Entrada/Salida en Java
Archivos

Universidad de Costa Rica
Sede de Guanacaste
Informática empresarial
IF-2000 Programación I
M.C.I. Kenneth Sánchez S.

Archivos

- Son el **almacenamiento secundario** por excelencia.

- Existen dos tipos de archivos según el tipo de información almacenada:
  - **archivos de texto**
  - **archivos binarios**

Archivos de texto

- Son aquellos archivos que sólo almacenan datos de tipos carácter o cadenas de caracteres.
- Son utilizados para:
  - Información del tipo registro.
  - Guardar información que no necesita de un formato específico.
- Guardar información de cualquier otro tipo en estos archivos no es una solución óptima.

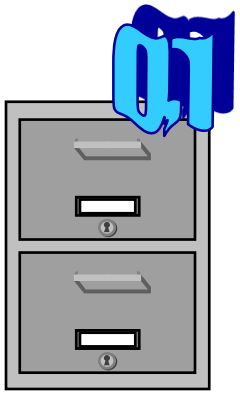
Archivos binarios

- A diferencia de los archivos de texto, **en estos archivos se almacenan datos de todo tipo** (char, long, float, int, etc.).
- Los tipos de datos que se almacenan en estos archivos **se guardan de manera binaria**, de acuerdo a la extensión que tengan los tipos de datos.

Archivos binarios

- Los datos se guardan de manera secuencial, es decir, uno detrás de otro.

- Cuando queremos accesar un dato en este tipo de archivos, tenemos que leer cada dato, hasta encontrar aquel que andamos buscando.



La utilización de objetos de E/S con archivos

- *Existen dos áreas que requieren atención adicional cuando manejamos E/S con archivos:*
  - La inicialización de los objetos de E/S con archivos, puede ser una tarea no tan exitosa. Puede ser que archivo que se desea abrir no encuentre, lo cual lanza la excepción *FileNotFoundException*.
  - Esto podría ser porque la cadena entregada al constructor puede ser el nombre de un archivo inexistente o que no se encuentra en el directorio de trabajo del programa.
  - Al momento de leer o escribir a un archivo se puede lanzar la excepción *IOException*.

Introducción
Archivos en java

- Un archivo en Java, es un flujo externo, una secuencia de bytes almacenados en un dispositivo externo (normalmente en disco).
- Si el archivo se abre para salida, es un flujo de archivo de salida.
- Si el archivo se abre para entrada, es un flujo de archivo de entrada

Archivos en java

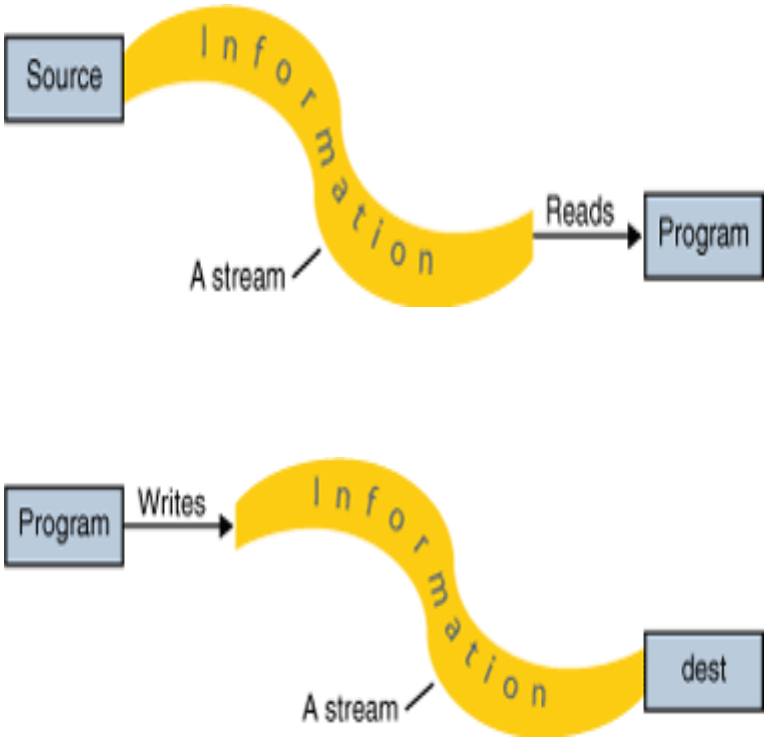
- Los programas leen o escriben en el flujo que puede estar conectado a un dispositivo
- El paquete *java.io* agrupa el conjunto de clases para el manejo de entrada y salida.
- Siempre que se vaya a procesar un archivo se tienen que utilizar clases de este paquete.

Archivos en java

- Los flujos de datos, de caracteres, de bytes se pueden clasificar en flujos de entrada (InputStream) y en flujos de salida(Output Stream)
- Por esto java declara dos clases que derivan directamente de la clase **Object**: *InputStream* y *OutputStream* ambas son **clases abstractas** que declaran métodos que deben redefinirse en sus clases derivadas.

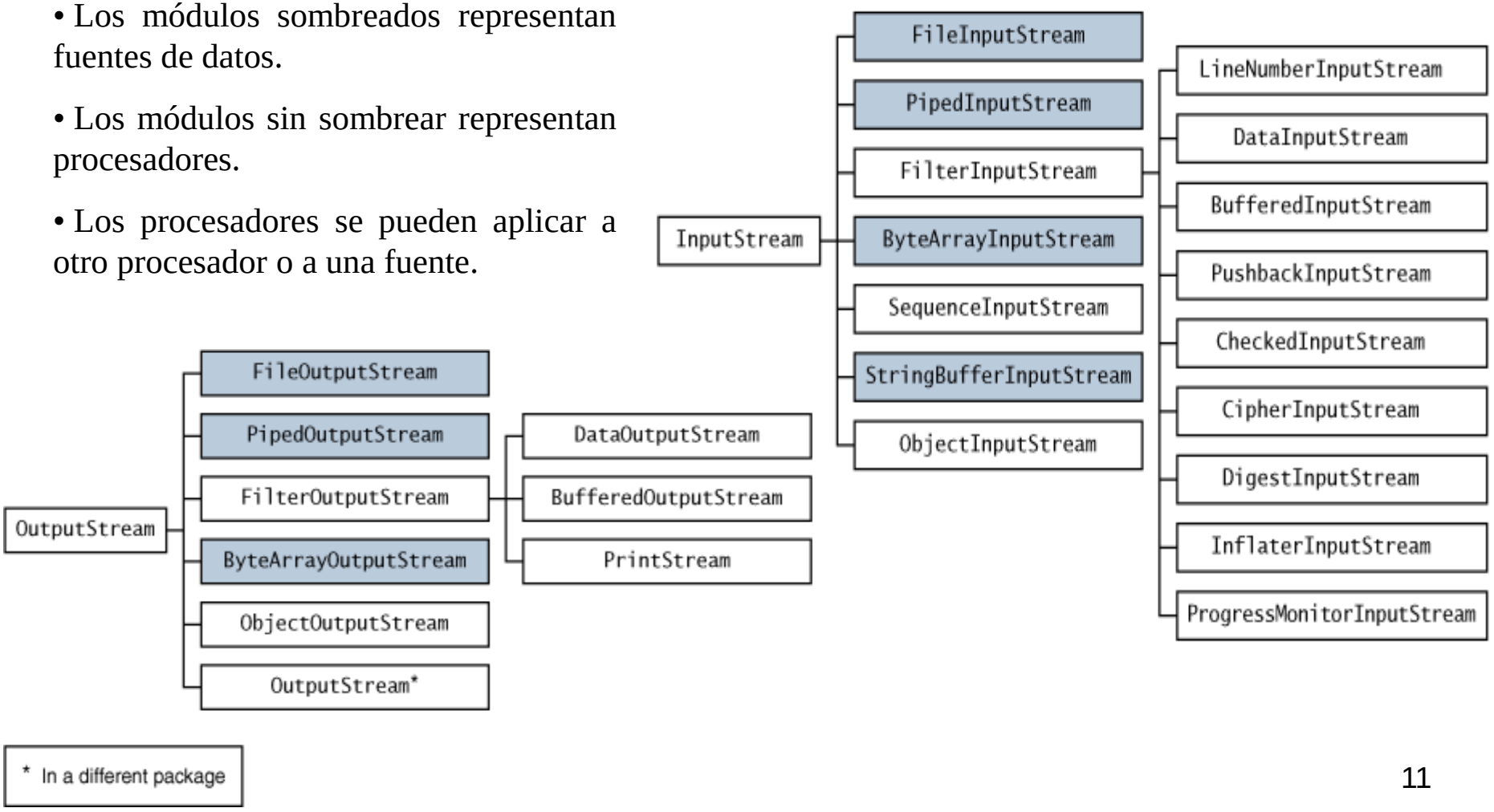
Streams

- 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



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.



