

Manejo de Archivos en Java

Índice

1. Introducción
2. Ficheros
 - Creación de un objeto File
 - Comprobaciones y Utilidades
3. Streams de Entrada
 - Objetos FileInputStream
 - ✓ Apertura de un FileInputStream
 - ✓ Lectura de un FileInputStream
 - ✓ Cierre de FileInputStream
 - ✓ Ejemplo: Visualización de un fichero
 - Objetos ObjectInputStream
 - Objetos DataInputStream
 - ✓ Apertura y cierre de DataInputStream
 - ✓ Lectura de un DataInputStream
 - Streams de entrada de URLs
 - ✓ Apertura de un Stream de entrada
4. Streams de Salida
 - Objetos FileOutputStream
 - ✓ Apertura de un FileOutputStream
 - ✓ Escritura en un FileOutputStream
 - ✓ Cierre de FileOutputStream
 - ✓ Ejemplo: Almacenamiento de Información
 - ObjectOutputStream
 - Streams de salida con buffer
 - ✓ Creación de Streams de salida con buffer
 - ✓ Volcado y Cierre de Streams de salida con buffer
 - Streams DataOutput
 - ✓ Apertura y cierre de objetos DataOutputStream
 - ✓ Escritura en un objeto DataOutputStream
 - ✓ Contabilidad de la salida
5. Ficheros de Acceso Aleatorio
 - Creación de un Fichero de Acceso Aleatorio
 - Acceso a la Información
 - Actualización de Información

1. Introducción

El manejo de archivos (persistencia), es un tema fundamental en cualquier lenguaje de programación. Pues nos permite interactuar con los dispositivos de almacenamiento externo para poder mantener la información en el tiempo. Java no es una excepción.

Cuando se desarrollan applets para utilizar en red, hay que tener en cuenta que la entrada/salida directa a fichero es una violación de seguridad de acceso. Muchos usuarios configurarán sus navegadores para permitir el acceso al sistema de ficheros, pero otros no.

Por otro lado, si se está desarrollando una aplicación Java para uso interno, probablemente será necesario el acceso directo a ficheros.

2. Ficheros

Para realizar operaciones sobre los ficheros, necesitamos contar con la información referente sobre un fichero (archivo). La clase **File** proporciona muchas utilidades relacionadas con ficheros y con la obtención de información básica sobre esos ficheros.

➤ Creación de un objeto File

Para crear un objeto File nuevo, se puede utilizar cualquiera de los tres constructores siguientes:

Constructores de la clase File	ejemplo
<code>public File(String pathname) ;</code>	<code>new File("/carpeta/archivo");</code>
<code>public File(String parent, String child) ;</code>	<code>new File("carpeta", "archivo");</code>
<code>public File(File parent, String child) ;</code>	<code>new File(new File("/carpeta"), "archive")</code>
<code>public File(URI uri) ;</code>	<code>new File(new URI(str);</code>

El constructor utilizado depende a menudo de otros objetos File necesarios para el acceso. Por ejemplo, si sólo se utiliza un fichero en la aplicación, el primer constructor es el mejor. Si en cambio, se utilizan muchos ficheros desde un mismo directorio, el segundo o tercer constructor serán más cómodos. Y si el directorio o el fichero es una variable, el segundo constructor será el más útil.

➤ Comprobaciones y Utilidades

Una vez que se haya creado un objeto de la clase File, se puede utilizar uno de los siguientes métodos para obtener información sobre el fichero:

Utilidad	Métodos (tipo y método)
Nombres del fichero	<code>String getName()</code>

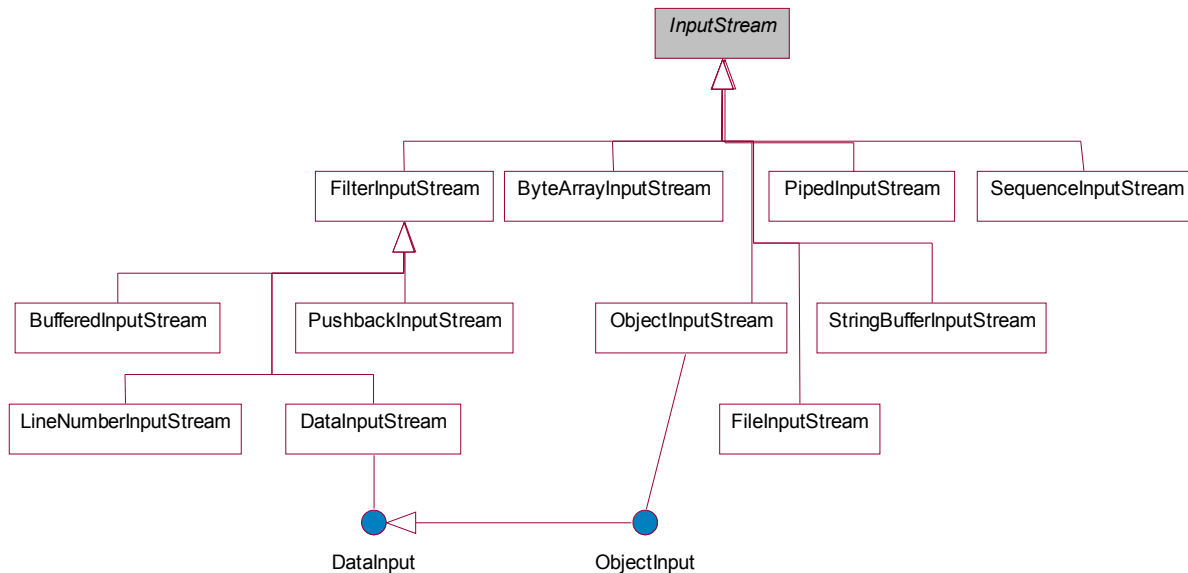
	String getPath() String getAbsolutePath() String getParent() boolean renameTo(File nuevoNombre)
Comprobaciones	boolean exists() boolean canWrite() boolean canRead() boolean isFile() boolean isDirectory() boolean isAbsolute()
Información general del fichero	long lastModified() long length()
Utilidades de directorio	boolean mkdir() String[] list()

Vamos a desarrollar una pequeña aplicación que muestra información sobre los ficheros pasados como argumentos en la línea de comandos:

```
import java.io.*;
class InformacionFichero {
    public static void main( String args[] ) throws IOException {
        if( args.length > 0 ){
            for( int i=0; i < args.length; i++ ){
                File f = new File( args[i] );
                System.out.println( "Nombre: "+f.getName() );
                System.out.println( "Camino: "+f.getPath() );
                if( f.exists() ){
                    System.out.print( "Fichero existente " );
                    System.out.print( (f.canRead() ? "y se puede Leer:"+""));
                    System.out.print( (f.canWrite()?"se puese Escribir:"+""));
                    System.out.println( "." );
                    System.out.println( "La longitud del fichero son "
                        +f.length()+" bytes" );
                }
                else{
                    System.out.println( "El fichero no existe." );
                }
            }
        }
        else
            System.out.println( "Debe indicar un fichero." );
    }
}
```

3. Streams de Entrada

Hay muchas clases dedicadas a la obtención de entrada desde un fichero. Este es el esquema de la jerarquía de clases de entrada por fichero:



➤ Objetos `FileInputStream`

La clase `FileInputStream` típicamente representan ficheros de texto accedidos en orden secuencial, byte a byte. Con `FileInputStream`, se puede elegir acceder a un byte, varios bytes o al fichero completo.

✓ Apertura de un `FileInputStream`

Para abrir un `FileInputStream` sobre un fichero, se le da al constructor un `String` o un objeto `File`: El ejemplo siguiente crea dos `FileInputStreams` que están utilizando el mismo archivo de disco real. Cualquiera de los dos constructores disponibles en esta clase puede lanzar una `FileNotFoundException`.

```

InputStream f1 = new FileInputStream("/archivo.txt");
File f = new File("/archivo.txt");
InputStream f2 = new FileInputStream(f);
FileInputStream fichero=new FileInputStream("/carpeta/archive.txt");
File fichero = new File( "/carpeta/archive.txt" );
FileInputStream ficheroSt= new FileInputStream( fichero );
  
```

Aunque probablemente el primer constructor es el que más se utiliza habitualmente, el segundo permite examinar el archivo más de cerca utilizando sus métodos antes de asignarlo a un flujo de entrada. Cuando se crea un `FileInputStream`, también se abre para lectura. `FileInputStream` sobrescribe seis de los métodos de la clase abstracta `InputStream`. Si se intentan utilizar los métodos `mark` o `reset` en un `FileInputStream` se generará una `IOException`.

