



Ciclo Formativo de Grado Superior: Desarrollo de Aplicaciones Multimedia

Módulo: Programación
Autor: Juan Carlos Moreno
Editorial: RA-MA®



Capítulo 6

Lectura y escritura de información

Índice

- 6.1 Flujos de datos
- 6.2 Clases relativas a flujos
- 6.3 Utilización de flujos
- 6.4 Ficheros de datos
- 6.5 Almacenamiento de objetos en ficheros.
persistencia. serialización
- 6.6 Interfaces de usuario
- 6.7 Concepto de evento y controladores de eventos
- 6.8 Generación de programas en entorno gráfico
- 6.9 Ejercicios resueltos
- 6.10 Ejercicios propuestos



Objetivos del capítulo:

- Conocer todas las clases relativas a flujos.
- Valorar la importancia de la persistencia.
- Almacenar datos y objetos de forma definitiva utilizando ficheros.
- Recuperar datos y objetos de ficheros.
- Diseñar aplicaciones con interfaz gráfica.
- Construir y controlar los eventos producidos en aplicaciones con interfaz gráfica.

- Un registro es una agrupación de datos.
 - Cada uno de estos elementos se denomina campo.
 - Estos campos pueden ser tipos elementales (int, char, etc.) o bien pueden ser a su vez otras estructuras de datos.
 - Es una agrupación generalmente heterogénea de campos tanto por sus tipos de datos como por su contenido.



6.1 FLUJOS DE DATOS

FLUJOS DE DATOS

- Los flujos de datos son flujos de información entre el programa y el origen o destino de la información.
- Se tratan en Java mediante objetos *stream*.
- Sirven para abstraer y simplificar la programación.
- Los programas se encargan de leer y escribir en los flujos sin importarles dónde se leen y se escriben los datos.

