Manejo de ficheros en Java

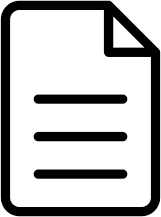
# El archivo

* Se considera un archivo (fichero) en informática a un conjunto de datos que se encuentran almacenados en un dispositivo
* Este conjunto de datos viene agrupado por un nombre, una ruta de acceso y una extensión
* El conjunto de los 3 elementos debe ser único
* Los ficheros se guardan en dispositivos de almacenamiento fijo como discos duros, pendrives, tarjetas, CD, etc.

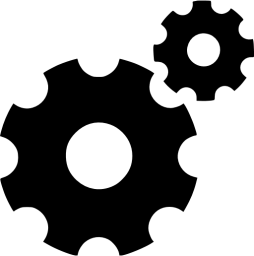
# Sistema de ficheros

* Los ficheros se suelen organizar de forma jerárquica
* No pueden existir ficheros con el mismo:
  + Nombre
  + Ruta
  + Extensión

# Tipos de ficheros

**Fichero estándar:** Fichero que contiene cualquier tipo de datos. Documentos, imágenes, audio, vídeo, etc.

**Directorio o carpeta:** Fichero que contiene otros ficheros. Sirve para organizar de forma jerárquica los diferentes ficheros

**Ficheros especiales:** Ficheros que sirven para controlar los diferentes periféricos conectados al ordenador

# Rutas

* Para acceder a un determinado fichero, se utiliza la **ruta** (path)
* Una ruta indica la dirección del fichero en el sistema de archivos
* En una ruta, cada nivel de la jerarquía se representa delimitado por el símbolo /. En Windows, el símbolo separador es \
* Además de /, existen 2 elementos especiales en la ruta
  + **.** Representa al directorio actual
  + **..** Representa al directorio padre en la jerarquía
* Existen 2 tipos de rutas:
  + **Absoluta**: Ruta al fichero desde el directorio principal (root). **Ej:** */home/Documentos/ejemplo.txt*
  + **Relativa**: Ruta al fichero desde el directorio actual. **Ej:** Estando en home, .*/Documentos/ejemplo.txt*

# Rutas. Ejemplo de árbol de directorios



**Root**

**Informática Automática**



**Programación Pract1**

**Pract2 Pract3**



**Linux**



**Pract4**

**Codigo Informe**

# Extensión de un fichero

* La extensión sirve para asociar un determinado fichero con un programa/s que lo ejecuten o interpreten
* Indica cómo debe ser interpretado el fichero



|  |
| --- |
| doc, docx, odt |
| html, xml, jsp |
| mp3, avi, wma |

* Se considera extensión lo existente a partir del último punto del nombre del archivo
  + Hola.doc  Extensión = doc
  + Hola.doc.mp3  Extensión = mp3

# El archivo

* En un ordenador, lo único que se es capaz de manejar es un conjunto de bits (0, 1)
* Estos bits se agrupan en la unidad mínima, el **byte**
* Podemos considerar a todo fichero en nuestro ordenador como un conjunto de bytes, independientemente de su extensión
* Interpretar el significado de estos bytes es responsabilidad del programa que manipula este fichero

# Operaciones con ficheros

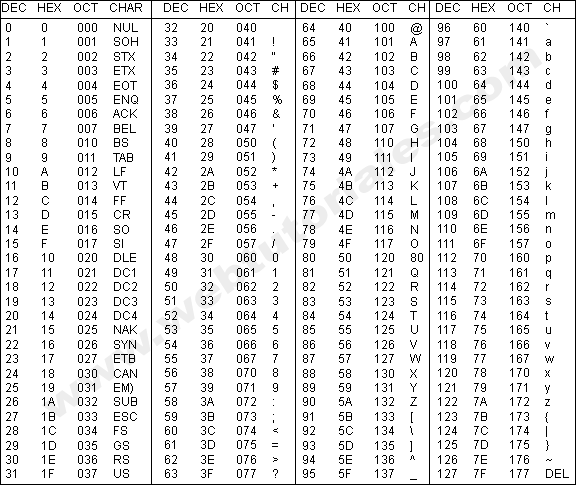
* **Apertura**: El programa abre el fichero y se prepara para leerlo o escribirlo. Suele “reservar” el fichero para sí
* **Cierre**: Indica que se ha finalizado con las operaciones sobre el fichero. Libera el fichero
* **Lectura**: Lee el fichero o una de sus partes
* **Escritura**: Permite escribir en el fichero, ya sea añadiendo datos o sobrescribiendo los ya existentes
* **Ejecución**: Similar a la lectura, pero utiliza los datos del fichero para ejecutar un software
* **Creación**: Crea un nuevo fichero con un nombre, extensión y ruta determinados
* **Eliminación**: Elimina un fichero determinado

# Permisos sobre ficheros

* El usuario que crea el fichero tiene derecho a decidir quién y cómo accede a su fichero
* Existen 3 grupos para los que se les puede definir permisos
  + **u**. Propietario
  + **g**. Grupo
  + **o.** Resto de usuarios
* Los permisos que se pueden dar son los siguientes
  + **r.** Lectura
  + **w**. Escritura
  + **x**. Ejecución

# Codificación de caracteres

* Para poder hacer la equivalencia entre byte y símbolo (número, letra, símbolo especial, etc.) se utiliza la tabla ASCII
* La tabla ASCII permite hasta 128 símbolos



# Codificación de caracteres

* En muchos casos, 128 símbolos es muy limitado (por ejemplo, en ASCII no existe la ñ o las tildes)
* Es por ello por lo que los sistemas utilizan diferentes codificaciones de caracteres, donde se utiliza más de un byte para

representar caracteres

* La elección de la configuración correcta indica que símbolos se pueden representar
* Algunas comunes:
  + **UTF-8, 16, 32**
  + ANSI
  + Estándar por idiomas ISO-XXXX

# Fichero en Java

* En Java se sigue el concepto de un fichero como un conjunto de bytes
* Por lo tanto, Java puede operar con ficheros de cualquier tipo
* Para encontrarlos, su ruta relativa parte de la ubicación del fichero ejecutable
  + Se permiten también rutas absolutas
* En Java se permiten hacer todas las operaciones con ficheros vistas anteriormente:
  + Apertura y cierre
  + Lectura y escritura
  + Creación y eliminación

# Apertura y cierre de un fichero

* Cuando se abre un fichero en Java, se “reserva” el fichero para operar con el
* Se establece un flujo de datos desde el fichero a una variable en Java, que representa al fichero
* A partir de esa variable, se pueden realizar todas las operaciones sobre fichero que se quieran
* Cuando se quiere dejar de usar el fichero, se debe **cerrar** el mismo, cortando el flujo de datos y liberando la variable





Variable Java

Fichero

# La clase File

* La clase que manipula los ficheros en Java se llama **File**
* Con esta clase se pueden hacer un gran número de operaciones sobre un fichero y sus propiedades, pero no se permite

leer ni escribir

* También permite obtener datos del fichero, como rutas, nombres, permisos e incluso si existe
* El resto de clases que manipulan ficheros parten de la existencia de una clase File, por lo que es la base de cualquier operación de manipulación de ficheros

# La clase File

* El constructor de la clase File tiene el siguiente esquema:

File variableFichero = new File(“ruta fichero”);

* Al crear el constructor, la variable **variableFichero** es un objeto con los datos del fichero que se encuentra en la ruta pasada por parámetro, si es que existe
  + La ruta puede ser absoluta o relativa
* Para cerrar un fichero, se usa la siguiente sentencia:

variableFichero.close();

* Con esto se cierra el fichero y la variable **variableFichero** pasará a ser nula (null)

# La clase File

* Además de la apertura y cierre, la clase File permite realizar un gran número de operaciones:
  + Comprobar si el fichero existe
  + Crear un fichero
  + Borrar el fichero
  + Obtener nombre, rutas (absolutas y relativas) y extensión del fichero
  + Decir si el fichero es un fichero o un directorio
  + Obtener el tamaño en bytes del fichero
  + Consultar y cambiar los permisos del fichero
  + Si es un directorio, obtener la lista de ficheros que contiene el mismo
  + Crear nuevos directorios
  + Renombrar fichero
  + Etc.
* Para ver todas las operaciones de una clase File, <https://docs.oracle.com/javase/7/docs/api/java/io/File.html>

# Ejemplo de uso de File

public static void main(String args[]) {

File fichero = new File(“FicheroEjemplo.txt");

if (fichero.exists()) {

System.out.println("Nombre del archivo "+ fichero.getName()); System.out.println(“Ruta "+ fichero.getPath()); System.out.println(“Ruta absoluta "+ fichero.getAbsolutePath()); System.out.println("Se puede escribir "+fichero.canRead()); System.out.println("Se puede leer "+fichero.canWrite()); System.out.println("Tamaño "+fichero.length());