✓ Lectura de un FileInputStream

Una vez abierto el FileInputStream, se puede leer de él. El método *read()* tiene muchas opciones:

```
int read();
```

Lee un byte y devuelve -1 al final del stream.

```
int read( byte b[] );
```

Llena todo el array, si es posible. Devuelve el número de bytes leídos o -1 si se alcanzó el final del stream.

```
int read( byte b[],int offset,int longitud );
```

Lee longitud bytes en **b** comenzando por **b[offset]**. Devuelve el número de bytes leídos o -1 si se alcanzó el final del stream.

✓ Cierre de FileInputStream

Cuando se termina con un fichero, existen dos opciones para cerrarlo: explícitamente, o implícitamente cuando se recicla el objeto (el *garbage collector* se encarga de ello).

Para cerrarlo explícitamente, se utiliza el método *close()*:

```
miFicheroSt.close();
```

✓ Ejemplo: Visualización de un fichero

Si la configuración de la seguridad de Java permite el acceso a ficheros, se puede ver el contenido de un fichero en un objeto JTextArea. El código siguiente contiene los elementos necesarios para mostrar un fichero:

```
FileInputStream fis;
JTextArea ta;
public void init() {
    byte b[] = new byte[1024];
    int i;
    // El buffer de lectura se debe hacer lo suficientemente grande
    // o esperar a saber el tamaño del fichero
    String s;
    try {
        fis = new FileInputStream( "/carpeta/archivo.txt");
    }
    catch( FileNotFoundException e ) {

    }
    try {
        i = fis.read( b );
    }
    catch( IOException e ) {

    }
}
```

```
s = new String( b,0 );
ta = new TextArea( s,5,40 );
add( ta );
}
```

Hemos desarrollado un ejemplo, Agenda.java en el que partimos de un fichero *agenda* que dispone de los datos que nosotros deseamos de nuestros amigos, como son: nombre, teléfono y dirección.

Si tecleamos un nombre, buscará en el fichero de datos si existe ese nombre y presentará la información que se haya introducido. Para probar, intentar que aparezca la información de Pepe.

```
import java.io.*;
class Direccion {
    protected String nombre;
    protected String telefono;
    protected String direccion;

    Direccion( String n,String t,String d ) {
        nombre = n;
        telefono = t;
        direccion = d;
    }

    public String getNombre() {
        return( nombre );
    }

    public String getTelefono() {
        return( telefono );
    }

    public String getDireccion() {
        return( direccion );
    }

    public void print() {
        System.out.println( nombre+"\n " +telefono+"\n " +
            direccion );
    }
}

class DireccionArray {
    protected Direccion lista[];
    final int max = 128;
    int tamano = 0;

    DireccionArray() {
        lista = new Direccion[max];
    }

    void printDireccion( String nombre ) {
```

```
        for(int i=0; i < tamano; i++ )
        {
            if( lista[i].getNombre().indexOf( nombre ) != -1 )
                lista[i].print();
        }
    }

    void addDireccion( Direccion direccion ) {
        if( tamano == max )
            System.exit( 1 );
        lista[tamano++] = direccion;
    }
}

public class Agenda {
    DireccionArray lista;
    FileInputStream agFichero;
    final int longLinea = 32;

    public static void main( String argv[] ) {
        Agenda agenda = new Agenda();
        agenda.bucle();
    }

    Agenda() {
        lista = cargaDirecciones();
    }

    void bucle() {
        String nombre;
        System.out.println( "Introduzca un nombre o <Enter>" );
        try {
            while( !"".equals( nombre = leeEntrada( System.in ) ) )
            {
                lista.printDireccion( nombre );
                System.out.println(
                    "Introduzca un nombre o <Enter>" );
            }
        }
        catch( Exception e ) {

        }
    }

    String leeEntrada( InputStream entrada ) throws IOException {
        byte chars[] = new byte[longLinea];
        int contador = 0;
        while( contador < longLinea &&( chars[contador++] =
(byte)entrada.read() ) != '\n' )
            if( chars[contador-1] == -1 )
                return( null );
        return( new String( chars,0,0,contador-1 ) );
    }

    Direccion cargaDireccion() throws IOException {
```

```

        String nombre = leeEntrada( agFichero );
        if( nombre == null )
            return( null );
        String telefono = leeEntrada( agFichero );
        String direccion = leeEntrada( agFichero ) + "\n ";
        +leeEntrada( agFichero ) + "\n " +leeEntrada( agFichero );
        return( new Direccion( nombre,telefono,direccion ) );
    }
    DireccionArray cargaDirecciones() {
        DireccionArray lista = new DireccionArray();
        Direccion nuevaDireccion;
        try {
            agFichero = new FileInputStream( "agenda" );
        }
        catch( FileNotFoundException e ) {
            System.out.println( "No hay fichero de Agenda" );
            return( lista );
        }
        while( true ) {
            try {
                nuevaDireccion = cargaDireccion();
                if( nuevaDireccion == null )
                    return( lista );
                lista.addDireccion( nuevaDireccion );
            }
            catch( Exception e ) {
                System.out.println( "Error cargando Agenda " );
                System.exit( 1 );
            }
        }
    }
}

```

➤ Objetos ObjectInputStream

Para hacer operaciones con el archivo binario serializado abierto se usa objetos de la clase ObjectInputStream, en este caso se usa para leer archivo y se usa el método readObject.

Ejemplo:

```

private A leer() throws IOException, ClassNotFoundException{
    input = new ObjectInputStream(new FileInputStream(fila));
    A obj = null;
    if( input != null){
        obj = (A) input.readObject();
    }
    if ( input != null)
        input.close();
    return obj;
}

```

➤ Objetos DataInputStream

Los objetos `DataInputStream` se comportan como los `FileInputStream`. Los streams de datos pueden leer cualquiera de las variables de tipo nativo, como *floats*, *ints* o *chars*. Generalmente se utilizan `DataInputStream` con ficheros binarios.

✓ Apertura y cierre de `DataInputStream`

Para abrir y cerrar un objeto `DataInputStream`, se utilizan los mismos métodos que para `FileInputStream`:

```
DataInputStream miDStream;
FileInputStream miFStream;

// Obtiene un controlador de fichero
miFStream = new FileInputStream("/etc/ejemplo.dbf");
//Encadena un fichero de entrada de datos
miDStream = new DataInputStream(miFStream);

// Ahora se pueden utilizar los dos streams de entrada para
// acceder al fichero (si se quiere...)
miFStream.read( b );
i = miDStream.readInt();

// Cierra el fichero de datos explícitamente
//Siempre se cierra primero el fichero stream de mayor nivel
miDStream.close();
miFStream.close();
```

✓ Lectura de un `DataInputStream`

Al acceder a un fichero como `DataInputStream`, se pueden utilizar los mismos métodos *read()* de los objetos `FileInputStream`. No obstante, también se tiene acceso a otros métodos diseñados para leer cada uno de los tipos de datos:

```
byte readByte()
int readUnsignedByte()
short readShort()
int readUnsignedShort()
char readChar()
int readInt()
long readLong()
float readFloat()
double readDouble()
String readLine()
```

Cada método leerá un objeto del tipo pedido.

Para el método *String readLine()*, se marca el final de la cadena con `\n`, `\r`, `\r\n` o con EOF.

