A. INVESTIGACIÓN SOBRE CLASES DEL PAQUETE JAVA.IO

¿Para qué se utilizan las siguientes clases?

- *File*

  Representa la ruta de un fichero o directorio. No lee ni escribe contenido. Sirve para trabajar con el sistema de ficheros: comprobar si existe (exist()), crear (createNewFile(), mkdir()), borrar (delete()), renombar, listar un directorio (list(), listFiles()) o consultar su tamaño y permisos.

- *FileReader*

  Flujo de entrada de caracteres que lee texto de un fichero. Lee carácter a carácter (read() devuelve un int, y -1 al llegar al final).

- *BufferedReader*

  Envuelve otro reader, y le añade un buffer en memoria. Su gran ventaja es readLine(), que lee el fichero línea a línea.

- *FileWriter*

  Flujo de salida de caracteres que escribe texto en un fichero. Puede sobreescribir el fichero o añadir al final con new FileWriter("f.txt", true).

- *BufferedWriter*

  Envuelve otro writer y acumula lo que escribes en un buffer antes de mandarlo al disco, incluye newLine(), que escribe el salto de línea propio del sistema operativo.

- *PrintWriter*

  Writer de "alto nivel " para escribir texto con formato de forma cómoda: print(), println(), printf() y format(), igual que con system.out.

¿Qué diferencia hay entre FileReader y BufferedReader?

- FileReader va directamente al fichero. Cada read() suele traducirse en una petición al flujo que hay por debajo (el disco), y solo sabe leer caracteres sueltos.
- BufferedReader lee un bloque grande de caracteres de una sola vez y lo guarda en memoria. Las siguientes lecturas salen de ese buffer, sin volver al disco. Además, ofrece readLine().

¿Cuál es más eficiente en el tratamiento de información? ¿Por qué?

El motivo es que acceder al disco es muy lento comparado con acceder a la RAM. La documentación oficial de BufferedReader lo explica así: sin buffer, cada llamada a read() o readLine() puede obligar a leer bytes del fichero, convertirlos a caracteres y devolverlos, y repetir eso en cada llamada "puede ser muy ineficiente". Por eso Oracle recomienda envolver con un BufferedReader cualquier FileReader.

¿Qué diferencia hay entre FileWriter, BufferedWriter y

PrintWriter?

	FileWriter	BufferedWriter	PrintWriter
Nivel	Bajo: conecta directamente con el fichero	Intermedio: envuelve otro Writer	Alto: envuelve un Writer, un OutputStream, un File o una ruta
Buffer	No (cada escritura va más directa al sistema)	Sí, más eficiente	Sí, cuando se crea a partir de un fichero
Métodos	write(), append()	write(), newLine()	print(), println(), printf(), format()
Excepciones	Lanza IOException	Lanza IOException	Sus métodos no lanzan IOException

Entre FileWriter y PrintWriter ¿Cuál de las dos clases lanza excepciones de tipo IOException?

FileWriter. Su constructor y sus métodos write(), flush() y close() declaran IOException, así que tienes que capturarla con try-catch o declararla con throws.

¿Cómo gestionará los errores la otra clase?

Según la documentación de PrintWriter, sus métodos nunca lanzan excepciones de E/S. Lo que hace es guardar el error en un indicador interno, y tú lo consultas llamando a checkError(), que devuelve true si ha habido algún fallo. Ojo: algunos constructores sí pueden lanzar excepciones. Por ejemplo, PrintWriter(String fileName) puede lanzar FileNotFoundException, que es una subclase de IOException.

¿Cuál de las dos clases usarías para añadir/escribir texto al final de un fichero?

FileWriter, con el constructor new FileWriter(fichero, true). El segundo parámetro, true, activa el modo append. PrintWriter no tiene un constructor con opción de añadir: si lo creas directamente con un nombre de fichero, borra el contenido existente o crea un fichero nuevo. Para usar la comodidad de PrintWriter y a la vez añadir al final, se envuelve un FileWriter en modo append: