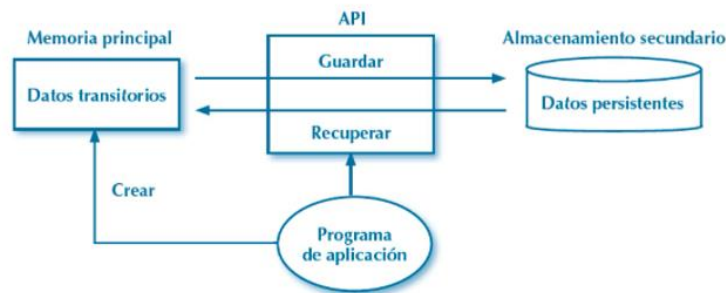


UT1. MANEJO DE FICHEROS

1. Introducción

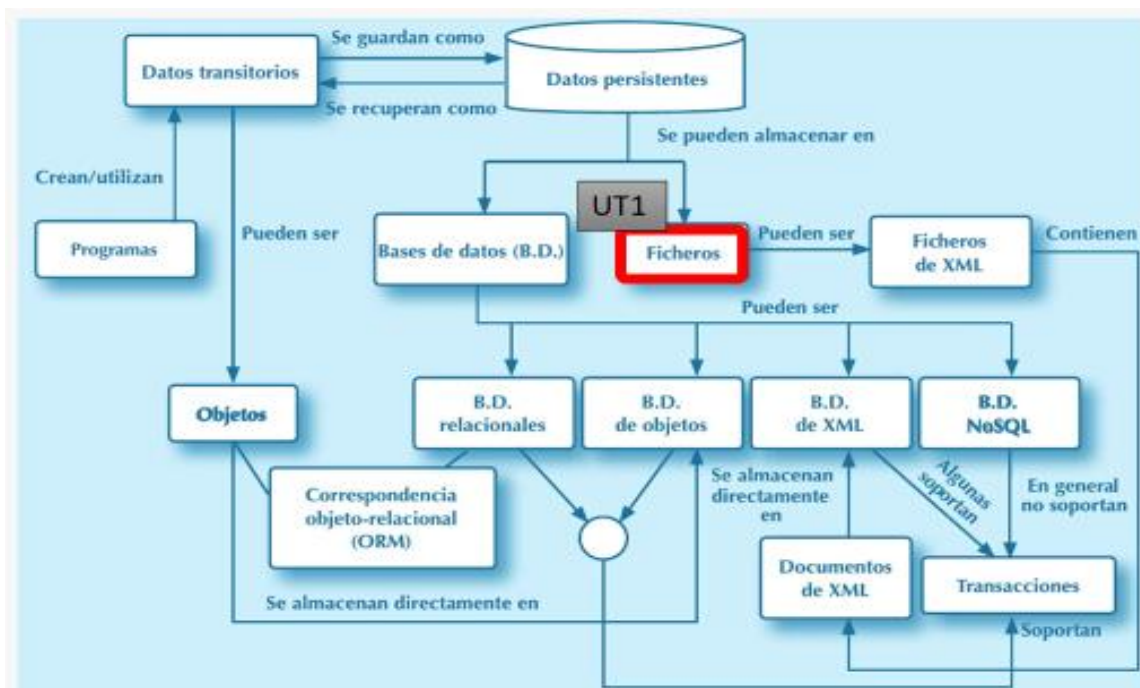
Tipos de almacenamiento de datos



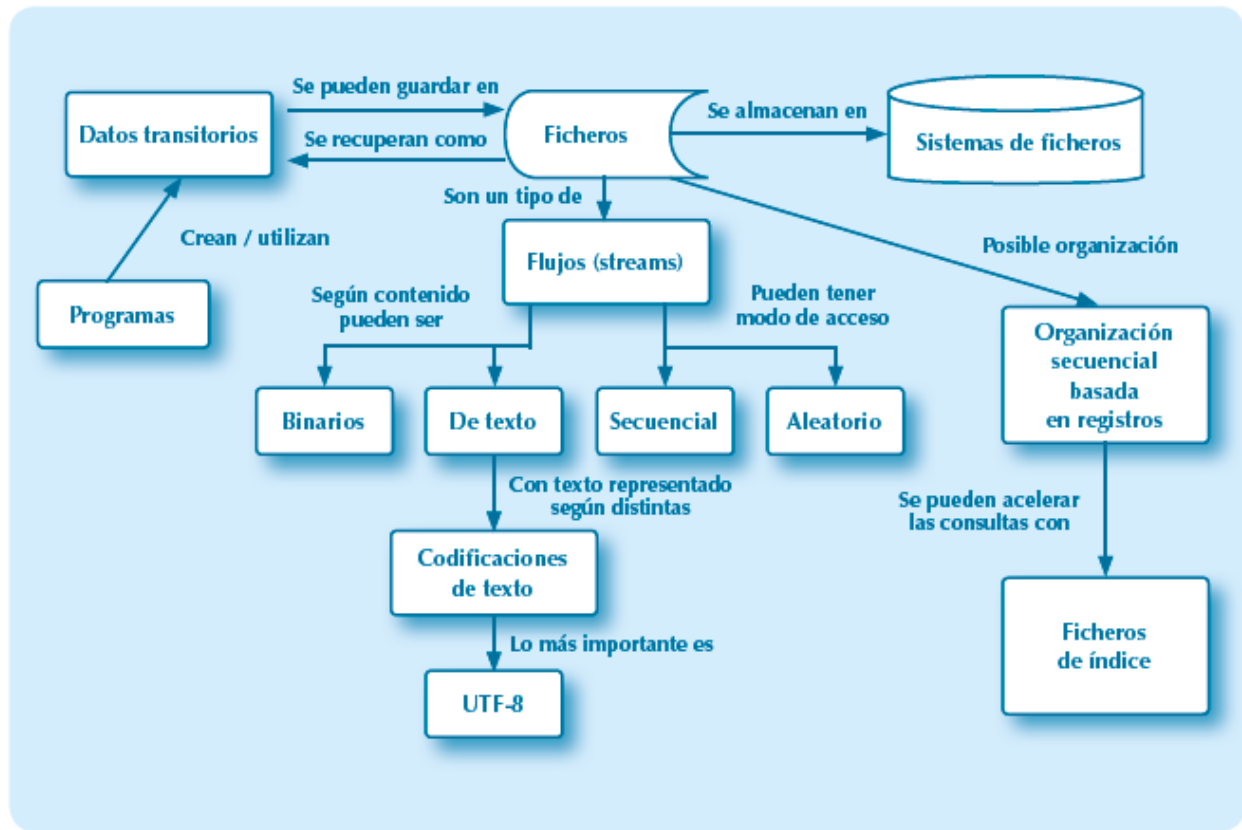
Memoria principal	Almacenamiento secundario
Volátil	Permanente
Baja capacidad	Alta capacidad
Rápida	Lenta
Accesible directamente	Se pueden transferir contenidos a memoria principal

- Medios de almacenamiento secundario (datos persistentes): discos duros, tarjetas de memoria, pen drive. Pueden encontrarse en un ordenador distinto a donde se ejecuta el programa.
- Los programas suelen acceder a los datos persistentes mediante funciones de alto nivel proporcionadas por diversas API (interfaces de programación de aplicaciones).

Sistemas de almacenamiento de datos persistentes



Almacenamiento de datos en ficheros



- En Java, el acceso a ficheros es tratado como un *flujo (stream)* de información entre el programa y el fichero. Para comunicar un programa con un origen o destino de cierta información (fichero) se usan *flujos* de información. Un flujo no es más que un objeto que hace de intermediario entre el programa y el origen o el destino de la información.

Por ejemplo, Java ofrece la clase *FileReader*, donde se implementa un flujo de caracteres que lee de un fichero de texto. Desde código Java, un programador puede manejar un objeto de esta clase para obtener el flujo de datos texto sacados del archivo que elija.

- Aunque nos centramos en ficheros, los flujos (*stream*) en Java se utilizan para acceder también a memoria o para leer/escribir datos en dispositivos de entrada/salida.
- En esta unidad vamos a ver cómo crear, consultar, modificar, procesar, etc. ficheros en Java.

2. Almacenamiento de datos en ficheros

Persistencia de datos en ficheros

- **Ficheros:** son el método de almacenamiento de información más elemental.
- En última instancia, todos los métodos de almacenamiento, por sofisticados que sean, almacenan los datos en ficheros.
- Relegados en los 1980's por las Bases de Datos Relacionales.

