	<p>Carátula para entrega de prácticas</p>
<p>Facultad de Ingeniería</p>	<p>Laboratorio de Docencia</p>

Laboratorio de Computación Salas A y B

Profesor: René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 1

Número de Práctica: 11

Integrante(s): 321573670

No. de Lista: N/A

Semestre: 2025-1

Fecha de Entrega: 4 / noviembre / 2024

Observaciones: _____

Calificación: _____

Índice

1. Introducción	2
2. Marco Teórico	2
3. Desarrollo	4
4. Resultados	8
5. Conclusiones	15
Referencias	16

1. Introducción

■ Planteamiento del problema:

El manejo de archivos es una habilidad esencial al momento de trabajar con una infinidad de tipo de datos, manejar memoria y realizar implementaciones de programas complejos, es por esto que es esencial que el alumnado tenga el conocimiento suficiente y adecuado para implementar soluciones con estos conocimientos.

Se busca que los programas descritos a continuación sirvan de apoyo en el proceso de adquisición de dichos conocimientos:

- Un programa para leer los contenidos de un archivo línea por línea.
- Un programa para el cual se lean los contenidos de un archivo línea por línea y sean almacenados en una variable.
- Un programa para almacenar los contenidos de un archivo, línea por línea, en un arreglo.
- Un programa para escribir y leer un archivo en texto plano.
- Un programa en el que se lean los contenidos de tres archivos diferentes y se escriban en uno nuevo.

■ Motivación:

Se espera que la correcta implementación de soluciones a los problemas propuestos permita al alumnado desarrollar una comprensión mucho más compleja de la funcionalidad del manejo de archivos en Java y sea capaz de implementar todos los conceptos teóricos en una gran variedad de aplicaciones.

■ Objetivos:

Se espera que el alumnado sea capaz de dar solución a los problemas propuestos, consolidando su conocimiento sobre la implementación del manejo de archivos en Java y así mejorando sus habilidades al respecto.

2. Marco Teórico

El manejo de archivos en Java es fundamental para la persistencia y manipulación de datos en aplicaciones. Permite que los programas interactúen

con el sistema de archivos, leyendo y escribiendo datos que pueden ser utilizados posteriormente. Las clases y métodos para realizar estas operaciones se encuentran principalmente en el paquete `java.io`, el cual proporciona una amplia gama de herramientas para trabajar con archivos de manera eficiente y segura [1].

La lectura de archivos se realiza comúnmente utilizando flujos de entrada, conocidos como “Input Streams”. Estos permiten que los datos sean leídos secuencialmente desde una fuente, como un archivo en disco. Clases como `FileReader` y `BufferedReader` son esenciales en este proceso. `FileReader` se utiliza para leer flujos de caracteres desde archivos, mientras que `BufferedReader` optimiza la lectura al permitir el almacenamiento en búfer de los datos, reduciendo así el número de operaciones de lectura necesarias [2].

Por otro lado, la escritura en archivos se lleva a cabo mediante flujos de salida, o Output Streams. Clases como `FileWriter` y `BufferedWriter` facilitan la escritura de datos en archivos. `FileWriter` escribe flujos de caracteres en archivos, y `BufferedWriter` mejora la eficiencia al proporcionar un búfer de escritura. Esto es especialmente útil cuando se escriben grandes cantidades de datos, ya que reduce las operaciones de escritura en el disco [3].

El manejo adecuado de excepciones es crucial al trabajar con archivos. Operaciones de entrada/salida son susceptibles a errores, como la ausencia del archivo o problemas de permisos. Excepciones como `FileNotFoundException` y `IOException` deben ser capturadas y manejadas correctamente para evitar que el programa falle inesperadamente y para proporcionar información útil al usuario sobre el error ocurrido [4].

El uso de `StringBuilder` es recomendado cuando se trabaja con cadenas de texto que requieren múltiples modificaciones, como la concatenación de líneas leídas desde un archivo. `StringBuilder` es más eficiente que la concatenación de cadenas utilizando el operador `+`, ya que modifica la cadena existente en lugar de crear una nueva en cada operación [1].

Finalmente, seguir buenas prácticas de programación es esencial al manipular archivos. Esto incluye verificar la existencia del archivo antes de leerlo o escribir en él, manejar adecuadamente las excepciones, y asegurar que los recursos se cierren correctamente. Estas prácticas no solo mejoran la robustez del programa sino que también facilitan su mantenimiento y legibilidad [3].

3. Desarrollo

- Ejercicio 0:

Toda la funcionalidad del programa para este ejercicio se definió dentro del método principal, primero se instanció un objeto **StringBuilder** denominado “sb” y a su vez un objeto **String** llamado “strLine”.

Posteriormente se declaró un bloque **try-catch** capaz de manejar dos excepciones, la excepción en el que no se encuentre el archivo a leer y la excepción en la que ocurre algún error al leer un archivo, cada una con su respectivo mensaje informado del error.

Dentro de la sección **try** se define un objeto **BufferedReader** “br” que sirve como “wrapper” de un objeto **FileReader** con la dirección al archivo dentro de su constructor. Una vez definido dicho objeto se declara un ciclo **while** que se ejecuta mientras que “strLine” no se encuentre **null**. Dentro del ciclo se comienza por imprimir el contenido de “strLine”, posteriormente se hace uso de “br” para leer una línea del archivo y asignarla a “strLine”, posteriormente se hace uso del método **append()** para concatenar el contenido de “strLine” al objeto **StringBuilder** y de la misma manera se le concatena un separador de línea.

Una vez que termina el ciclo y se imprimió a pantalla el contenido entero del archivo se cierra el objeto “br”.

- Ejercicio 1:

De manera similar al ejercicio anterior, toda la funcionalidad de este programa se encuentra en su método **main()**. En este caso se comienza por crear dos objetos tipo **String** denominados “strLine” y “str_data”.

Una vez hecho esto se declara un bloque **try-catch** para manejar las mismas dos excepciones del programa anterior. Dentro de la sección **try** se define nuevamente un objeto **BufferedReader** con un **FileReader** en su constructor con la dirección al archivo. Después se define el mismo ciclo **while** pero dentro de este se empieza por definir un condicional para el caso en el que “strLine” sea **null** en cuyo caso detiene la ejecución del ciclo. Pasada dicha condicional se le concatena a “str_data” el contenido de “strLine” y se le asigna a “strLine” la línea leída por el objeto **BufferedReader**.

Finalmente, fuera del ciclo, se imprime el contenido de “str_data” y se libera la memoria para el **BufferedReader**.

■ Ejercicio 2:

Como ya vimos que ocurre en estos ejercicios, toda la funcionalidad se encuentra declarada dentro del método principal. Primero se instancia un **StringBuilder** “sb”, un **String** “strLine” y finalmente una lista de cadenas denominada “list” con el constructor **ArrayList<>()**.

Se vuelve a implementar el mismo bloque **try-catch** para las mismas excepciones. En este caso dentro de la sección **try** se instancia un objeto **BufferedReader** y se inicializa el mismo ciclo **while**. Dentro de dicho ciclo se comienza por asignar a “strLine” lo leído por “br”, después al objeto “sb” se le concatena el contenido de “strLine” y un separador de línea, posteriormente se verifica si “strLine” es **null** en cuyo caso se rompe el ciclo, si se logra pasar dicha condicional se le añade “strLine” al objeto “list” haciendo uso del método **add()**.

Una vez terminado el ciclo se manda a imprimir los contenidos de “list” con una secuencia de transformaciones a arreglo y a cadena y finalmente se libera la memoria de “br”.

■ Ejercicio 3:

Nuevamente toda la funcionalidad del programa se definió en la clase principal. En este caso solamente se empieza por instancias un **StringBuilder** nuevamente denominado “sb” y una cadena “strLine”. De la misma manera se define el bloque necesario para manejar excepciones pero esta vez solamente maneja la **IOException**, ya no maneja el caso en el que no encuentra el archivo pues no es necesario.

Dentro del **try** se define una cadena “filename” con la dirección al archivo al cual se le escribirá, esta misma cadena es usada como parte del constructor para el objeto “fw” de tipo **FileWriter**, inmediatamente después de hace uso del método **write()** para “fw” con el mensaje desado y se libera el mismo.

Una vez escrito lo deseado al archivo, se define un objeto **BufferedReader** llamado “br” de la misma manera que en ejercicios anteriores. Después se vuelve a definir el mismo ciclo con la cadena “strLine”. Dentro del ciclo primero se le concatena a “sb” los contenidos de “strLine” y un

separados de línea, posteriormente se le asigna a “strLine” lo leído por br y finalmente se imprime “strLine”.

Para concluir se libera la memoria del objeto **BufferedReader**

■ Practica 11:

Para la realización del programa solicitado para este último punto se declararon tres clases dentro de un paquete, estas fueron **Lectura**, **Escritura** y **Practica11**, cada una con su respectiva funcionalidad análoga a su nombre.

Comenzando por la clase **Lectura**, para esta se definió un método estático **realizarLectura()** con un parámetro tipo cadena y definiendo que retornará un objeto **StringBuilder**.

Se comenzó por instanciar un objeto **StringBuilder** denotado “sb” para posteriormente declarar el bloque **try-catch** con el cual se manejan las excepciones de los primeros ejercicios. Dentro del **try** primero se instancia un **BufferedReader** “br” de la misma manera que en los ejercicios previos de modo que funciona como “wrapper” para un **FileReader** con el parámetro para la dirección en su constructor.

Posteriormente se define una cadena “strLine” misma que es usada dentro del ciclo definido posteriormente para la condición de continuación, pero esta vez la condición radica en saber si “strLine” menos lo que lee el objeto “br” es nulo o no. Dentro del ciclo simplemente se le concatena “strLine” a “sb” junto al separador de línea y se imprime “strLine”. Finalmente se libera la memoria del **BufferedReader**. Al finalizar el método se retorna el objeto “sb”.

Ahora respecto a la clase **Escritura**, se define un método similar al de la anterior clase con el nombre de **realizarEscritura()** solo que esta vez no retorna nada y tiene como parámetro un objeto **StringBuilder**.

Dentro del método primero se define una cadena con el nombre del archivo en el que se realizará la escritura, posteriormente dentro de un bloque **try-catch** con la capacidad de manejar la excepción en la que no se puede realizar la escritura, simplemente se crea un objeto **FileWriter** “fw” haciendo uso de la dirección como parámetro y posteriormente se manda a llamar el método **write()** de dicho objeto con “sb” como parámetro.

Finalmente en la clase `Practica11` se define el método `main()`, dentro de dicho método se instancia un `StringBuilder` y un arreglo de cadenas con las direcciones de los tres archivos que serán leídos. Después se usa un ciclo `for-each` con el cual se itera sobre el arreglo y para cada una de las cadenas se le concatena al `StringBuilder` el resultado del método estático “`realizarLectura()`” con cada archivo como parámetro. Por último se llama al método `realizarEscritura()` y se le pasa como parámetro el `StringBuilder` ya con los contenidos de todos los archivos.

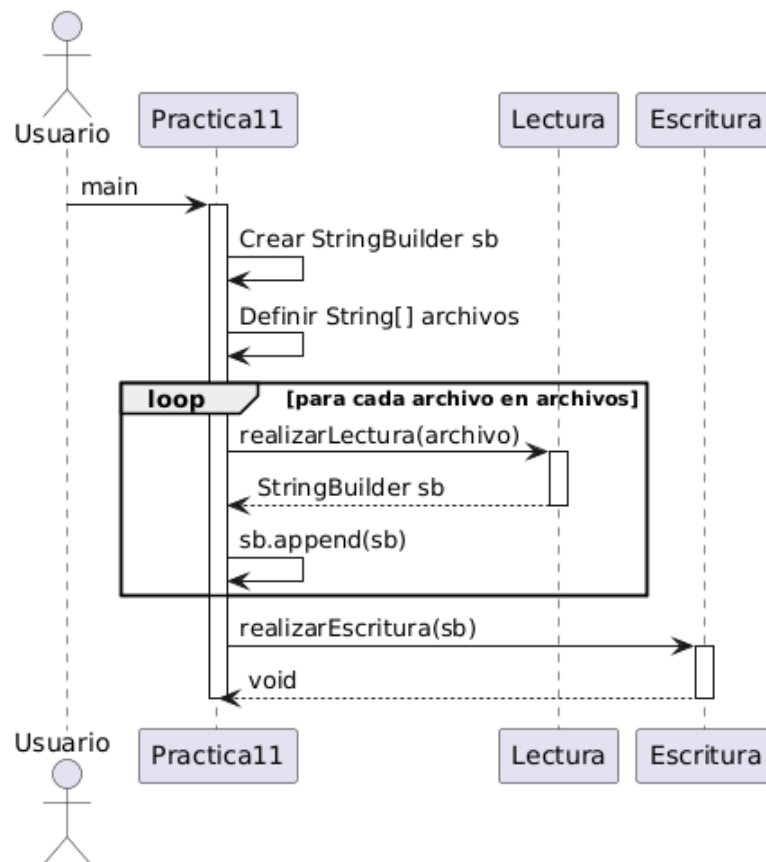


Figura 1: Diagrama UML de secuencia

4. Resultados

- Ejercicio 0:

Código:

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Ejercicio0{
    Run | Debug | Run main | Debug main
    public static void main(String [] args){
        StringBuilder sb = new StringBuilder();
        String strline = "";
        try {
            BufferedReader br = new BufferedReader(new FileReader(fileName:"C:\\Users\\isaac\\OneDr
            while (strline != null){
                System.out.println(strline);
                strline = br.readLine();
                sb.append(strline);
                sb.append(System.lineSeparator());
            }
            br.close();
        } catch (FileNotFoundException e) {
            System.err.println(x:"Archivo no encontrado");
        } catch (IOException e){
            System.err.println(x:"Error al leer el archivo");
        }
    }
}
```

Figura 2: Clase y método principal para el ejercicio 0, se ejecuta la funcionalidad completa del programa y se incuye el manejo de dos excepciones.

Ejecución:

```
Buenas tardes.
Buenas noches.
Buenas mañanas.
Malas tardes.
Malas noches.
holaaaaaaaaaaaaaaaaaaaaa waaaaaaa. hola.
```

Figura 3: Ejecución en la que se imprime en terminal el contenido del archivo leído.

- Ejercicio 1:

Código:

```

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Ejercicio1{
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        String strline = "";
        String str_data = "";
        try {
            BufferedReader br = new BufferedReader(new FileReader(fileName:"C:\\Users\\isaac\\OneDr
            while (strline != null){
                if (strline == null)
                    break;
                str_data += strline;
                strline = br.readLine();
            }
            System.out.println(str_data);
            br.close();
        } catch (FileNotFoundException e) {
            System.err.println(x:"Archivo no encontrado...");
        } catch (IOException e){
            System.err.println(x:"No es posible leer el archivo...");
        }
    }
}

```

Figura 4: Método principal en el que se define la funcionalidad completa del programa para el ejercicio 1, se almacenan los datos leídos del archivo en una variable que después se imprime, se hace uso del manejo de excepciones.

Ejecución:

```

Buenas tardes.Buenas noches.Buenas mañanas.Malas tardes.Malas noches.holaaaaaaaaaaaaaaaaaa waaaaaaa. hola.

```

Figura 5: Ejecución del código, se nos presenta la impresión de todo lo leído del archivo.

■ Ejercicio 2:

Código:

```

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Ejercicio2{
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        String strLine = "";
        List<String> list = new ArrayList<String>();
        try {
            BufferedReader br = new BufferedReader(new FileReader(fileName:"C:\\Users\\isaac\\OneDr
            while(strLine!=null){
                strLine = br.readLine();
                sb.append(strLine);
                sb.append(System.lineSeparator());
                strLine = br.readLine();
                if(strLine == null){
                    break;
                }
                list.add(strLine);
            }
            System.out.println(Arrays.toString(list.toArray()));
            br.close();
        } catch (FileNotFoundException e) {
            System.err.println(x:"Archivo no encontrado...");
        } catch (IOException e) {
            System.err.println(x:"No se puede leer el archivo...");
        }
    }
}

```

Figura 6: Método principal para el ejercicio 2, se escribe toda la funcionalidad del programa dentro de un bloque `try-catch` para manejar excepciones. Para este programa se agrega la información leída en un `ArrayList` de cadenas.

Ejecución:

```
[Buenas noches., Malas tardes., holaaaaaaaaaaaaaaaaaaaaa waaaaaaa. hola.]
```

Figura 7: Ejecución en la que vemos la impresión de los contenidos del arreglo.

■ Ejercicio 3:

Código:

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Ejercicio3 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        String strLine = "";
        try {
            String filename = "C:\\Users\\isaac\\OneDrive\\Desktop\\esuelita\\3er semestre\\POO\\pro
            FileWriter fw = new FileWriter(filename, append: false);
            fw.write(str: "Me la estoy pasando bien raro...\n");
            fw.close();
            BufferedReader br = new BufferedReader(new FileReader(fileName: "C:\\Users\\isaac\\OneDr
            while (strLine != null) {
                sb.append(strLine);
                sb.append(System.lineSeparator());
                strLine = br.readLine();
                System.out.println(strLine);
            }
            br.close();
        } catch (IOException e) {
            System.err.println(x: "IOException...");
        }
    }
}

```

Figura 8: Clase principal que alberga al método principal, en la cual haciendo uso de un conjunto de objetos `FileWriter`, `BufferedReader` y `StringBuilder` se escribe una frase a un archivo, después se lee de ese archivo y se concatena a una cadena y se imprime.

Ejecución:

```

Me la estoy pasando bien raro...
null

```

Figura 9: Ejecución del programa, se observa la impresión de lo escrito, cabe notar la impresión de "null" debido a una lectura demás.

■ Practica 11:

Código:

```

package mx.unam.fi.poo.g1.p11;

/**
 * Clase principal
 * @author Campos Cortés Isaac Jareth
 * @version Noviembre 2024
 */
public class Practica11 {
    /**
     * Función principal en la que se ejecuta toda la funcionalidad del programa
     * @param args -> Arreglo por defecto de la función main
     */
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        String[] archivos = {
            "C:/Users/isaac/OneDrive/Desktop/esuelita/3er semestre/P00/practica11/ejercicio/doc1",
            "C:/Users/isaac/OneDrive/Desktop/esuelita/3er semestre/P00/practica11/ejercicio/doc2",
            "C:/Users/isaac/OneDrive/Desktop/esuelita/3er semestre/P00/practica11/ejercicio/doc3"
        };
        for (String archivo : archivos) {
            sb.append(Lectura.realizarLectura(archivo));
        }
        Escritura.realizarEscritura(sb);
    }
}

```

Figura 10: Clase en la que se define el método principal para el programa desarrollado, aquí se instancia un objeto **StringBuilder** y se crea un arreglo de cadenas en el que se contienen las direcciones de los archivos, posteriormente se le adjunta al **Stringbuilder** cada texto leído en cada archivo para finalmente ser escrito en uno nuevo.

```

public class Lectura {
    /**
     * Método para realizar la lectura del archivo
     * @param direccion -> Parámetro para acceder al archivo
     * @return sb -> Retorna un StringBuilder con el contenido del archivo
     */
    public static StringBuilder realizarLectura(String direccion){
        StringBuilder sb = new StringBuilder();

        try {
            BufferedReader br = new BufferedReader(new FileReader(direccion));
            String strLine;
            while ((strLine = br.readLine()) != null) {
                sb.append(strLine);
                sb.append(System.lineSeparator());
                System.out.println(strLine);
            }
            br.close();
        } catch (FileNotFoundException e) {
            System.err.println("Archivo no encontrado: " + direccion);
        } catch (IOException e) {
            System.err.println("No es posible leer el archivo: " + direccion);
        }
        return sb;
    }
}

```

Figura 11: Clase Lectura, en esta clase se define el método `realizarLectura()`, podemos ver que su funcionalidad se encuentra dentro de un bloque `try-catch` con el cual se manejan dos excepciones, este método se dedica en leer los contenidos de un archivo y concatenarlos a un `StringBuilder` el cual retorna al finalizar.

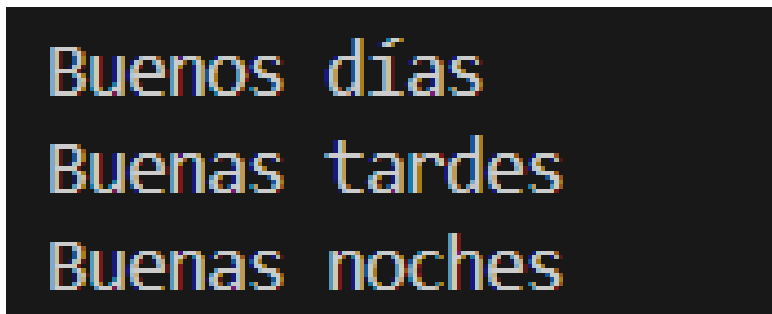
```

public class Escritura {
    /**
     * Método para realizar la escritura del archivo final
     * @param sb -> El StringBuilder final con el contenido de todos los archivos que se va a escri
     */
    public static void realizarEscritura(StringBuilder sb){
        String filename = "C:/Users/isaac/OneDrive/Desktop/esuelita/3er semestre/POO/practica11/eje
        try (FileWriter fw = new FileWriter(filename, append:false)) {
            fw.write(sb.toString());
        } catch (IOException e) {
            System.err.println(x:"No es posible escribir en el archivo de salida...");
        }
    }
}

```

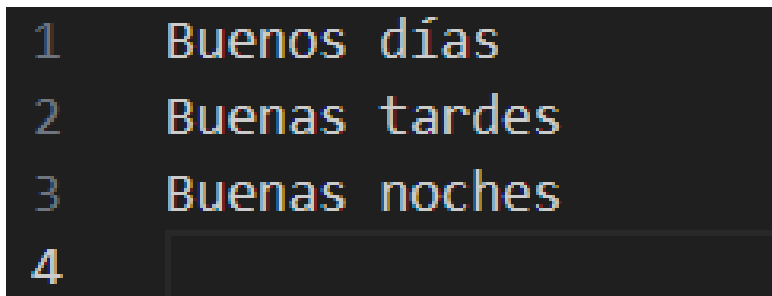
Figura 12: Clase Escritura, de manera similar a la clase anterior se declara un método `realizarEscritura()` cuya funcionalidad se encuentra dentro de un bloque `try-catch`, la funcionalidad de esta clase radica en la escritura de lo que contiene en objeto `StringBuilder` a un cierto archivo.

Ejecución:



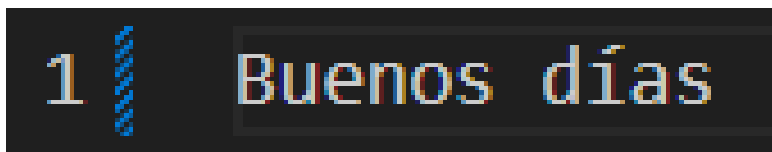
```
Buenos días
Buenas tardes
Buenas noches
```

Figura 13: Ejecución del código en el que podemos visualizar la impresión del contenido del archivo resultante.



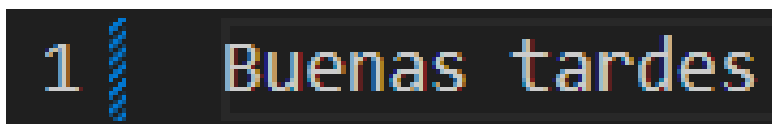
```
1 Buenos días
2 Buenas tardes
3 Buenas noches
4 
```

Figura 14: Contenidos del archivo resultante.



```
1 | Buenos días
```

Figura 15: Contenido del primer archivo.



```
1 | Buenas tardes
```

Figura 16: Contenido del segundo archivo-

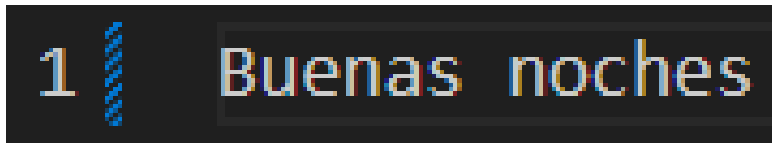


Figura 17: Contenido del tercer archivo.

5. Conclusiones

La práctica fue satisfactoriamente completada, esto indica que los conceptos presentados fueron exitosamente comprendidos por el alumnado pues se consiguió su correcta implementación y utilización para dar solución a los problemas conceptualizados inicialmente.

Personalmente la práctica me ayudó a comprender de manera mucho más tangible la manera de manejar archivos en Java y todo lo que esto implica, además me agrada que poco a poco vamos construyendo sobre el resto de conocimientos que hemos visto durante la duración del curso.

Referencias

- [1] C. S. Horstmann, *Core Java Volume I–Fundamentals*. Pearson Education, 2019.
- [2] B. Eckel, *Thinking in Java*. Prentice Hall Professional, 2006.
- [3] J. Bloch, *Effective Java*, 3rd. Addison-Wesley Professional, 2018.
- [4] Oracle. “Exceptions.” (2023), dirección: <https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>.