackTrace();
}
finally{
 if (descriptor!=null) ...
 if (mibuf!=null) ...
 if (cin!=null) ...
}
```

Ficheros de texto: Lectura

1. Abrir fichero para lectura
   ⇒ Clase `FileReader`

```
FileReader dfe=null;
try{
  dfe=new FileReader("entrada.txt");
}catch(IOException e){System.err.println(e);}
```

   Devuelve en `dfe` el descriptor de un fichero de entrada de texto.
2. Crear un buffer de entrada para leer líneas enteras a partir del descriptor
   ⇒ Clase `BufferedReader`

```
BufferedReader entrada=new BufferedReader(dfe);
```

3. Hemos vinculado el objeto `dfe`, descriptor del fichero `entrada.txt`, con el objeto `entrada` que permite leer líneas enteras.

Ejercicio: Leer un fichero de texto

- Hacer un programa en java que lea el fichero de texto que hemos generado en el ejemplo anterior y lo muestre por pantalla.

```
FileReader ->  el fichero de lectura.
BufferedReader -> lectura por líneas
```


Ejercicio: Leer un fichero de texto

```
try{
  FileReader entrada=new FileReader(args[0]);
  BufferedReader mibuf=new BufferedReader(entrada);
  String cadena=null;
  cadena=mibuf.readLine();
  while(cadena!=null){
    System.out.println("---"+cadena+"----");
    cadena=mibuf.readLine();
  }
}catch(IOException ex){
 ex.printStackTrace();}
finally{
 if (mibuf!=null)
   try{ mibuf.close(); }catch(IOException ex){
     ex.printStackTrace();}
 if (entrada!=null)
   try{ entrada.close(); }catch(IOException ex){
     ex.printStackTrace();}
}
```


Tipos de ficheros binarios

En Java existen distintas clases para el tratamiento de los ficheros binarios en función de lo que se necesite.

- `FileInputStream/FileOutputStream`: Tratamiento secuencial de los datos sin diferenciar tipos básicos.
- `DataInputStream/DataOutputStream`: Tratamiento secuencial de la información diferenciando el tipo de dato.
- Además:
  - `RandomAccessFile`
  - `ObjectInputStream/ObjectOutputStream`
  - `BufferedInputStream/BufferedOutputStream`
  - `ByteArrayInputStream/ByteArrayOutputStream`
  - `...`

Ficheros binarios: Escritura secuencial

1. Abrir fichero para escritura
   - Clase `FileOutputStream`

```
FileOutputStream salida=null;
try{
  salida=new FileOutputStream("fichero.dat");
}catch(IOException e){System.err.println(e);}
```

2. Métodos:
   - `void write(int c)`: Escribe el byte que contiene `c`.

Ejercicio: Generar un fichero binario a partir de un fichero de texto

- Hacer un programa en java que lea un fichero de texto y escriba su contenido en un fichero binario.

```
FileReader para abrir el fichero de lectura.
FileOutputStream para abrir fichero de escritura.
Escribir los números como números (no como
caractereres):
 if ((c>='0')&&(c<='9'))
   i=i-(int)'0';
 descriptor.write(i);
```


Ejercicio: Generar un fichero binario a partir de un fichero de texto

```
try{
  ...
  while(i!=-1){
  ...
  }
}catch(IOException ex){System.err.println(ex);}
finally{
 if (lectura!=null) {
   try{
    lectura.close();
   }catch(IOException ex){System.err.println(ex);} }
 if (escritura!=null) {
   try{
    escritura.close();
   }catch(IOException ex){System.err.println(ex);} }
}
```


Ficheros binarios: Lectura secuencial

1. Abrir fichero para lectura
   - Clase `FileInputStream`

```
FileInputStream entrada=null;
try{
  entrada=new FileInputStream("fichero.dat");
}catch(IOException e){System.err.println(e);}
```

2. Métodos:
   - `int read()`: Lee un byte y lo devuelve como entero.
   - `int read(byte[] bbuf)`: Lee tantos bytes como quepan en bbuf o los que queden hasta final de fichero.

Ejemplo: Leer un fichero binario

- Hacer un programa en java que lea el contenido de un fichero binario y lo muestre por pantalla.

