LABORATORIO DE PROGRAMACIÓN ORIENTADA A OBJETOS

DRA. MARICELA BRAVO

CBI DIVISION DE CIENCIAS BASICAS E INGENIERIA
UAM - Azcapotzalco

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
Azcapotzalco

PROGRAMACIÓN CON FLUJOS

LABORATORIO DE PROGRAMACIÓN ORIENTADA A OBJETOS

2

ARCHIVOS

- Los archivos tienen como finalidad guardar datos de forma permanente.
- Una vez que acaba la aplicación los datos almacenados están disponibles para que otra aplicación pueda recuperarlos para su consulta o modificación.

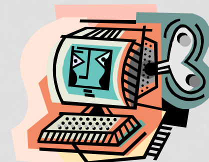
3

ARCHIVOS

La organización de un archivo define la forma en la en que se estructuran u organizan los datos.

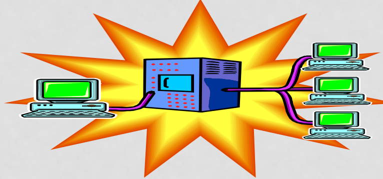
Formas de organización fundamentales:

- **Secuenciales:** los registros se insertan en el archivo en orden de llagada. Las operaciones básicas permitidas son: escribir, añadir al final del archivo y consultar .
- **Directa o aleatoria:** cuando un registro es directamente accesible mediante la especificación de un índice.



Flujos (Streams)

Es una abstracción, que representa a un flujo de datos entre un origen y un destino en Java. Todo proceso de entrada y salida en Java se hace a través de flujos.

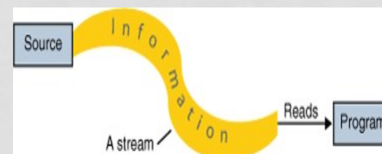


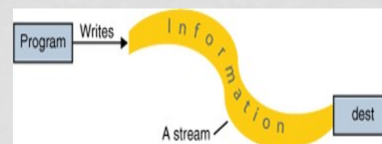
Entre el origen y el destino debe existir un canal , por el que viajan datos.

Cuando se abre un archivo se establece una conexión entre el programa y el dispositivo que contiene ese archivo , por el canal fluirá la secuencia de datos.
Igual ocurre al intentar escribir en un archivo.

STREAMS

- Un stream representa un flujo de información:
  - procedente de una fuente (teclado, file, memoria, red, etc.) o
  - dirigida a un destino (pantalla, file, etc.)
- Los streams comparten una misma interfaz que hace abstracción de los detalles específicos de cada dispositivo de E/S.
- Todas las clases de streams están en el paquete java.io

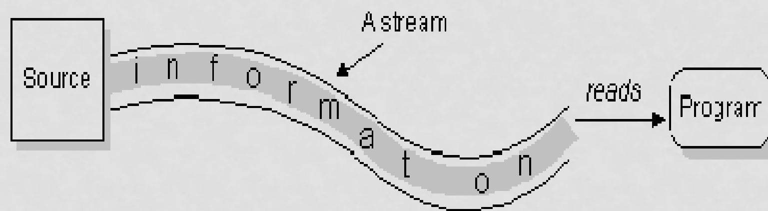




6

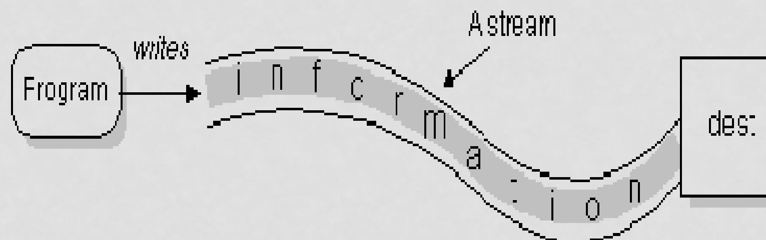
Archivos y Flujos (Streams)

Para obtener información de una fuente un programa abre un stream y lee la información secuencialmente



ARCHIVOS Y FLUJOS (STREAMS)

De igual forma, un programa puede enviar información a un destino externo abriendo un stream al destino y escribiendo la información secuencialmente .



ARCHIVOS Y FLUJOS (STREAMS)

No importa el tipo de datos ni de donde proviene ni a donde se dirige, los algoritmos para la lectura y escritura de datos son escencialmente los mismos

```
Abrir el stream
mientra haya información
   leer información
cerrar el stream
```

```
Abrir el stream
mientra haya información
   escribir la información
cerrar el stream
```

ARCHIVOS Y FLUJOS

- Java posee una colección de clases stream las cuales soportan la lectura y escritura.
- Para utilizar las clases stream el programa deberá importar el paquete java.io donde se encuentran todas las clases necesarias para dar entrada/salida a las aplicaciones.

