Ficheros

- Los ficheros permiten almacenar información de forma permanente en un soporte externo. Están habitualmente formados por registros. Un registro es una estructura heterogénea de datos, denominados campos y a los que accedemos por su nombre. En Java no existen estructuras de registro, si bien el adecuado diseño de clases posibilita la gestión de los mismos.
- Java, a diferencia de otros lenguajes, no tiene métodos específicos para abrir archivos.
- La acción de construir un *FileReader*, *FileWriter, FileInputStream* o *FileOutputStream* equivale a una operación de apertura.

Lectura de archivos de texto clase Reader

- Para leer un fichero de texto es necesario crear un *FileReader* y un *BufferedReader*
- *FileReader* es una clase derivada de *InputStreamReader*, que permite leer ficheros de caracteres utilizando la codificación por defecto.
- *BufferedReader* es una clase que permite la lectura de bloques de datos.
- Una vez creados ambos objetos, se puede leer el fichero, línea a línea, utilizando el método *readLine*() o carácter a carácter utilizando el método *read*().

Lectura de archivos de texto clase Reader

```
public class Main {

  public static void main(String args[]){
  String texto="";
       try{
          FileReader fichero=new FileReader("Agenda.txt");
          BufferedReader fE=new BufferedReader(fichero);
          texto = fE.readLine();
          while(texto != null){
                 int posi=texto.indexOf(",");
                 String nombre=texto.substring(0,posi);
                 String teléfono=texto.substring(posi+1);
                 System.out.print("Nombre: "+nombre);
                 System.out.println(", Teléfono: "+teléfono);
                texto = fE.readLine();
           }
        fE.close();
       }catch(IOException e){
          System.out.println(e.getMessage());
       }
   }
  }
```

Lectura de archivos de texto clase Reader

- El fichero a leer en este caso, "*Agenda.txt*", está formado por nombres y teléfonos, separados por "," estructurados en líneas. El programa lee hasta el final del fichero (*while texto!=null*), extrayendo los datos y visualizándolos.
- Al llegar al final del fichero, éste se cierra utilizando el método *close*().
- Es obligatorio controlar excepciones.

Escritura en archivos de texto clase Writer

- *FileWriter* es una clase derivada de *OutputStreamWriter (*derivada de *Writer)* que se utiliza para escribir *streams* de caracteres en un archivo.
- Tiene varios constructores, entre ellos, uno que permite crear un objeto para añadir datos a un fichero ya existente:

  FileWriter (String filename, boolean append)

- La clase *BufferedWriter* escribe texto almacenado en el *buffer* asociado con el objeto *Writer* especificado. Permite una gestión más eficiente que *FileWriter*.
- Puede utilizarse en método *write()* para escribir datos en un *stream*.

Escritura en archivos de texto clase Writer

```
public class Main{
public static void main(String args[]){
Scanner sc = new Scanner(System.in);
String nombre;
try{
   FileWriter flS=new FileWriter("Agenda.txt",true);
   BufferedWriter fS=new BufferedWriter(flS);
   do{
   System.out.print("Introduce un nombre: ");
   nombre = sc.nextLine();
   if(nombre.length()>0){
       System.out.print("Teléfono:");
       String teléfono = sc.nextLine();
       fS.write(nombre+","+teléfono);
        fS.newLine();
    }
  } while(nombre.length()>0);
  fS.close();
  }catch(IOException e){
   System.out.println(e.getMessage());
   }
}
}
```

Escritura en archivos de texto clase Writer

- Se añaden datos al fichero *Agenda.txt* , ya que se utiliza *true* como segundo argumento. En caso de no utilizarlo, si el fichero ya existiera, quedaría destruido.
- Se introducen nombres y teléfonos separados por comas y se graban utilizando el método *write()*.
- El método *newline()* inserta un salto de línea entre datos, en lugar de utilizar '\n'.
- El programa finaliza al pulsar *Enter* en el nombre.
- Al llegar al final del fichero, éste se cierra utilizando el método *close*(), con esto quedarían grabados los datos que pudieran quedan en el *buffer* pendientes de almacenar. Si queremos almacenar esos datos antes de cerrar, podemos utilizar el método *flush*().
- Es obligatorio controlar excepciones.

Lectura de archivos de datos clase InputStream

- *DataInputStream* es una clase que permite leer directamente desde el *stream* datos primitivos: *boolean, byte, double, float, int, long, short.*
- El constructor tiene como argumento un objeto de la clase *FileInputStream*.
- Los métodos de lectura que pueden aplicarse son los siguientes:

readBoolean(), readByte(), readDouble(), readFloat(), readInt(), readLong(), readShort(),readChar()

Lectura de archivos de datos clase InputStream

- Cada método de entrada devuelve el tipo de datos primitivo indicado por el nombre del método.
- Estos métodos no devuelven ningún valor que pueda utilizarse como indicador de fin de fichero. Cuando un método de lectura alcanza el final del fichero se arroja una excepción de fin de archivo (*EOFException*). Por tanto, el bucle que lee los datos debe de ir encerrado entre un bloque *try-catch*

Lectura de archivos de datos clase InputStream

```
public class Main {
    public static void main(String[] args) {
    int i;
    double nota;
    try {
    FileInputStream fich=new FileInputStream("binario.dat");
    DataInputStream datos = new DataInputStream(fich);
       try{
          while (true)
          {
              i=datos.readInt();
              System.out.println("Número: "+i);
              nota=datos.readDouble();
              System.out.println("Nota: "+nota);

           }
         }catch (EOFException eof){
            datos.close();}
    } catch (IOException ex) {
         System.out.println("Error "+ex.getMessage()); }
    }}
```

Escritura en archivos de datos clase OutputStream

- *DataOutputStream* es una clase que permite escribir datos primitivos: *boolean, byte, double, float, int, long, short, char* a un *stream*.
- El constructor tiene como argumento un objeto de la clase *FileOutputStream*.
- Los métodos de escritura que pueden aplicarse son los siguientes:

writeBoolean(), writeByte(), writeDouble(), writeFloat(), writeInt(), writeLong(), writeShort(),writeChar()

Escritura en archivos de datos clase OutputStream

```
public class Main {
    public static void main(String[] args) {
    int i;
    double nota;
    Scanner teclado=new Scanner(System.in);
    try {
        FileOutputStream fich=newFileOutputStream("binario.dat");
        DataOutputStream datos = new DataOutputStream(fich);
        System.out.println("Número , 0 finaliza");
        i=teclado.nextInt();
        while (i!=0)
        {
             datos.writeInt(i);
             System.out.println("Nota (0-10)");
             nota=teclado.nextDouble();
             datos.writeDouble(nota);
             System.out.println("Número , 0 finaliza");
             i=teclado.nextInt();
         }
          datos.close();
    } catch (IOException ex) {
         System.out.println("Error: "+ex.getMessage());}
    }
}
```

Lectura de archivos clase Scanner

- La clase *Scanner* también puede utilizarse para leer ficheros.
- El constructor debe tener como argumento un objeto de tipo *File*().
- Puede utilizarse el método *nextLine*() para leer línea a línea.
- Puede utilizarse el método *hasNextline*() para comprobar si existen más líneas en el fichero.

Lectura de archivos clase Scanner

```
public class Main{
    public static void main (String args[]){
    try{
    Scanner sc = new Scanner(new File ("binario.dat"));
           while (sc.hasNextLine()) {
               String texto = sc.nextLine();
               System.out.println(texto);

           }
        }catch(Exception e){}
    }
}
```

Modos de acceso a ficheros

- Las clases vistas hasta ahora permiten acceder a los ficheros de modo secuencial.
- Existe una clase que permite la creación de ficheros de acceso directo: *RandomAccessFile*.
- El constructor tiene como primer argumento un fichero y como segundo el modo de acceso al mismo. Los principales modos de acceso son los siguientes:
  - "r" ->acceso para lectura.
  - "rw"->acceso para lectura y escritura. Si el fichero no existe, se creará.

Modos de acceso a ficheros

- Puede utilizarse el método *seek* para situar el puntero de lectura y escritura en cualquier parte del fichero.
- Existen métodos para leer en el fichero, dependiendo del tipo de dato a leer: *readChar, readBoolean, readByte, readInt, readDouble,* etc.
- Existen métodos para escribir en el fichero, dependiendo del tipo de dato a escribir: *writeChar, writeBoolean, writeByte, writeInt, writeDouble,* etc.

```
//grabación de datos
public class Main {
    public static void main(String argumentos[]){
    try {
    Scanner sc = new Scanner(System.in);
    RandomAccessFile t = new RandomAccessFile("Datos.dat","rw");
    int numRegistro,longitudRegistro=56;
   //int numRegistro(4)+ int edad(4)+ double peso(8)+ String nombre(40->2 por carácter)
            if (t.length()!=0)//si hay registros
            {   numRegistro=(int)t.length()/longitudRegistro;
                t.seek(t.length());//sitúa el puntero al final del archivo
            }
            else
                numRegistro=0;
    String nombre="";
    int edad;
    double peso;
    System.out.print("Nombre: Enter finaliza");
    nombre = sc.nextLine();
    while (!nombre.isEmpty())
    {
        System.out.print("Edad: ");
        edad = Integer.parseInt(sc.nextLine());
        System.out.print("Peso: ");
        peso = Double.parseDouble(sc.nextLine());
        numRegistro++;
        t.writeInt(numRegistro);
        for (int i=0;i<20;i++)
        if (i<nombre.length())
            t.writeChar(nombre.charAt(i));
        else
            t.writeChar(' ');//rellena con espacios en blanco hasta 20
        t.writeInt(edad);
        t.writeDouble(peso);
        System.out.print("Nombre: Enter finaliza");
        nombre = sc.nextLine();
     }
    t.close();
 }catch (IOException ex)System.out.println("error"+ex.getMessage());}}}
```

```
//lectura
public class Main {
    public static void main(String[] args) {
        try {
            RandomAccessFile t = new RandomAccessFile("Datos.dat", "r");
            int numRegistro,longitudRegistro=56;
            String nombre;
            int edad;
            double peso;
            //número de registros
            long numReg=t.length()/longitudRegistro;
            for (int num=1;num<=numReg;num++)
            {
                nombre = "";//inicializa para cada registro
                numRegistro=t.readInt();
                for (int i = 0; i < 20; i++) {
                    nombre += t.readChar();
                }
                edad=t.readInt();
                peso=t.readDouble();
                System.out.println("Número: "+numRegistro);
                System.out.println("Nombre: "+nombre);
                System.out.println("Edad: "+edad);
                System.out.println("Peso: "+peso);
            }
            t.close();
        } catch (IOException ex) {
            System.out.println("error"+ex.getMessage());
        }
    }
}
```

Serialización

- La serialización es el proceso por el cual un objeto o una colección de objetos se convierten en una secuencia de *bytes*, que pueden ser almacenados en un fichero y recuperados posteriormente para su uso.
- Cuando se serializa un objeto se almacena la estructura de la clase y todos los objetos referenciados por éste.
- Java proporciona distintos *interfaces* del paquete *java.io* para resolver distintos casos de serialización.
- La serialización más sencilla es la que proporciona la *interface Serializable*. Por tanto, los objetos que deseamos serializar deben implementar dicha *interface*.

Serialización

- Se utilizan las clases *ObjectInputStream* y *ObjectOutputStream* con los métodos *writeObject* y *readObject*.
- Al serializar un objeto, se serializan todas las propiedades, tanto primitivas como objetos miembros.
- Los objetos de tipo *static* no son serializados, por lo que deben ser guardados y recuperados de forma independiente.
- Los métodos *writeInt, writeDouble, readInt* y *readDouble*, entre otros, permiten serializar de forma explícita variables primitivas de sus respectivos tipos.

```
public class Serializable1{
    public static void main(String argumentos[]){
        try{
            Scanner sc = new Scanner(System.in);
            System.out.print("Nombre vulgar: ");
            String texto=sc.nextLine();
            Arbol p = new Arbol(texto);
            System.out.print("Nombre cientifico: ");
            texto=sc.nextLine();
            p.ponNombreCientifico(texto);
            //escritura
            FileOutputStream f=new FileOutputStream("tmp");
            ObjectOutputStream fis=new ObjectOutputStream(f);
            fis.writeObject(p);
            fis.close();
            //lectura
            FileInputStream fe = new FileInputStream("tmp");
            ObjectInputStream fie=new ObjectInputStream(fe);
            Arbol q = (Arbol)fie.readObject();
            System.out.println(q.muestraArbol());
            fie.close();
        }catch ( Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

```
public class Arbol implements Serializable{
    private String nombreVulgar;
    private String nombreCientifico;

    Arbol(String nombre){
        this.nombreVulgar=nombre;
    }

    public void ponNombreCientifico(String nombre){
        this.nombreCientifico=nombre;
    }

    public String muestraArbol(){
        return nombreVulgar+ " "+nombreCientifico;
    }
}
```

Gestión de ficheros y directorios clase File

- Representa un fichero o un directorio.
- El constructor puede tener:
  - Un solo argumento: un *String* indicando una ruta o el nombre de un archivo.
  - Dos argumentos: el primero debe ser un *String* o un *File* con una ruta y el segundo un *String* con el nombre del archivo.

Clase File
Métodos

Tipo devuelto	métodos	Acción
boolean	*isDirectory()*	comprueba si es directorio
boolean	*isFile()*	comprueba si es fichero
boolean	*exists()*	comprueba si existe el fichero o directorio
String	*getName()*	devuelve el nombre del archivo
String	*getPath()*	devuelve el Path
String[]	*list()*	devuelve una lista con archivos y directorios
boolean	*canRead()*	comprueba si el fichero se puede leer
boolean	*canWrite()*	comprueba si se puede escribir sobre el fichero
boolean	*canExecute()*	comprueba si el fichero se puede ejecutar

Clase File
Métodos

boolean	*mkdirs()*	crea los directorios necesarios de un path
boolean	*delete()*	borra un archivo o directorio vacío
long	*length()*	tamaño del fichero en bytes
long	*lastModified()*	devuelve la fecha de la última modificación
boolean	*renameTo()*	cambia el nombre del fichero
static File[]	*listRoots()*	devuelve los volúmenes de almacenamiento
File[]	*listFiles()*	devuelve archivos y directorios

Ejemplo atributos

```
public class Main{
    public static void main(String args[]){
        String nombreF;
        InputStreamReader flujo =new InputStreamReader(System.in);
        BufferedReader teclado = new BufferedReader(flujo);
        try{
            System.out.println("Introduce el nombre del fichero");
            nombreF = teclado.readLine();
            File f = new File(nombreF);
            System.out.println( "Nombre: "+f.getName() );
            System.out.println( "Camino: "+f.getPath() );
            if(f.exists()){
              System.out.println( "El fichero existe" );
              if(f.canRead())
                System.out.println("Se puede leer");
              if(f.canWrite())
                System.out.println( "Se puede modificar");
              System.out.println("La longitud del fichero es:"
                                 + f.length()+" bytes" );
            }else
                System.out.println( "El fichero no existe." );
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

Ejemplo mkdir, delete

```
public class Main{
public static void main(String args[]){
 File directorio = new File("directorio");
 if (directorio.mkdir())
  System.out.println("Se ha creado directorio");
 else
  System.out.println("No se ha podido crear el directorio");
 if(directorio.delete())
  System.out.println("Se ha borrado el directorio");
 else
 System.out.println("No se ha podido borrar el directorio");
   }
}
```