Para leer un *long*, por ejemplo:

```
long numeroSerie;
numeroSerie = miDStream.readLong();
```

➤ Streams de entrada de URLs

Además del acceso a ficheros, Java proporciona la posibilidad de acceder a URLs como una forma de acceder a objetos a través de la red. Se utiliza implícitamente un objeto URL al acceder a sonidos e imágenes, con el método *getDocumentBase()* en los applets:

```
String imagenFich = new String( "imagenes/pepe.gif" );  
imagenes[0] = getImage( getDocumentBase(),imagenFich );
```

No obstante, se puede proporcionar directamente un URL, si se quiere:

```
URL imagenSrc;  
imagenSrc = new URL( "http://enterprise.com/~info" );  
imagenes[0] = getImage( imagenSrc,"imagenes/pepe.gif" );
```

✓ Apertura de un Stream de entrada

También se puede abrir un stream de entrada a partir de un URL. Por ejemplo, se puede utilizar un fichero de datos para un applet:

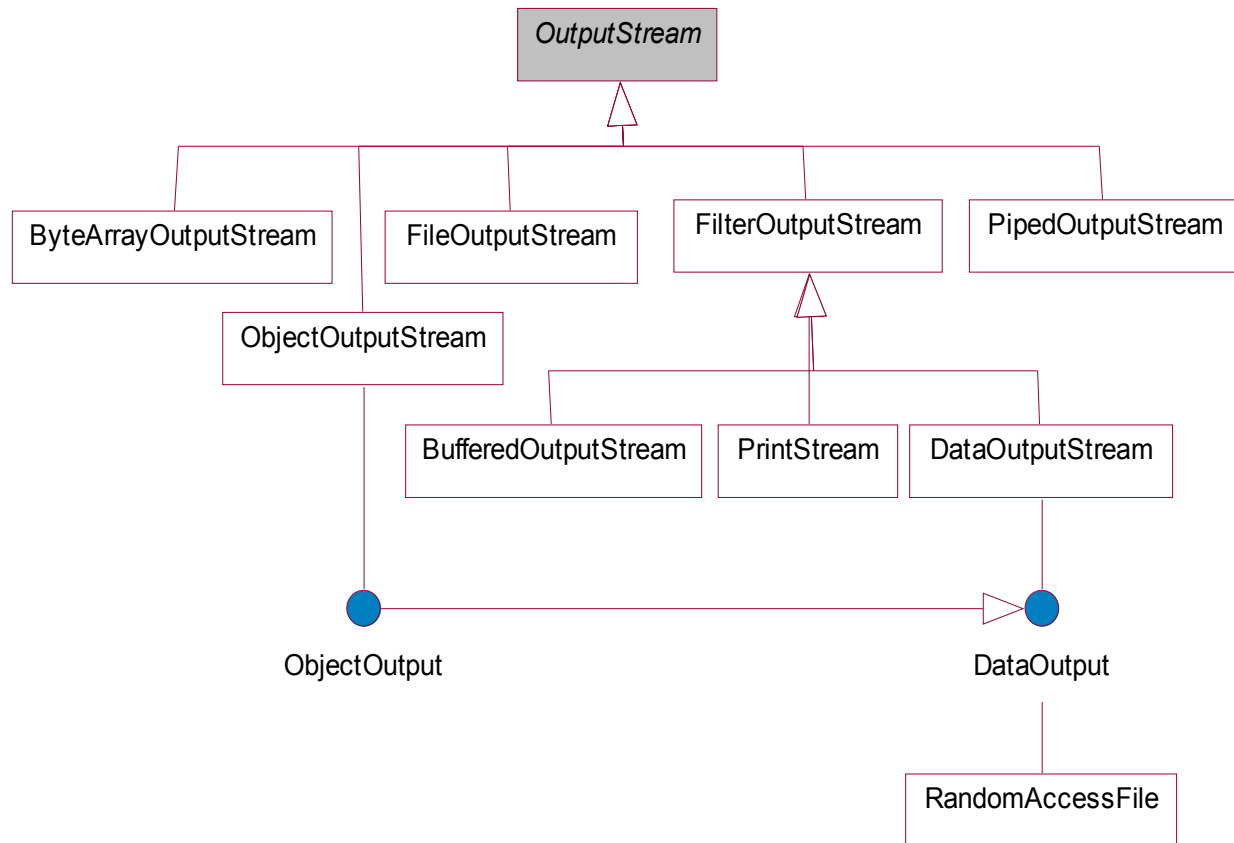
```
InputStream is;  
byte buffer[] = new byte[24];  
is = new URL( getDocumentBase(),datos).openStream();
```

Ahora se puede utilizar **is** para leer información de la misma forma que se hace con un objeto *FileInputStream*:

```
is.read( buffer,0,buffer.length );
```

4. Streams de Salida

La contrapartida necesaria de la lectura de datos es la escritura de datos. Como con los Streams de entrada, las clases de salida están ordenadas jerárquicamente:



Examinaremos las clases **`FileOutputStream`** y **`DataOutputStream`** para complementar los streams de entrada que se han visto.

➤ Objetos `FileOutputStream`

Los objetos `FileOutputStream` son útiles para la escritura de ficheros de texto. Como con los ficheros de entrada, primero se necesita abrir el fichero para luego escribir en él.

✓ Apertura de un `FileOutputStream`

Para abrir un objeto `FileOutputStream`, se tienen las mismas posibilidades que para abrir un fichero stream de entrada. Se le da al constructor un `String` o un objeto `File`.

```
FileOutputStream miFicheroSt;  
miFicheroSt = new FileOutputStream( "/etc/kk" );
```

Como con los streams de entrada, también se puede utilizar:

```
File miFichero  
FileOutputStream miFicheroSt;  
miFichero = new File( "/etc/kk" );  
miFicheroSt = new FileOutputStream( miFichero );
```

✓ Escritura en un `FileOutputStream`

Una vez abierto el fichero, se pueden escribir bytes de datos utilizando el método `write()`. Como con el método `read()` de los streams de entrada, tenemos tres posibilidades:

```
void write( int b );
Escribe un byte.
void write( byte b[] );
Escribe todo el array, si es posible.
void write( byte b[],int offset,int longitud );
Escribe longitud bytes en b comenzando por b[offset].
```

✓ Cierre de `FileOutputStream`

Cerrar un stream de salida es similar a cerrar streams de entrada. Se puede utilizar el método explícito:

```
miFicheroSt.close();
```

O, se puede dejar que el sistema cierre el fichero cuando se recicle `miFicheroSt`.

✓ Ejemplo: Almacenamiento de Información

Este programa, pregunta al usuario una lista de nombres y números de teléfono. Cada nombre y número se añade a un fichero situado en una localización fija. Para indicar que se ha introducido toda la lista, el usuario especifica "Fin" ante la solicitud de entrada del nombre.

Una vez que el usuario ha terminado de teclear la lista, el programa creará un fichero de salida que se mostrará en pantalla o se imprimirá. Por ejemplo:

```
475123,Juanito
564878,Luisa
123456,Pepe
347698,Antonio
354762,Maria
```

El código fuente del programa es el siguiente:

```
import java.io.*;
class Telefonos {
    static FileOutputStream fos;
    public static final int longLinea = 81;
    public static void main( String args[] ) throws IOException {
        byte tfno[] = new byte[longLinea];
        byte nombre[] = new byte[longLinea];
        fos = new FileOutputStream( "telefono.dat" );
        while( true ) {
            System.err.println( "Teclee un nombre ('Fin' termina)" );
            leeLinea( nombre );
            if( "fin".equalsIgnoreCase( new String( nombre,0,3 ) ) )
```

```

        break;
        System.err.println( "Teclee el numero de telefono" );
        leeLinea( tfno );
        for( int i=0; tfno[i] != 0; i++ )
            fos.write( tfno[i] );
        fos.write( ',' );
        for( int i=0; nombre[i] != 0; i++ )
            fos.write( nombre[i] );
        fos.write( '\n' );
    }
    fos.close();
}

private static void leeLinea( byte linea[] ) throws IOException {
    int b = 0;
    int i = 0;
    while( ( i < ( longLinea-1 ) ) && ( ( b = System.in.read() ) != '\n' ) )
        linea[i++] = (byte)b;
    linea[i] = (byte)0;
}
}

```

➤ Objetos ObjectOutputStream

Los objetos de la clase ObjectOutputStream nos permite escribir en el archivo, o sea proporciona un flujo de comunicación con los dispositivos de almacenamiento.

