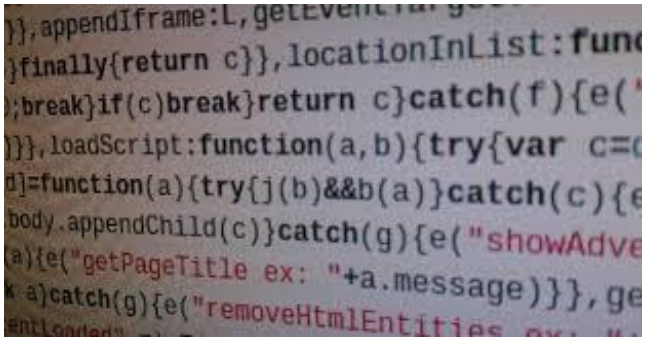


UT 12

REGISTROS, FICHEROS Y OPERACIONES. 2ª Parte.

IES JUAN DE LA CIERVA
DPTO. INFORMÁTICA

CONTENIDOS
DE LA
PRESENTACIÓN

Objetivos

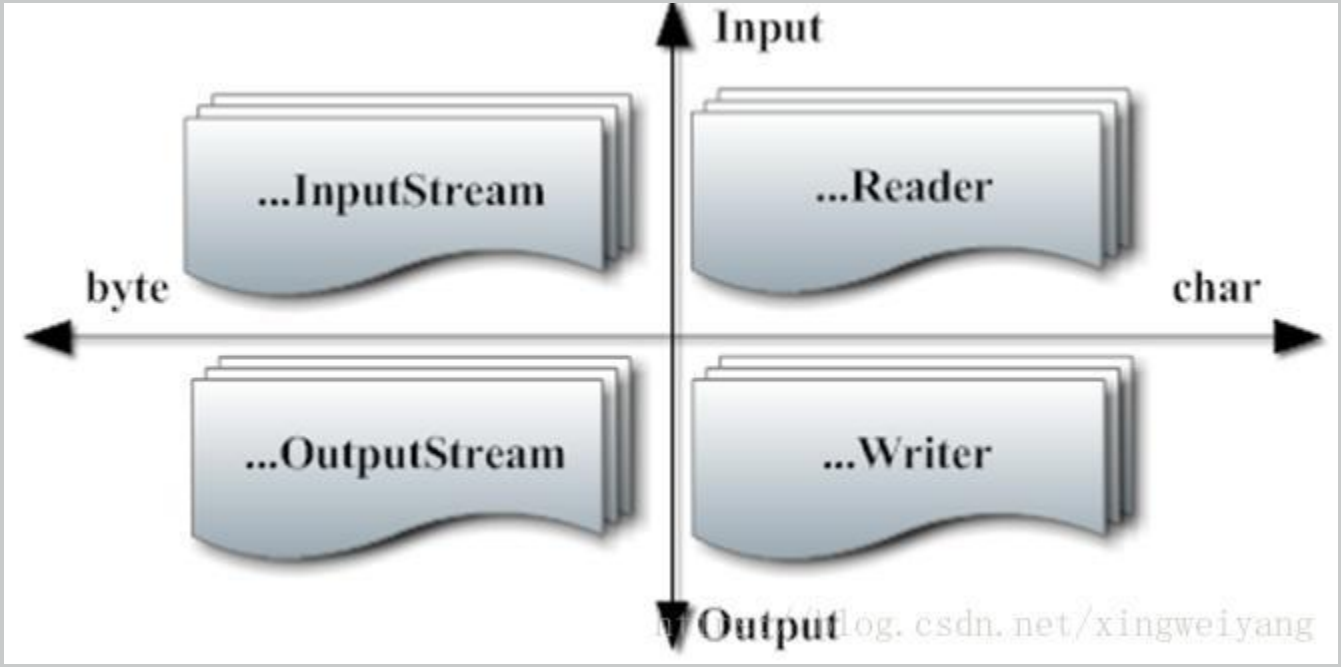
La unidad permite al alumno conocer los conceptos de los distintos tipos de ficheros con los que nos podemos encontrar (byte y carácter), los distintos tipos de operaciones que podemos realizar con ellos y su manejo.

Contenidos

- Conceptos **de ficheros.**
- Tipos de ficheros (secuenciales y directos).

Operaciones sobre ficheros.

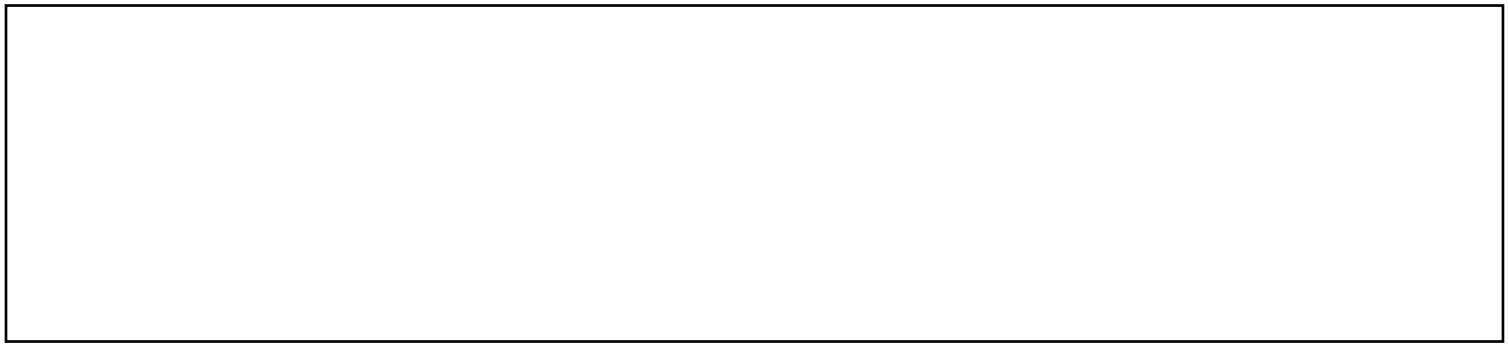
- Creación, lectura y escritura ficheros de Texto.
- **Creación, lectura y escritura ficheros binarios.**
- **Modificación y actualización de ficheros.**
- **Eliminación de ficheros.**



Ficheros:Acceso con Flujo de Bytes

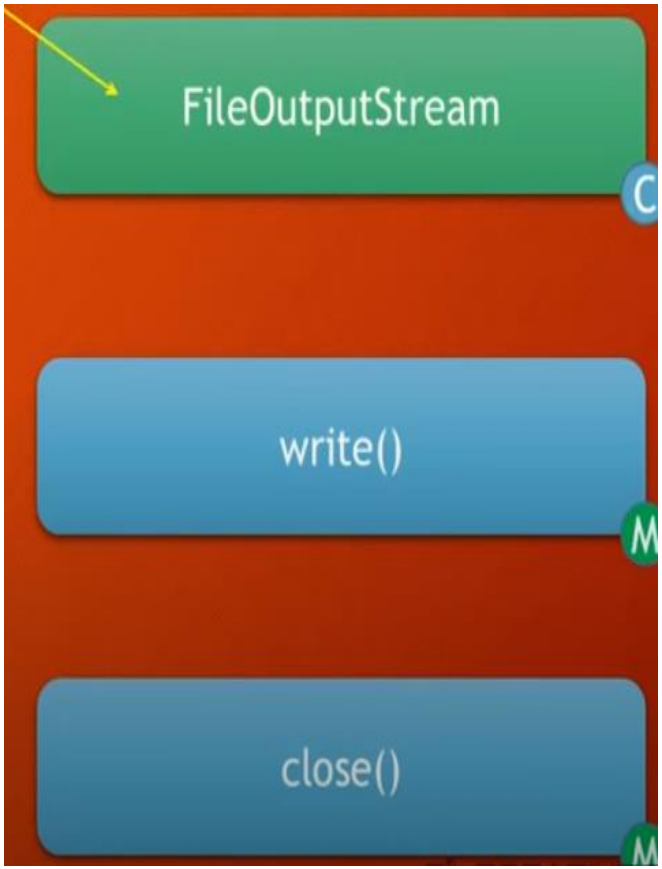
CLASES DE FLUJOS DE BYTES

CLASES DE FLUJOS DE BYTES



```
62  55 247 114 163 100  94 202 128  10 1
46 156 152  68  64  94 137 120 135  47
10 156 100 226 149  17 184 173 120   4 2
61  96 203 109  74  91 230  71  86 105 1
23 175 113 192  11 100 198 248  29 136
17 230   5 180 163 249   7 123 102 101
14  24 193 113 226 190  27 193 139  91 1
62  46 121 199  38   6 234 247  20  42 1
28  52 123  55 126  70 228  35  85 168
16 124 213 154 102  61 180  40 171  41 1
66  37   6 211 104  21 182 222 122  85
44 194  92  93 252 151 222 122  87  65 2
09 191 131 101 250  21 152 100 204 130
69  64 164 132 236 118 138 113 159  48
60 169 101 138  59  53 217 165 240  17 2
14  11 164 129  17   7 203 247 156 232
```