```
FileInputStream para abrir el fichero de lectura.
Un array de bytes (para leer en bloques):
  byte[] vector=new byte[50];
Mostrar bytes como caracteres:
  System.out.print((char)vector[i]);
Mostrar números como números:
  if ((vector[i]>=0) && (vector[i]<=9))
       c=(char)(vector[i]+(char)'0');
```


Ejemplo: Leer un fichero binario

```
try{
 descriptor=new FileInputStream(nomfich);
 int leidos = descriptor.read(vector);
 while(leidos>0) {
  StringBuffer str=new StringBuffer();
  for (int i=0;i<vector.length;i++) {
     if ((vector[i]>=0) && (vector[i]<=9))
       c=(char)(vector[i]+(char)'0');
     else c=(char)vector[i];
     str.append(c);
    }
   System.out.print(str);
   leidos = descriptor.read(vector);
  }
}catch(IOException ex){System.err.println(ex);}
finally{
 if (descriptor!=null)
   try{ descriptor.close();
   }catch(IOException ex){System.err.println(ex);}
}
```


Ficheros binarios: Escritura de datos primitivos

1. Identificar el fichero a abrir:
   - Clase `FileOutputStream`

     `FileOutputStream fs=new FileOutputStream("fichero.dat");`
   - Crea en `fs` un flujo de salida, pero esta clase sólo tiene un método de escritura genérico.
2. Crear un flujo de salida que permita escribir datos de forma portable.
   - Clase `DataOutputStream`

     ```
     DataOutputStream salida=new DataOutputStream(fs);
     ```
3. Hemos vinculado el objeto `fs` que identifica el nombre del fichero con el objeto `salida` que permite escribir de forma portable.

Objeto `DataOutputStream`

Métodos más frecuentes:

- Todos los métodos son del tipo `void writeXXX(...)` donde XXX es el tipo de dato que escribe:
  1. `void writeInt(int v)`
  2. `void writeChar(int v)`
  3. `void writeDouble(double v)`
  4. `void writeBoolean(boolean v)`
  5. `void writeUTF(String s)`

Ejemplo: Generar un fichero a partir de datos primitivos

```
double[] numeros={12.30,45,67.20,89.99,2.05};
int[] enteros={2,3,4,5,6};
String[] cadenas={"uno","dos","tres","cuatro","cinco"};
  try  {
      fob=new FileOutputStream(args[0]);
      dob=new DataOutputStream(fob);
      System.out.println("Generacion de "+args[0]);
      for(i=0;i<numeros.length;i++)
      {
        dob.writeUTF(cadenas[i]);
        dob.writeInt(enteros[i]);
        dob.writeDouble(numeros[i]);
      }
     }catch (IOException e) {System.err.println(e);  }
// falta cerrar los streams
```


Ficheros binarios: Lectura de datos primitivos

Importante:

Los datos deben ser leídos con el método correspondiente al que fueron escritos y en el mismo orden.

1. Identificar el fichero a abrir:
   - Clase `FileInputStream`

     `FileInputStream fe=new FileInputStream("fichero.dat");`
   - Crea en `fe` un flujo de entrada: esta clase sólo tiene dos métodos genéricos de lectura.
2. Crear un flujo de entrada que permita leer datos de forma portable.
   - Clase `DataInputStream`

     ```
     DataInputStream entrada=new DataInputStream(fe);
     ```
3. Hemos vinculado el objeto `fe` que identifica el nombre del fichero con el objeto `entrada` que permite leer de forma portable.

Objeto `DataInputStream`

Métodos más frecuentes:

- Todos los métodos son del tipo `... readXXX()` donde XXX es el tipo de dato que lee:
  1. `int readInt()`
  2. `char readChar()`
  3. `double readDouble()`
  4. `boolean readBoolean()`
  5. `String readUTF()`
- El final de fichero se detecta cuando salta la excepción `EOFException`.

Ejemplo: Leer el fichero generado en el ejemplo anterior

```
try  {
 fib=new FileInputStream(args[0]);
 dib=new DataInputStream(fib);
 System.out.println("Lectura de "+args[0]);
 try
  {
   while(true) {
    cadena=dib.readUTF();
    entero=dib.readInt();
    numero=dib.readDouble();
    System.out.print("cadenas:"+cadena+" enteros:"+entero);
    System.out.println("numeros:"+numero);}
  }catch(EOFException ex){}
}catch (IOException e) {System.err.println(e);  }
//falta cerrar los streams
```

Ejercicio

Implementar una aplicación que:

1. Lea líneas de la entrada estándar y las escriba en un fichero de texto hasta encontrar el fin de fichero. La entrada introducida en todos los casos serán secuencias de dígitos.
2. Lea el fichero de texto generado en el punto anterior, convierta cada línea leída del fichero en un número y lo escriba en un fichero binario.