FILEREADER, FILEWRITER

- Los streams para archivos se manejan con los objetos asociados a la clase File(FileReader y FileWriter) para leer y escribir a un archivo en asociación con los métodos: read() y write().

- Un stream para archivos se puede crear a partir de
  - Un String con el nombre del archivo
  - Un objeto tipo File
  - Un objeto tipo FileDescriptor

ARCHIVOS Y FLUJOS

```
/* El siguiente programa utiliza un objeto FileReader y uno
FileWriter para copiar el contenido de un archivo denominado
cancion.txt a un archivo denominado salida*/

import java.io.*;

public class Copy
{
    public static void main(String[] args) throws IOException
    {
        File inputFile = new File("cancion.txt");
        File outputFile = new File("salida");

        FileReader in = new FileReader(inputFile);
        FileWriter out = new FileWriter(outputFile);
        int c;

        while ((c = in.read()) != -1)
           out.write(c);

        in.close();
        out.close();
    }
}
```

MANIPULACIÓN DE FLUJOS

FLUJOS Y ENTRADA/SALIDA

- Clases básicas para el manejo de archivos en IO
  - FileInputStream, para lectura de un archivo
  - FileOutputStream, para la escritura en un archivo
- Ejemplo:

Para abrir un archivo para "miArchivo.txt" para lectura
FileInputStream fis = new FileInputStream("myArchivo.txt");

Para abrir un archivo “archSalida.txt" para escritura
FileOutputStream fos = new FileOutputStream (“archSalida.txt");

14

MUESTRA EL CONTENIDO DE UN ARCHIVO

```
public class ContenidoArchivo
{
   public static void main(String args[])
   {
      try
      {
         FileInputStream archivo = new FileInputStream("lista.txt");
         byte buffer[] = new byte[50];
         int numBytes;
         do
         {
            numBytes = archivo.read(buffer);
            System.out.write(buffer, 0, numBytes);
         } while (numBytes == buffer.length);
      }
      catch (FileNotFoundException e)
      {
         System.err.println("No se encontro el archivo");
      }
      catch (IOException e)
      {        System.err.println("Fallo la lectura");
      } }}
```

15

FILTROS

- Una vez que se ha abierto un stream (por ejemplo un archivo) se pueden asociar filtros.
- Los filtros hacen que la lectura/escritura sea más eficiente.

- Algunos filtros populares:
- Para los tipos básicos
  - *DataInputStream*, *DataOutputStream*
- Para objetos
  - *ObjectInputStream*, *ObjectOutputStream*

16

ESCRITURA DE DATOS EN UN ARCHIVO USANDO FILTROS

```
import java.io.*;
public class EscrituraconFiltros
{
   public static void main(String args[])
   {
      try
      {
         FileOutputStream fos = new FileOutputStream("datos.dat");
         DataOutputStream dos = new DataOutputStream(fos);
         dos.writeInt(2);
         dos.writeDouble(2.7182818284590451);
         dos.writeDouble(3.1415926535);
         dos.close(); fos.close();
      }
      catch (FileNotFoundException e)
      {
          System.err.println("Archivo no encontrado");
      }
      catch (IOException e) {
         System.err.println("Falla de lectura o escritura");
      }
   }
}
```

17

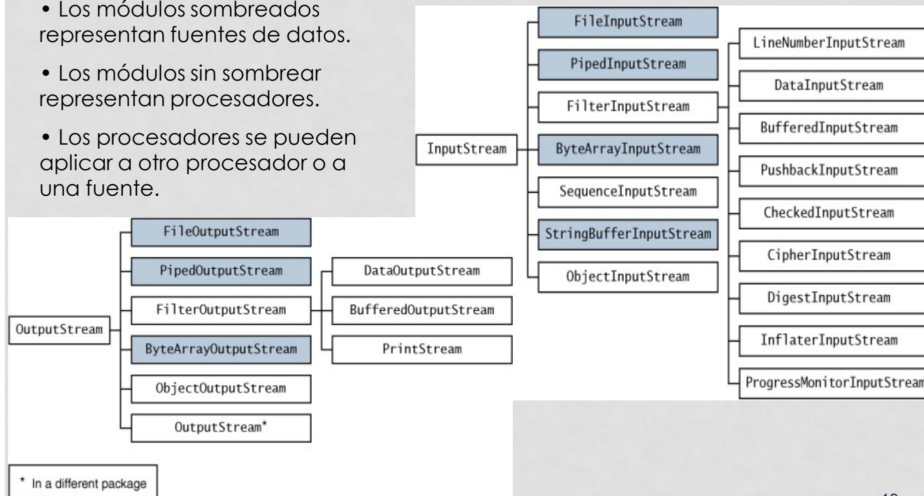
LECTURA CON FILTROS

```
import java.io.*;
public class LecturaconFiltros
{
   public static void main(String args[])
   {
      try {
         FileInputStream fis = new FileInputStream("datos.dat");
         DataInputStream dis = new DataInputStream(fis);
         int n = dis.readInt();
         System.out.println(n);
         for( int i = 0; i < n; i++ ) { System.out.println(dis.readDouble());
         }
         dis.close(); fis.close();
      }
      catch (FileNotFoundException e) {
         System.err.println("Archivo no encontrado");
      }
      catch (IOException e) { System.err.println("Falla de lectura o escritura");
      }
   }
}
```

