|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | ***Ing. Guadalupe Lizeth Parrales Romay*** |
| *Asignatura:* | ***Programación Orientada a Objetos*** |
| *Grupo:* | ***04*** |
| *No de Práctica(s):* | ***11*** |
| *Integrante(s):* | ***Dávila Ortega Jesús Eduardo - No. Cuenta: 317199860***  ***Díaz Hernández Marcos Bryan - No. Cuenta: 317027253***  ***Pareja Ávila Emiliano - No. Cuenta: 317081345***  ***Vázquez Zavala Oliver Alexis - No. Cuenta: 317202263*** |
| *No. de Equipo de cómputo empleado:* |  |
| *No. de Lista o Brigada:* | ***12,14, 30, 37*** |
| *Semestre:* | ***2021-1*** |
| *Fecha de entrega:* | ***12 de enero de 2021*** |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

***OBJETIVO.***

Implementar el intercambio de datos (lectura y escritura) entre fuentes externas (archivos

y/o entrada y salida estándar) y un programa (en un lenguaje orientado a objetos).

***INTRODUCCIÓN.***

* ***Flujo de Datos.***

Todas las aplicaciones requieren que se les proporcione información de entrada para realizar las operaciones que cada proceso resuelve, así mismo requiere mecanismos de salida para entregar los resultados. La forma de representar estas entradas y salidas son streams que son flujos de datos. Un stream se comporta como un medio de comunicación entre el programa y la fuente o destino de los datos. La información se transmite en serie a través de este medio.

* ***Clase File***

Permite manejar archivos o carpetas, es decir, crear y borrar tanto archivos como carpetas entre otras funciones. Cuando se crea una instancia de la clase file no se crea ningún archivo o carpeta solamente se crea una referencia hacia un objeto de este tipo.

* ***Clase FileInputStream:***

Permite la lectura de flujos de bytes desde un archivo de texto plano

* ***Clase FileOutputStream:***

Permite crear y escribir un flujo de bytes en un archivo de texto plano.

* ***Clase FileWriter:***

Permite escribir un flujo de caracteres en un archivo de texto plano.

* ***Clase BufferedWriter:***

Permite crear un buffer para realizar una escritura eficiente de caracteres desde la aplicación hacia el archivo destino.

* ***Clase PrintWriter:***

Permite escribir de forma sencilla en un archivo de texto plano. Posee los métodos print y println, idénticos a los de System.out y el método close(), el cual cierra el stream de datos.

* ***Clase FileReader:***

Permite leer flujos de caracteres de un archivo de texto plano.

* ***Clase InputStreamReader:***

Convierte los streams de bytes a streams de caracteres. Sysytem.in es el objeto de la clase InputStream el cual recibe datos desde la entrada estándar del teclado

* ***Clase BufferedReader:***

Crea un buffer para realizar una lectura eficiente de caracteres. Dispone del método readLine que permite leer una línea de texto y tiene como valor de retorno un string.

* ***Clase StringTokenizer:***

Permite separar una cadena de texto por palabras (espacios) o por algún otro carácter.

* ***Clase Scanner:***

Permite leer flujos de bytes desde la entrada estándar, pero también puede hacerlo desde otra fuente.

* ***Clase Console***:

Permite recibir flujos de datos desde la línea de comandos(entrada estándar).

* ***Clase System:***

En Java, la entrada desde el teclado y la salida a pantalla se utilizan a través de la clase System. Esta clase pertenece al paquete java.lang y posee atributos y métodos que se relacionan directamente con el sistema local.

* ***Atributo in:***

Objeto preparado para recibir datos desde la entrada estándar del sistema que generalmente es el teclado.

* ***Atributo out:***

Objeto utilizado para imprimir los datos en la salida estándar del sistema que generalmente es la pantalla.

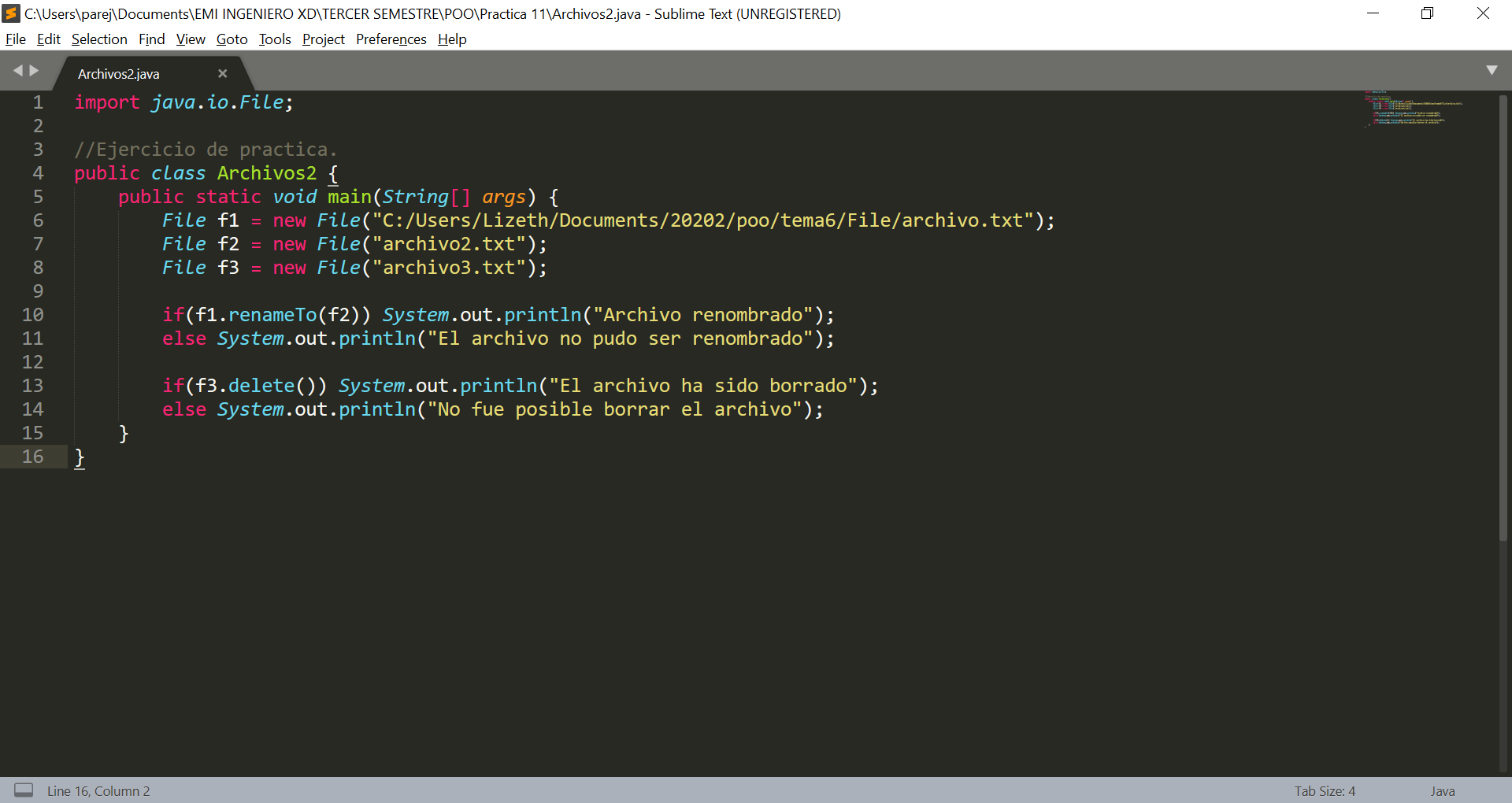
* ***Atributo err:***

Objeto utilizado para mensajes de error presentados por defecto en pantalla.

***Análisis de los ejercicios.***

***Ejercicio 1***

* **Análisis previo**



El análisis previo que hicimos sobre este programa es que primero se importa de la librería java.io la biblioteca File, que tiene la clase File que nos permitirá modelar archivos. Después se instancia la clase Archivos 2 de carácter público, y en el método main, se instancian 3 archivos que son objetos de la clase File, y de acuerdo con los diferentes métodos constructores de la clase File, se instancian estos archivos.

El primero llamado f1, su método constructor recibe como parámetro la ruta absoluta en donde se guardará el archivo, aunque en este caso no encontrará la ruta absoluta, puesto a que el archivo aún no se ha creado en memoria y además el programa se está ejecutando en una computadora diferente a la de la profesora que es en la que originalmente se ejecutó el programa. El segundo archivo f2, también es un objeto de la clase File y su método constructor recibe como parámetro el nombre del archivo que será archivo2.txt, y el archivo f3 al igual que los anteriores es un objeto de la clase File y su método constructor recibe como parámetro una cadena que será el nombre del archivo llamado archivo3.txt.

Cabe aclarar que estos archivos no existen en memoria aún, ya que nunca se crearon por medio del método de la clase File createNewFile(), por tanto estos archivos que se instanciaron son solo abstracciones de archivos.

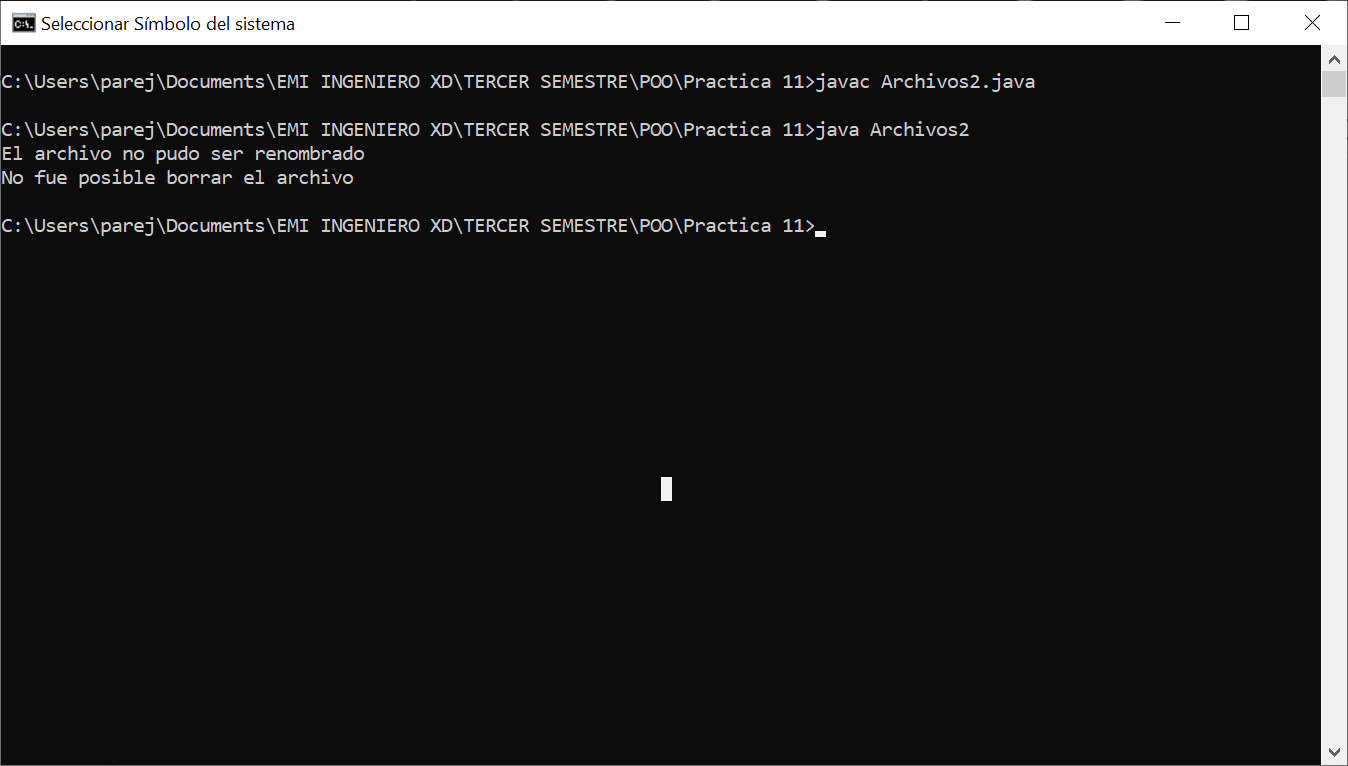
Luego, en una estructura de control de flujo if-else, se hace una validación, llamando al método renameTo() del archivo f1 y como parámetro se pasa al archivo f2, este método devuelve un valor booleano true en caso de que el archivo si se haya podido renombrar y un false si no se pudo renombrar. En caso de que el método devuelva un true, se imprimirá a pantalla que el archivo fue renombrado, y en caso de que regrese un false, se imprime a pantalla que el archivo no pudo ser renombrado.

Después en una estructura de control de flujo if-else se hace una llamada al método delete() del archivo f3, el cual se encarga de eliminar al archivo, dicho método no recibe parámetros y su valor de retorno es un booleano, este método devuelve un true si el archivo se eliminó correctamente y un false en caso de que no se haya eliminado correctamente, en caso de que el valor de retorno sea un true, se imprimirá a pantalla que el archivo fue borrado, en caso contrario se imprime a pantalla que el archivo no fue borrado.

* **Desarrollo**

Como ya mencionamos anteriormente estos archivos que se instanciaron, no existen en la memoria, por tanto el programa no los encontrará, entonces al hacer la llamada al método renameTo, el archivo f1 no se renombrará como el archivo f2, y es el mismo caso para el archivo f3, al hacer la llamada del método delete, el método buscará en los archivos de la carpeta donde está almacenado el código y no encontrará ningún archivo con el nombre archivo3.txt, puesto a que no existe en la memoria porque no se creó mediante el método createNewFile(), por tanto intuimos que la salidas del programa serán, que primero se imprimirá a pantalla que el archivo f1 no pudo ser renombrado y que el archivo 3 no pudo ser eliminado.

* **Ejecución del programa**



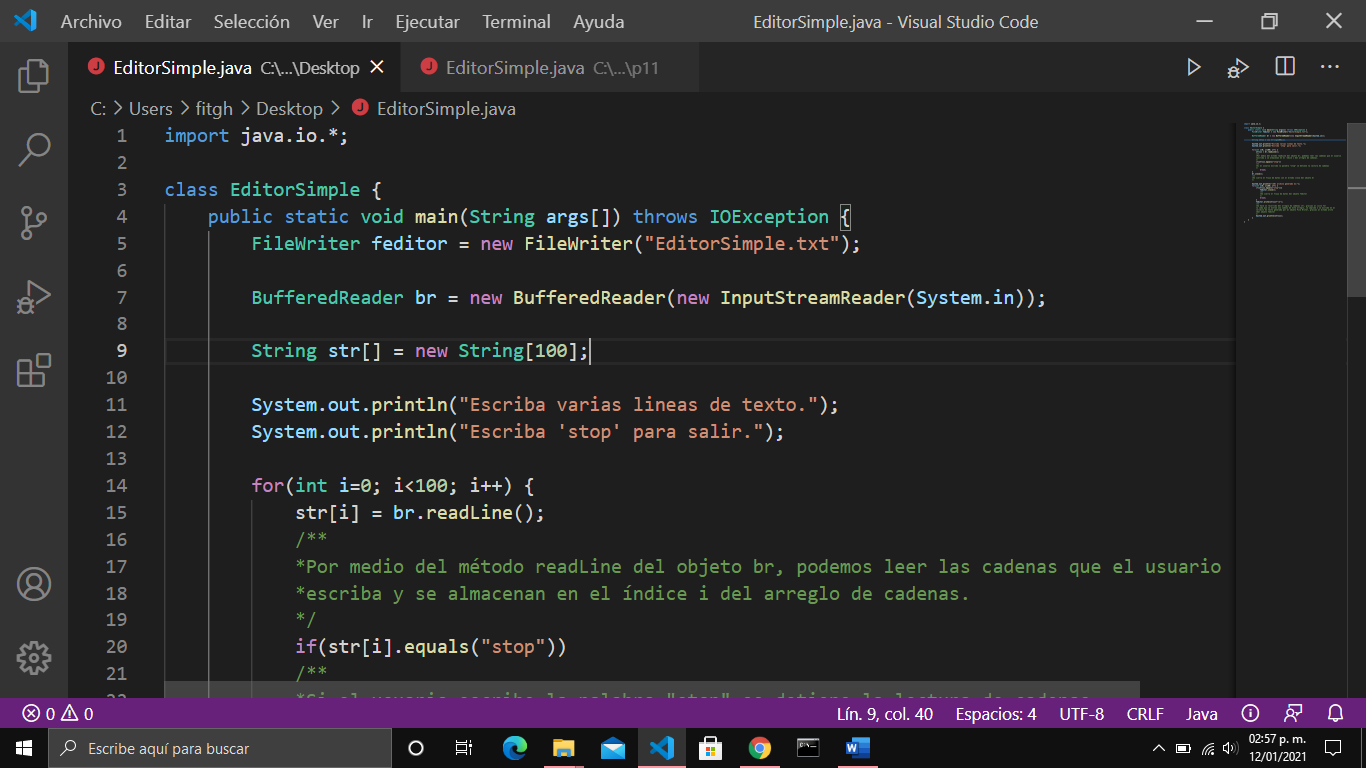
***Ejercicio 2***

* **Análisis previo.**

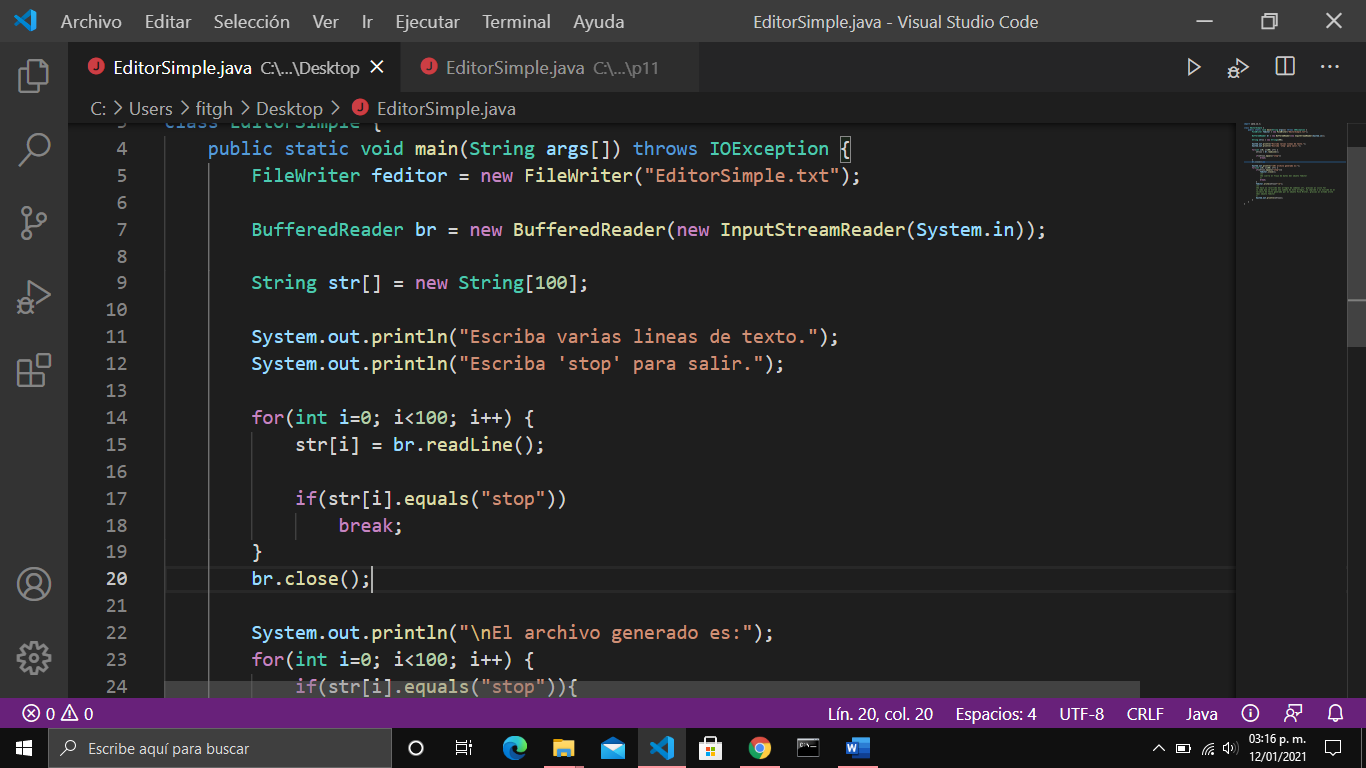
Tomando como base el Código ya escrito en EditorSimple.java, para poder guardar algún texto recibido por línea de comandos en un archivo de texto plano mediante un objeto de tipo FileWriter, inicialmente se debe de crear un objeto de la clase FileWriter, cuyo constructor debe contener el nombre del archivo de texto con su respectiva extensión, para que posteriormente con la ayuda del ciclo for previamente desarrollado, el cual se encarga de hacer el recorrido sobre el respectivo arreglo de cadenas con la información introducida, poder guardar e imprimir las respectivas cadenas que el usuario escribe hasta encontrarse con la palabra "stop", a partir de la cual se detiene la lectura de cadenas.

* **Desarrollo**

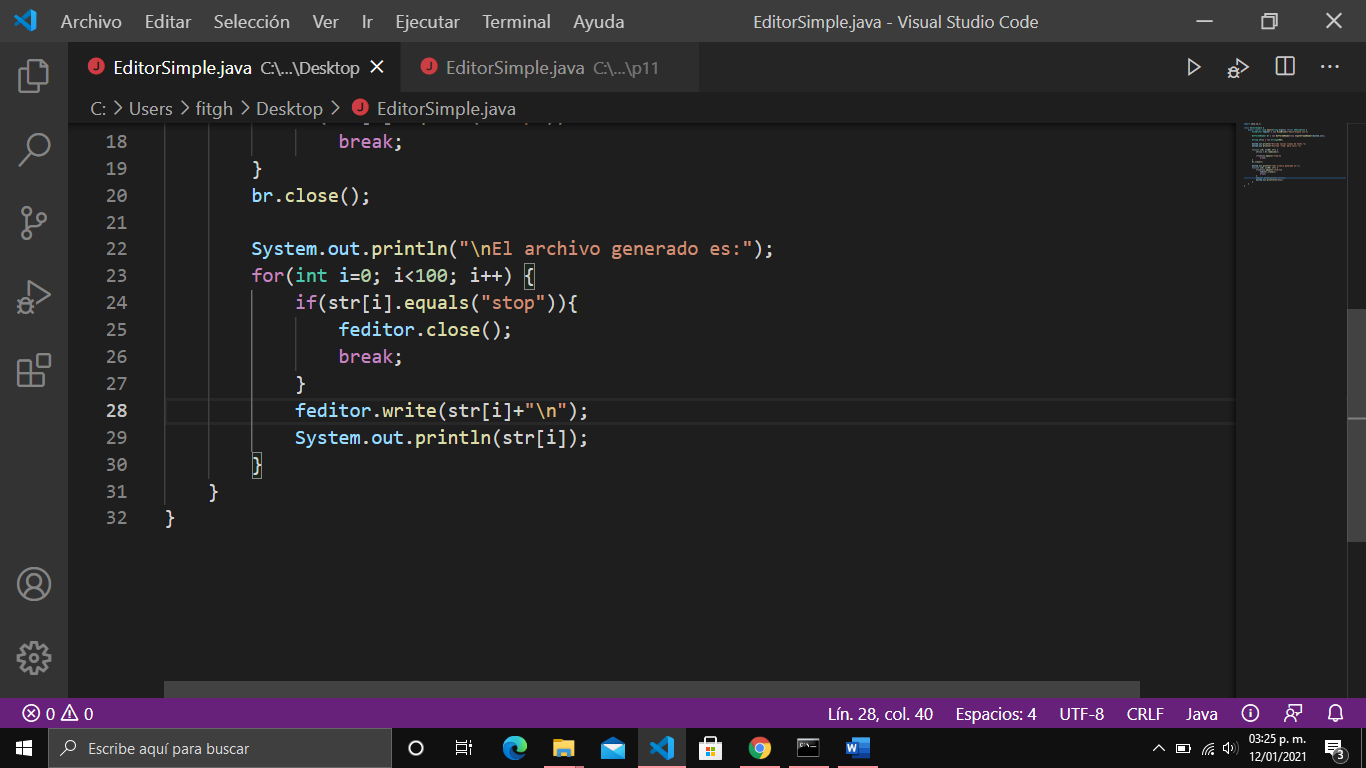
Inicialmente se creó el archivo “feditor” que es un objeto de la clase “FileWriter”, para permitir guardar las respectivas cadenas de caracteres o flujos de bytes en un archivo de texto plano, y en su método constructor recibe como parámetro el nombre del archivo de texto en este caso “EditorSimple.txt”, y como originalmente se tenía se crea un objeto de la clase BufferedReader llamado “br”, que nos permitirá leer las respectivas cadenas.



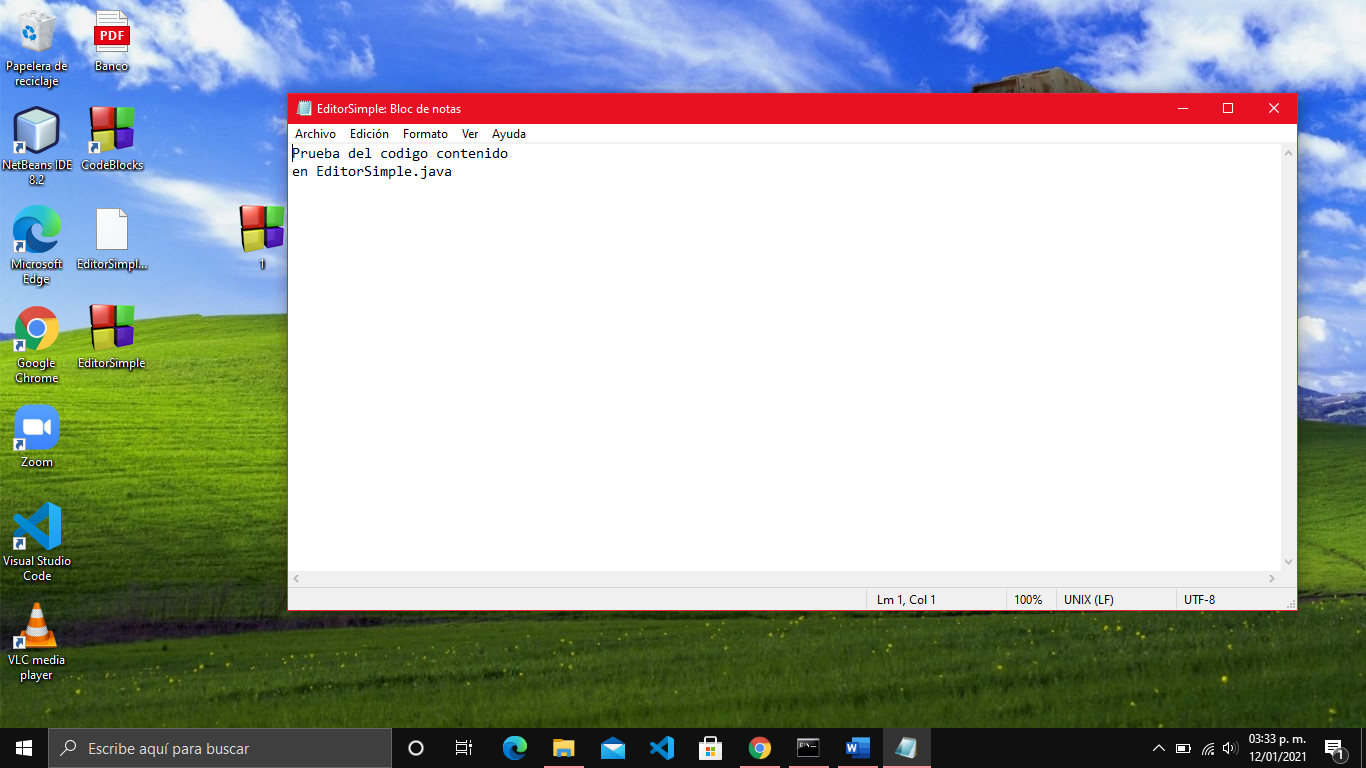
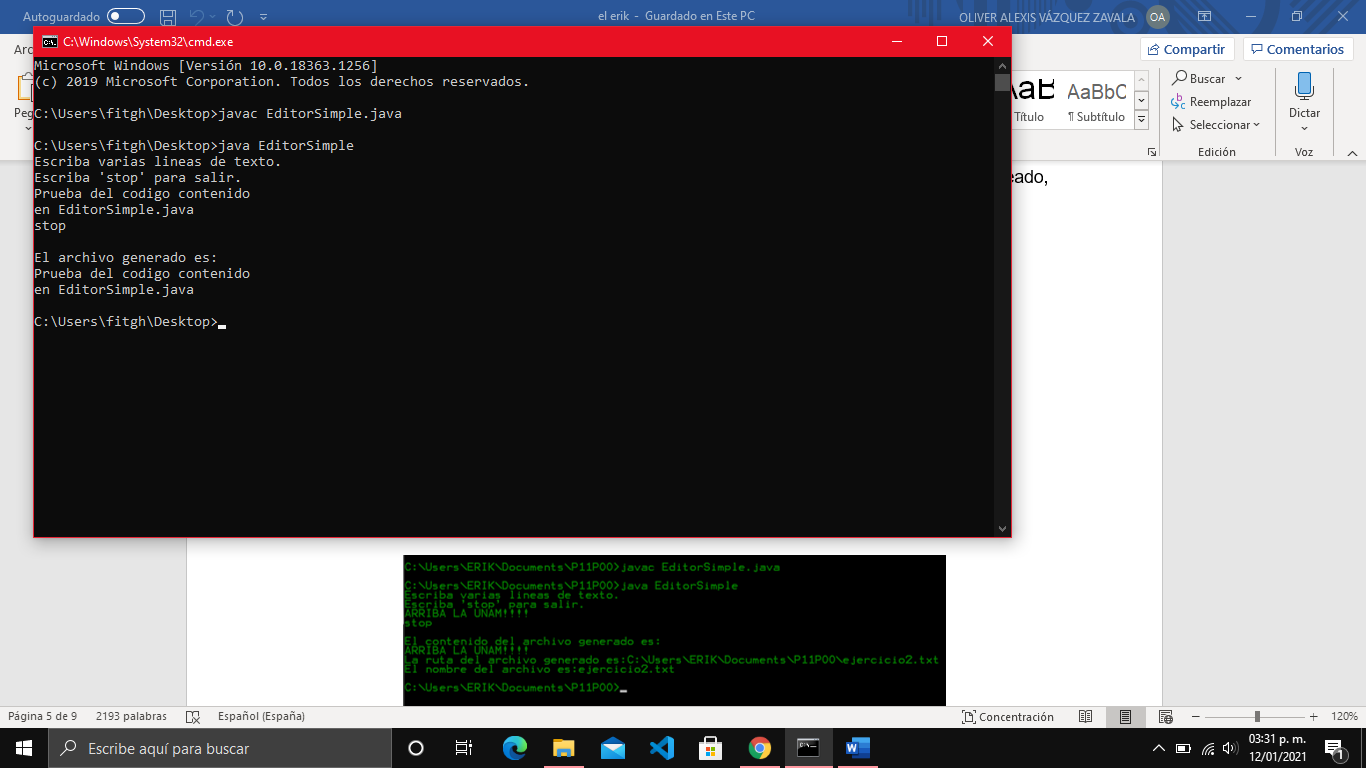
Posteriormente en el primer ciclo por medio del método readLine del objeto de la clase BufferedReader, podemos leer las cadenas que el usuario escriba y se almacenan en el índice i del arreglo de cadenas y si el usuario escribe la palabra "stop" se detiene la lectura de cadenas.

****

Por último en el otro ciclo se hace un recorrido del arreglo de cadenas y cada que se extrae una cadena de este arreglo, se imprime y se escribe en el archivo de texto generado por el objeto File Writer, con el método write del objeto.



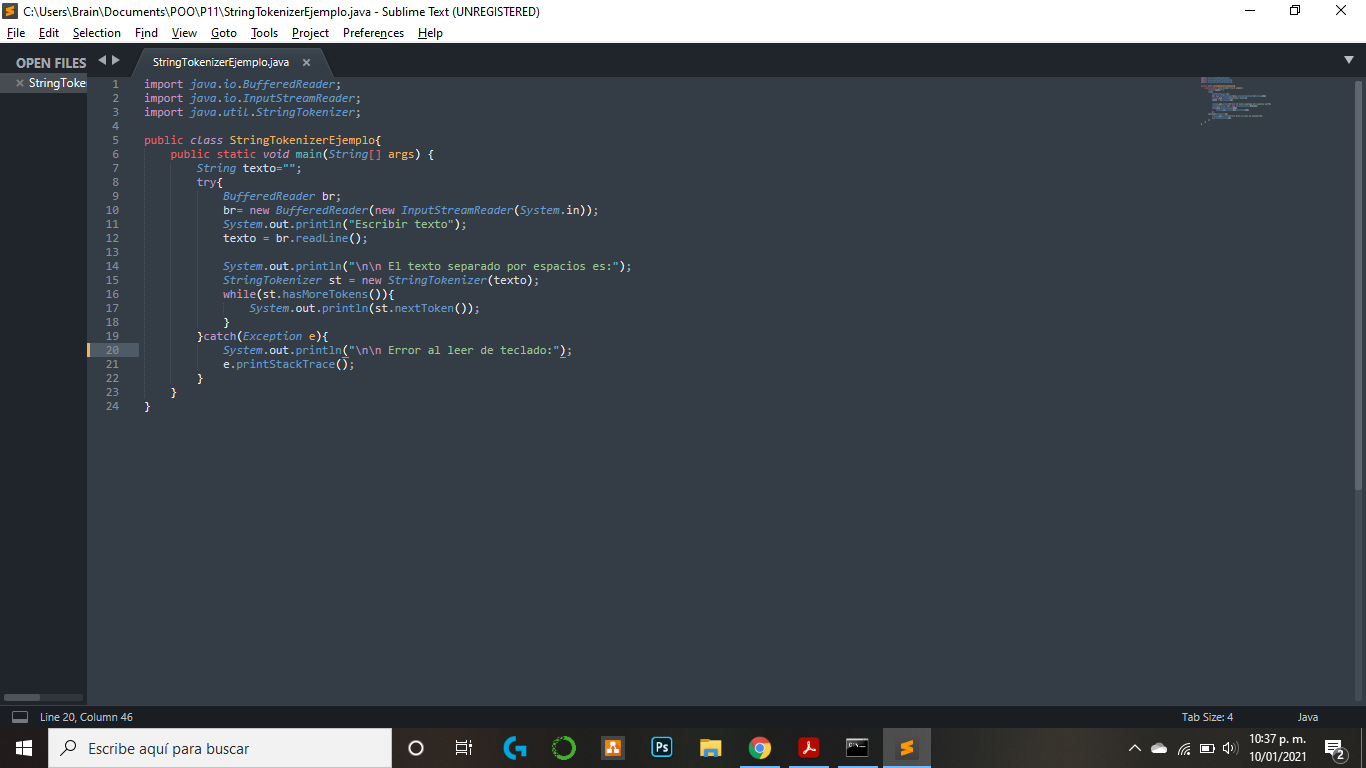
* **Ejecución del programa**



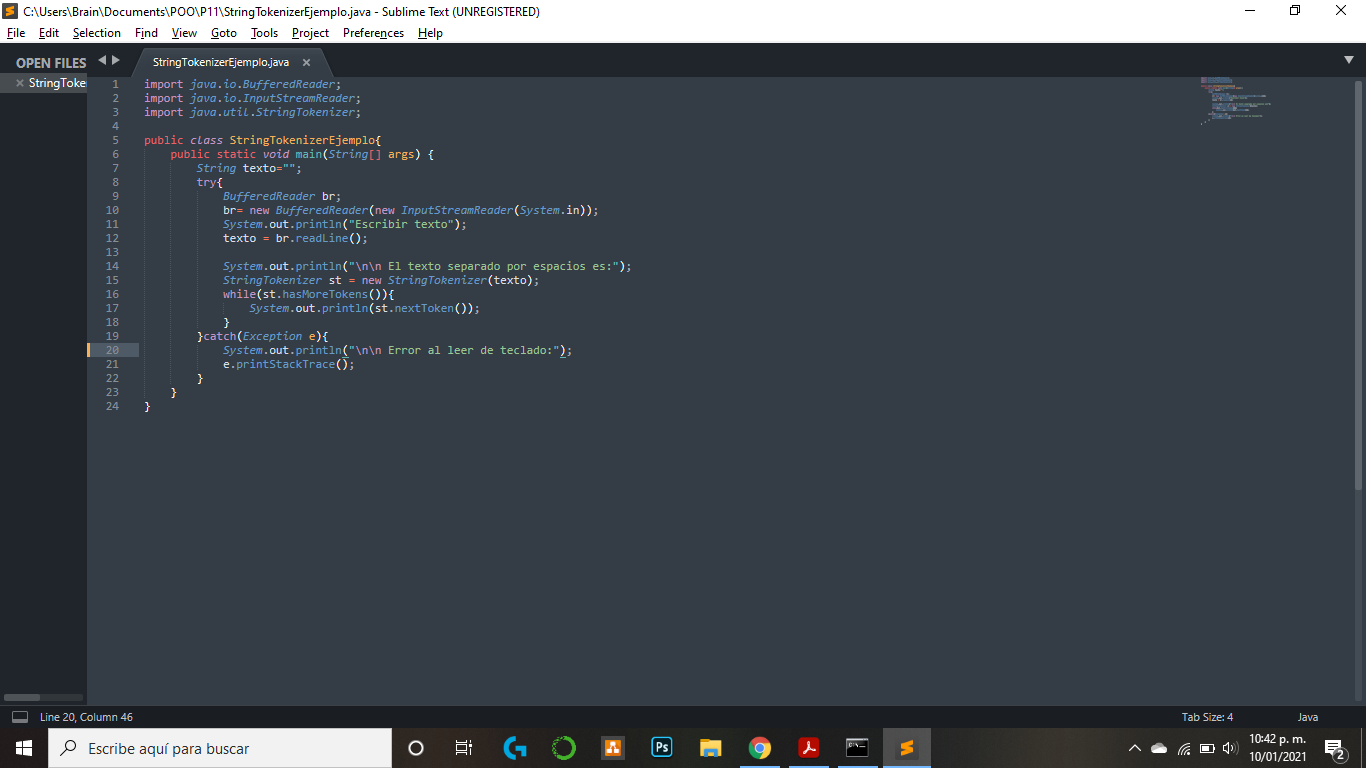
***Ejercicio 3***

* **Análisis.**

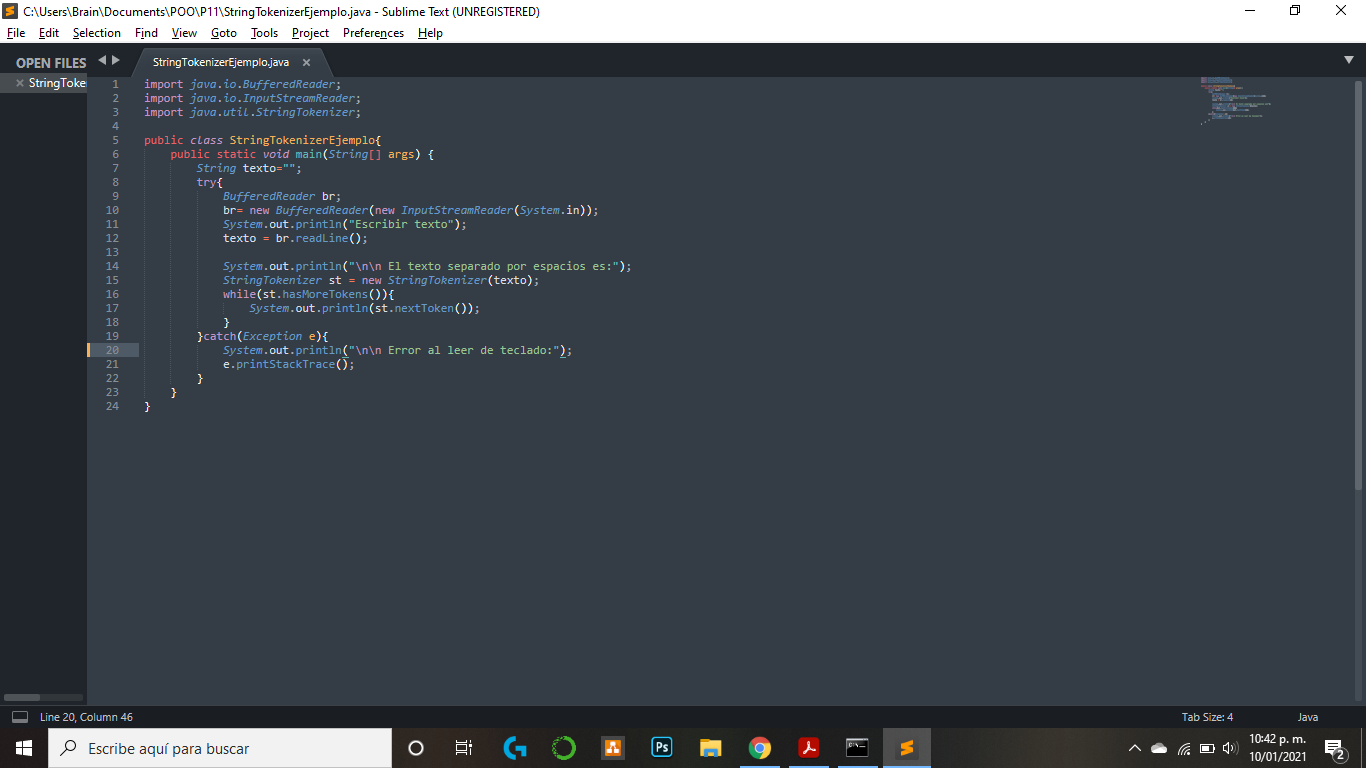
La clase BufferedReader permite la creación de un buffer que permita la correcta lectura de los caracteres, en este caso el constructor recibe una clase reader, que en este caso es un InputStreamReader, la cual permite que se introduzcan bytes y se traduzcan a caracteres. En este caso se utiliza la instrucción que permite la mayor eficiencia posible.



El método readLine(), permite que se lea una línea desde el teclado, que se traduce en una línea de texto. Por la apertura de un flujo de caracteres al crear el buffer, se debe colocar un try-catch en caso de que no sea posible hacer la reserva del buffer y que no se pueda la lectura del teclado.



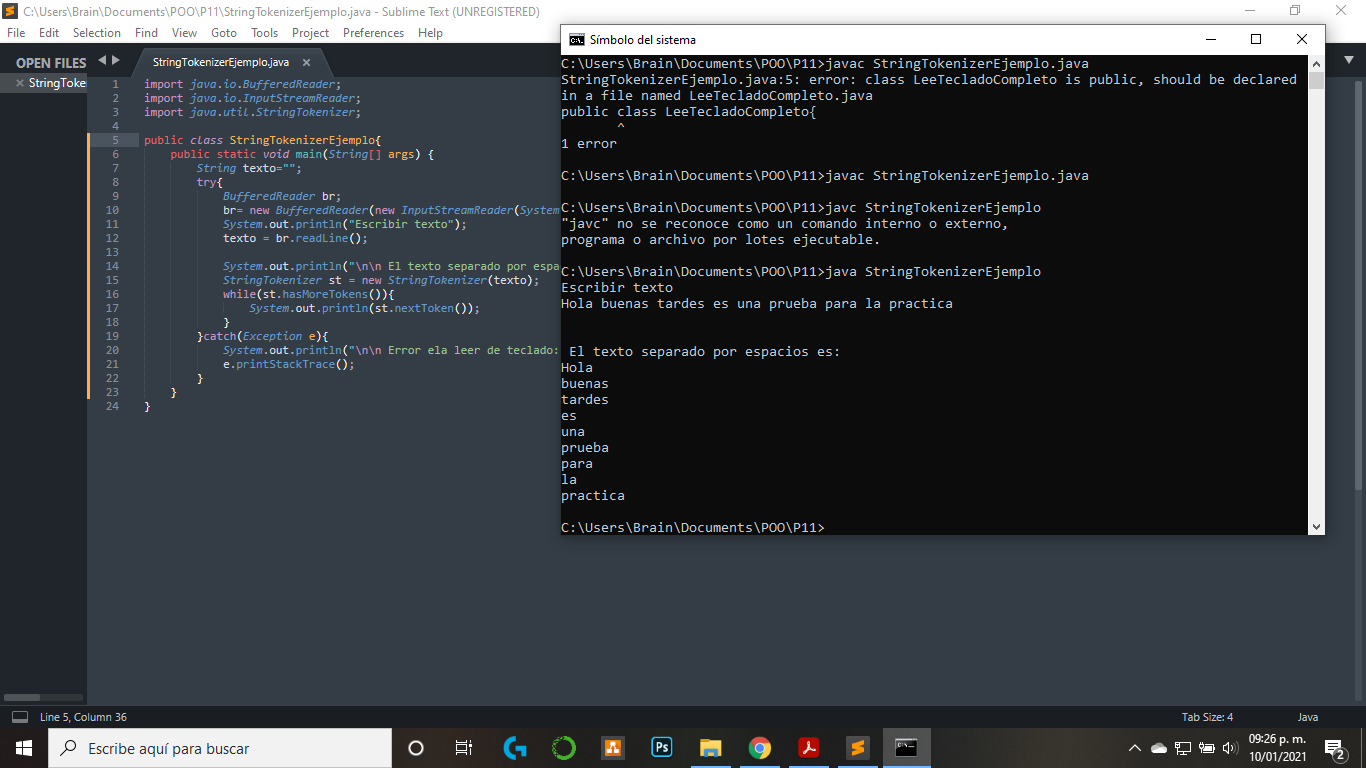
La clase StringTokenizer permite convertir una cadena en una serie de tokens, esto significa que la cadena será partida por cada elemento que la compone, entonces una cadena de n palabras será descompuesta en estas n palabras. El método constructor de la clase puede recibir el texto a separar en tokens y también se pueden colocar elementos, como comas, punto, guiones, etc. que permitan la creación de tokens de forma más selectiva.



Por medio de esta clase se tienen varios métodos que verifican la existencia de los tokens, y la salida de los tokens, el método hasMoreTokens(), permite verificar que existan tokens dentro del objeto que se puedan utilizar, y el método nextToken() regresa el token primero o el siguiente dentro del objeto.

Se implementa un while() para poder imprimir todos los tokens, y de esta forma en el programa ver cada uno de estos que componen la cadena.

* **Ejecución del programa**

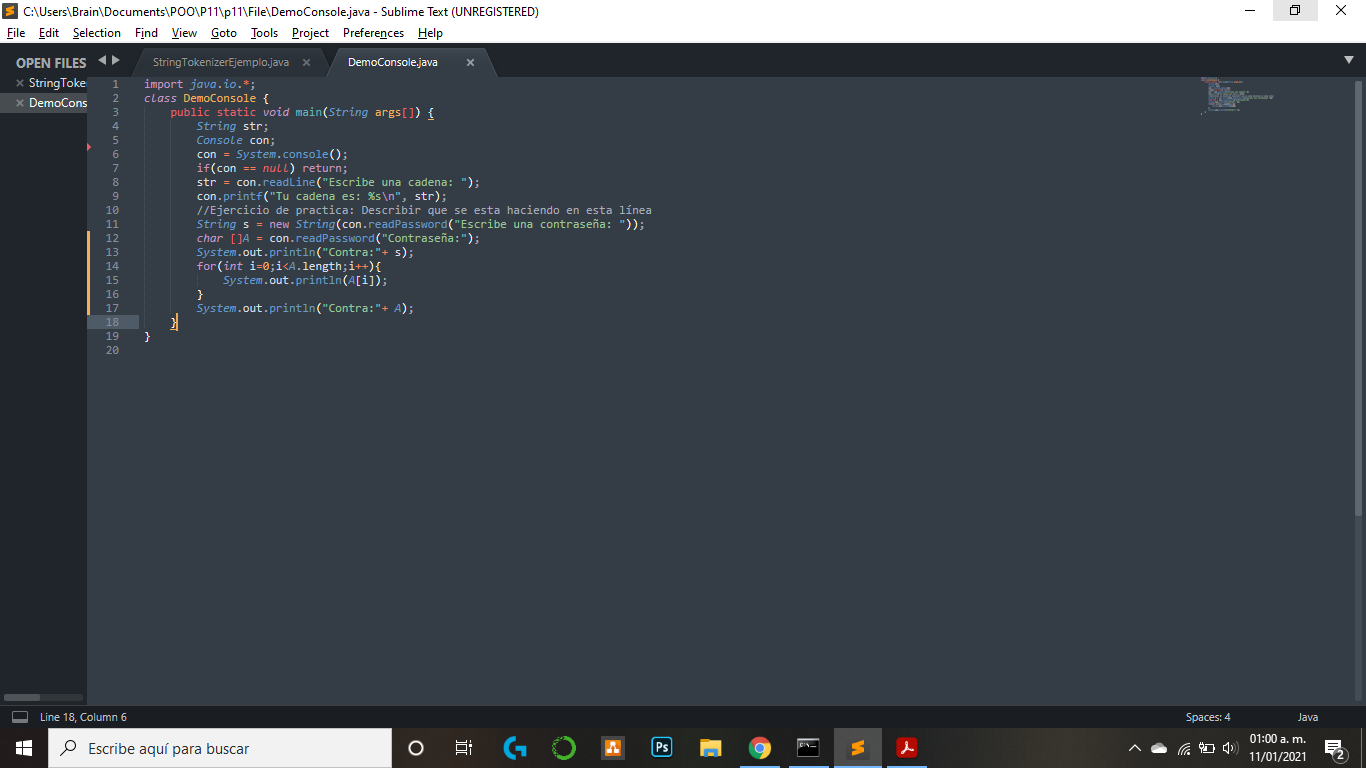


***Ejercicio 4***

* **Análisis.**

La máquina virtual permite tener una instancia la clase de consola, esta es únicamente una por lo que se invoca por medio de la sentencia System.console(), y en caso de que se retorne un null, significa que la máquina virtual no puede tener una instancia de la consola.

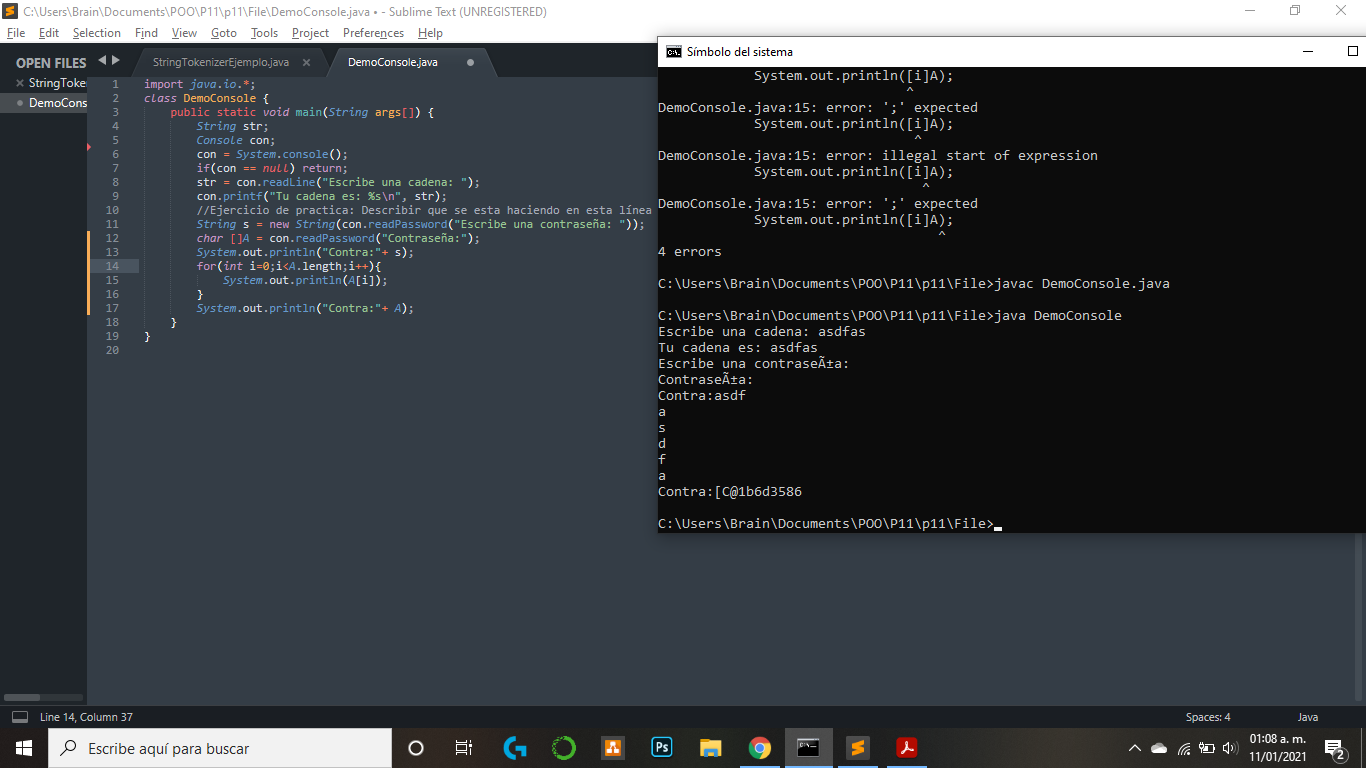
Con respecto a la siguiente parte:

****

Con la línea indicada se utiliza el constructor de la clase String, para poder tomar los valores de regreso del método readPassword() , el cual es un arreglo de chars por lo que el constructor toma el conjunto de ese arreglo y lo representa como una cadena, es algo similar a un casteo o aun tokenizer inverso. Al momento de colocar la cadena en la consola no se muestran los caracteres debido a que se ejecuta de esta forma por seguridad.

¿Es necesario convertir a cadena la contraseña obtenida del ejemplo de la clase Console?. En este caso no es necesario, debido a que el método regresa un arreglo de caracteres, y por medio de este es posible guardar la contraseña en otro arreglo de caracteres. Y a través de un for se puede acceder a cada uno de los elementos de la contraseña y de igual forma se puede mandar a imprimir por el println, pero solo que se manda a imprimir el toString() y al ser un arreglo se tiene una combinación de caracteres que representan al arreglo.

* **Ejecución del programa**

****

***Conclusiones***

* Dávila Ortega Jesús Eduardo:

Durante el desarrollo de la práctica logre comprender la abstracción que se lleva acabo durante el manejo de archivos en java, y como identificar que tipo de clases son necesarias dependiendo de lo que requiera el código para funcionar de una manera correcta, así como también logre comprender un poco mas sobre el manejo de flujos de datos y caracteres en java logrando así los objetivos de la practica.

* Díaz Hernández Marcos Bryan:

Al desarrollar la práctica y los ejemplos de esta, el flujo de datos fue más sencillo de visualizar en los archivos, mediante el flujo de los bytes y el flujo de los caracteres, el cual se me hizo más sencillo de entender por medio de los ejercicios, de esta forma pude entender el funcionamiento de las clases que se utilizan para el flujo y al final poder cumplir con el objetivo de la práctica.

* Pareja Ávila Emiliano:

Pude comprender la abstracción de un archivo dentro de un lenguaje de programación, y su concepto como un objeto que tiene métodos propios, se como identificar y utilizar las clases de archivos que se requieran para entrada o salida de datos, y también se como los archivos pueden recibir información a partir de un flujo de bytes o de cadenas. Por lo tanto, se cumplieron los objetivos de la práctica.

* Vázquez Zavala Oliver Alex:

Con el desarrollo de esta práctica pude reforzar conceptos relacionados con el manejo de archivos en Java, tarea de la cual pude darme cuenta que es bastante común en la programación al momento de desarrollar un proyecto, así como poder comprender para qué sirve cada una de las diversas clases empleadas en el desarrollo de la práctica, notando las diversas operaciones y funciones que tienen los archivos, las cuales hacen de los archivos una herramienta de bastante utilidad ya que en estas entidades podemos almacenar diversa información, procesos, composiciones etc., que son de gran utilidad para nuestros programas dependiendo de que sea sobre lo que se esté trabajando.

***Bibliografía:***

* Flores Fernández, Héctor Arturo(2012) ”Programación Orientada a Objetos usando Java”