Ejemplo:

```

File fila=new File("miArchivo.bin");
FileOutputStream fos=new FileOutputStream(fila);
ObjectOutputStream out=new ObjectOutputStream(fos);

```

Luego se escribe con el metodo writeObject(objeto), de la clase ObjectOutputStream.

```

if (out != null)
    out.writeObject(obj);
if ( out != null)
    out.close();

```

➤ Streams de salida con buffer

Si se trabaja con gran cantidad de datos, o se escriben muchos elementos pequeños, será una buena idea utilizar un stream de salida con buffer. Los streams con buffer ofrecen los mismos métodos de la clase **FileOutputStream**, pero toda salida se almacena en un buffer. Cuando se llena el buffer, se envía a disco con una única operación de escritura; o, en caso necesario, se puede enviar el buffer a disco en cualquier momento.

✓ Creación de Streams de salida con buffer

Para crear un stream BufferedOutput, primero se necesita un stream FileOutput normal; entonces se le añade un buffer al stream:

```
FileOutputStream miFileStream;  
BufferdOutpurStream miBufferStream;  
// Obtiene un controlador de fichero  
miFileStream = new FileOutputStream( "/tmp/kk" );  
// Encadena un stream de salida con buffer  
miBufferStream = new BufferedOutputStream( miFileStream );
```

✓ Volcado y Cierre de Streams de salida con buffer

Al contrario que los streams FileOutput, cada escritura al buffer no se corresponde con una escritura en disco. A menos que se llene el buffer antes de que termine el programa, cuando se quiera volcar el buffer explícitamente se debe hacer mediante una llamada a *flush()*:

```
// Se fuerza el volcado del buffer a disco  
miBufferStream.flush();  
// Cerramos el fichero de datos. Siempre se ha de cerrar primero el  
// fichero stream de mayor nivel  
miBufferStream.close();  
miFileStream.close();
```

➤ Streams DataOutput

Java también implementa una clase de salida complementaria a la clase **DataInputStream**. Con la clase **DataOutputStream**, se pueden escribir datos binarios en un fichero.

✓ Apertura y cierre de objetos DataOutputStream

Para abrir y cerrar objetos **DataOutputStream**, se utilizan los mismos métodos que para los objetos **FileOutputStream**:

```
DataOutputStream miDataStream;  
FileOutputStream miFileStream;  
BufferedOutputStream miBufferStream;  
  
// Obtiene un controlador de fichero  
miFileStream = new FileOutputStream( "/tmp/kk" );  
// Encadena un stream de salida con buffer (por eficiencia)  
miBufferStream = new BufferedOutputStream( miFileStream );  
// Encadena un fichero de salida de datos  
miDataStream = new DataOutputStream( miBufferStream );  
  
// Ahora se pueden utilizar los dos streams de entrada para  
// acceder al fichero (si se quiere)  
miBufferStream.write( b );  
miDataStream.writeInt( i );  
  
// Cierra el fichero de datos explícitamente. Siempre se cierra  
// primero el fichero stream de mayor nivel  
miDataStream.close();  
miBufferStream.close();  
miFileStream.close();
```

✓ Escritura en un objeto `DataOutputStream`

Cada uno de los métodos `write()` accesibles por los `FileOutputStream` también lo son a través de los `DataOutputStream`. También encontrará métodos complementarios a los de `DataInputStream`:

```
void writeBoolean( boolean b );
void writeByte( int i );
void writeShort( int i );
void writeChar( int i );
void writeInt( int i );
void writeFloat( float f );
void writeDouble( double d );
void writeBytes( String s );
void writeChars( String s );
```

Para las cadenas, se tienen dos posibilidades: bytes y caracteres. Hay que recordar que los bytes son objetos de 8 bits y los caracteres lo son de 16 bits. Si nuestras cadenas utilizan caracteres Unicode, debemos escribirlas con `writeChars()`.

✓ Contabilidad de la salida

Otra función necesaria durante la salida es el método `size()`. Este método simplemente devuelve el número total de bytes escritos en el fichero. Se puede utilizar `size()` para ajustar el tamaño de un fichero a múltiplo de cuatro. Por ejemplo, de la forma siguiente:

```
int numBytes = miDataStream.size() % 4;
for( int i=0; i < numBytes; i++ )
    miDataStream.write( 0 );
```

5. Ficheros de Acceso Aleatorio

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.

➤ Creación de un Fichero de Acceso Aleatorio

Hay dos posibilidades para abrir un fichero de acceso aleatorio:
Con el nombre del fichero:

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

Con un objeto `File`:

```
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`). Por ejemplo, se puede abrir un fichero de una base de datos para actualización:

```
RandomAccessFile miRAFile;  
miRAFile = new RandomAccessFile( "/tmp/kk.dbf","rw" );
```

➤ Acceso a la Información

Los objetos `RandomAccessFile` esperan información de lectura/escritura de la misma manera que los objetos `DataInput/DataOutput`. Se tiene acceso a todas las operaciones *read()* y *write()* de las clases **`DataInputStream`** y **`DataOutputStream`**.

También se tienen muchos métodos para moverse dentro de un fichero:

```
long getFilePointer();
```

Devuelve la posición actual del puntero del fichero

```
void seek( long pos );
```

Coloca el puntero del fichero en una posición determinada. La posición se da como un desplazamiento en bytes desde el comienzo del fichero. La posición 0 marca el comienzo de ese fichero.

```
long length();
```

Devuelve la longitud del fichero. La posición *length()* marca el final de ese fichero.

➤ Actualización de Información

Se pueden utilizar ficheros de acceso aleatorio para añadir información a ficheros existentes:

```
miRAFile = new RandomAccessFile( "/tmp/kk.log","rw" );  
miRAFile.seek( miRAFile.length() );  
// Cualquier write() que hagamos a partir de este punto del código  
// añadirá información al fichero
```

Vamos a ver un pequeño ejemplo, `Log.java`, que añade una cadena a un fichero existente:

```
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();  
    }  
}
```


Ejemplo 01. Crear una aplicación usando archivos serializados con las opciones de nuevo archivo, guardar archivo, guardar como, abrir, salir, etc.

Crearemos una aplicación sencilla para poder entender mejor el uso de las clases que nos permite el manejo de archivos, para ello definimos una clase llamado [Archivo.java](#) que nos permite manipular los archivos. Esta clase es una plantilla que nos permite manejar cualquier tipo de objetos serializados

```
/*
    Name:                Archivo®
    Author:              ALEJANDRO REYES MARZANO
    Description:          es una clase que nos permite manejar archivos serializados
                        de cualquier objeto. es una plantilla
    Date:                20/06/09 15:17
    Copyright:           ©2008-2009 alemsoft corp
*/
package datos;
import java.io.*;
public class Archivo <A>{
    /**file sirve para encapsular la interaccion de nuestros programas con el
    sistema
    * de archivos. Mediante la clase File no nos limitamos a leer el
    contenido de un
    * archivo, ademas podemos obtener toda la informacion del archivo
    nombre, fecha etc
    **/
    private File fila;

    /**La clase InputStream nos sirve para leer datos del cliente*/
    private ObjectInputStream input;

    /**es un objeto que nos permite escribir en el archivo*/
    private ObjectOutputStream output;

    /**construye un archivo con el objeto file en la que se encapsula el
    archivo con
    * todas las propiedades nombre, fecha, etc*/
    public Archivo(File fila) {
        this.fila=fila;
    }

    private void escribir(A obj) throws IOException{
        output = new ObjectOutputStream(new FileOutputStream(fila));
        if (output != null)
            output.writeObject(obj);
        if ( output != null)
            output.close();
    }

    private A leer() throws IOException, ClassNotFoundException{
        input = new ObjectInputStream(new FileInputStream(fila));
        A obj = null;
        if( input != null){
            obj = (A) input.readObject();
        }
    }
}
```