CLASES DE FLUJOS DE BYTES



CLASES DE FLUJOS DE BYTES

Clase de flujo de bytes	Significado
BufferedInputStream **BufferedOutputStream**	Flujo de entrada y salida en búfer
ByteArrayInputStream **ByteArrayOutputStream**	Flujo de entrada y salida en una matriz de bytes
DataInputStream **DataOutputStream**	Flujo de entrada y salida que contiene métodos para leer los tipos de datos estándar de JAva
FileInputStream **FileOutputStream**	Flujo de entrada y salida que lee y escribe desde un archivo
FilterInputStream **FilterOutputStream**	Implementa InputStream y OutputStream
InputStream **OutputStream**	Clase abstracta que describe flujos de entrada y salida
ObjectdInputStream **ObjectOutputStream**	Flujo de entrada y salida de objetos
PipeddInputStream **PipedOutputStream**	Conducción de entrada y salida en búfer
PrintStream	Flujo de salida que contiene print() y println()
PushbackInputStream	Flujo de entrada que permite devolver bytes de flujo
SecuenceInputStream	Flujo de entrada que combina dos o más flujos de entrada que se leen secuencialmente, uno tras otro

CLASES DE FLUJOS DE BYTES

Clases InputStream y OutputStream: Clases abstractas que describe flujos de entrada y salida.

MÉTODOS de la clase InputStream	
Método	Descripción
int available()	Devuelve el número de bytes de entrada disponibles para lectura.
void close()	Cierra el origen de entrada. Los posteriores intentos de lectura generan IOException.
void mark(int numBytes)	Añade una marca al punto actual de flujo de entrada que se valida hasta que se lean numBytes.
boolean markSupported()	Devuelve true si el flujo de invocación admite mark()/reset().
int read()	Devuelve una representación entera del siguiente byte de entrada disponible. Devuelve -1 al llegar al final del flujo.
int read (byte buffer[])	Intenta leer hasta los bytes de buffer.length en búfer y devuelve el número real de bytes leídos correctamente. Se devuelve -1 al llegar al final del flujo.
int read(byte buffer[], int desplazamiento, int numBytes)	Intenta leer hasta los bytes de numBytes en búfer comenzando en búfer[desplazamiento] y devuelve el número de bytes leidos correctamente. Devuelve -1 al llegar al final del flujo.
void reset()	Restablece el puntero de entrada a la marca definida previamente.
long skip(long numBytes)	Ignora los bytes de entrada indicados por numBytes y devuelve el número de bytes ignorados

CLASES DE FLUJOS DE BYTES

Clases InputStream y OutputStream: Clases abstractas que describe flujos de entrada y salida.

MÉTODOS de la clase OutputStream	
Método	**Descripción**
void flush()	Envía a su destino la salida que se haya guardado en búfer. Es decir, vacía el búfer de salida.
void close()	Cierra el origen de salida. Los posteriores intentos de escritura generan IOException.
void write(int b)	Escribe un solo byte en un flujo de salida. El parámetro es int, lo que le permite invocar write() con expresiones sin tener que convertirlas de nuevo a byte.
void write(byte buffer [])	Escribe una matriz completa de bytes en un flujo de salida.
void write(byte buffer[], int desplazamiento, int numBytes)	Escribe un subintervalo de numBytes bytes desde el búfer de matriz, empezando por búfer[desplazamiento]

LECTURA DE UN ARCHIVO DE BYTES

1. **Apertura del fichero** : mediante la creación de un objeto **FileInputStream.**

   Constructor:
   FileInputStream(String nombreArchivo) throw FileNotFoundException

2. **Lectura del Fichero.**
   int read() throws IOException

   - Cada vez que se invoca, read() lee un byte del archivo y lo devuelve como valor entero.
   - Devuelve -1 al llegar al final del fichero.
   - Genera IOException en caso de error.

3. **Cerrar el Fichero.**
   void close() throws IOException
   Al cerrar un archivo se liberan los recursos del sistema asociados al mismo.

EJEMPLO USO DE FICHERO DE BYTES

EJEMPLO : Leer un Fichero de Imagen.

Sitúa una imagen en una carpeta en el escritorio y recupérala utilizando un programa en Java.

Muestra los bytes según los vaya leyendo y cuenta cúantos bytes ha leído.

ESCRIBIR EN UN ARCHIVO DE BYTES

1. **Apertura del fichero** : mediante la creación de un objeto **FileOutputStream.**

   Constructor:
   - FileOutputStream(String nombreArchivo)
     throws FileNotFoundException //destruye ficheros con mismo nombre
   - FileOutputStream(String nombreArchivo, boolean append)
     throws FileNotFoundException //si append true añade al final
     //si append false sobreescribe

2. **Escritura del Fichero.**
   void write(int valbyte) throws IOException

   - Este método escribe el byte especificado en valbyte en el archivo.
   - Aunque valbyte se declara int, sólo se escriben en el archivo los 8 bits de orden inferior.
   - Si se produce un error de escritura, se genera un IOException.

