UT 6: Archivos

1. Entrada/salida.

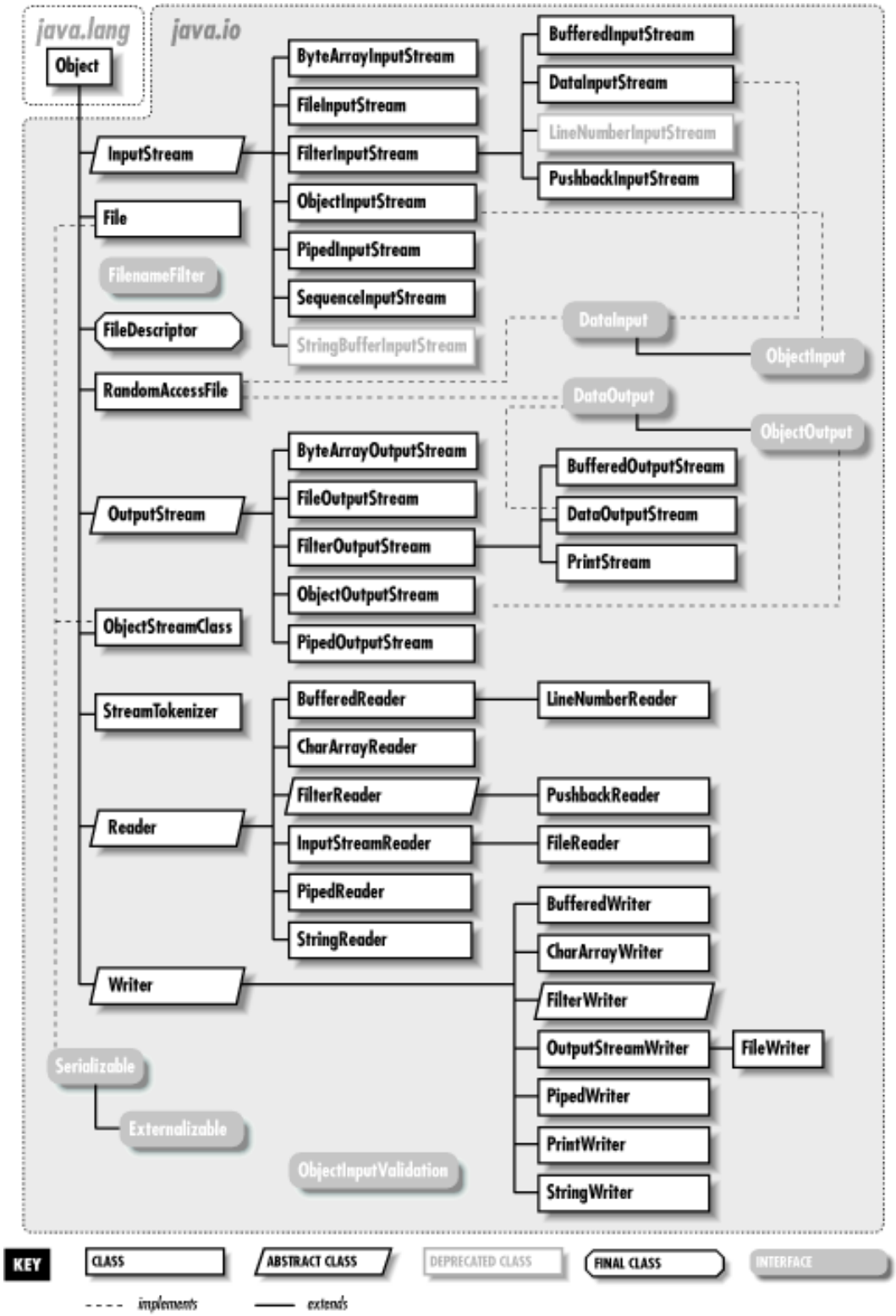
El paquete java.io contiene todas las clases relacionadas con las funciones de entrada (*input*) y salida (*output*). Se habla de E/S (o de I/O) refiriéndose a la entrada y salida. En términos de programación se denomina entrada a la posibilidad de introducir datos hacia un programa; salida sería la capacidad de un programa de mostrar información al usuario.

Todas las clases relacionadas con la entrada y salida de datos están en el paquete "*java.io*"

2. Archivos o ficheros. Java IO.

El apartado de ficheros es realmente importante puesto que permite a un programa almacenar información de forma "permanente" una vez acabada su ejecución o leer información de archivos físicos.

Para la gestión de ficheros existen una serie de clases importantes dentro de "java.io":



Algunas de las más importantes sería:

- **File**: es una representación abstracta de un archivo o directorio. Esta clase no se utiliza para escribir o leer datos, sino para trabajar a alto nivel. Es decir, para crear, buscar, eliminar archivos y para trabajar con directorios y rutas.

- **Reader/Writer**: son clases abstractas que permiten la E/S de ficheros de texto, es decir, permiten la lectura/escritura de *arrays* de caracteres. Las anteriores gestionan ficheros binarios y estas ficheros de texto.
    - **InputStreamReader/OutputStreamWriter:** son clases concretas que heredan de Reader/Writer.
    - **BufferedReader/BufferedWriter:** permiten almacenar temporalmente los datos para su tratamiento. Los métodos más importantes son **readLine**() o **write**(String)
    - **PrintWriter:** pensada para la impresión de textos**.**
- **InputStream/ OutputStream**: son clases **abstractas** que permiten streams de E/S, es decir, corrientes de datos binarios recogidos byte a byte.
    - **DataInputStream/DataOutputStream:** son clases concretas que heredan de InputStream/OutputStream.
        - Permiten el manejo de bytes y los adaptan a los tipos primitivos: **readBoolean(), writeBoolean(boolean), readInt(), writeInt(int), etc**.
        - También pueden manejar bytes, líneas y lecturas con formato: **readByte() , readLine(), readUTF(), etc**. (lo mismo para write).
    - **BufferedInputStream/BufferedOutputStream:** similar a BufferedReader/BufferedWriter pero para la lectura de bytes.
    - **ObjectInputStream/ObjectOutputStream:** también son clases concretas de InputStream/OutputStream. Solo permiten leer bytes, siempre que los datos almacenados sean objetos.
        - **readObject(), writeObject(Object)**

Importante: los métodos de las clases anteriores suelen lanzar varias excepciones (*IOException*, habitualmente y algunas más), por tanto, habría que gestionar bien las Excepciones.

2.1. Serializable

Para poder enviar/recibir un objeto por la red o escribir/leer objetos en ficheros se necesitan convertirlos en una secuencia de bytes. Esta tarea se puede llevar a cabo gracias a la interfaz Serializable.

Simplemente se debe implementar esa interfaz y nada más, puesto que no tiene métodos.

```
3  import java.io.Serializable;
4
5  public class Cliente implements Serializable{
6
7      private static final long serialVersionUID = 5066111006632621208L;
8
9
10     protected String nombre;
11     protected int tfno;
12     protected String email;
13
14
15     public Cliente(String nombre, int tfno, String email) {
16         this.nombre = nombre;
17         this.tfno = tfno;
18         this.email = email;
19     }
20
```

La única recomendación de *Serializable* sería generar un *long* (*serialVersionUID*) que nos permite controlar las versiones de ese objeto, de forma que tanto en el emisor como en el receptor debemos tener la misma versión de esa clase.

2.2. Ejemplo escritura de un fichero de texto con *PrintWriter*

```
    public static void escrituraFicheroTextoPW() {
        FileWriter fw = null;
        PrintWriter pw = null;
        try {
            fw = new FileWriter("Ficheros//pruebaFicheroIO.txt");

            // O si se quiere añadir info al final de un fichero existente:
//          fichero = new FileWriter("Ficheros//pruebaFicheroIO.txt", true);
            pw = new PrintWriter(fw);

            System.out.println("Comenzamos a escribir...");

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

            System.out.println("Fin de la escritura.");

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (fw != null) {
                    fw.close();
                }

                if (pw != null) {
                    pw.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

2.3. Ejemplo escritura de un fichero de texto con *BufferedWriter*

```
public static void escrituraFicheroTextoBW() {
    FileWriter fw = null;
    BufferedWriter bw = null;

    try {

        fw = new FileWriter("Ficheros//pruebaFicheroIO_BW.tx");
        bw = new BufferedWriter(fw);

        System.out.println("Comenzamos a escribir...");

        for (int i = 0; i < 10; i++) {
            bw.write("Linea " + i + "\n");
        }

        System.out.println("Fin de la escritura.");

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (fw != null) {
                fw.close();
            }

            if (bw != null) {
                bw.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2.4. Ejemplo lectura de un fichero de texto

```
public static void lecturaFicheroTexto() {
    File archivo = null;
    FileReader fr = null;
    BufferedReader br = null;

    try {
        // Apertura del fichero y creacion de BufferedReader para poder
        // hacer una lectura comoda (dispone del metodo readLine()).
        archivo = new File("Ficheros//pruebaFicheroIO.txt");
        fr = new FileReader(archivo);
        br = new BufferedReader(fr);

        // Lectura del fichero
        String linea;

        while ((linea = br.readLine()) != null) {
            System.out.println(linea);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (fr != null) {
                fr.close();
            }

            if (br != null) {
                br.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2.5. Ejemplo escritura de un objeto en un fichero binario

```java
public static void escrituraFicheroBinarioObjeto() {
    Cliente cliente = new Cliente("Fran", 666777888, "fran@fran.es");

    FileOutputStream fos = null;
    ObjectOutputStream oos = null;

    try {
        System.out.println("Vamos a escribir un cliente en un fichero: ");
        fos = new FileOutputStream("Ficheros/cliente.bin");
        oos = new ObjectOutputStream(fos);

        // Escribo el cliente en el fichero:
        oos.writeObject(cliente);
        System.out.println("Cliente escrito con éxito!");

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (fos != null) {
                fos.close();
            }

            if (oos != null) {
                oos.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

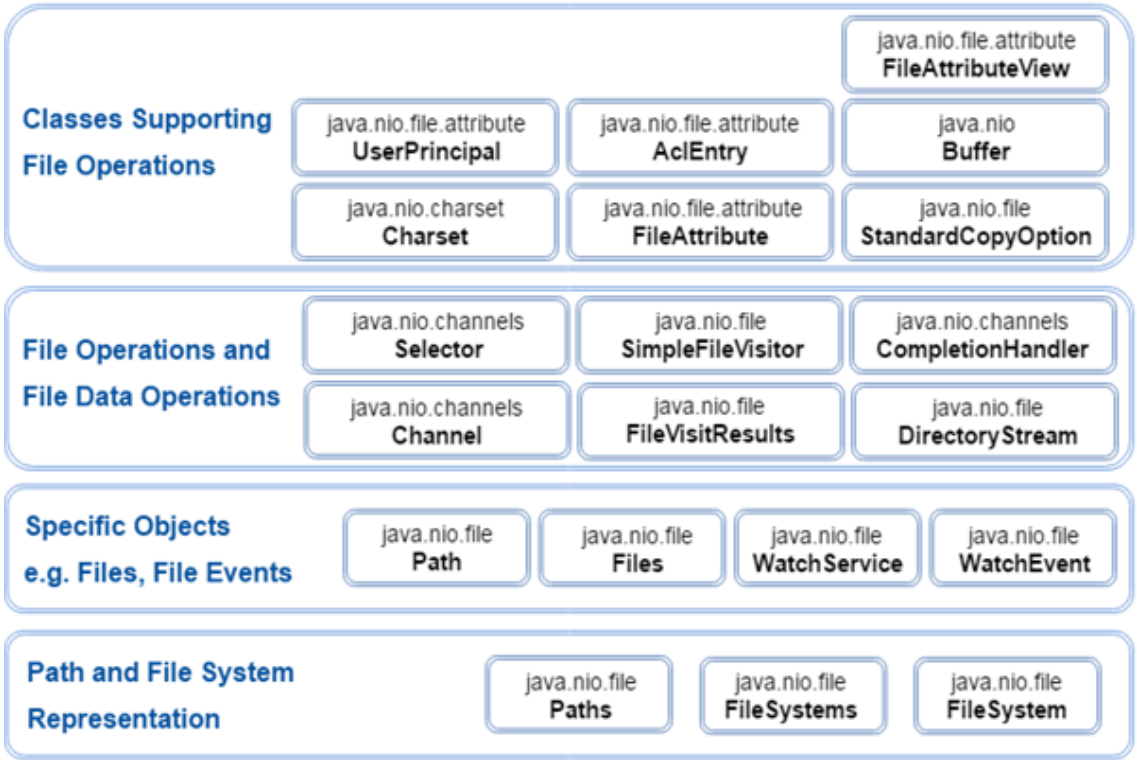
Nota: el objeto cliente debe ser *Serializable*, tanto para lectura como para escritura.

2.6. Ejemplo lectura de un objeto en un fichero binario

```java
public static void lecturaFicheroBinarioObjeto() {

    Cliente cliente = null;
    FileInputStream fis = null;
    ObjectInputStream ois = null;

    try {
        System.out.println("Vamos a leer un objeto cliente de un fichero: ");
        fis = new FileInputStream("Ficheros/cliente.bin");
        ois = new ObjectInputStream(fis);

        // Leo tantos clientes como pueda.
        while (true) {
            try {
                cliente = (Cliente) ois.readObject();
                System.out.println("Cliente leído: " + cliente + "\n");
            } catch (EOFException e) {
                // Cuando lleguemos al final del fichero, salimos con el break.
                System.out.println("Fin de la lectura");
                break;
            }
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        try {
            if (fis != null) {
                fis.close();
            }

            if (ois != null) {
                ois.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3. Ficheros con Java 7 o posterior. Java NIO.

A partir de Java 7 se incluyen en el paquete "java.nio" una serie de clases que facilitan el trabajo con ficheros.



3.1. Try-with-resources

Además del paquete "*nio*", surge otro concepto importante, "*try-with-resources*". En Java 6 todos los objetos relacionados con ficheros, BBDD, etc. tienen que cerrarse de forma explícita, normalmente en el *finally*, sin embargo, con Java 7 no es necesario, siempre y cuando estos objetos implementen la interfaz "*java.lang.AutoCloseable*" o "*java.io.Closeable*".

Por ejemplo:

```
7   */
8  public interface ObjectInput extends DataInput, AutoCloseable {
9      /**
0       * Read and return an object. The class that implements this interface
1       * defines where the object is "read" from.
2       *
3       * @return    the object read from the stream
4       * @throws    java.lang.ClassNotFoundException If the class of a serialized
5       *            object cannot be found.
6       * @throws    IOException If any of the usual Input/Output
7       *            related exceptions occur.
8       */
```

3.2. Ejemplo de escritura en un fichero de texto

```
public static void escrituraFicheroPath() {
    String prueba = "Esto es una prueba\nPrueba";
    Path file = Paths.get(RUTA+"/pruebaFicheroNIO.txt");

    // Try-with-resources
    // Dentro del try se definen todos los objetos que necesitan cerrarse.
    // De forma automática, se cerrarán una vez que salgan del try.
    try(BufferedWriter bw = Files.newBufferedWriter(file, StandardCharsets.UTF_8)) {

        System.out.println("Escribimos en el fichero...");

        bw.write(prueba, 0, prueba.length());

        System.out.println("Fin de la escritura.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.3. Ejemplo de lectura en un fichero de texto

```
public static void lecturaFicheroPath() {
    Path file = Paths.get(RUTA+"/pruebaFicheroNIO.txt");

    try(BufferedReader br = Files.newBufferedReader(file, StandardCharsets.UTF_8)) {
        System.out.println("Vamos a leer un fichero...");
        String line = null;
        while((line = br.readLine()) != null) {
            System.out.println(line);
        }

        System.out.println("Fin de la lectura.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.4. Ejemplo de lectura en un fichero de binario con Objetos

```
public static void lecturaFicheroObjetoPath() {
    Cliente cliente = null;

    Path file = Paths.get(RUTA+"/pruebaFicheroObjetoNIO.bin");

    try(InputStream is = Files.newInputStream(file);
            ObjectInputStream ois = new ObjectInputStream(is);) {

        System.out.println("Leemos el objeto cliente en un fichero...");

        // Leo tantos clientes como pueda.
        while (true) {
            try {
                cliente = (Cliente) ois.readObject();
                System.out.println("Cliente leído: " + cliente + "\n");
            } catch (EOFException e) {
                // Cuando lleguemos al final del fichero, salimos con el break.
                System.out.println("Fin de la lectura");
                break;
            }
        }

    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```