1) File

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase File representa un elemento cualquiera del sistema de archivos, lo cual puede ser un fichero o un directorio, y da métodos que nos permiten obtener información sobre él.

File
+ File (String ruta) + boolean exists() + boolean canRead() + String getName() + String getPath() + boolean isFile() + boolean isDirectory() + long length() + boolean renameTo(File f) + boolean createNewFile() + boolean mkdirs() + boolean delete() + File[] listFiles()

- El **constructor** permite obtener un objeto File a partir de una ruta. Si dicha ruta tiene subdirectorios, se recomienda usar la barra **/** para separarlos, ya que funciona bien en Windows y Linux.
- **exists**: Es true si el elemento del sistema de archivos definido en el objeto File existe.
- **canRead**: Es true si el programa tiene permiso de lectura sobre el elemento del sistema de archivos definido en el objeto File.
- **getName**: Devuelve el nombre del elemento definido en el objeto File.
- **getPath**: Devuelve la ruta completa del elemento definido en el objeto File.
- **isFile**: Devuelve true si el elemento definido en el objeto File es un archivo.
- **isDirectory**: Devuelve true si el elemento definido en el objeto File es un directorio.
- **length**: Devuelve el tamaño en bytes del elemento definido en el objeto File.
- **renameTo**: Cambia el nombre del elemento definido en el objeto File al que pasamos como parámetro.
- **createNewFile**: Crea un archivo vacio en la ruta indicada en el objeto File, siempre que no exista ya un archivo que se llame así.
- **mkdirs**: Crea un directorio vacio en la ruta indicada en el objeto File, creando además todos los subdirectorios necesarios para ello que no existan y se encuentren en la ruta.
- **delete**: Borra el elemento definido en el objeto File. Devuelve false si no se puede.
- **listFiles**: Cuando el objeto File representa un directorio, devuelve una lista con todos los archivos y subdirectorios que hay en él.

2)OutputStream

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase OutputStream (*traducida como canal de salida, flujo de datos de salida*) representa un lugar genérico al que un programa puede enviar datos. Un OutputStreames por tanto, cualquier cosa que puede recibir datos en forma de bytes desde un programa, como por ejemplo un archivo, una impresora, la tarjeta de sonido, etc.

OutputStream
+ intwrite (int b) throws IOException + void write (byte[] b) throws IOException + void write (byte[] b, intinicio, intcantidad) throws IOException + void flush() throws IOException + void close() throws IOException

- El primer método **write** escribe un byte sobre el canal de salida. Es necesario recordar que un byte no es más que una pequeña cantidad de datos y que todos los tipos de datos están compuestos por bytes (por ejemplo, un int son 4 bytes). Del int que se pasa como parámetro, solo el último byte es escrito sobre el canal, ignorándose los restantes bytes. Por tanto, con este método solo se puede escribir un número entre 0 y 255. Se lanza una excepción si se produce un error de escritura sobre el medio.

- El segundo método **write** escribe en el medio de salida todos los bytes que hay almacenados en el array que se pasa como parámetro.

- El tercer método **write** escribe la parte del array de bytes que se pasa como parámetro, cuyo inicio y tamaño se indican como segundo y tercer argumento.

- El método **flush** hace que el medio grabe físicamente los bytes que posee en su caché interna, si es que la tiene. Algunos dispositivos, para ir más rápido, poseen una memoria (caché) donde guardan los datos que reciben desde el programa y que cada cierto tiempo van grabandode forma física[1].

- El método **close** cierra el contacto con el canal de salida y finaliza el intercambio de datos entre el programa y el medio. Debe usarse siempre que se ha terminado de trabajar con el medio.

[1] Debe tenerse en cuenta que el acceso a memoria siempre es mucho más rápido que el acceso al dispositivo físico.

3)FileOutputStream

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase FileOutputStreamrepresenta un canal de salida de datos hacia un archivo del ordenador. Por tanto, permite a un programa enviar bytes a un archivo.

FileOutputStreamextendsOutputStream
+ FileOutputStream(String ruta) throws IOException + FileOutputStream (String ruta, booleanañadir) throws IOException

- El primer **constructor** crea un FileOutputStreampara el archivo cuya ruta pasamos como parámetro, teniendo muy en cuenta que:

  - Si el archivo no existe, se crea nuevo.
  - Si el archivo ya existe, se borra y se crea nuevo.

- El segundo **constructor** crea un FileOutputStreampara el archivo cuya ruta pasamos como parámetro, teniendo muy en cuenta que:

  - Si el archivo no existe, se crea nuevo.
  - Si el archivo ya existe y el segundo parámetro es false, se borra y se crea nuevo.
  - Si el archivo ya existe y el segundo parámetro es true, el archivo original no se borra, sino que los bytes que se graben se añadirán al final.

4)InputStream

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase InputStream(*traducida como canal de entrada, flujo de datos de entrada*) representa un lugar genérico que puede proporcionar datos a un programa. Un InputStreamespor tanto, cualquier cosa que puede aportar datos en forma de bytes a un programa, como por ejemplo un archivo, un teclado, un escáner, un joystick, un ordenador de la red local, etc.