```
// para transformar un string en int
entero = Integer.parseInt(cadena);
```

3. Lea el fichero binario creado en el punto anterior y lo muestre por pantalla.

En este ejercicio la salida y la entrada del programa deben ser exactamente iguales.

Lectura/escritura de objetos

- Para poder leer/escribir objetos completos de fichero, dichos objetos deben implementar la interfaz **`Serializable`**.
- Esta interfaz no tiene métodos ni constantes; solo sirve para identificar la semántica de ser serializable.
- Para ello asocia a cada clase serializable un número de versión: `serialVersionUID`.
- Se usa durante la deserialización para verificar que remitente y receptor de un objeto serializado tienen clases que sean compatibles con respecto a la serialización.
- Si el receptor tiene una clase con un `serialVersionUID` diferente que el de la clase del remitente correspondiente se lanza una excepción `InvalidClassException`.

Escritura de objetos

1. Identificar el fichero a abrir:
   - Clase `FileOutputStream`

     `FileOutputStream fs=new FileOutputStream("objetos.dat");`
   - Crea en `fs` un flujo de salida, pero esta clase tiene un método de escritura genérico, no se puede escribir el contenido de un objeto como entidad.
2. Crear un flujo de salida que permita escribir datos de forma portable.
   - Clase `ObjectOutputStream`

```
ObjectOutputStream out=new ObjectOutputStream(fs);
```

3. Hemos vinculado el objeto `fs` que identifica el nombre del fichero con el objeto `out` que permite escribir objetos de forma portable.
4. El método utilizado para escribir objetos en fichero es `void writeObject(Object obj)`.

Lectura de objetos

1. Identificar el fichero a abrir:
   - Clase `FileInputStream`

     `FileInputStream fe=new FileInputStream("objetos.dat");`
   - Crea en `fe` un flujo de entrada: esta clase sólo tiene métodos genéricos de lectura.
2. Crear un flujo de entrada que permita leer y deserializar la secuencia de datos correspondiente a un objeto.
   - Clase `ObjectInputStream`

```
ObjectInputStream in=new ObjectInputStream(fe);
```

3. Hemos vinculado el objeto `fe` que identifica el nombre del fichero con el objeto `in` que permite leer de forma portable.
4. El método utilizado para leer objetos de fichero es `Object readObject()`.

Ejemplo: Serialización (I)

- Clase cuyos objetos vamos a serializar:

```
public class Simple implements Serializable{
private String cadena;
private int valor;
public  Simple(String str1,int inicial){
 if(str1!=null)
  cadena=new String(str1);
 else
  cadena=null;
 valor=inicial;
}
public void muestra(){
 System.out.println("Cadena: "+cadena);
 System.out.println("Entero: "+valor);
}
}
```


Ejemplo: Serialización (II)

- Clase principal:

```
public class Principal{
 try{
   FileOutputStream sal=new FileOutputStream(args[0]);
   ObjectOutputStream out=new ObjectOutputStream(sal);
   Simple[] vector=new Simple[3];
   for(i=0;i<3;i++) vector[i]=new Simple("cadena",i);
   for(i=0;i<3;i++)
         out.writeObject(vector[i]);
   out.close();
   FileInputStream entra=new FileInputStream(args[0]);
   ObjectInputStream in=new ObjectInputStream(entra);
   Simple[] array=new Simple[3];
   for(i=0;i<3;i++){
     try {
        array[i]=(Simple) in.readObject();
        System.out.println("Simple copia"+i+":");
        array[i].muestra();
     }catch (ClassNotFoundException e) {System.err.println(e);}
   }
   in.close();
  }catch(IOException e){
     e.printStackTrace();
     System.exit(0);
  }
}
```


Clases de ficheros en Java

Nombre Clase (java.io)	Tipo información almacenada	Lectura/ escritura información	Acceso datos
BufferedInputStream/ BufferedOutputStream	binaria	bytes	secuencial
BufferedReader/BufferedWriter	texto	ASCII	secuencial
DataInputStream/ DataOutputStream	binaria	datos	secuencial
InputStreamReader/ OutputStreamWriter	texto	ASCII	secuencial
FileInputStream/ FileOutputStream	binaria	bytes	secuencial
FileReader/FileWriter	texto	ASCII	secuencial
ObjectInputStream/ ObjectOutputStream	binaria	objetos	secuencial
PrintStream/PrintWriter	texto	String	secuencial
RandomAccessFile	binaria	bytes	directo