ALGORITMO DE LECTURA DE FICHERO

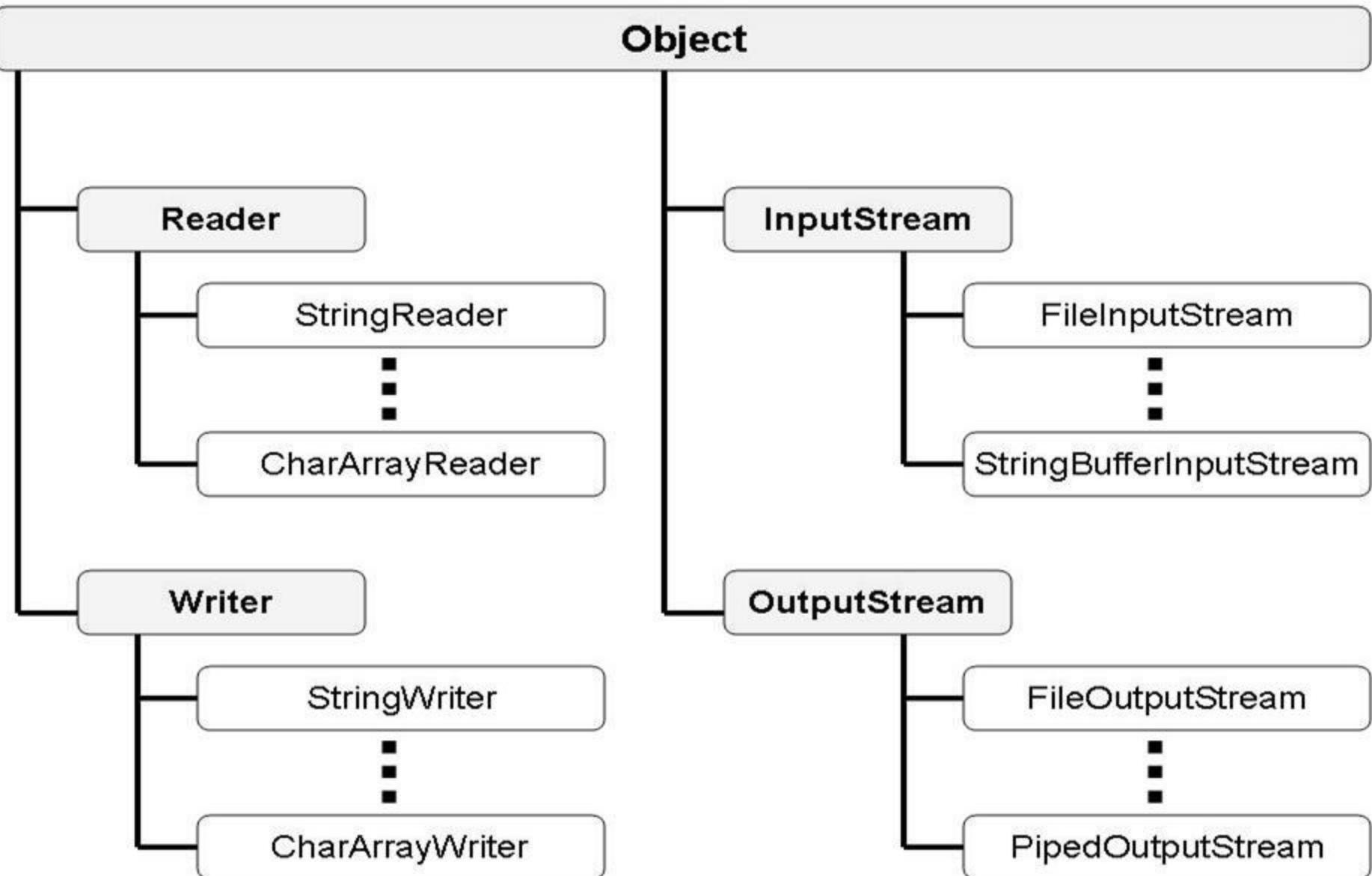


ALGORITMO DE ESCRITURA DE FICHERO





6.2 CLASES RELATIVAS A FLUJOS



El paquete java.io

Si se le pasa un *null* como argumento a un constructor o método en las clases o interfaces de este paquete, el programa lanzará una excepción del tipo `NullPointerException`.



Las tuberías

Son flujos de datos que permiten conectar dos programas o procesos entre sí transmitiéndose información entre uno y otro.

Medios y flujos asociados a los mismos

Juan Carlos Moreno

Medio	Flujo de caracteres	Flujo de bytes
Memoria	CharArrayReader CharArrayWriter	ByteArrayInputStream ByteArrayOutputStream
	StringReader StringWriter	StringBufferInputStream
Archivo	FileReader FileWriter	FileInputStream FileOutputStream
Pipes/Tuberías	PipedReader PipedWriter	PipedInputStream PipedOutputStream

JDBC
Stream
Protected
Overload casting
FLOAT
Class API
Class
OVERLOADING
Integer CORBA
SDK swing
Reader
package Objeto
Exception
INT
Programación
Liberaria
release
return
INTERNET
indí JAVAWrapper
JAVA while
bytecode
public
constructor
overloading
Compiler
JDK
Herencia
bytecode
PUBLIC

6.3 UTILIZACIÓN DE FLUJOS

Clases

Características o Función que pueden realizar

BufferedReader(C)

BufferedWriter(C)

BufferedInputStream(B)

BufferedOutputStream(B)

Tienen la propiedad de añadir un buffer al funcionamiento del objeto. En el siguiente ejemplo se puede observar la utilización del método `readLine()` el cual hace más eficiente el uso de estos objetos.

InputStreamReader(C)

OutputStreamWriter(C)

Utilizadas como clases puentes las cuales transforman streams que utilizan bytes en otros que utilizan caracteres.

ObjectInputStream(B)

ObjectOutputStream(B)

Tienen la propiedad de la seriación o serialización.

FilterReader(C)

FilterWriter(C)

FilterInputStream(B)

FilterOutputStream(B)

Pueden aplicar filtros a los stream de datos.

(C) Operan con flujos de caracteres

(B) Operan con flujos de bytes

Clases

Características o Función que pueden realizar

`DataInputStream(B)`

`DataOutputStream(B)`

Tienen la capacidad de la transformación de datos. Leen y escriben en formatos propios de Java y facilitan las transmisiones entre equipos de distinto funcionamiento.

`PushbackReader(C)`

`PushbackInputStream(B)`

Tienen la capacidad de poder mirar cuál es el siguiente carácter de la entrada y devolverlo.

`PrintWriter(C)`

`PrintStream(B)`

Tienen métodos para imprimir las variables Java con apariencia normal.

`SequenceInputStream(B)`

Tienen la capacidad de la concatenación.



Lectura de cadena por teclado

```
public static String leercadena(){
    String cad="";
    BufferedReader br;
    br = new BufferedReader(new
    ● InputStreamReader(System.in));
    ■ try{
        cad = br.readLine();
    }catch (IOException e ) {
        e.printStackTrace();
    }
    return cad;
}
```



Escritura por pantalla

```
PrintWriter pantalla = new PrintWriter(System.out);
char[] array = { 'M', 'o', 'r', 'e', 'n', 'o' };
String str = new String("Juan Carlos");
pantalla.write(str);
pantalla.print(" ");
pantalla.write(array, 0, 6);
pantalla.println("");
pantalla.flush();
```



6.4 FICHEROS DE DATOS

Método de acceso



- **Acceso secuencial.** El acceso al registro n implica la lectura previa de los registros del 1 al $n-1$.
- **Acceso directo.** Los registros se acceden expresando su dirección en el fichero.
- **Acceso por índice.** El acceso a los datos se hace mediante una clave. La clave se busca en una tabla, la cual tiene asociados clave y dirección relativa de los registros. Una vez que se conoce la dirección relativa se accede a los datos.
- **Acceso dinámico.** Se puede acceder a los datos mediante cualquiera de las formas anteriormente citadas.

Escritura secuencial en un archivo

```
FileOutputStream f=null; char c=0;  
String s = "En un lugar de la mancha de cuyo  
nombre no quiero acordarme...";  
try{  
    f=new FileOutputStream("datos.txt");  
    for (int i=0; i<s.length();i++){ c=s.charAt(i);  
        f.write((byte)c); }  
}catch(IOException e){  
    e.printStackTrace();}  
}finally{  
    try{  
        f.close();  
    }catch(IOException e){ e.printStackTrace(); }  
}
```

Lectura secuencial de un archivo



```
FileInputStream f=null; String s=""; char c;
try{
    f=new FileInputStream("datos.txt");
    int size = f.available();
    for (int i=0;i<size;i++){
        c=(char)f.read(); s=s+c;
    }
} catch(IOException e){ e.printStackTrace();
} finally{
    System.out.println(s);
    try{
        f.close();
    } catch(IOException e){ e.printStackTrace(); }
}
```



LA CLASE FILE

- La clase *File* desciende directamente de la clase *Object*.

Recuerda

El objeto *File* puede trabajar tanto con ficheros como con directorios.

Constructores de la clase File

Constructores	Descripción
public File (String ruta_absoluta)	Este constructor crea un objeto <i>File</i> al cual hay que pasarle toda la ruta incluido el nombre de archivo.
public File(String ruta, String nombre)	En este constructor hay que pasarle la ruta relativa y el nombre del fichero.
public File(File ruta, String nombre)	Como el objeto <i>File</i> puede representar un directorio o un fichero, el primer parámetro funcionará como directorio (ruta relativa) y el segundo parámetro será el nombre del fichero a crear.