Ficheros. Desventajas

- Si hay que realizar **consultas complejas** o que requieren relacionar mucha información diversa, será difícil escribir un programa para realizarlas.
- Si el **volumen de datos** para manejar es muy grande, o si es necesario realizar con mucha frecuencia **operaciones de borrado o modificación** de datos, el rendimiento será muy pobre.
- **Concurrencia:** Permitir que varios programas realicen a la vez operaciones de consulta / actualización puede introducir inconsistencias en los datos e incluso dañar los ficheros.
- Es difícil establecer y hacer que se cumplan **restricciones de integridad** sobre los datos.
- También es difícil evitar **redundancias** en los datos:
 - Se desperdicia espacio de almacenamiento.
 - Pueden surgir inconsistencias cuando se añaden o modifican datos: si la misma información está en más de un lugar, puede acabar teniendo un valor distinto en cada uno.

Tipos de ficheros según su contenido

- En principio un fichero puede almacenar cualquier tipo de información.
- Vamos a diferenciar el tratamiento de:
 - Ficheros de texto
 - Ficheros binarios
- Ficheros **de texto:** contienen únicamente caracteres.
 - Caracteres imprimibles: letras, números signos de puntuación, etc.
 - Espacios, tabuladores, retornos de carro, etc.
 - Su contenido se puede visualizar y modificar con cualquier editor de texto (Bloc de notas, gedit, nano, etc.).
- **Ficheros binarios:** el resto de ficheros.
 - Almacenan cualquier tipo de información.
 - En general, hacen falta programas especiales para mostrar la información que contienen.

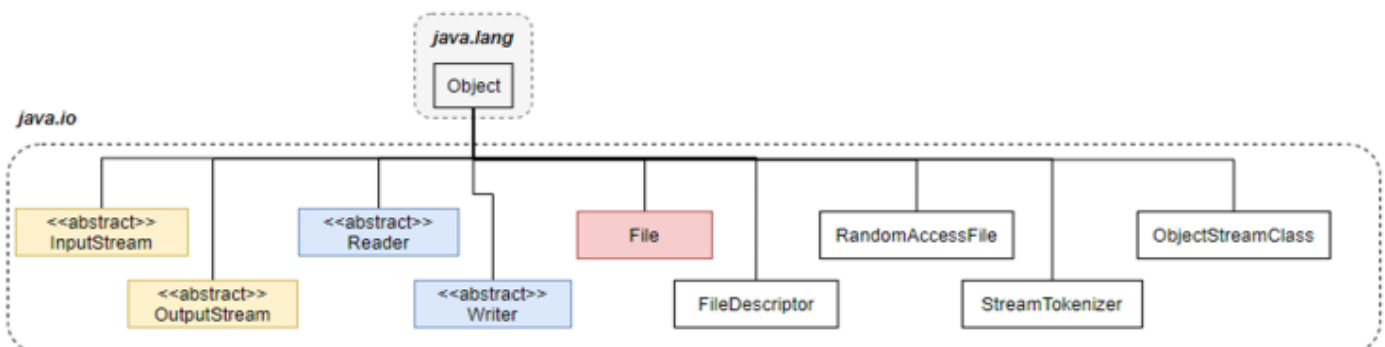
Tipos de ficheros según su modo de acceso

- Existen dos formas de acceder a los contenidos de los ficheros (tanto para lectura como para escritura);
 - Acceso secuencial:** para llegar a cualquier parte de un fichero hay que pasar antes por los contenidos anteriores, empezando siempre por el principio del fichero.
 - Acceso aleatorio:** se accede directamente a los datos situados en cualquier posición del fichero.

3. Clases Java para trabajar con ficheros

- A las operaciones que facilitan el intercambio de información el programa con el exterior (por ejemplo, con un fichero) se les conoce como **operaciones de Entrada/Salida (E/S)**.
- Las operaciones de E/S en Java las proporciona el paquete estándar denominado `java.io`. Este paquete incorpora:
 - Clases:**
 - Clases para la gestión de ficheros y directorios en el sistema de ficheros local.
 - Clases para leer desde un flujo de datos.
 - Clases para escribir desde un flujo de datos.
 - Clases para gestionar la serialización de objetos (convertir objetos en bytes).
 - Interfaces**
 - Excepciones**

El paquete java.io



Control de excepciones

- Algunas de las operaciones que se realizan sobre ficheros generan excepciones que hay que capturar, y por tanto deben estar incluidas en un bloque try-catch. Esto nos permitirá mostrar mensajes de error y terminar el programa de una forma ordenada en caso de que se produzca algún fallo como por ejemplo si el fichero no existe, no tenemos permiso para acceder a él, etc.

