Resumen Java The Complete Reference

Entre las páginas 556 y 566 encontramos información de suma importancia para la manipulación de archivos.

Se comienza por mostrar la lista con todas las clases que se encuentran en la paquetería java.io, así como las interfaces.

Una vez mostrada esta lista con todas estas clases e interfaces que nos permiten la manipulación de flujos de entrada y salida, se comienza por hablar de la clase File, la cual no opera con flujos, si no que, trabaja directamente con archivos y archivos en el sistema. A través de esta clase, nosotros podemos obtener la información necesaria para manipular un archivo en el sistema o para crear el nuestro. Para crear un archivo de la clase File, se encuentran diversos constructores, a los cuales se les puede pasar la dirección de donde se encuentra un archivo o donde queremos crear el archivo, la dirección y el nombre, entre otros.

Dentro de esta clase File, encontramos varios métodos importantes para obtener las propiedades de un archivo, como:

- *getName():* Obtiene el nombre del archivo.
- *getPath():* Obtiene la ruta del archivo.
- *getAbosultePath():* Obtiene la ruta absoluta de un archivo.
- *getParent():* Obtiene el directorio donde se encuentra el archivo.
- *exists():* Comprueba si el archivo actual existe.
- *canWrite():* Si el archivo se ha abierto en modo de escritura.
- *canRead():* Si el archivo se ha a abierto en modo de lecutra.
- *isDirectory():* Si el archivo en realidad es un direcotorio.
- *isFIle():* Si es un archivo de texto plano común y no un directorio.
- *isAbsolute():* Si una ruta especificada es abosluta o no.
- *lastModified():* La última vez que se modificó el archivo.
- *length():* La longitud en bytes del archivo.
- *renameTo():* Renombra el nombre de un archivo.
- *delete():* Borra un archivo o directorio. Solo borra el directorio si está vacío. De este método, encontramos otro bastante parecido, el cual es *deleteOnExit(),* el cual borrará un archivo en cuanto el programa deje de ejecutarse.
- *list():* Utilizada en directorios. Este método nos muestra todos los archivos o subdirectorios en un directorio como un arreglo de tipo String, con los nombres de los archivos . Este método tiene dos sobrecargas, en la cual, nosotros podemos pasarle un FilenameFilter o FileFilter, el cual nos permite filtrar aquellos archivos que contengan cierta cadena en el nombre del archivo, por lo que, solamente mostrará aquellos archivos que cumplan dicho nombre.
- *listFiles():* Funciona prácticamente igual que *list(),* y también cuenta con dos sobrecargas, las cuales consisten en filtrar archivos. La diferencia con este método y *list(),* es que este método retorna un arreglo de tipo File.
- *mkDir()* y *mkDirs():* Crean un directorio en una ruta especificada. *mkDirs()* creará el directorio aún si la ruta especificada no existe, por lo que, creará todas las carpetas necesarias hasta crear toda la ruta especificada, mientras que, *mkDir()* solo creará el directorio siempre y cuando exista la ruta especificada.

Junto con las clases previamente definidas, también se definen las clases Closeable y Flushable. La clase Closeable sirve para cerrar flujos mediante el método *close().* La clase Flushable permite escribir toda aquella información almacenada en algún tipo de Buffer a través del método *flush().*

Ya que se definió esto, se procede a hablar de los dos tipos de flujos disponibles, los cuales son los flujos de bytes y de caracteres, los cuales se utilizan dependiendo del tipo de información con la que se requiera tratar. En este caso, se hablan de las clases que nos permiten la manipulación de flujos de bytes, las cuales son InputStream y OutputStream.

Dichas clases son abstractas. InputStream se refiere a la entrada de bytes y OutputStream a la salida de bytes.

En InputStream, los métodos a resaltar son *available()* y *read().* Con el primer método obtenemos la cantidad de bytes disponibles para leer y con el segundo método se leen los bytes, y regresa la cantidad de bytes que se pudieron leer con éxito, regresando -1 cuando se llega al final del archivo.

En OutputStream, el método a resaltar es *write(),* con el cual podemos escribir bytes.

De la clase InputStram hereda la clase FileInputStream. Esta clase nos permite recibir un flujo de bytes de un archivo, es decir, nos permite la lectura de la información de un archivo, manipulándolos como bytes.

De la clase OutputStream hereda la clase FileOutputStream. Esta clase nos permite escribir un flujo de bytes en un archivo.

Los ejemplos propuestos en estas páginas se encuentran explicados, además de que se proporcionan las capturas de ejecuciones, por lo que, con dichos ejemplos podemos entender mejor la idea de estas clases y como se emplean sus métodos.