# UNIDAD VI

## ARCHIVOS

M.C. ELIZABETH CASTELLANOS NOLASCO

- Existen 2 tipos de archivos:
  - Archivos de acceso Secuencial
    - Archivos de texto
    - Archivos con datos primitivos
  - Archivos de acceso aleatorio.

# Clase File

- La clase *File*, no se utiliza para transferir datos sino que **nos proporciona información acerca de los archivos**, de sus atributos, de los directorios, etc. incluso para creación o eliminación de éstos.
- La clase *File* tiene tres constructores:

  *File(String path)*

  *File(String path, String name)*

  *File(File dir, String name)*

# Algunos de los métodos más importantes que describe esta clase son los siguientes:

- *String getName()*
- *String getPath()*
- *String getAbsolutePath()*
- *boolean exists()*
- *boolean canWrite()*
- *boolean canRead*
- *boolean isFile()*
- *boolean isDirectory()*
- *long length()*
- *boolean mkdir()*
- *boolean renameTo(File nuevo)*
- *boolean delete()*

# Creación de un objeto File

- Creamos un objeto *fichero* de la clase *File*, pasándole el nombre del archivo, en este caso, el nombre del archivo "*datos.txt*"

```
File fichero=new File("datos.txt");
```

ó

```
String cd;
File fichero =new File(cd);
```

- *Para crear un directorio:*

```
File fichero2=new File("midirectorio");
```

ó

```
String cd;
File fichero =new File(cd);
```

# Información de un archivo

- *boolean canWrite()* :Indica si se puede escribir en el archivo.
- *boolean canRead():* Indica si se puede leer el archivo.
- *boolean exists():* Verifica si existe o no el archivo o directorio.
- *boolean isFile()* : Si el objeto File hace referencia o no a un fichero.
- *boolean isDirectory()* : Si el objeto File hace referencia o no a un directorio.

- *String getName()* : Devuelve el nombre del archivo sin ruta.
- *String getPath()* : Devuelve la ruta relativa del archivo.
- *String getAbsolutePath(): Devuelve la ruta completa donde está el archivo.*
- *boolean delete()* : Elimina el archivo o directorio especificado por el objeto File.

*(un directorio solo podrá ser eliminado si está vacío).*

```
if(fichero.exists()){

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

- El proceso para leer o escribir datos consta de tres pasos:
  - Abrir el archivo de datos
  - Mientras exista información, leer o escribir los datos
  - Cerrar el archivo de datos

- Todas estas clases se encuentran en el paquete ***java.io***, por lo que al principio del código fuente tendremos que escribir la sentencia

```
import java.io.*;
```

# Lectura de un archivo de texto

- Para recuperar cadenas de caracteres de un archivo, el paquete java.io proporciona dos clases:
  - *FileReader*
  - *BufferedReader*

# Creación del objeto FileReader

- Para recuperar información es necesario crear un objeto FileReader asociado al mismo.
- Este objeto representa un fichero de texto "abierto" para la lectura de datos.
- Esto lo hace porque es capaz de adaptar la información recuperada del archivo a las características de una aplicación java, transformando los bytes almacenados en caracteres unicode.

- Se puede construir el objeto a través de los siguientes constructores:
  - *FileReader (String path)*
  - *FileReader (File archivo)*
- Ejemplo:

```
File f = new File (“datos.txt”);
FileReader fr = new FileReader(f);
```

- Cuando FileReader lee la información del archivo, primero los recupera de éste como tipo byte y posteriormente los convierte en String.

- Por esta razón es más fácil recurrir a la clase BufferedReader para realizar esta operación, utilizando el objeto FileReader como puente para crear un objeto de este tipo.

# Creación de un objeto BufferedReader

- Una vez creado el objeto FileReader, el segundo paso consiste en crear un BufferedReader, para lo cual utilizamos:

```
BufferedReader bf = new BufferedReader(fr);
```

- Una vez creado el objeto, puede utilizarse el método *readLine()* para leer líneas de texto en forma similar como se lee de teclado.

- Dado que un archivo puede estar formado por más de una línea, necesitamos utilizar un ciclo *while* para recuperar todas las líneas de texto en forma secuencial.

- Ahora bien, si lo que se quiere leer es caracter por caracter en vez de línea por línea, debemos utilizar el método *read()* en lugar de *readLine().*

- El método *read()* devuelve un entero que representa el código unicode del caracter leído, siendo el resultado -1 si no hay mas caracteres para leer.

- También podemos utilizar la clase Scanner, para ello se pasará el objeto File asociando el archivo al constructor del Scanner.

# Los pasos para leer y escribir en disco son los siguientes:

- Se crean dos objetos de las clases *FileReader* y *FileWriter*, llamando a los respectivos constructores a los que se les pasa los nombres de los archivos o bien, objetos de la clase *File*, respectivamente

```
entrada=new FileReader("Archivo3.java");
salida=new FileWriter("copia.java");
```

- Se lee mediante *read* los caracteres del flujo de entrada, hasta llegar al final (la función *read* devuelve entonces -1), y se escribe dichos caracteres en el flujo de salida mediante *write*.