El bloque try-catch tiene el siguiente formato:

```
try {
    Instrucciones que se pretenden ejecutar
    (si se produce una excepción puede que
    no se ejecuten todas ellas).
} catch {
    Instrucciones que se van a ejecutar
    cuando se produce una excepción.
} finally {
    Instrucciones que se van a ejecutar tanto
    si se producen excepciones como si no
    se producen.
}
```

O más concretamente, en el caso de manejo de ficheros:

```
try {
    Operaciones_con_fichero
} catch (tipo_de_error nombre_de_variable) {
    Tratamiento_del_error
} finally {
    Instrucciones que se van a ejecutar tanto
    si se producen excepciones como si no
    se producen.
}
```

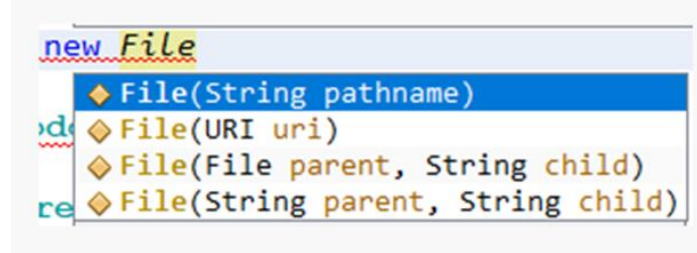
- Se pueden especificar varios catch para controlar diferentes excepciones.
- La parte `finally` es opcional.

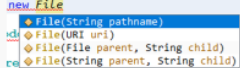
4. Clase File para la gestión de ficheros y directorios

- La clase **File** de Java, que se encuentra en el paquete `java.io`, permite realizar todas operaciones sobre archivos: listar archivos en un directorio, ver sus propiedades (como permisos de lectura y escritura), y crear o borrar archivos y directorios.
- Esta clase cuenta con una amplia lista de métodos y sobrecarga de constructores.
- Aprenderemos a utilizar la clase **File** para crear y eliminar ficheros/directorios, obtener propiedades de ficheros/directorios, filtrar ficheros que tengan unas determinadas características, etc.

Clase File: Constructores

- La clase tiene cuatro constructores, uno de los cuales será utilizado para especificar la ruta de un archivo.





Ruta de un directorio o de un archivo

File(String pathname):

- en Linux: `new File("/directorio/fichero.txt");`
- en plataformas Microsoft Windows:
`new File("C:\\directorio\\fichero.txt");`

File(String directorio, String nombrefichero):

`new File("directorio", "fichero.txt");`

File(File directorio, String fichero):

`new File(new File("directorio"), "fichero.txt");`

Clase File: métodos principales

Método	Descripción
<code>boolean canRead()</code>	Devuelve true si el objeto File (archivo o directorio) se puede leer.
<code>boolean canWrite()</code>	Devuelve true si se puede escribir en el objeto File (archivo o directorio).
<code>boolean canExecute()</code>	Devuelve true si se puede ejecutar el objeto File (archivo o directorio).
<code>boolean createNewFile()</code>	Crea un fichero vacío asociado al objeto File. Devuelve true si se ha podido crear. Para poder crearlo el fichero no debe existir. Lanza una excepción del tipo IOException .
<code>File createTempFile()</code>	Crea un fichero temporal y devuelve un objeto File que apunta a él. Es útil para crear archivos temporales, que luego se borran, asegurándonos de tener un nombre no repetido. Lanza una excepción del tipo IOException .
<code>boolean delete()</code>	Borra el fichero o directorio asociado al objeto File. Si es un directorio debe estar vacío. Devuelve true si se ha podido eliminar.
<code>void deleteOnExit()</code>	El fichero se borra cuando finaliza la ejecución de la máquina virtual de Java.
<code>boolean exists()</code>	Devuelve true si el fichero/directorio existe.
<code>String getName()</code>	Devuelve el nombre del fichero o directorio.