3. **Cerrar el Fichero.**
   void close() throws IOException
   Al cerrar un archivo se liberan los recursos del sistema asociados al mismo.

EJEMPLO USO DE FICHERO DE BYTES

EJEMPLO : CopiarFichero

Partiendo del ejemplo anterior haz una copia del fichero imagen en otra carpeta distinta a la que de origen.
Al fichero nuevo dale el nombre FicheroCopia.xxx

CERRAR AUTOMATICAMENTE UN ARCHIVO

Desde JDK7 hay una forma de gestionar los recursos. Es lo que se llama **try con recursos** o administración automática de recursos.

Su principal **ventaja** es que evita situaciones en las que un archivo (u otro recurso) no se libera después de que deje de ser necesario.

Formato general:

```
try (especificación-recurso){
    // usar el recurso
}
```

Ejemplo:

```
try (FileInputStream fin = new FileInputStream(args[0])){
        ……..……
} //(*) se pueden especificar varios ficheros separados por ;
```

PRÁCTICA 13.2

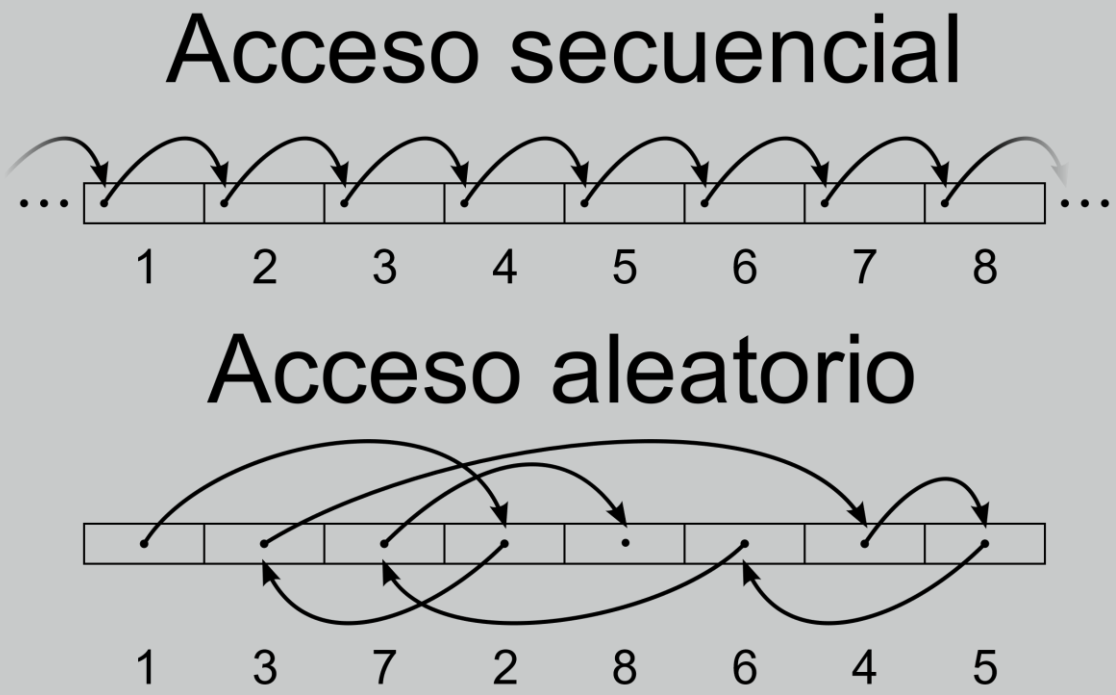
Desarrolla una utilidad para comparar archivos de bytes secuenciales. La llamarás **CompararFicheros**.

Abre dos archivos para comparar y después lee y compara cada bytes leído.

Si detecta una diferencia, mostrar el mensaje de que los archivos son distintos.

Si se llega simultáneamente al final de cada archivo y no hay diferencias, los archivos son iguales.

- Utiliza la nueva instrucción try con recursos.
- Mejóralo con distintas opciones. Por ejemplo, añada una opción que ignore la diferencia entre mayúsculas y minúsculas de las letras.
- Hacer que CompararFicheros muestre la posición del archivo, en que se ha encontrado la primera diferencia.



Ficheros de Acceso Aleatorio

ARCHIVOS DE ACCESO ALEATORIO

Java también permite acceder a los contenidos de un archivo de forma aleatoria.