Ejemplos de creación de objetos de la clase File

//primer constructor

```
File f1 = new File("/home/jcmoreno/datos.txt");
```

//segundo constructor

```
File f2 = new File("/home/jcmoreno","datos.txt");
```

//tercer constructor

```
File dir = new File("/home/jcmoreno");
```

```
File f3 = new File(dir,"datos.txt");
```

Algunos métodos del objeto *File* (I)

Juan Carlos Moreno

Métodos	Descripción
public boolean isDirectory()	Estos dos métodos sirven para saber si se está trabajando con ficheros o con directorios.
public boolean isFile()	Con este método nos cercioramos si el fichero existe realmente.
public boolean exists()	Permite borrar el fichero o directorio al que referencia el objeto <i>File</i> .
public boolean renameTo(File dest)	Permite renombrar el fichero o directorio al que referencia el objeto File. El nuevo nombre lo toma como parámetro.
public boolean canRead()	Estos métodos nos permiten averiguar si tenemos permisos de lectura y escritura sobre el fichero.
public boolean canWrite()	

Algunos métodos del objeto *File* (II)

Métodos	Descripción
public String getPath()	Devuelve la ruta relativa del archivo.
public String getAbsolutePath()	Devuelve la ruta absoluta del archivo (incluye el nombre del archivo).
public String getName()	Devuelve el nombre del archivo.
public String getParent()	Devuelve el directorio padre (del que cuelga el fichero o directorio).
public long length()	Devuelve el tamaño del archivo en bytes, si lo que se quiere son los kilobytes habrá que dividirlo entre 1024 y si se quieren los megabytes dividirlo nuevamente por 1024.
public boolean equals(Object obj)	Este método permitirá comparar objetos <i>File</i> para saber si son iguales o no.

Métodos para directorios	Descripción
public boolean mkdir()	Crea un directorio.
public String[] list()	Devuelve un array de objetos String con los nombres de los ficheros/directorios del directorio al que apunta el objeto <i>File</i> .
public String[] list(filtro)	Devuelve un array de objetos String con los nombres de los ficheros que cumplen con un determinado filtro (por ejemplo “*.java”).
public File[] listFiles()	Devuelve un array de objetos <i>File</i> con los ficheros/directorios del directorio al que apunta el objeto <i>File</i> .

CLASES FILEWRITER Y FILEREADER

- Ambas clases trabajan con flujos de caracteres (*char*) en modo lectura y escritura sobre un fichero.