```
while((c=entrada.read())!=-1){
salida.write(c);          }
```

- Finalmente, se cierran ambos flujos llamando a sus respectivas funciones *close* en bloques **try..catch**

```
entrada.close();
salida.close();
```

ArchivoEscribir.java

ArchivoLeer.java

El archivo de datos es: datos.txt

# Leer y escribir datos primitivos

- La clase *DataInputStream* es útil para leer datos del tipo primitivo de una forma portable. Esta clase tiene un sólo constructor que toma un objeto de la clase *InputStream* o sus derivadas como parámetro.
- Se crea un objeto de la clase *DataInputStream* vinculándolo a un un objeto *FileInputStream* para leer desde un archivo en disco denominado pedido.txt.

- FileInputStream fileIn=new FileInputStream("pedido.txt");

  DataInputStream entrada=new DataInputStream(fileIn);

## o en una sola línea

- DataInputStream entrada=new DataInputStream(new FileInputStream("pedido.txt"));

- La clase *DataInputStream* define diversos métodos *readXXX* que son variaciones del método *read* de la clase base para leer datos de tipo primitivo

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

- La clase *DataOutputStream* es útil para escribir datos del tipo primitivo de una forma portable. Esta clase tiene un sólo constructor que toma un objeto de la clase *OutputStream* o sus derivadas como parámetro.
- Se crea un objeto de la clase *DataOutputStream* vinculándolo a un un objeto *FileOutputStream* para escribir en un archivo en disco denominado pedido.txt.

- FileOutputStream fileOut=new FileOutputStream("pedido.txt");
  DataOutputStream salida=new DataOutputStream(fileOut);

o en una sola línea

- DataOutputStream salida=new DataOutputStream(new FileOutputStream("pedido.txt"));

- La clase *DataOutputStream* define diversos métodos *writeXXX* que son variaciones del método *write* de la clase base para escribir datos de tipo primitivo

*void writeBoolean(boolean v);*
*void writeByte(int v);*
*void writeBytes(String s);*
*void writeShort(int v);*
*void writeChars(String s);*
*void writeChar(int v);*
*void writeInt(int v);*
*void writeLong(long v);*
*void writeFloat(float v);*
*void writeDouble(double v);*

# *Ejemplo: un pedido*

- En este ejemplo, se escriben datos a un archivo y se leen del mismo, que corresponden a un pedido
- La descripción del item, un objeto de la clase *String*
- El número de unidades, un dato del tipo primitivo **int**
- El precio de cada item, un dato de tipo **double**.

| | **Escritura** | **Lectura** |
|---|---|---|
| Un caracter | *writeChar* | *readChar* |
| Un entero | *writeInt* | *readInt* |
| Un número decimal | *writeDouble* | *readDouble* |
| Un string | *writeChars* | *readLine* |

Veamos el código que escribe los datos a un archivo pedido.txt en disco.

1. Se parte de los datos que se guardan en los arrays denominados *descripciones*, *unidades* y *precios*
2. Se crea un objeto de la clase *DataOutputStream* vinculándolo a un objeto *FileOutputStream* para escribir en un archivo en disco denominado pedido.txt..
3. Se escribe en el flujo de salida los distintos datos llamando a las distintas versiones de la función *writeXXX* según el tipo de dato (segunda columna de la tabla).
4. Se cierra el flujo de salida, llamando a su función miembro *close*.

```
double[] precios={1350, 400, 890, 6200, 8730};

int[] unidades={5, 7, 12, 8, 30};

String[] descripciones={"paquetes de papel", "lápices", "bolígrafos", "carteras", "mesas"};


DataOutputStream salida=new DataOutputStream(new FileOutputStream("pedido.txt"));

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

    salida.writeChars(descripciones[i]);

    salida.writeChar('\n');

    salida.writeInt(unidades[i]);

    salida.writeChar('\t');

    salida.writeDouble(precios[i]);

}

salida.close();
```

Pedido.java
Pedidoimprime.java

Veamos el código que lee los datos a un archivo pedido.txt en disco

1. Se crea un objeto de la clase *DataInputStream* vinculándolo a un un objeto *FileInputStream* para leer en un archivo en disco denominado pedido.txt..
2. Se lee el flujo de entrada los distintos datos en el mismo orden en el que han sido escritos, llamando a las distintas versiones de la función *readXXX* según el tipo de dato (tercera columna de la tabla).
3. Se guardan los datos leídos en memoria en las variables denominadas *descripcion*, *unidad* y *precio* y se usan para distintos propósitos
4. Se cierra el flujo de entrada, llamando a su función miembro *close*.

```
  double precio;
  int unidad;
  String descripcion;
  double total=0.0;

  DataInputStream entrada=new DataInputStream(new
FileInputStream("pedido.txt"));
  try {
      while ((descripcion=entrada.readLine())!=null) {
          unidad=entrada.readInt();
          entrada.readChar();       //lee el carácter tabulador
          precio=entrada.readDouble();
          System.out.println("has pedido "+unidad+" "+descripcion+" a
"+precio+" pts.");
          total=total+unidad*precio;
      }
  }catch (EOFException e) {}
  System.out.println("por un TOTAL de: "+total+" pts.");
  entrada.close();
```

# El final del archivo

```
try {
   while(true){
         descripcion=entrada.readLine();
         unidad=entrada.readInt();
         entrada.readChar();      //lee el caracter tabulador
         precio=entrada.readDouble();
         System.out.println("has pedido "+unidad+"
         "+descripcion+" a "+precio+" pts.");
         total=total+unidad*precio;
   }
 }catch (EOFException e) {
   System.out.println("Excepción cuando se alcanza el
   final del archivo");
```

# Otros ejemplos

- Agenda completa
  - Programa: AgendaCompleta.java
  - Archivo de datos: agenda.txt

  Este ejercicio captura alumnos e imprime el archivo completo.

- Atributos
  - Programa : Atributos.java
  - Este programa solicita el nombre de archivo que deseas

  revisar y regresa algunos de sus atributos por ejemplo si ponemos pedido.txt o si ponemos un nombre de carpeta que este en la misma ruta.