```

    }
    if ( input != null)
        input.close();
    return obj;
}

public boolean salvar(A objetos){
    try{
        if(!fila.exists())
            fila.createNewFile();
        escribir(objetos);
        return true;
    }
    catch(java.io.IOException excepcion){
        javax.swing.JOptionPane.showMessageDialog(null, "Error de archivo
inexistente\n"+excepcion.getMessage());
        return false;
    }
}

public A recuperar(){
    A aux=null;
    try{
        aux=leer();
        return aux;
    }
    catch(java.io.EOFException eof){
        return aux;
    }
    catch(java.io.IOException ex){
        javax.swing.JOptionPane.showMessageDialog(null, "Error en la
lectura de archivo\n"+ex.getMessage());
        return aux;
    }
    catch (ClassNotFoundException f) {
        javax.swing.JOptionPane.showMessageDialog(null, "Archivo
inexistente"
        + f.getMessage());
        return aux;
    }
}
}

```

Aquí crearemos la clase [Persona.java](#), una clase serializado con sus respectivos atributos y métodos. Es una clase sencilla pues aquí se trata de explicar el uso de archivos, ya el lector puede implementar clases mas sofisticadas de acuerdo a sus necesidades

```

/*
Name:          Persona®
Author:        ALEJANDRO REYES MARZANO
Description:    es una clase que usaremos para nuestra aplicacion
Date:          20/06/09 15:17

```

```
Copyright:      ©2008-2009 alemsoft corp
*/

package datos;
public class Persona implements java.io.Serializable{
    private String nombre;
    private String apellidos;
    private String dni;
    private int edad;

    /**
     * @construye un objeto persona y establece los valores nulos
     * a los parametros correspondientes*/
    public Persona() {

    }

    /**
     * @construye un objeto persona y establece los atributos con los
     * datos pasados como parametro */
    public Persona(String nombre, String apellidos, String dni, int edad) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.dni = dni;
        this.edad = edad;
    }

    /**
     * @returna el nombre
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * @asigna el nombre
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * @returna las apellidos
     */
    public String getApellidos() {
        return apellidos;
    }

    /**
     * @asigna los apellidos
     */
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    /**
```

```
        * @returna el dni
        */
    public String getDni() {
        return dni;
    }

    /**
     * @asigna el dni
     */
    public void setDni(String dni) {
        this.dni = dni;
    }

    /**
     * @returna la edad
     */
    public int getEdad() {
        return edad;
    }

    /**
     * @asigna la edad
     */
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

Clase [Personas.java](#), es una clase que encapsula un objeto de la clase HashMap que nos permite manejar como una estructura donde se almacenaran las personas registradas. El lector puede implementar mas métodos como métodos de búsqueda y otros.

```
/*
    Name:          Personas®
    Author:         ALEJANDRO REYES MARZANO
    Description:    es una estructura de objetos
    Date:          20/06/09 15:17
    Copyright:     ©2008-2009 alemsoft corp
*/
package datos;
public class Personas implements java.io.Serializable{

    private java.util.HashMap<String ,Persona> personas;

    public Personas() {
        personas =new java.util.HashMap<String ,Persona>();
    }

    /**
     *metodo que nos permite insertar un objeto persona a la lista de
     personas.
     *recibe como parametro un objeto persona y luego obtiene el dni como
     identificador
     *con ese identificador busca en la lista si ya se encuentra o no el
     objeto
    */
}
```

```

        *si lo encuentra retorna el objeto encontrado con ese identificador, si
no encuentra
        *retorna null. El valor se asigna en la variable aux, si es null se
inserta en otro
        *caso no se inserta. para saber si inserto o se usa un boolean y se
retorna ese valor*/
    public boolean adicionarPersona(Persona persona){
        boolean insertado=false;
        Persona aux=getPersonas().get(persona.getDni());
        if(aux==null){
            getPersonas().put(persona.getDni(), persona);
            insertado=true;
        }
        return insertado;
    }

    /**
     * @returna la lista personas
     */
    public java.util.HashMap<String, Persona> getPersonas() {
        return personas;
    }

    /**
     * @establece la lista personas con el parametro pasado
     */
    public void setPersonas(java.util.HashMap<String, Persona> personas) {
        this.personas = personas;
    }

    public void guardar(java.io.File file){
        Archivo archivo =new Archivo(file);
        archivo.salvar(personas);
    }

    public void cargar(java.io.File file){
        Archivo archivo =new Archivo(file);
        personas=(java.util.HashMap<String ,Persona>)archivo.recuperar();
    }
}

```

La clase [FrameArchivo.java](#) es la clase principal donde se muestra la aplicación

```

/*
Name:                FrameArchivo®
Author:              ALEJANDRO REYES MARZANO
Description:          es la clase principal
Date:                20/06/09 15:17
Copyright:           ©2008-2009 alemsoft corp
*/
package ventanas;

```

```
import datos.*;
import javax.swing.*;
import java.io.*;

public class FrameArchivo extends javax.swing.JFrame {
    Personas personas=new Personas();
    private JFileChooser jFileChooserOperaciones;
    private boolean dirty=false;
    private String currFileName=null;
    private Archivo archivo;

    public FrameArchivo() {
        super(".:Manejo de Archivos:.");
        initComponents();
        jFileChooserOperaciones = new JFileChooser();
        this.jMenuItemNuevo.addActionListener(new Nuevo(this));
        this.jMenuItemGuardarC.addActionListener(new GuardarArchivo(this));
        this.jMenuItemAbrir.addActionListener(new AbrirArchivo(this));
        this.jMenuItemSalir.addActionListener(new Salir(this));
        this.jMenuItemGuardar.addActionListener(new Guardar(this));
    }

    @SuppressWarnings("unchecked")
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jTextFieldNOMBRE = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        jTextFieldApellidos = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        jTextFieldDni = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        jTextFieldEdad = new javax.swing.JTextField();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextAreaMostrar = new javax.swing.JTextArea();
        jButtonAgregar = new javax.swing.JButton();
        jButtonMostrar = new javax.swing.JButton();
        jMenuBar1 = new javax.swing.JMenuBar();
        jMenu1 = new javax.swing.JMenu();
        jMenuItemNuevo = new javax.swing.JMenuItem();
        jMenuItemAbrir = new javax.swing.JMenuItem();
        jMenuItemGuardar = new javax.swing.JMenuItem();
        jMenuItemGuardarC = new javax.swing.JMenuItem();
        jSeparator1 = new javax.swing.JSeparator();
        jMenuItemSalir = new javax.swing.JMenuItem();
        jMenu2 = new javax.swing.JMenu();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jPanel1.setLayout(new java.awt.GridLayout(4, 2, 5, 5));

        jLabel1.setText("Nombre");
        jPanel1.add(jLabel1);
        jPanel1.add(jTextFieldNOMBRE);

        jLabel2.setText("Apellidos");
```

```
jPanel1.add(jLabel2);
jPanel1.add(jTextFieldApellidos);

jLabel3.setText("Dni");
jPanel1.add(jLabel3);
jPanel1.add(jTextFieldDni);

jLabel4.setText("Edad");
jPanel1.add(jLabel4);
jPanel1.add(jTextFieldEdad);

jTextAreaMostrar.setColumns(20);
jTextAreaMostrar.setRows(5);
jScrollPane.setViewportView(jTextAreaMostrar);

jButtonAgregar.setText("Agregar");
jButtonAgregar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonAgregarActionPerformed(evt);
    }
});

jButtonMostrar.setText("Mostrar");
jButtonMostrar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonMostrarActionPerformed(evt);
    }
});

jMenu1.setText("Archivo");

jMenuItemNuevo.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.
    awt.event.KeyEvent.VK_N, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemNuevo.setText("Nuevo Lista");
jMenu1.add(jMenuItemNuevo);

jMenuItemAbrir.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.
    awt.event.KeyEvent.VK_A, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemAbrir.setText("Abrir");
jMenu1.add(jMenuItemAbrir);

jMenuItemGuardar.setAccelerator(javax.swing.KeyStroke.getKeyStroke(ja
    va.awt.event.KeyEvent.VK_G, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemGuardar.setText("Guardar");
jMenu1.add(jMenuItemGuardar);

jMenuItemGuardarC.setAccelerator(javax.swing.KeyStroke.getKeyStroke(ja
    va.awt.event.KeyEvent.VK_C, java.awt.event.InputEvent.CTRL_MASK));
jMenuItemGuardarC.setText("Guardar Como ...");
jMenu1.add(jMenuItemGuardarC);
jMenu1.add(jSeparator1);

jMenuItemSalir.setText("Salir");
jMenu1.add(jMenuItemSalir);

jMenuBar1.add(jMenu1);
```

```

jMenu2.setText("Otros");
jMenuBar1.add(jMenu2);

setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(67, 67, 67)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jButtonMostrar)
                        .addComponent(jButtonAgregar)))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addContainerGap(41, Short.MAX_VALUE))
                )
            .setVerticalGroup(
                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jButtonAgregar)
                                .addGap(18, 18, 18)
                                .addComponent(jButtonMostrar)))
                        .addGap(33, 33, 33)
                        .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                    )
            )
        )
    );

    pack();
}