InputStream
+ intread() throws IOException + int read (byte[] b) throws IOException + int read (byte[] b, intinicio, intlongitud) throws IOException + booleanmarkSupported() throws IOException + void mark (int bytes) throws IOException + void reset () throws IOException + intavailable() throws IOException + long skip (int bytes) throws IOException + void close() throws IOException

- El primer método **read** lee un solo byte desde el canal de entrada.Conviene recordar que un byte es la unidad mínima de la que están formados todos los tipos de datos y se corresponde con un número entre 0 y 255. Este método lanza una excepción si se produce un error de lectura, o devuelve -1 si el medio de entrada ya no tiene más datos para leer.
- El segundo método **read** rellena un array con tantos bytes de máximo como indica su capacidad. Devuelve el número de bytes leidos.
- El tercer método **read** rellena la parte del array pasado como parámetro definida por su posición de inicio y longitud. Devuelve el número de bytes leidos.
- El método **markSupported** devuelve true si el medio de entrada de datos admite los métodos mark y reset. No todos los medios los admiten.
- Cuando se llama al método **mark**, el canal crea una señal interna que se conserva durante el número de bytes de lectura indicados como parámetro. Si durante esos bytes se llama al método reset, el canal retrocede y se posiciona nuevamente en el punto donde se puso la marca, volviéndose a leer los mismos bytes otra vez.
- El método **reset** hace que el canal de datos vuelva a la posición donde se puso la marca.
- El método **available** devuelve el número de bytes que pueden ser leídos desde la caché del canal antes de que se produzca un acceso físico al medio a por más datos. No todos los medios lo soportan.
- El método **skip** descarta el número de bytes pasado como parámetro. No todos los medios lo soportan.
- El método **close** cierra el canal de entrada de datos. Debe usarse para liberar recursos tras finalizar el trabajo con el stream.

5)FileInputStream

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase FileInputStreamrepresenta un canal de salida de datos procedentes de un archivo del ordenador.

FileInputStreamextendsInputStream
+ FileInputStream (String ruta) throws IOException

- El **constructor** crea un FileInputStream a partir del archivo cuya ruta pasamos como parámetro. Por tanto, sirve para introducir bytes a un programa procedentes del archivo.

6)PrintWriter

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase PrintWriter permite crear y escribir en un archivo de texto.

PrintWriter extends Writer
+ PrintWriter (String ruta) throws FileNotFoundException + PrintWriter (String ruta, boolean añadir) throws FileNotFoundException + void println(String texto) + void print(int dato) + void print(long dato) + void print(double dato) + void print(boolean dato) + void close()

- El **constructor** crea un objeto PrintWriter para escribir texto en el archivo cuya ruta se pasa como parámetro. El archivo se sobreescribe si no existe
- El método **println** escribe el texto indicado en el archivo
- Las diferentes sobrecargas[2] del método **print** sirven para escribir variables de tipo básico en el archivo.
- El método **close** cierra el archivo y libera los recursos asociados. Debe usarse siempre que el PrintWriter deja de utilizarse.

[2] Una sobrecarga de un método es una versión diferente del mismo método que recibe parámetros distintos.

7) BufferedReader

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase BufferedReader sirve para leer String desde una fuente de caracteres (Reader). Normalmente la fuente de caracteres es un archivo de texto (extensión .txt).

BufferedReader extends Reader
+ BufferedReader(Reader origen) + String readLine() throws IOException + void close()

- El **constructor** crea un objeto BufferedReader que puede leer String desde la fuente de caracteres que se le suministra como parámetro Reader.
- El método **readLine** devuelve un String procedente de la fuente de caracteres. Este String es una línea de texto que abarca hasta el siguiente “salto de línea” que haya en la fuente de caracteres. Si se acaba la fuente de caracteres, el método devuelve **null**
- El método **close** libera los recursos que el sistema operativo ha asignado al objeto para leer datos. Es necesario llamarlo siempre que se termine de usar este objeto, para no consumir recursos innecesariamente.

8) Writer

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase Writer representa un medio en el que es posible escribir caracteres. Por ejemplo, un archivo, la pantalla, etc. Esta clase no puede ser instanciada directamente, por lo que es necesario acudir a sus clases hijas.

Writer
+ write(String s) + write(char c) + close() throws IOException

- El primer método **write** sirve para escribir un String en el medio.
- El segundo método **write** sirve para escribir un char en el medio.
- El método **close** libera los recursos que el sistema operativo ha asignado al objeto para escribir datos. Es necesario llamarlo siempre que se termine de usar este objeto, para no consumir recursos innecesariamente.

9) FileWriter

- **Librería:***Librería de entrada y salida de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.io*

La clase FileWriter es un Writer que permite la escritura de texto en un archivo.

FileWriter extends Writer
+ FileWriter(String archivo) throws IOException + FileWriter(String archivo, boolean añadir) throws IOException

- El primer **constructor** crea un objeto para escribir texto en el archivo que se pasa como parámetro. Este constructor siempre sobreescribe el archivo, borrando todo su contenido previo.
- El segundo método **constructor** crea un objeto para escribir texto en el archivo que se pasa como parámetro. Se comporta así:
    - Si el archivo no existe, lo crea.
    - Si el archivo existe y el boolean "añadir" vale true → Añade el nuevo texto al final del archivo, sin borrar lo que ya hay
    - Si el archivo existe y el boolean "añadir" vale false → Sobreescribe el archivo