}

fichero.close();

}

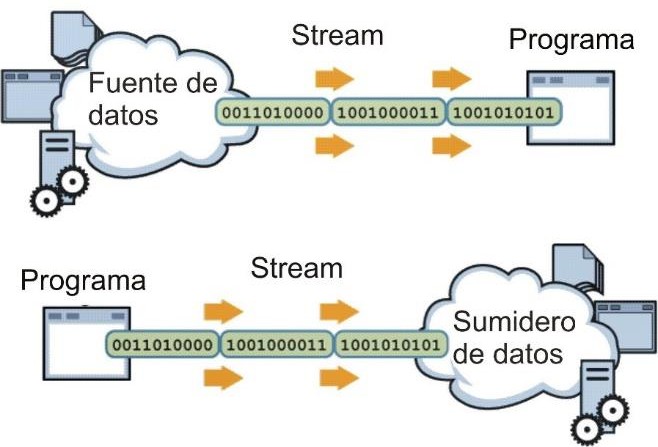
Nombre del archivo: FicheroEjemplo.txt Ruta: FicheroEjemplo.txt

Ruta absoluta: C:\DirectorioEjemplo\FicheroEjemplo.txt Se puede escribir: false

Se puede leer: true Tamaño: 1366

Stream de datos

* Para poder realizar las operaciones de lectura y escritura de ficheros, Java establece lo que se conoce como un Stream de datos
* Crea una vía de comunicación entre programa y fichero que permite “moverse” por las distintas partes del fichero
* Existe un puntero que apunta a las partes del fichero



Variable

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Lectura de datos de un fichero

* Para leer datos de un fichero **de texto**, utilizaremos las siguientes clases:
* **Clase File:** Para representar el fichero que se quiere leer

File fichero = new File(“ruta fichero”);

* **Clase FileReader:** Establece el stream de datos de lectura del fichero. Tiene una serie de métodos para leer carácter a carácter. Al constructor del FileReader recibe el objeto File

FileReader reader = new FileReader(fichero);

* **Clase BufferedReader:** Crea un buffer a través del FileReader, que permite leer mas de un carácter. El constructor recibe el FileReader como parámetro

BufferedReader buffer = new BufferedReader (reader);

# Lectura de datos de un fichero

* Utiliza la función del BufferedReader llamada **readLine()**, la cual:
  + Devuelve la siguiente línea de texto si existe
  + Si no existe, devuelve null

String linea = buffer.readLine();

* Teniendo en cuenta el funcionamiento de readLine(), se puede leer todo el fichero utilizando un bucle while

String linea;

while((linea=buffer.readLine()) != null) {

System.out.println(linea);

}

# Lectura de datos de un fichero

import java.io.\*;

class LeeFichero {

public static void main(String [] arg) { File archivo = null;

FileReader reader = null; BufferedReader buffer = null;

try {

archivo = new File("C:\\directorioArchivo\\archivo.txt"); reader = new FileReader (archivo);

buffer = new BufferedReader(reader);

String linea;

while( (linea=buffer.readLine()) != null) { System.out.println(linea);

}

}

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

}finally{ try{

if( null != fr ){

fr.close();

}

}catch (Exception e2){ e2.printStackTrace();

}

}

}

}

# Escritura de datos de un fichero

* Para escribir datos en un fichero **de texto**, utilizaremos las siguientes clases:
* **Clase File:** Para representar el fichero que se quiere leer

File fichero = new File(“ruta fichero”);

* **Clase FileWriter:** Establece el stream de datos de escritura del fichero. Tiene una serie de métodos para escribir en ficheros. Al constructor del FileWriter recibe el objeto File

FileWriter writer = new FileWriter(fichero);

* **Clase PrintWriter:** Crea un buffer a través del FileWriter, que permite extender los métodos del FileWriter por otros similares a los que tenemos en la salida de pantalla. El constructor recibe el FileWriter como parámetro

PrintWriter pw = new PrintWriter(writer);

# Escritura de datos de un fichero

* Entre las funciones que tenemos en el PrintWriter, las más comunes son:
  + **print(“texto”)**. Imprime el texto pasado por parámetro
  + **println(“texto”)**. Imprime el texto pasado por parámetro y hace un salto de línea
* El constructor del FileWriter puede recibir un segundo parámetro **boolean** que indica si queremos escribir el fichero desde cero (false) o si queremos añadir texto al existente (true)