```



```
private void jButtonAgregarActionPerformed(java.awt.event.ActionEvent evt)
{
    boolean f=false;
    if(this.verString(this.datos().getDni()))
        f=personas.adicionarPersona(this.datos());
    this.reset();
    if(!f){
        javax.swing.JOptionPane.showMessageDialog(this,"ya se encuentra
registrado");
    }
}

private void jButtonMostrarActionPerformed(java.awt.event.ActionEvent evt)
{
    this.mostrar();
}

public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new FrameArchivo().setVisible(true);
        }
    });
}

private Persona datos(){
    Persona p=new Persona();
    String nombre=this.jTextFieldNombre.getText();
    String apellidos=this.jTextFieldApellidos.getText();
    String dni=this.jTextFieldDni.getText();
    int edad=Integer.parseInt(this.jTextFieldEdad.getText());
    p.setApellidos(apellidos);
    p.setDni(dni);
    p.setNombre(nombre);
    p.setEdad(edad);
    return p;
}

private boolean verString(String s){
    if(!(s==null))
        return true;
    return false;
}

private void mostrar(){
    String s="\n\tLista de Personas\n";
    java.util.Collection co=personas.getPersonas().values();
    for(Object ob: co){
        s+=((Persona)ob).getDni()+"\t"+((Persona)ob).getNombre()+"\t"+
((Persona)ob).getApellidos()+"\t\t"+((Persona)ob).getEdad()+"\n";
    }
    this.jTextAreaMostrar.setText(s);
}

private void reset(){
```

```

        this.jTextFieldApellidos.setText("");
        this.jTextFieldDni.setText("");
        this.jTextFieldEdad.setText("");
        this.jTextFieldNOmbre.setText("");
        this.jTextFieldNOmbre.requestFocus();
    }

    private javax.swing.JButton jButtonAgregar;
    private javax.swing.JButton jButtonMostrar;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItemAbrir;
    private javax.swing.JMenuItem jMenuItemGuardar;
    private javax.swing.JMenuItem jMenuItemGuardarC;
    private javax.swing.JMenuItem jMenuItemNuevo;
    private javax.swing.JMenuItem jMenuItemSalir;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JTextArea jTextAreaMostrar;
    private javax.swing.JTextField jTextFieldApellidos;
    private javax.swing.JTextField jTextFieldDni;
    private javax.swing.JTextField jTextFieldEdad;
    private javax.swing.JTextField jTextFieldNOmbre;

    public boolean saveAsFile() {
        this.repaint();
        if (JFileChooser.APPROVE_OPTION
==jFileChooserOperaciones.showSaveDialog(this)) {
            currFileName =
jFileChooserOperaciones.getSelectedFile().getPath();
            this.repaint();
            return saveFile();
        }
        else {
            this.repaint();
            return false;
        }
    }

    //para manejo de archivos
    public boolean saveFile() {
        if(currFileName == null)
            return saveAsFile();
        try {
            File file = new File (currFileName);
            this.personas.guardar(file);
            this.dirty = false;
            tituloAplicacion();
            return true;
        }
    }

```

```
        catch(Exception ex){
            ex.printStackTrace();
        }
        return false;
    }

    public boolean cerrarAplicacion() {
        if(!dirty)
            return true;
        int value= JOptionPane.showConfirmDialog(this, "Desea Guardar los cambios ?", "Ventanita", JOptionPane.YES_NO_CANCEL_OPTION) ;
        switch(value) {
            case JOptionPane.YES_OPTION:
                return saveFile();
            case JOptionPane.NO_OPTION:
                return true;
            case JOptionPane.CANCEL_OPTION:
            default:
                return false;
        }
    }

    public void tituloAplicacion() {
        String caption;
        if (currFileName == null) {
            caption = "Lista contactos 01";
        }
        else {
            caption = currFileName;
        }
        if(dirty) {
            caption = "*" + caption;
        }
        caption = "contactos - " + caption;
        this.setTitle(caption);
    }

    public void nuevo(java.awt.event.ActionEvent e) {
        if(cerrarAplicacion()) {
            personas= new Personas();
            currFileName = null;
            dirty = false;
            tituloAplicacion();
            this.jTextAreaMostrar.setText("");
        }
    }

    void abrir(String fila) {
        try {
            File file = new File(fila);
            this.personas.cargar(file);
            this.currFileName = fila;
            this.dirty = false;
            tituloAplicacion();
        }
        catch(Exception ex){
```

```
        ex.printStackTrace();
    }
}

public void abrirArchivo() {
    if (!cerrarAplicacion())
        return;
    if (JFileChooser.APPROVE_OPTION ==
jFileChooserOperaciones.showOpenDialog(this)) {
        abrir(jFileChooserOperaciones.getSelectedFile().getPath());
    }
    this.repaint();
}

public void abre(java.awt.event.ActionEvent e) {
    abrirArchivo();
}

public void salvarComo(java.awt.event.ActionEvent e) {
    saveAsFile();
}

@Override
protected void processWindowEvent(java.awt.event.WindowEvent e) {
    super.processWindowEvent(e);
    if(e.getID() == java.awt.event.WindowEvent.WINDOW_CLOSING)
        salir(null);
}

public void salir(java.awt.event.ActionEvent e) {
    if(cerrarAplicacion())
        System.exit(0);
}

void salvar(java.awt.event.ActionEvent e) {
    saveFile();
}
}

/**clase para el manejo de archivos*/
class Nuevo implements java.awt.event.ActionListener {
    FrameArchivo nuevo;
    public Nuevo(FrameArchivo nuevo) {
        this.nuevo = nuevo;
    }

    @Override
    public void actionPerformed(java.awt.event.ActionEvent e) {
        nuevo.nuevo(e);
    }
}

class AbrirArchivo implements java.awt.event.ActionListener {
    FrameArchivo abrir;

    public AbrirArchivo(FrameArchivo abrir) {
```

```
        this.abrir = abrir;
    }

    @Override
    public void actionPerformed(java.awt.event.ActionEvent e) {
        abrir.abre(e);
    }
}
class GuardarArchivo implements java.awt.event.ActionListener {
    FrameArchivo guardar;

    public GuardarArchivo(FrameArchivo guardar) {
        this.guardar = guardar;
    }

    @Override
    public void actionPerformed(java.awt.event.ActionEvent e){
        guardar.salvarComo(e);
    }
}
class Salir implements java.awt.event.ActionListener {
    FrameArchivo salir;

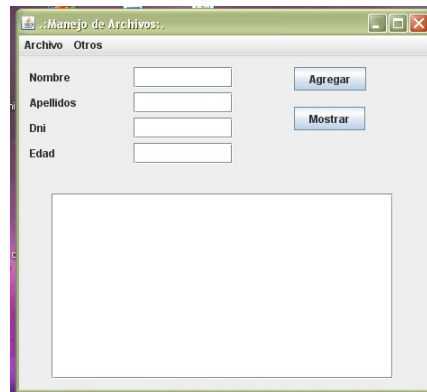
    public Salir(FrameArchivo salir) {
        this.salir=salir;
    }

    @Override
    public void actionPerformed(java.awt.event.ActionEvent e) {
        salir.salir(e);
    }
}
class Guardar implements java.awt.event.ActionListener {
    FrameArchivo g;

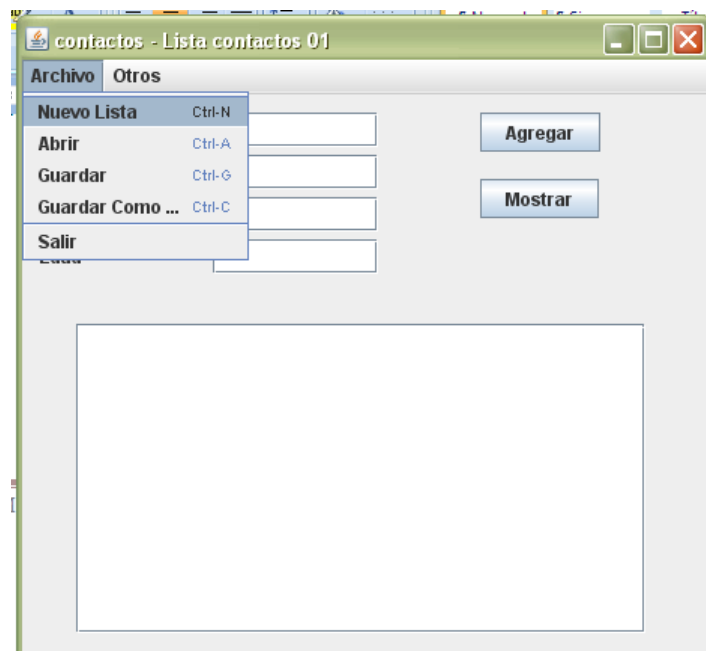
    public Guardar(FrameArchivo g) {
        this.g = g;
    }

    @Override
    public void actionPerformed(java.awt.event.ActionEvent e) {
        g.salvar(e);
    }
}
```