java.lang.Object

java.io.Reader

java.io.InputStreamReader

java.io.FileReader

java.lang.Object

java.io.Writer

java.io.OutputStreamWriter

java.io.FileWriter

FLUJOS DE DATOS DATAOUTPUTSTREAM Y DATAINPUTSTREAM

java.lang.Object

java.io.InputStream

java.io.FilterInputStream

java.io.DataInputStream

java.lang.Object

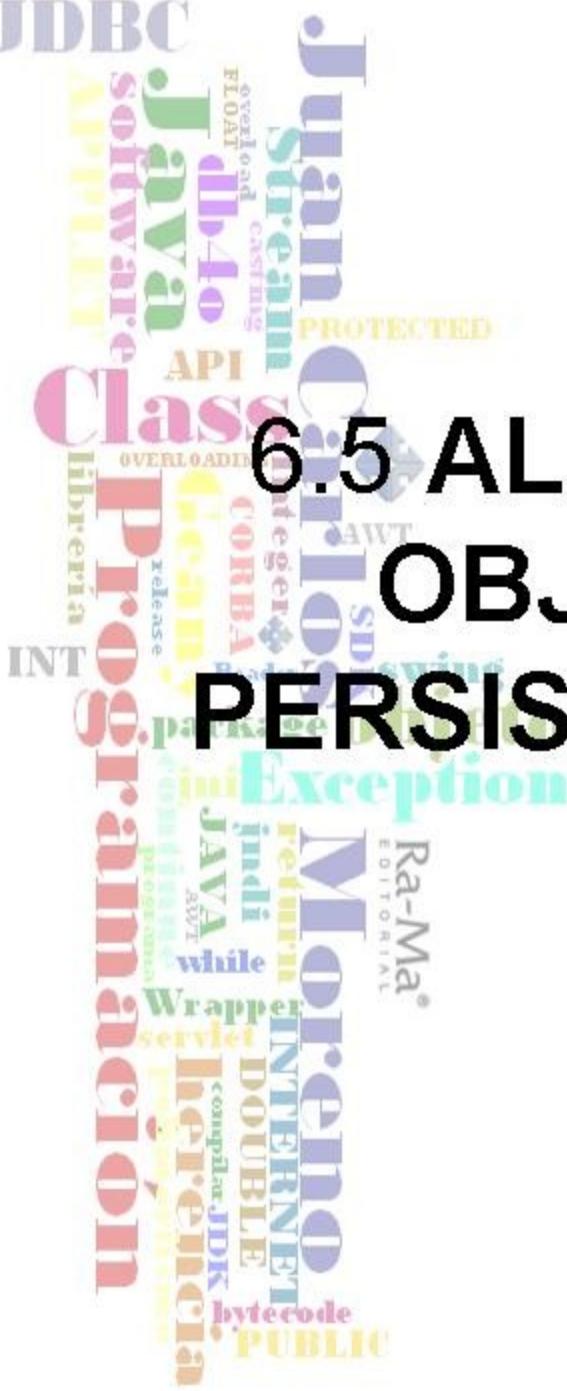
java.io.OutputStream

java.io.FilterOutputStream

java.io.DataOutputStream

- Como se puede observar, ambas clases descienden de una clase filtro (FilterInputStream y FilterOutputStream).
- Las clases filtro en Java es la modificación/transformación de los datos.

6.5 ALMACENAMIENTO DE OBJETOS EN FICHEROS. PERSISTENCIA. SERIALIZACIÓN





La serialización de objetos

- Consiste en transformar un objeto en una secuencia o serie de bytes de tal manera que se represente el **estado** de dicho objeto.
- Una vez serializado se puede enviar a un fichero, se podría enviar por la red, etc.
- Si por ejemplo tenemos el objeto seriado y almacenado en un fichero sería posible recomponer el objeto.

Es posible no serializar algunos de los atributos de un objeto, esto se realiza utilizando el modificador **transient**.

Por ejemplo:

```
protected transient int dato;
```

En este caso, el campo dato no interesa que sea persistente.



Recuerda

Para poder serializar un objeto de una clase es necesario que implemente la interfaz `java.io.Serializable`.

Clases *ObjectOutputStream* y *ObjectInputStream*

Juan Carlos Moreno

- Para el almacenamiento y la recuperación de objetos es posible utilizar las clases *ObjectOutputStream* y *ObjectInputStream*.

`java.lang.Object`

`java.io.InputStream`

`java.io.ObjectInputStream`

`java.lang.Object`

`java.io.OutputStream`

`java.io.ObjectOutputStream`

JDBC
Stream
String
PROTECTED
FLOAT
Overload casting
Class API
Class
OVERLOADING
Integer CORBA
CORBA Reader
swing
package Objeto
Exception
INT
RELEASE
Programación
Liberaria
Ra-Ma® EDITORIAL
Moreno INTERNET
return
indí JAVAWT
while Wrapper
public
bytecode
Heredencia
compilar JDK
PUBLIC

6.6 INTERFACES DE USUARIO

Librerías gráficas en Java

- **AWT** (*Abstract Window Toolkit* – Kit de herramientas abstracto para ventanas). Es parte de la JFC (*Java Foundation Classes* – Clases base de Java). El desarrollar con este kit tiene la ventaja de que las aplicaciones se parecen mucho al kit de herramientas nativo subyacente.
- **Swing**. La ventaja de Swing frente a AWT es que los componentes utilizados por la librería gráfica de Swing están programados con código no nativo, lo cual lo hacen más portable. Estos componentes son más potentes que los anteriores y se identifican con una J antes del nombre del componente (por ejemplo JButton).

NUESTRA PRIMERA APLICACIÓN CON SWING

```
import javax.swing.*;  
public class holamundoswing{  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("Ventana Hola  
        Mundo");  
        frame.setDefaultCloseOperation(WindowConsta  
        nts.EXIT_ON_CLOSE);  
        JLabel label= new JLabel("Hola Mundo");  
        frame.getContentPane().add(label);  
        frame.pack();  
        frame.setLocationRelativeTo(null);  
        frame.setVisible(true);  
    } }
```

Contenedores Swing de alto nivel

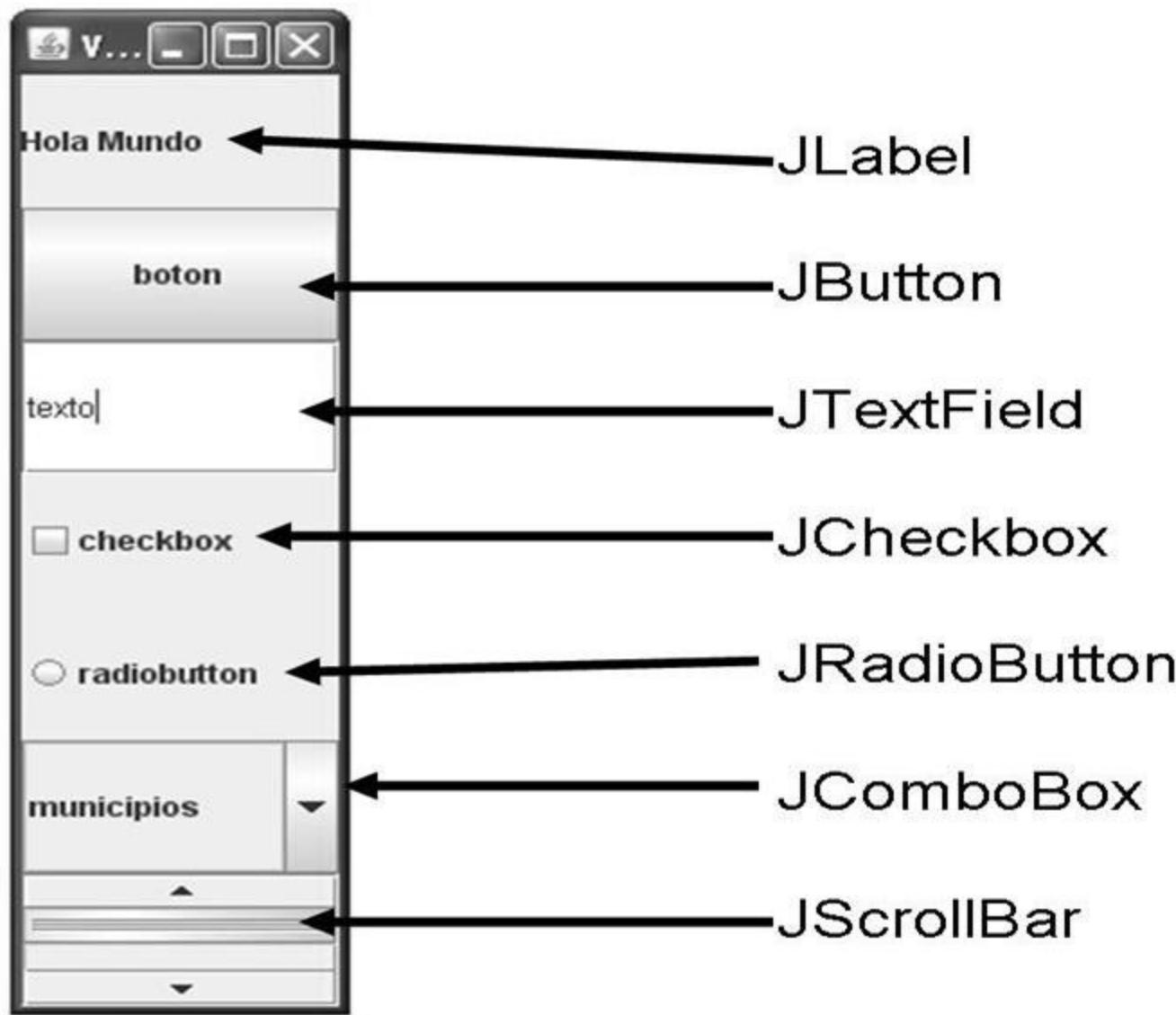
- **JFrame.** Implementa una ventana. Las ventanas principales de una aplicación deberían de ser un JFrame.
- **JDialog.** Implementa una ventana tipo diálogo. Este tipo de ventanas se utilizan como ventanas secundarias y generalmente son llamadas por ventanas padre del tipo JFrame.
- **JApplet.** Un *applet* es una aplicación Java que se ejecuta dentro de un navegador web en la máquina del cliente.