18

CLASES DE STREAMS (STREAMS ORIENTADOS A BYTE)

- Los módulos sombreados representan fuentes de datos.
- Los módulos sin sombrear representan procesadores.
- Los procesadores se pueden aplicar a otro procesador o a una fuente.



19

SUBCLASES DE INPUTSTREAM

- FileInputStream: lectura de files byte a byte
- ObjectInputStream: lectura de files con objetos.
- FilterInputStream:
  - BufferedInputStream: lectura con buffer, más eficiente.
  - DataInputStream: lectura de tipos de datos primitivos (int, double, etc.).

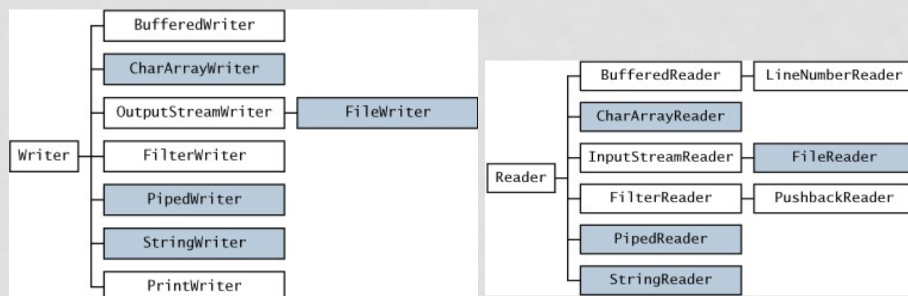
20

SUBCLASES DE OUTPUTSTREAM

- FileOutputStream: escritura de files byte a byte
- ObjectOutputStream: escritura de files con objetos.
- FilterOutputStream:
  - BufferedOutputStream: escritura con buffer, más eficiente.
  - DataOutputStream: escritura de tipos de datos primitivos (int, double, etc.).

CLASES DE STREAMS (STREAMS ORIENTADOS A CARACTER)

- Soportan UNICODE (16 bits para un char).
- Módulos sombreados son fuentes, y sin sombrear son procesadores.



SUBCLASES DE READER

- InputStreamReader: convierte un stream de bytes en un stream de chars.
  - FileReader: se asocia a files de chars para leerlos.
- BufferedReader: proporciona entrada de caracteres a través de un buffer (más eficiencia).

23

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.

24

OTRAS CLASES DE JAVA.IO

- File: permite realizar operaciones habituales con files y directorios.
- RandomAccessFile: permite acceder al n-ésimo registro de un file sin pasar por los anteriores.
- StreamTokenizer: permite "trocear" un stream en tokens.

25

TÍPICOS USOS DE LOS STREAMS (LECTURA POR LÍNEAS)

```
public static void main(String[] args) throws IOException {
  // 1a. Se lee un file línea a línea
  BufferedReader in = new BufferedReader( new
                FileReader("IOStreamDemo.java"));
  String s, s2 = new String();
  while((s = in.readLine())!= null)
       s2 += s + "\n";
  in.close();
  // 1b. Se lee una línea por teclado
  BufferedReader stdin = new BufferedReader( new
                        InputStreamReader(System.in));
  System.out.print("Enter a line:");
  System.out.println(stdin.readLine());
}
```

26

PARSING DE TIPOS BÁSICOS

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

- También están disponibles: parseDouble(), parseLong()

27

TÍPICOS USOS DE LOS STREAMS (ESCRITURA POR LÍNEAS)

```
// throws IOException
String []s = {“hola“, “que”, “tal”};
// Se inicializa s
PrintWriter out1 = new PrintWriter( new
  BufferedWriter( new FileWriter("IODemo.out")));
int lineCount = 1;
for (int i=0; i<s.length(); i++)
  out1.println(lineCount++ + ": " + s[i]);
out1.close();
```

28

DUDAS O COMENTARIOS???