Método	Descripción
<code>String getAbsolutePath()</code>	Devuelve la ruta absoluta asociada al objeto File.
<code>String getCanonicalPath()</code>	Devuelve la ruta única absoluta asociada al objeto File. Puede haber varias rutas absolutas asociadas a un File, pero sólo una única ruta canónica. Lanza una excepción del tipo IOException .
<code>String getPath()</code>	Devuelve la ruta con la que se creó el objeto File. Puede ser relativa o no.
<code>String getParent()</code>	Devuelve un String con el nombre del directorio padre, o null si no tiene directorio padre.
<code>File getParentFile()</code>	Devuelve un objeto File conteniendo el directorio padre del File, o null si no tiene directorio padre.
<code>boolean isAbsolute()</code>	Devuelve true si la ruta es absoluta.
<code>boolean isDirectory()</code>	Devuelve true si el objeto File corresponde a un directorio válido.
<code>boolean isFile()</code>	Devuelve true si el objeto File corresponde a un fichero válido.
<code>long lastModified()</code>	Devuelve un valor en milisegundos que representa la última vez que se ha modificado (medido desde las 00:00:00 GMT, del 1 de enero de 1970) Devuelve 0 si el fichero no existe o ha ocurrido un error.
<code>long length()</code>	Devuelve el tamaño del fichero en bytes. Devuelve 0 si no existe. Devuelve un valor indeterminado si es un directorio.
<code>String[] list()</code>	Devuelve un array de String con los nombres de ficheros y directorios que contiene el directorio asociado al objeto File. Si no es un directorio devuelve null. Si el directorio está vacío devuelve un array vacío.
<code>String[] list(FileNamedFilter filtro)</code>	Devuelve un array de String con los nombres de ficheros y directorios que contiene el directorio asociado al objeto File que cumplen con el filtro indicado.
<code>File[] listFiles()</code>	Devuelve un array de File conteniendo los ficheros y directorios dentro del directorio representado por el objeto File.
<code>boolean mkdir()</code>	Crea un directorio. Devuelve true si se ha podido crear
<code>boolean mkdirs()</code>	Crea un directorio incluyendo los directorios no existentes especificados en la ruta padre del directorio a crear. Devuelve true si se ha podido crear el directorio y los directorios no existentes de la ruta padre.
<code>String pathSeparator ()</code>	Representa, en formato String, el separador de rutas correspondiente al sistema operativo en cuestión ("/" o "\").
<code>boolean renameTo(File nuevoNombre)</code>	Renombra el fichero/directorio asociado al objeto File con el nombre nuevoNombre.
<code>boolean setLastModified()</code>	Establecer la fecha y hora de modificación del archivo. Ejemplo: <code>File("prueba.txt").setLastModified(new Date().getTime());</code>
<code>boolean setExecutable (boolean executable)</code>	Establece el permiso de ejecución al valor que se pasa como argumento (true o false). Devuelve true si operación exitosa.
<code>boolean setReadable (boolean readable)</code>	Establece el permiso de lectura al valor que se pasa como argumento (true o false). Devuelve true si operación exitosa.
<code>boolean setReadOnly()</code>	Establece permiso de solo lectura para el objeto File (archivo o directorio). Devuelve true si operación exitosa.
<code>boolean setWritable(boolean writable)</code>	Establece el permiso de escritura al valor que se pasa como argumento (true o false). Devuelve true si operación exitosa.

Clase File. Ejemplo_01: crear objetos de la Clase File

```
/*
 * Crear un objeto de la clases File, imprimir la ruta absoluta y comprobar si
 * el archivo que referencia existe.
 * Utilizar el constructor que recibe la ruta y nombre del archivo en un único String.
 * La ruta y nombre de archivo se pasan al programa como argumentos al main()
 * Probar con distintos nombres archivos, tanto existentes como no
 */
package ut1_24_25_ejemplos;
import java.io.File;

public class UT1_24_25_ClaseFileEjemplo_01 {

    public static void main(String[] args) {

        //Creamos un objeto File que encapsula el fichero

        File archivo = new File(args[0]);

        //Imprimimos la ruta a la que referencia el objeto
        System.out.println("Ruta absoluta del fichero:" + archivo.getAbsolutePath());

        // Comprobamos si el archivo referenciado existe
        System.out.println("El fichero existe?: " + archivo.exists());

    }
}
```

Ejemplo de ejecución:

```
run:
Ruta absoluta del fichero:/home/alumno/NetBeansProjects/UT1_24_25/UT1_24_25_Ejemplos/manifest.mf
El fichero existe?: true
BUILD SUCCESSFUL (total time: 0 seconds)
```


Clase File. Ejemplo_02: listar el contenido de un directorio

```
/*
 * Lista el contenido de un directorio cuyo nombre se pasa como argumento al main
 */
package ut1_24_25_ejemplos;

import java.io.File;

public class UT1_24_25_ClaseFileEjemplo_02 {

    public static void main(String[] args) {

        // Creamos objeto File para la ruta recibida en el main
        File ruta = new File(args[0]);
        // Mostramos la ruta
        System.out.println(ruta.getAbsolutePath());

        // Si existe el directorio en esa ruta, mostramos los nombres de los archivos
        // De lo contrario se imprime un mensaje de error
        if (ruta.exists()){

            String[] nombres_archivos = ruta.list();
            for (int i = 0; i < nombres_archivos.length; i++) {
                System.out.println(nombres_archivos[i]);
            }
        }
        else {
            System.out.println("ERROR: no existe el directorio");
        }
    }
}
```

Ejemplo de ejecución:

```
run:
/home/alumno/NetBeansProjects
UT1_24_25
LeeArchivoTexto
ut1_23_24_ejercicios
ut1_23_24
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejercicio: Ampliar el Ejemplo_02 anterior para que se muestre también el contenido de los subdirectorios que haya dentro del directorio (sólo el primer nivel).

Nota: tendrás que utilizar el constructor `File (String parent, String child)` y dos bucles `for`.

