A. Investigación sobre clases del paquete java.io

¿Para qué se utilizan las siguientes clases?

- **File**: representa una ruta de acceso a un archivo o directorio en el sistema de ficheros. No lee ni escribe contenido, sino que gestiona metadatos: comprobar si existe, crear directorios, eliminar archivos, obtener el tamaño, etc.
- **FileReader**: clase orientada a caracteres para leer el contenido de un archivo de texto. Lee los datos carácter por carácter directamente desde el disco.
- **BufferedReader**: lee texto de una corriente de entrada de caracteres almacenando previamente los datos en un búfer en memoria. Permite una lectura eficiente y aporta el método `readLine()` para procesar el texto línea a línea.
- **FileWriter**: clase orientada a caracteres para escribir datos en un archivo de texto. Escribe los caracteres directamente en el archivo en disco.
- **BufferedWriter**: escribe texto en una corriente de salida guardando los datos en un búfer en memoria antes de volcarlos al disco, optimizando las operaciones de escritura.
- **PrintWriter**: clase avanzada de escritura de texto que aporta métodos de formato como `println()`, `print()` y `printf()`. Captura internamente las excepciones de E/S para facilitar su uso.

¿Qué diferencia hay entre FileReader y BufferedReader? ¿Cuál es más eficiente? ¿Por qué?

- Diferencia principal: FileReader accede directamente al disco para leer carácter por carácter, mientras que BufferedReader envuelve a otro lector y almacena un bloque de caracteres en un búfer de memoria RAM.
- Más eficiente: BufferedReader.
- ¿Por qué? Las operaciones de acceso al disco físico (E/S) son costosas en tiempo de ejecución. FileReader realiza un acceso al disco por cada carácter leído. BufferedReader realiza una sola lectura en disco para llenar su búfer interno (por defecto 8 KB) y entrega los datos desde la memoria RAM, reduciendo drásticamente las llamadas al sistema operativo.

¿Qué diferencia hay entre FileWriter, BufferedWriter y PrintWriter?

- **FileWriter**: clase básica de bajo nivel para escribir caracteres directamente en un fichero de texto en disco. No usa almacenamiento intermedio.
- **BufferedWriter**: añade una capa de búfer sobre un Writer (como FileWriter). Acumula los caracteres en memoria y los vuelca al disco en bloque, mejorando el rendimiento. Incluye el método `newLine()` para saltos de línea adaptados al sistema operativo.
- **PrintWriter**: añade capacidades de formato de alto nivel (`print`, `println`, `printf`). Puede envolver tanto a FileWriter como a BufferedWriter. No obliga a capturar `IOException` en cada llamada.

Entre FileWriter y PrintWriter:

- **¿Cuál lanza excepciones IOException?** FileWriter lanza obligatoriamente excepciones comprobadas de tipo `IOException` en casi todos sus métodos (`write()`, `close()`, `flush()`).
- **¿Cómo gestiona los errores la otra clase?** PrintWriter no lanza `IOException` en sus métodos de escritura. En su lugar, captura internamente la excepción y establece una bandera de estado de error. Para comprobar si ha ocurrido algún fallo, hay que invocar explícitamente el método `checkError()`, que devuelve `true` si se produjo un error.
- **¿Cuál usarías para añadir texto al final de un fichero?** Se usa FileWriter (o PrintWriter envolviendo a un FileWriter), indicando el parámetro booleano `append` como `true` en el constructor:

```
FileWriter fw = new FileWriter("fichero.txt", true);
```

Si además quieres formatear o escribir línea a línea con comodidad, la combinación idónea es:

```
PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter("fichero.txt", true)));
```