FileWriter writer = new FileWriter(fichero);

FileWriter writer = new FileWriter(fichero, true);

FileWriter writer = new FileWriter(fichero, false);

Desde cero Añadir texto

# Escritura de datos de un fichero

import java.io.\*;

public class EscribirFichero

{

public static void main(String[] args)

{

File fichero = null; FileWriter writer = null; PrintWriter pw = null; try

{

fichero = new File("C:\\directorioArchivo\\archivo.txt");

writer = new FileWriter(fichero); pw = new PrintWriter(writer);

for (int i = 0; i < 10; i++) { pw.println("Linea " + i);

}

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

} finally { try {

if (null != fichero) { fichero.close();

}

} catch (Exception e2) { e2.printStackTrace();

}

}

}

}

# Borrado de un fichero

* Para borrar un fichero, se utiliza la clase **File**
* Existe la función **delete()** que elimina el fichero

public static void main(String args[]) {

File fichero = new File(“FicheroEjemplo.txt");

if (fichero.exists()) {

System.out.println("Nombre del archivo "+ fichero.getName()); System.out.println(“Ruta "+ fichero.getPath()); System.out.println(“Ruta absoluta "+ fichero.getAbsolutePath()); System.out.println("Se puede escribir "+fichero.canRead()); System.out.println("Se puede leer "+fichero.canWrite()); System.out.println("Tamaño "+fichero.length());

}

fichero.delete();

}

# Posibles problemas

* En las operaciones con ficheros pueden surgir una serie de problemas
* Algunos son:
  + Intentar leer/borrar/escribir sobre un fichero que no existe
  + Intentar leer/borrar/escribir sobre un fichero que ya está siendo modificado por otro programa
  + Intentar escribir o leer un fichero cuando el puntero haya llegado al final del fichero (EOF)

# Control de excepciones

* Java tiene un sistema para controlar los errores que se pueden producir durante la ejecución, llamado control de

**excepciones**

* Se basa en que cuando se produce un error que no se quiere controlar en ese punto, se lanza una excepción
* En otro punto, se controla la excepción y se maneja el error en consecuencia

Controla excepción

catch(Exception)

Lanza excepción

thow Exception()

# Control de excepciones

* Permite una mayor modularidad, ya que facilita la creación de librerías
  + La librería lanza la excepción
  + El código controla la excepción
* Facilita el control de errores, ya que distintos tipos de excepciones se pueden controlar de la misma forma
* La obligación de control de excepciones garantiza un mejor control de los errores

# Sentencias try-catch

* Para controlar las excepciones, se utiliza la sentencia try-catch
  + Dentro del bloque try, se ejecutan las sentencias sobre las que se quiere controlar las excepciones
  + Dentro del bloque catch, se muestran las sentencias que se deben ejecutar en caso de error

try {

//SENTENCIAS A EJECUTAR

} catch (Exception e) {

//SENTENCIAS QUE SE EJECUTAN SI HAY ERROR

}

* Como parámetro, el catch pone el tipo de excepción a controlar. Si se quieren controlar todas, se usa

**Exception**

* La sentencia finally es opcional, y contiene las sentencias que se van a ejecutar exista o no una excepción

# Control de excepciones

import java.io.\*;

public class EscribirFichero

{

public static void main(String[] args)

{

File fichero = null; FileWriter writer = null; PrintWriter pw = null; **try**

**{**

fichero = new File("C:\\directorioArchivo\\archivo.txt");

writer = new FileWriter(fichero); pw = new PrintWriter(writer);

for (int i = 0; i < 10; i++) { pw.println("Linea " + i);

}

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

**} finally { try {**

if (null != fichero) { fichero.close();

}

**} catch (Exception e2) { e2.printStackTrace();**

**}**

**}**

}

}

# Control de excepciones

* Si las excepciones no se controlan, se muestra el detalle por consola. Muestra la pila de ejecución

**Tipo excepción**

**Descripción excepción**

**Pila ejecución**

StudentException: Error finding students

at StudentManager.findStudents(StudentManager.java:13) at StudentProgram.main(StudentProgram.java:9)

Caused by: DAOException: Error querying students from database at StudentDAO.list(StudentDAO.java:11)

at StudentManager.findStudents(StudentManager.java:11)

... 1 more

Caused by: java.sql.SQLException: Syntax Error

at DatabaseUtils.executeQuery(DatabaseUtils.java:5) at StudentDAO.list(StudentDAO.java:8)

... 2 more