LOS COMPONENTES SWING (I)

Objeto	Descripción
JButton	Botón estándar.
JLabel	Etiqueta de texto estandar.
JTextField	Cuadro de texto.
JTextArea	Cuadro de texto multilínea.
JCheckBox	Checkbox o casilla de verificación.
JRadioButton	Radiobutton o botones de opción.
JComboBox	Lista desplegable.
JScrollBar	Barra de desplazamiento.

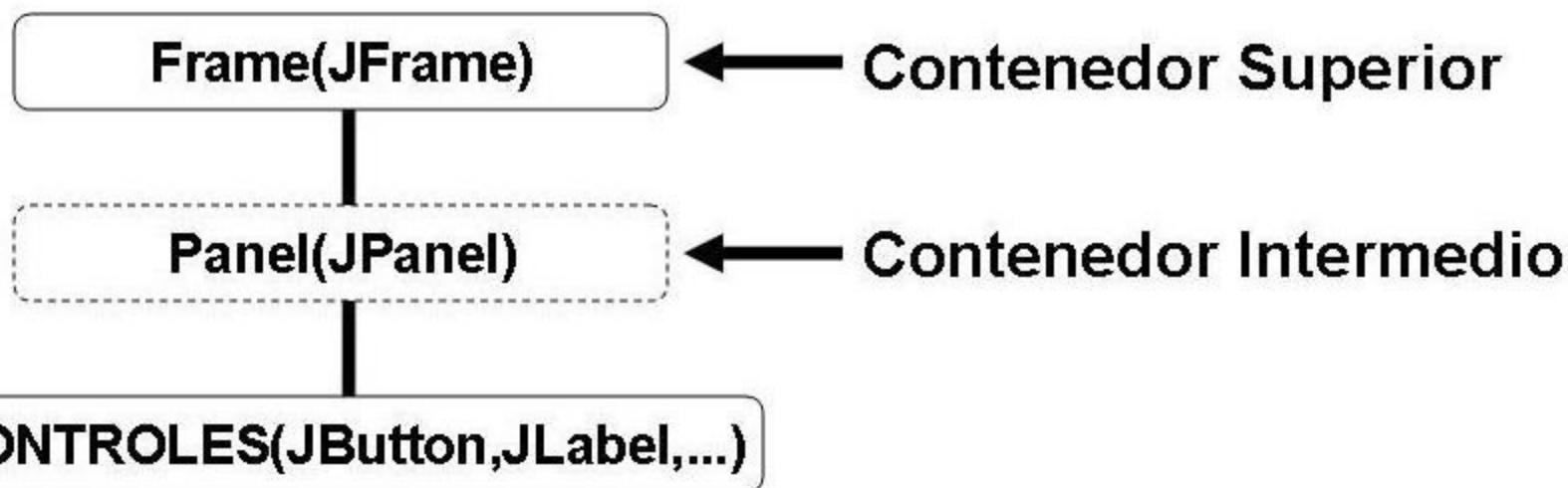


LOS COMPONENTES SWING (II)



LOS CONTENEDORES SWING

- Los paneles (JPanel). Contenedores de componentes ligeros y estos a su vez pueden contener otros paneles.



ORGANIZACIÓN DE LOS CONTROLES EN UN CONTENEDOR

- Para organizar los controles en un objeto que implemente la interfaz LayoutManager (por ejemplo los paneles) es necesario establecer en ella un Layout Manager o administración de diseño
- Los *Layout Manager* se pueden establecer al crear el objeto (*constructor*).



Tipos de Layout Manager

- **FlowLayout.** Coloca los componentes en el contenedor de izquierda a derecha. Es el Layout Manager por defecto en los paneles.
- **BorderLayout.** Divide el contenedor en cinco partes (norte, sur, este, oeste y centro).
- **CardLayout.** Permite colocar grupos de componentes diferentes en momentos diferentes de la ejecución del programa.
- **GridLayout.** Coloca los componentes en filas y columnas.
- **GridBagLayout.** Coloca los componentes en filas y columnas, pero un componente puede ocupar más de una columna.
- **BoxLayout.** Coloca los componentes en una fila o columna ajustándose.

APARIENCIA DE LAS VENTANAS

- El *look and feel* genérico de las ventanas puede ser modificado según diferentes estilos.
- Java tiene un Gestor de la interfaz del usuario UIManager que controla la apariencia genérica de las ventanas y de los componentes que las componen.
- Este UIManager se encuentra en la clase javax.swing.UIManager.

Estableciendo el Look&Feel

```
// Establecer el look and feel metal.
```

```
// Un L&F muy Java
```

```
try{
```

```
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
```

```
}catch (Exception e){e.printStackTrace();}
```

El método getSystemLookAndFeelClassName()

Plataforma	Valor devuelto por el método getSystemLookAndFeelClassName()
Multi plataforma	"javax.swing.plaf.metal.MetalLookAndFeel" Es el valor devuelto por el método getCrossPlatformLookAndFeelClassName. Este look and feel se le llama metal.
Windows®	"com.sun.java.swing.plaf.windows.WindowsLookAndFeel" Look and feel de sistemas Windows.
Solaris®	"com.sun.java.swing.plaf.motif.MotifLookAndFeel" Este look and feel puede usarse en cualquier plataforma.
Mac®	"javax.swing.plaf.mac.MacLookAndFeel" Look and feel de sistemas Mac.
Unix/Linux	"com.sun.java.swing.plaf.gtk.GTKLookAndFeel" Para sistemas GTK.

6.7 CONCEPTO DE EVENTO Y CONTROLADORES DE EVENTOS





EVENTO Y CONTROLADORES DE EVENTOS



LIS...NER



EVENTO

RESPUESTA





EVENTO Y CONTROLADORES DE EVENTOS

- Existen una serie de manejadores/controladores de eventos (*listener*) los cuales deberán de ser asociados al componente para que este ejecute la respuesta necesaria.
- Estos *listener* son diferentes dependiendo de los eventos a los que van a dar respuesta.
- Los *listener* están especializados dependiendo del evento ocurrido.

Listeners asociados a componentes (I)

Listener	Componentes	Acción a la que responden
ActionListener	1. JButton 2. JTextField 3. JComboBox 4. ...	1. Presionar el botón. 2. Pulsar intro. 3. Elegir una opción. 4....
AdjustmentListener	1. JScrollBar 2. ...	Mover la barra de desplazamiento. ...
FocusListener	1. JButton 2. JTextField 3. JComboBox 4. ...	Las acciones de este <i>listener</i> son obtener y perder el foco (colocarnos en el componente e irnos del mismo cuando estaba activo).
ItemListener	1. JCheckbox 2. ...	1. Seleccionar y deseleccionar la opción.

Listeners asociados a componentes (II)

Listener	Componentes	Acción a la que responden
KeyListener	1. JTextField 2. JTextArea 3. ...	Pulsar una tecla cuando el componente tiene el foco.
MouseListener	Múltiples componentes	Acciones como presionar el botón del ratón.
MouseMotionListener	Múltiples componentes	Acciones como arrastrar (<i>drag</i>) o pasar por encima del objeto.
WindowListener	1. JFrame	Acciones relativas a la ventana como por ejemplo cerrarla.

Programación del manejo de eventos (I)



1. Se crea el componente.
2. Se añade el *listener* adecuado al componente y el *listener* escuchará la acción sobre el componente.
3. Dependiendo del componente o la acción se ejecutará la acción necesaria.

Programación del manejo de eventos (II)