Subclases de InputStream

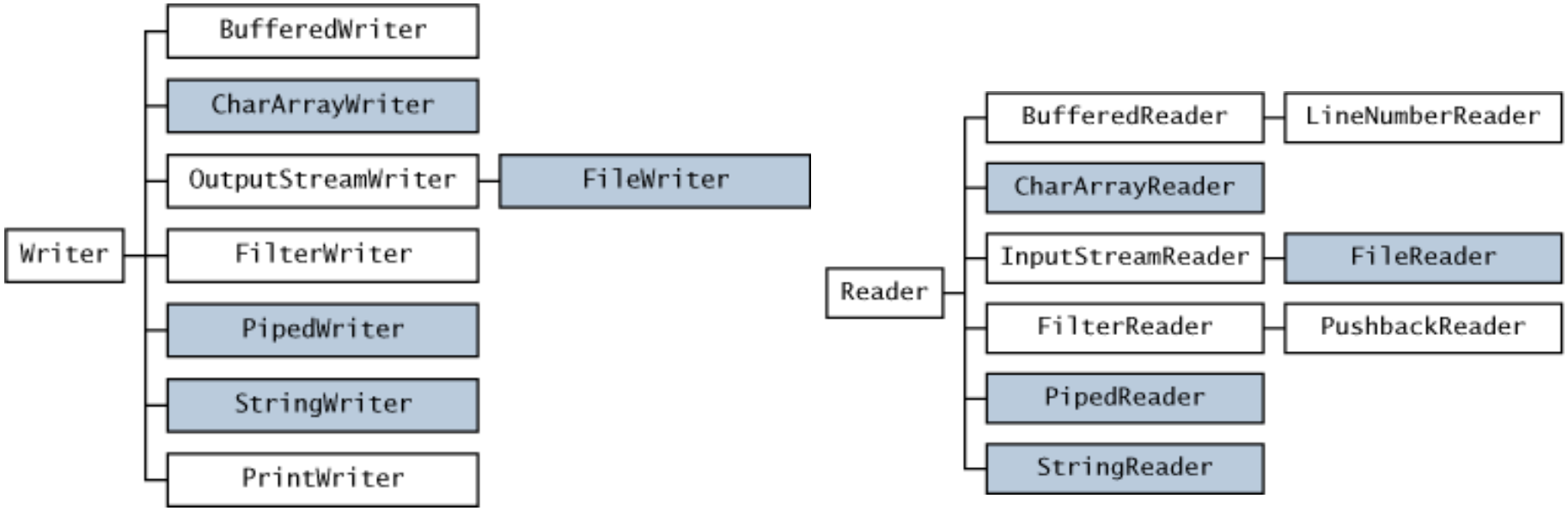
- *FileInputStream:* lectura de files byte a byte
- *ObjectInputStream:* lectura de files con objetos.
- *FilterInputStream:*
  - *BufferedInputStream:* lectura con buffer, más eficiente.
  - *DataInputStream:* lectura de tipos de datos primitivos (int, double, etc.).

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 archivos de chars para leerlos.
- *BufferedReader*: proporciona entrada de caracteres a través de un buffer (más eficiencia).

