**BufferedReader:**

Es utilizado para leer datos de entrada de manera eficiente, especialmente desde archivos de texto.

Su método read() lee un solo carácter a la vez, mientras que readLine() lee una línea completa de texto.

close() se utiliza para cerrar el flujo de entrada una vez que se ha terminado de leer.

open() no es un método de BufferedReader. Se utiliza el constructor de BufferedReader para abrir un flujo de entrada a partir de un objeto File, un objeto InputStreamReader o un objeto InputStream.

No proporciona métodos para escribir datos en un archivo.

**BufferedWriter**:

Es utilizado para escribir datos en un archivo de texto de manera eficiente.

Su método write() se utiliza para escribir una cadena de caracteres o una porción de un arreglo de caracteres en el archivo.

close() se utiliza para cerrar el flujo de salida una vez que se ha terminado de escribir.

open() no es un método de BufferedWriter. Se utiliza el constructor de BufferedWriter para abrir un flujo de salida a partir de un objeto File, un objeto OutputStreamWriter o un objeto OutputStream.

No proporciona métodos para leer datos desde un archivo.

***true***

El parámetro true pasado al constructor de FileWriter indica que se desea abrir el archivo en modo de "append" (añadir). Esto significa que si el archivo ya existe, los nuevos datos que se escriban se agregarán al final del archivo, en lugar de sobrescribir su contenido existente.

En el contexto de tu código, cuando abres el archivo firmas.txt para escribir en él, pasas true como segundo argumento al constructor de FileWriter:

*BufferedWriter out = new BufferedWriter(new FileWriter("src/app16/firmas.txt", true));*

Esto asegura que cuando se escribe un nuevo nombre en el archivo, este se agregue al final del archivo en lugar de sobrescribir su contenido existente. Esto es útil para mantener todas las firmas almacenadas en el archivo sin perder ninguna información previa.

**Scanner:**

Es utilizado para analizar tipos de datos primitivos y cadenas desde una fuente de entrada, como el teclado o un archivo de texto.

Proporciona métodos como nextInt(), nextDouble(), next(), y nextLine() para leer diferentes tipos de datos.

close() se utiliza para cerrar el Scanner una vez que se ha terminado de leer.

open() no es un método de Scanner. Se utiliza el constructor de Scanner para crear un nuevo objeto Scanner que se asocie con la fuente de entrada deseada.

No proporciona métodos para escribir datos en un archivo.

En resumen, BufferedReader y BufferedWriter son más adecuados para trabajar con archivos de texto, mientras que Scanner es más útil para leer datos de entrada del usuario desde el teclado. Cada clase tiene sus propios métodos específicos para leer y escribir datos según el contexto de uso.

***next y nextLine***

En la clase Scanner de Java, los métodos next() y nextLine() se utilizan para leer datos de entrada del usuario. Ambos métodos son útiles para leer cadenas de texto, pero tienen diferencias importantes en cuanto a cómo manejan el formato de entrada y los caracteres especiales.

next():

next() lee la entrada del usuario hasta el próximo espacio en blanco (espacio, tabulación, salto de línea, etc.).

Devuelve la siguiente cadena de caracteres que encuentra.

No incluye el carácter de nueva línea ('\n') en la cadena devuelta.

Por ejemplo:

*Scanner scanner = new Scanner(System.in);*

*System.out.println("Ingrese su nombre y apellido: ");*

*String nombre = scanner.next();*

*System.out.println("Nombre ingresado: " + nombre);*

*Si el usuario ingresa "John Smith", el valor de nombre sería "John".*

nextLine():

nextLine() lee toda la línea de entrada del usuario hasta encontrar un salto de línea ('\n').

Devuelve la cadena de caracteres completa, incluyendo los espacios en blanco y el carácter de nueva línea ('\n') al final.

Por ejemplo:

*Scanner scanner = new Scanner(System.in);*

*System.out.println("Ingrese su nombre completo: ");*

*String nombreCompleto = scanner.nextLine();*

*System.out.println("Nombre completo ingresado: " + nombreCompleto);*

*Si el usuario ingresa "John Smith", el valor de nombreCompleto sería "John Smith".*

En resumen, next() se utiliza para leer una sola palabra o un token hasta el siguiente espacio en blanco, mientras que nextLine() se utiliza para leer toda una línea de entrada del usuario, incluyendo espacios en blanco y caracteres de nueva línea. La elección entre next() y nextLine() depende del formato de entrada esperado y de cómo se desea manejar la entrada del usuario.

***hasNext():***

Este método devuelve true si hay otro token en la entrada del scanner, de lo contrario devuelve false.

Se puede utilizar para verificar si hay más datos disponibles antes de intentar leer el próximo token con next().

*Scanner scanner = new Scanner(System.in);*

*if (scanner.hasNext()) {*

*String token = scanner.next();*

*System.out.println("Token: " + token);*

*} else {*

*System.out.println("No hay más tokens disponibles.");*

*}*

***hasNextLine():***

Similar a hasNext(), pero específicamente verifica si hay otra línea de entrada en el scanner.

Se puede utilizar para verificar si hay más líneas disponibles antes de intentar leer la próxima línea con nextLine().

*Scanner scanner = new Scanner(System.in);*

*if (scanner.hasNextLine()) {*

*String linea = scanner.nextLine();*

*System.out.println("Línea: " + linea);*

*} else {*

*System.out.println("No hay más líneas disponibles.");*

*}*

**DIFERENCIA ENTRE SCANNER Y BUFFEREDREADER**

*//Usando BufferedReader:*

*public class BufferedReaderExample {*

*public static void main(String[] args) {*

*try (BufferedReader br = new BufferedReader(new FileReader("archivo.txt"))) {*

*String linea;*

*while ((linea = br.readLine()) != null) {*

*System.out.println(linea);*

*}*

*} catch (IOException e) {*

*System.err.println("Error al leer el archivo: " + e.getMessage());*

*}*

*}*

*}*

*//Usando Scanner:*

*public class ScannerExample {*

*public static void main(String[] args) {*

*try {*

*File archivo = new File("archivo.txt");*

*Scanner scanner = new Scanner(archivo);*

*while (scanner.hasNextLine()) {*

*String linea = scanner.nextLine();*

*System.out.println(linea);*

*}*

*scanner.close();*

*} catch (FileNotFoundException e) {*

*System.err.println("Archivo no encontrado: " + e.getMessage());*

*}*

*}*

*}*