Para ello se utiliza **RandomAccessFile**, que contiene un archivo de acceso aleatorio.

Esta clase permite leer y escribir en los archivos el registro que nosotros le indiquemos sin tener que leer los registros guardados en posiciones físicas anteriores a él.

ARCHIVOS DE ACCESO ALEATORIO

Constructor:

RandomAccessFile(String nombreArchivo, String acceso) throws FileNotFoundException

Al abrir el fichero debemos no sólo indicar el nombre del archivo o el FILE sino el modo de apertura.

Acceso: **“r”** para leer

“rw” el archivo se abre en modo lectura y escritura

ARCHIVOS DE ACCESO ALEATORIO

Las excepciones que pueden aparecer son:

EOFException.
FileNotFoundException.
IOException.

Algunos de los métodos que tiene esta clase son:

seek(long pos) - permite colocarse en una posición concreta, contada en bytes, en el archivo. Lo que se coloca es el puntero de acceso que es la señal que marca la posición a leer o escribir.

void seek(long nuevaPos) throws IOException

ARCHIVOS DE ACCESO ALEATORIO

Podremos leer y escribir los tipos primitivos, con métodos como:

- **readInt()**
- **readLine()**
- **writeDouble()**

long getFilePointer( ) - posición actual del puntero de acceso.

long length( ) - devuelve el tamaño del archivo.

Int skipbytes(int d) - desplaza el puntero del fichero las posiciones en bytes que le estamos indicando.

ARCHIVOS DE ACCESO ALEATORIO

Para leer o escribir utilizamos los métodos de las clases **DataInputStream** y **DataOutputStream**.

MÉTODOS DE SALIDA	MÉTODOS DE ENTRADA
void writeBoolean(boolean val)	boolean readBoolean()
void writeByte(int val)	byte readByte()
void writeChar(int val)	char readChar()
void writeDouble(double val)	Double readDouble()
void writeFloat(float val)	float readFloat()
void writeInt(int val)	Int readInt()
void writeLong(float val)	long readLong()
void writeShort(float val)	short readShort()

EJEMPLO DE ACCESO ALEATORIO

Ejemplo que ilustra la E/S de acceso aleatorio.

Escribe seis valores double en un archivo y después los lee de forma no secuencial.

Ejemplo: Ejemplo6_EscrituraYAccesoAleatorio

ALGUNAS CONSIDERACIONES. ACCESO DIRECTO

- Los registros deben ser de longitud fija, por lo que debemos leer/escribir siempre el número de bytes correspondientes de cada registro.

- Podemos tener posiciones de registros en los que no hemos grabado nada.

- Antes de grabar en el fichero, debemos comprobar que no está ocupada esas posiciones por otro registro ya que perderíamos sus datos. Por lo que debemos posicionarnos, leer lo que hay, volver a posicionarnos para escribir si está vacío o sustituir por otro valor. Puede darse el caso de colisiones (dos registros tienen la misma ubicación en el archivo), en ese caso deberíamos guardar el registro al final del fichero en posiciones consecutivas (llamada zona de excedentes).

ALGUNAS CONSIDERACIONES. ACCESO DIRECTO

- Si vamos a procesar todo el fichero antes de realizar la operación deseada sobre el registro, comprobar que tiene información. Ya que puede que no tenga nada grabado en dicha posición.

- Podemos preparar el fichero antes de grabar los datos correctos, enviando un registro nulo a todas las posiciones que puede ocupar el fichero. De esta forma sabremos qué valor tenemos en los campos o en alguno de ellos si no ha sido grabado.

- Para salir del fichero podemos buscar una posición inexistente.

- El formato writeUTF, guarda dos bytes más para para saber cuántos bytes tiene que leer.

PRÁCTICA 13_3. BONOLOTO

Hacer un programa que realice la siguiente secuencia de operaciones utilizando arrays, ficheros y los correspondientes accesos aleatorios o secuenciales.

1. Generar 200 número aleatorios comprendidos entre 1 y 49 y almacenar el número de veces que ha aparecido cada valor en un vector de 49 posiciones.
2. Grabar el contenido del array en un fichero de bytes que trataremos con acceso aleatorio.
3. Recorrer el fichero y visualizar su contenido por pantalla.
4. Programar un proceso en bucle en el que podamos consultar la frecuencia de aparición de un número determinado.

Creación y Eliminación de Ficheros desde el programa

CREACIÓN Y ELIMINACIÓN DE ARCHIVOS

Clase: File

Métodos:

- createNewFile()
- delete()

Ejemplo: Ejemplo7_CrearBorrarFichero