Subclases de Writer

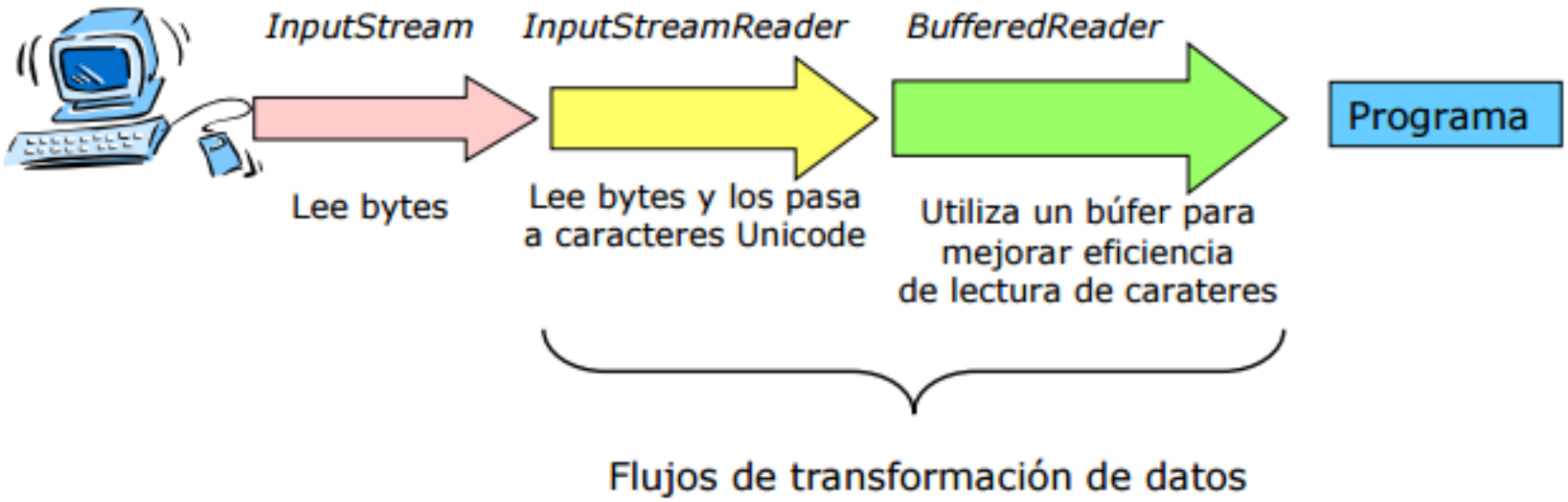
- *OutputStreamWriter*: convierte un stream de bytes en un stream de chars.
  - *FileWriter*: se asocia archivos de chars para modificarlos.
- *BufferedWriter*: proporciona salida de caracteres a través de un buffer (más eficiencia).
- *PrintWriter*: métodos *print()* y *println()* para distintos tipos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

Otras Clases de java.io

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

Combinación de flujos

- Los flujos se pueden combinar para obtener la funcionalidad deseada



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("Ingrese una línea de texto:");
    System.out.println(stdin.readLine());
}
```

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();
```

Típicos Usos de los Streams (escritura de tipos básicos)

```
// throws IOException
DataOutputStream out2 = new
      DataOutputStream(new
            BufferedOutputStream(new
                  FileOutputStream("Data.txt")));

out2.writeDouble(3.14159);
out2.writeBytes("Este es el valor de pi\n");
out2.writeChars( "Este es el valor de pi\n");
out2.writeDouble(1.41413);
out2.writeUTF("Raíz cuadrada de 2");
out2.close();
```

Típicos Usos de los Streams (lectura de tipos básicos)

```
// throws IOException
DataInputStream in5 = new DataInputStream( new
   BufferedInputStream( new
      FileInputStream("Data.txt")));

System.out.println(in5.readDouble());
System.out.println(in5.readLine());
System.out.println(in5.readDouble());
System.out.println(in5.readUTF());
```

Típicos Usos de los Streams (Object Stream)

- Java permite guardar objetos en ficheros, pero esos objetos deben implementar la interfaz *Serializable*:

  public class MySerializableClass implements Serializable { ... }

- Ejemplo:

```
// throws IOException
FileOutputStream out = new FileOutputStream(“ElTiempo.dat");
ObjectOutputStream s = new ObjectOutputStream(out);
s.writeObject("Hoy");
s.writeObject(new Date(1,1,2014));
s.close();
// throws IOException y ClassNotFoundException
FileInputStream in = new FileInputStream(“ElTiempo.dat");
ObjectInputStream s = new ObjectInputStream(in);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
s.close();
```

Ficheros con Objetos (Listas enlazadas)

```
File F = new File (“ListaEnlazada.dat");
try {
FileOutputStream mf ;
mf = new FileOutputStream (F);
DataOutputStream es = new DataOutputStream (mf);
A.Guardar(es);
}catch (FileNotFoundException e){
e.printStackTrace ();
}
```