```

public static void main(String[] args) {
    // TODO Auto-generated method stub

    File ruta=new File("C:/Users/Juan/Desktop/java_ser");

    System.out.println(ruta.getAbsolutePath());

    String[] nombres_archivos=ruta.list();

    for(int i=0;i<nombres_archivos.length;i++){

        System.out.println(nombres_archivos[i]);

        File f=new File(ruta.getAbsolutePath(),nombres_archivos[i]);

        if(f.isDirectory()){

            String[] archivos_subcarpeta=f.list();

            for(int j=0;j<archivos_subcarpeta.length;j++){

                System.out.println(archivos_subcarpeta[j]);

            }

        }

    }
}

```

Clase File. Ejemplo_03: crear un fichero

```

//----- Crear fichero -----
// Creamos el objeto que encapsula el fichero
// En Windows:
// File fichero = new File("d:\\prueba\\miFichero.txt");

// En Linux:
File fichero = new File("/home/profe/carpetal/fichero.txt"); // <-- El directorio carpetal debe existir
// A partir del objeto File creamos el fichero físicamente
try {
    if (fichero.createNewFile()) {
        //true si el fichero no existe y se ha creado con éxito
        //false si el fichero ya existe
        System.out.println("El fichero " + fichero.getAbsolutePath() + " se ha creado correctamente");
    } else {
        System.out.println("El fichero " + fichero.getAbsolutePath() + " ya existe");
    }
} catch (IOException ioe) {
    System.out.println("Se ha producido una excepción en tiempo de ejecución. Mensaje: " + ioe.getMessage());
}

```

Ejemplos de ejecución:

```

run:
El fichero /home/profe/carpetal/fichero.txt se ha creado correctamente
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

run:
El fichero /home/profe/carpetal/fichero.txt ya existe
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

run:
Se ha producido una excepción en tiempo de ejecución. Mensaje: Permiso denegado
BUILD SUCCESSFUL (total time: 0 seconds)

```

Clase File. Ejemplo_04: borrar un fichero

```
//----- Borrar fichero -----
// Previamente se tiene que haber creado el objeto File que encapsula al fichero que queremos borrar
if (fichero.exists()) { //comprobamos si el fichero existe
    System.out.println("Borramos el fichero " + fichero.getPath());
    fichero.delete();
}
else System.out.println("No existe el fichero " + fichero.getPath());
```

Ejemplos de ejecución:

```
run:
Borramos el fichero /home/profe/carpetal/fichero.txt
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
No existe el fichero /home/profe/carpetal/fichero.txt
BUILD SUCCESSFUL (total time: 0 seconds)
```

Clase File. Ejemplo_05: crear directorios (versión 1)

```
//----- CREAR DIRECTORIO/S -----
// Declaración de variables con distintos ejemplos de rutas
String ruta1 = "/home/alumno/prueba2";
String ruta2 = "carpetal/carpet2/carpet3"; // <-- ruta relativa (desde directorio del proyecto)
String ruta3 = "/home/alumno/ad/prueba2/carpet4/carpet5/carpet6"; // <-- ruta absoluta

// Crear directorio prueba2 -----
File directorio1 = new File(ruta1);
boolean exito = directorio1.mkdir();
if (exito) {
    System.out.println("Directorio: " + ruta1 + " creado");
}

// Crear varios directorios con ruta relativa (1) -----
File directorio2 = new File(ruta2);
exito = directorio2.mkdirs();
if (exito) {
    System.out.println("Directorios " + ruta2 + " creados.");
}

// Crear varios directorios con ruta absoluta (2) -----
File directorio3 = new File(ruta3);
exito = directorio3.mkdirs();
if (exito) {
    System.out.println("Creada la estructura: " + ruta3);
}
```

Ejemplo de ejecución:

```
run:
Directorio: /home/alumno/prueba2 creado
Directorios carpetal/carpet2/carpet3 creados.
Creada la estructura: /home/alumno/ad/prueba2/carpet4/carpet5/carpet6
BUILD SUCCESSFUL (total time: 0 seconds)
```

Clase File. Ejemplo_06: crear directorios (versión 2)