Prueba: Así se ve el programa

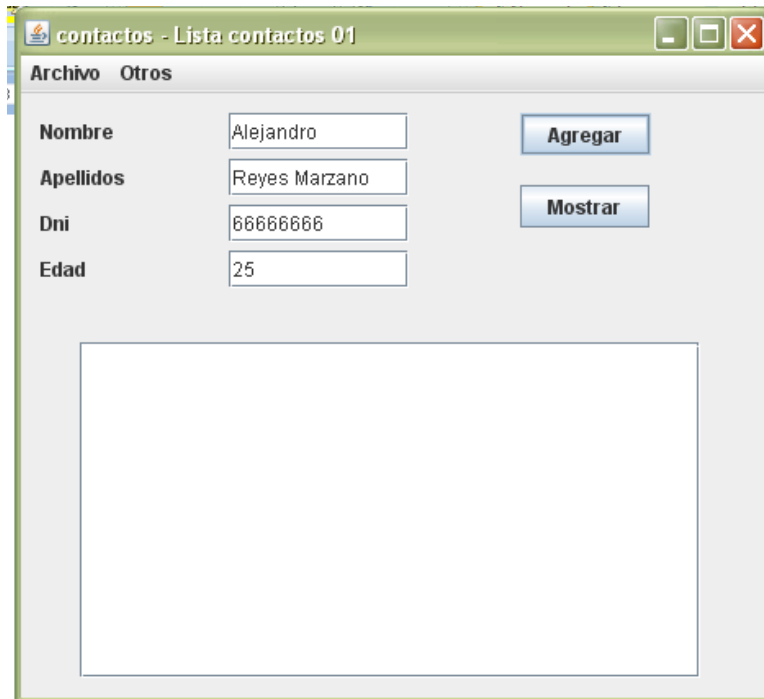


En la barra de menú elegimos la opción Archivo luego elegimos el menú ítem nuevo para crear una nueva lista de personas a registrar. Y se creara una lista vacía.

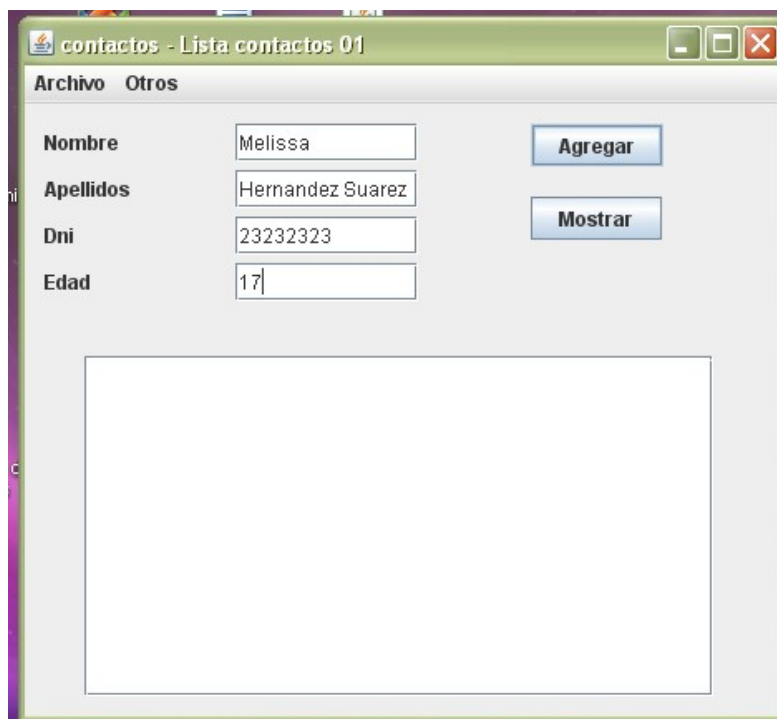


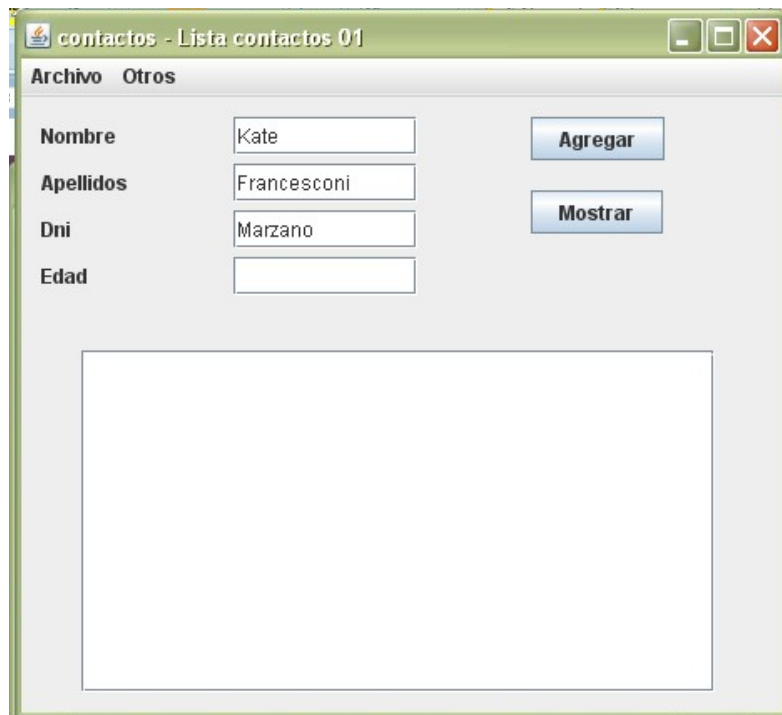
Después de haber creado la lista ya podemos ingresar personas en la lista.

Para ello llénanos los datos de la persona en los campos respectivos. Como se ve en la imagen.

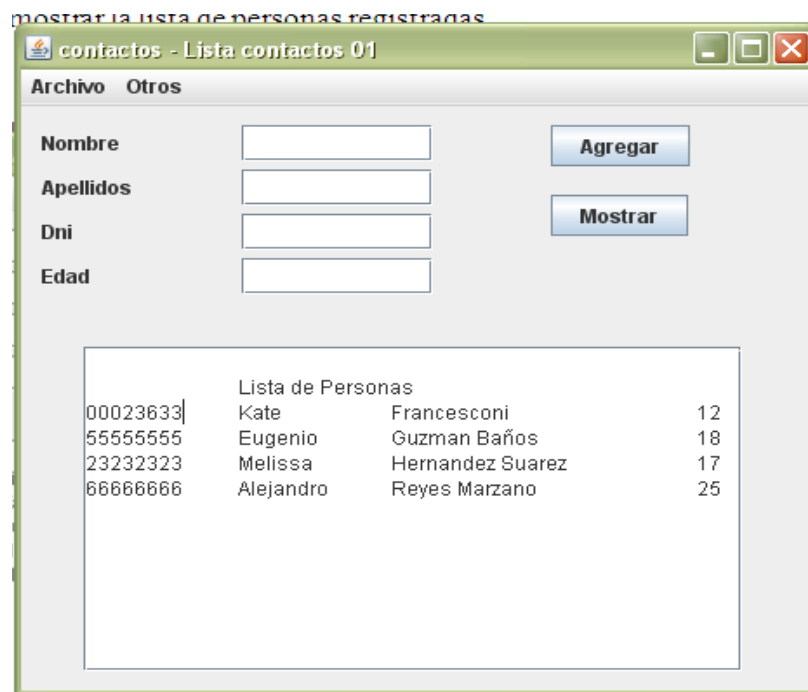


Luego damos agregar y la persona será agregando a la lista de personas. Así podemos seguir ingresando mas personas a la lista.





Ahora demos click en mostrar para mostrar la lista de personas registradas

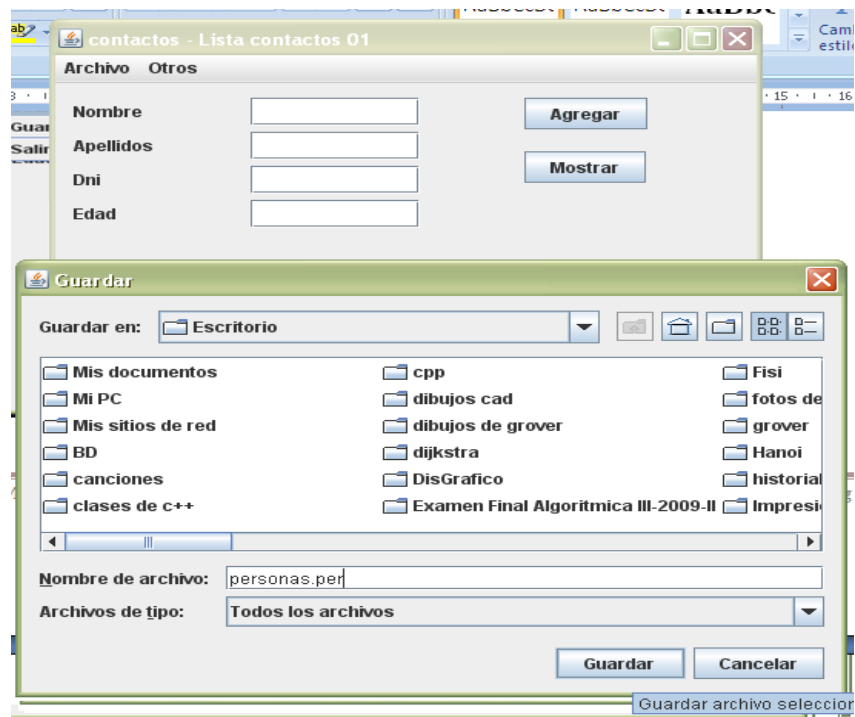


Dni	Nombre	Apellidos	Edad
00023633	Kate	Francesconi	12
55555555	Eugenio	Guzman Baños	18
23232323	Melissa	Hernandez Suarez	17
66666666	Alejandro	Reyes Marzano	25

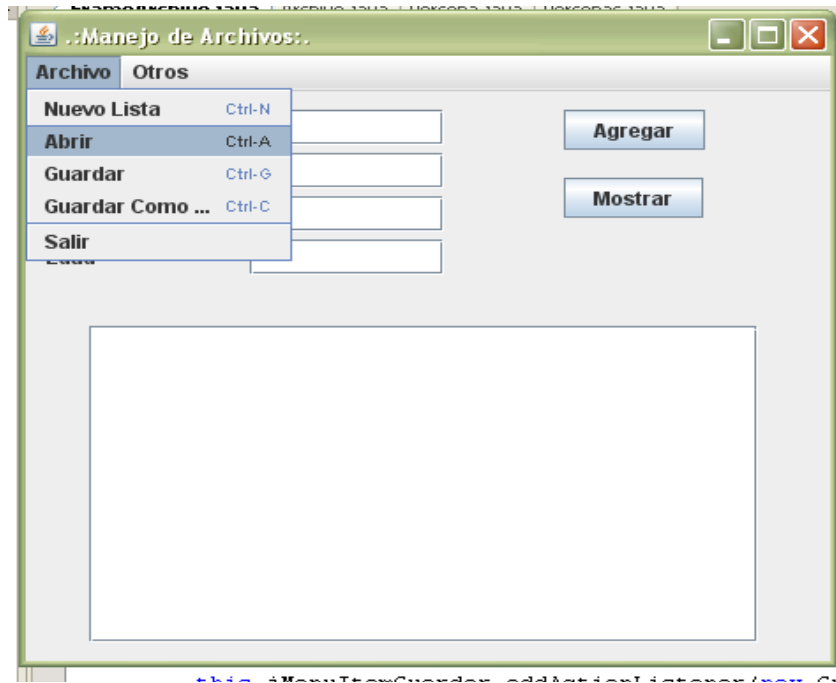
En el menú archivo seleccionamos guardar como es la primera vez puedes hacer con la opción guardar o guardar como, da lo mismo pero cuando el archivo ya existe y estamos trabajando con ese archivo si queremos actualiza podemos hacer con la opción guardar y si queremos guardar con otro nombre lo hacemos con guardar como, hay que tener cuidado.



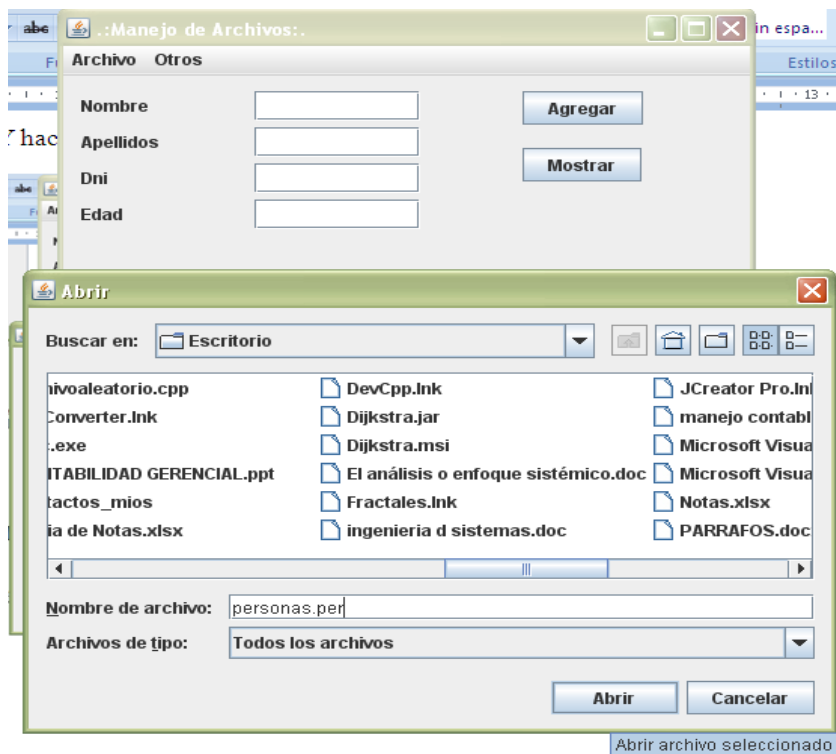
Luego le asignamos un nombre y seleccionamos la dirección a donde se quiere guardar y hacemos click en guardar, si hacemos click en cancelar no se guardará.



Para abrir seleccionamos la opción abrir del menú archivo.



Seleccionamos el fichero y hacemos click en abrir



Finalmente mostramos la lista que habíamos guardado



Bueno, espero que les sirva en algo eso es todo por esta vez.