Ficheros con Objetos (Listas enlazadas) Almacenar

```
public void Guardar(DataOutputStream cliente){
Nodo actual = inicio;
while(actual!=null){
   try{
cliente.writeUTF(actual.dat.nombre);
cliente.writeUTF(actual.dat.DNI);
cliente.writeUTF(actual.dat.NombreComida);
cliente.writeInt(actual.dat.cantidadComida);
cliente.writeInt(actual.dat.cantidadBebida);
cliente.writeUTF(actual.dat.tipoPago);
cliente.writeDouble(actual.dat.MontoCancelar);
   }
   catch ( IOException e){
throw new RuntimeException(e);}
   actual = actual.siguient;
   }
}
```

Crear y recuperar archivo

```
public class Persona  {
private String nombre;
private String cedula;
private float promedio;
Persona(){}

public  String getNombre() {
return nombre;

}
public void setNombre(String nombre) {
this.nombre = nombre;
}
public  String getCedula() {
return cedula;
}
public  void setCedula(String cedula) {
this.cedula = cedula;
}
public  float getPromedio() {
return promedio;
}
public  void setPromedio(float promedio) {
this.promedio = promedio;
} }
```

```
import java.io.*;
import javax.swing.JOptionPane;
public class Registro {

public static void main(String[] args) {

 escribir();
 leer();
                //borrar          String
Strsize=JOptionPane.showInputDialog("cantidad a
ingresar");
                }//main
                /*******escribir******/
private static void escribir(){
                                    try {
                  File f=new File("register.txt");
                FileWriter fw=new FileWriter(f);
                BufferedWriter bw= new BufferedWriter(fw);
                   Persona Obj[]=new Persona[50];//vector
```

Crear y recuperar archivo (cont.)

```
String Strsize=JOptionPane.showInputDialog("cantidad a ingresar");//definiendo tamaño
                    int size=Integer.parseInt(Strsize);//parse
                    for(int x=0;x<size;x++){

                                String nombre=JOptionPane.showInputDialog("Ingrese nombre persona "+"#"+(x+1));
                                                    String cedula=JOptionPane.showInputDialog("Ingrese cedula"+"#"+(x+1));
                                                    String Strpromedio=JOptionPane.showInputDialog("Ingrese promedio:"+"#"+(x+1));
                                float promedio=Float.parseFloat(Strpromedio);

                                Obj[x]=new Persona();
                                Obj[x].setNombre(nombre);
                                Obj[x].setCedula(cedula);

                Obj[x].setPromedio(promedio);
```

```
bw.write(nombre+"\n");
                                bw.write(cedula+"\n");
                                bw.write(Strpromedio+"\n");

                    }
                    bw.close();
                    }catch(Exception ex){ex.printStackTrace();}                    }//

                /*******Leer******/
private static void leer(){
  try {
  File f=new File("register.txt");
FileReader fr=new FileReader(f);
BufferedReader br= new BufferedReader(fr);
Persona Obj[]=new Persona[50];//vector
String Str=null;//no hace falta creo...
int x=0;
float temp=0;//para el parse {
while((Str=br.readLine())!=null){
```

Crear y recuperar archivo (cont.)

```
Obj[x]=new Persona();
Obj[x].setNombre(Str);//asigna
Str=br.readLine();//lee la cedula
Obj[x].setCedula(Str);//asigna
Str=br.readLine();//lee el Promedio
temp=Float.parseFloat(Str);
Obj[x].setPromedio(temp);//asigna

//if(br.readLine()==null){JOptionPane.showMessageDialog(null,
"BREAK");break;}
                                    x++;}//while
                                    br.close();
                                    Mostrar(Obj);//llama a mostrar


                                     }catch(Exception
ex){ex.printStackTrace();}
                    }//met
```

```
private static void Mostrar(Persona[] temp){
                    int x=0;
                    while(temp[x].getNombre()!=null){
String
Strtemp="\nNombre:"+temp[x].getNombre()+"\nCed
ula:"+temp[x].getCedula()+
                    "\nPromedio:"+temp[x].getPromedio();
JOptionPane.showMessageDialog(null,Strtemp);
                    x++;
                    }//while


  }
}
```