```
//----- Crear directorio/s v2 (versión abreviada - menos código) -----
// Declaración de variables
String ruta1 = "/home/profe/prueba2";
String ruta2 = "carpetal/carpet2/carpet3"; // <-- ruta relativa (desde directorio del proyecto)
String ruta3 = "/home/profe/prueba3/carpet4/carpet5/carpet6"; // <-- ruta absoluta

// Crear directorio prueba2
if ((new File(ruta1)).mkdir()) {
    System.out.println("Directorio: " + ruta1 + " creado");
}
else System.out.println("No se ha podido crear el directorio: " + ruta1);

// Crear varios directorios con ruta relativa (1)
if ((new File(ruta2)).mkdirs()) {
    System.out.println("Directorios " + ruta2 + " creados.");
}
else System.out.println("No se han podido crear los directorio: " + ruta1);

// Crear varios directorios con ruta absoluta (2)
if ((new File(ruta3)).mkdirs()) {
    System.out.println("Creada la estructura: " + ruta3);
}
else System.out.println("No se han podido crear los directorios: " + ruta1);
```

Ejemplo de ejecución:

```
run:
Directorio: /home/profe/prueba2 creado
Directorios carpetal/carpet2/carpet3 creados.
Creada la estructura: /home/profe/prueba3/carpet4/carpet5/carpet6
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Otros métodos y propiedades útiles para el manejo de ficheros

- La clase **System** contiene varios campos y métodos de clase que permiten el acceso a propiedades definidas externamente y variables de entorno:
 - String System.getProperty(String key)**: devuelve el valor de la propiedad de sistema especificada en el parámetro *key*.
 - Properties System.getProperties()**: devuelve las propiedades del sistema en un objeto de la clase **Properties**.

Estas son algunas de las propiedades de sistema que vamos a utilizar en el manejo de ficheros:

Key	Descripción del valor asociado
file.separator	Carácter que separa los componentes de la ruta de un archivo. En UNIX es "/" y en Windows es "\"
user.name	Nombre de usuario
user.home	Directorio home del usuario
user.dir	Directorio actual del usuario

- La clase **File** tiene el campo estático de tipo **String** llamado **separator**, que representa el carácter que separa los componentes de una ruta en el sistema operativo en cuestión ("/" en Unix y "\" en Windows). Podemos usar este campo para que las rutas funcionen independientemente del sistema operativo. Por ejemplo, la ruta "C:\Usuarios\alumno" se pondría como:

```
"C:" + File.separator + "Usuarios" + File.separator + "alumno"
```

Ejemplo: programa para mostrar propiedades del sistema y utilizar el campo `File.separator` para construir rutas.

```
package demos;
import java.io.File;

/**
 * Programa que muestra el valor de variables del sistema
 * También muestra el uso del campo File.separator para construir rutas independientes del S.O.
 */
public class ClaseProperties {
    public static void main(String[] args) {

        String cadenal;

        //Muestra el carácter file.separator
        System.out.println("file.separator: " + System.getProperty("file.separator"));

        // Muestra el directorio actual
        System.out.println("user.dir: " + System.getProperty("user.dir"));

        // Muestra el directorio home del usuario
        System.out.println("user.home: " + System.getProperty("user.home"));

        // Muestra el nombre de usuario
        System.out.println("user.name: " + System.getProperty("user.name"));

        // Muestra todas las propiedades del sistema
        System.out.println("Estas son todas las propiedades del sistema: " + System.getProperties());

        // Construye una ruta a partir del directorio de usuario y la muestra
        cadenal = System.getProperty("user.home") + File.separator + "Documents"
            + System.getProperty("file.separator") + "prueba";
        System.out.println(cadenal);

    } //main
} //Fin clase ClaseProperties
```

➤ Realizar las actividades 1.1 a 1.5

6. La clase JOptionPane

La clase `JOptionPane` de Java se utiliza para mostrar cuadros de diálogo estándar en una interfaz gráfica de usuario (GUI), como mensajes de advertencia, confirmación, introducción de datos, etc. Esta clase forma parte del paquete `javax.swing` y facilita la interacción del usuario con el programa mediante ventanas emergentes.

Método `showInputDialog()`

- Muestra un cuadro de diálogo que solicita la entrada de datos al usuario.

Algunas implementaciones del método son:

- `static String showInputDialog (Object mensaje):` Shows a question-message dialog requesting input from the user.
- `static String showInputDialog (Component parentComponent, Object mensaje):` Shows a question-message dialog requesting input from the user parented to parentComponent.

- `static String showInputDialog (Component parentComponent, Object mensaje, String tituloVentana, int messageType):` Shows a dialog requesting input from the user parented to parentComponent with the dialog having the title title and message type messageType.
- `static Object showInputDialog (Component parentComponent, Object mensaje, String tituloVentana, int messageType, Icon icono, Object[] valores, Object valorInicialSeleccionado):` Prompts the user for input in a blocking dialog where the initial selection, possible selections, and all other options can be specified.
- **Ejemplo:**

```
String nombre = JOptionPane.showInputDialog("Ingrese su nombre:");
```

Método showMessageDialog()

- Muestra un cuadro de diálogo simple con un mensaje de información.
- Se puede utilizar en lugar de System.out.print

Algunas implementaciones del método son:

- `static void showMessageDialog(Component parentComponent, Object message):` Brings up an information-message dialog titled "Message".
- `static void showMessageDialog(Component parentComponent, Object message, String title, int messageType):` Brings up a dialog that displays a message using a default icon determined by the messageType parameter.
- `static void showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon):` Brings up a dialog displaying a message, specifying all parameters.
- **Ejemplo:**

```
JOptionPane.showMessageDialog(null, "Este es un mensaje de información.");
```

Nota: el parámetro **parentComponent** determina el frame (marco) en el que se muestra el diálogo; si es nulo, o si el componente principal no tiene frame, se utiliza un frame predeterminado

Método showConfirmDialog()

- Muestra un cuadro de diálogo de confirmación donde el usuario puede seleccionar opciones como "Sí", "No" o "Cancelar".
- Todas sus implementaciones devuelven un int
- Valores devueltos:
 - `JOptionPane.CLOSED_OPTION`: si se cierra la ventana sin elegir una opción, devuelve -1
 - `JOptionPane.CANCEL_OPTION`: si se pulsa cancelar, devuelve 2
 - `JOptionPane.NO_OPTION`: si se elige NO devuelve 1
 - `JOptionPane.YES_OPTION`: si se elige SI devuelve 0

Algunas implementaciones del método son:

@@@@@ Ver API de Java


• Ejemplo:

```
int respuesta = JOptionPane.showConfirmDialog(null, "¿Desea continuar?");
if (respuesta == JOptionPane.YES_OPTION) {
    // Acción si el usuario elige "Sí"
}
```

El parámetro messageType:

• Parámetro messageType:

- JOptionPane.QUESTION_MESSAGE
- JOptionPane.INFORMATION_MESSAGE
- JOptionPane.WARNING_MESSAGE
- JOptionPane.ERROR_MESSAGE
- JOptionPane.PLAIN_MESSAGE

Icon description	Java look and feel	Windows look and feel
question		
information		
warning		
error		

Ejemplos de uso

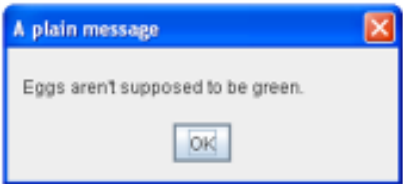
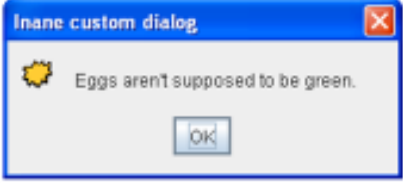




```
//default title and icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.");

//custom title, warning icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane warning",
    JOptionPane.WARNING_MESSAGE);

//custom title, error icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);
```

```
//custom title, no icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "A plain message",
    JOptionPane.PLAIN_MESSAGE);

//custom title, custom icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane custom dialog",
    JOptionPane.INFORMATION_MESSAGE,
    icon);
```



Clase JOptionPane. Ejemplo_01

```
import javax.swing.JOptionPane;

//Ejemplos de uso de JOptionPane
public class UT1_24_25_JOptionPaneEjemplo_01 {

    public static void main(String[] args) {

        // Muestra un mensaje en un InputDialog, guarda el valor introducido en una
        // variable String, y muestra el valor de variable en un MessageDialog

        //Pedir info al usuario
        String nombre = JOptionPane.showInputDialog("Introduce tu nombre:");

        //Mostrar por pantalla
        String saludo = "Hola, " + nombre;
        JOptionPane.showMessageDialog(null, saludo);

    }
}
```

Ejemplo de ejecución:

