	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): _____

Asignatura: _____

Grupo: _____

No de Práctica(s): _____

Integrante(s): 321172974

_____

_____

_____

No. de lista o brigada: _____

Semestre: _____

Fecha de entrega: _____

Observaciones: _____

_____

CALIFICACIÓN: _____

ÍNDICE

Introducción

Planteamiento del problema

Diseñar e implementar un programa que permita leer tres archivos de texto plano y escribir los contenidos de los archivos en otro archivo diferente

Motivación

El manejo de archivos es primordial para poder manejar sistemas que puedan almacenar información en la memoria secundaria, de tal forma que no dependan únicamente de la memoria principal para poder recordar o consultar datos previamente guardados.

De esta forma, la lectura y escritura en archivos es esencial para la construcción de sistemas completos.

Objetivos

Implementar una aplicación que resuelva el problema planteado además de:

- ☑ Tener el programa encapsulado
- ☑ Definirlo en un paquete
- ☑ Contar con un jar y su documentación javadoc generados
- ☑ Incluir su diagrama de clase

Marco Teórico

El manejo de archivos en Java es fundamental para aplicaciones que requieren almacenar y recuperar datos de manera persistente. El lenguaje proporciona una amplia gama de clases en el paquete java.io para este propósito, permitiendo operaciones de lectura y escritura de archivos de texto y binarios.

Para leer archivos se tienen las clases FileReader y BufferedReader, donde se puede decir según **[1],** que la clase BufferedReader es comúnmente utilizada debido a su capacidad de leer datos de manera más eficiente mediante un búfer.

BufferedReader

Objeto que lee texto de una entrada de caracteres, poniéndolos en un buffer para así leerlos de manera eficiente caracteres, arreglos y líneas.

El BufferedReader es una clase que envuelve un objeto de tipo Reader, añadiendo funcionalidad de búfer. Esto permite leer líneas completas de texto, lo cual es ideal para procesar archivos grandes. Su uso involucra crear una instancia que reciba un FileReader y posteriormente leer el contenido del archivo línea por línea.

FileWriter

La clase FileWriter facilita la escritura de caracteres en un archivo. Es particularmente útil para crear o modificar archivos de texto. Su método write permite escribir cadenas de texto directamente al archivo. Para garantizar que los datos se escriban correctamente, se utiliza el método flush y se cierra el flujo con close una vez completadas las operaciones

Métodos afines[1]

Objeto	Método	Descripción
BufferedReader	`readLine()`	Lee una línea de texto
BufferedReader / FileWriter	`close()`	Cierra la entrada/salida y libera los recursos del sistema asociados a ella

[1] Tabla construida con la API de Java **[2], [3], [4], [5]**

FileWriter	**`write(String)`**	Escribe una porción de un arreglo de caracteres o una cadena
StringBuilder	**`append(String)`**	Añade la cadena especificada en la secuencia de caracteres
System	**`lineSeparator()`**	Regresa el separador de línea establecido por el sistema

Excepciones

Las operaciones de lectura y escritura de archivos pueden generar excepciones como IOException o sus subclases específicas, como FileNotFoundException. Estas excepciones deben ser manejadas mediante bloques try-catch para asegurar la robustez del programa y evitar fallos inesperados. Java también permite declarar excepciones utilizando la palabra clave throws en los métodos que realizan estas operaciones

Excepción	**Descripción**
IOException	Alguna excepción del tipo I/O ha ocurrido (Input or Output)
FileNotFoundException[2]	Específicamente se lanza cuando un constructor de algún FileInputStream falla al intentar abrir un archivo con una ruta que no existe, o el archivo es inaccesible

[2] **[6]**

Desarrollo

Ejercicios en clase

`Ejercicio0`	
Planteamiento	**Solución**
Programa que lea el contenido de un archivo línea por línea	Se construye un bloque try catch para verificar si se puede leer un archivo al momento de crear un objeto del tipo BufferedReader con la ruta especificada [3] Intenta leer mientras strLine no sea nulo, es decir mientras no haya llegado al final del archivo, por lo que va añadiendo al StringBuilder lo leído al mismo tiempo que lo imprime en pantalla

`Ejercicio1`	
Planteamiento	**Solución**
Programa que lea un archivo línea por línea y lo vaya almacenando en una variable	Se construye un bloque try-catch para manejar excepciones relacionadas con la lectura de un archivo. Se utiliza la clase BufferedReader para leer el contenido del archivo especificado en la ruta ./test.txt. Dentro del bloque try, se inicializa una cadena str_data que concatenará cada línea leída del archivo en un solo texto. El programa realiza un ciclo while que verifica si strLine no es null para leer línea

[3] Si la ruta no existe arroja una excepción del tipo FileNotFoundException

	por línea del archivo. Cada línea leída se agrega a la cadena str_data. Si strLine es null, el ciclo se interrumpe, indicando que se ha alcanzado el final del archivo. Una vez terminado el proceso de lectura, el contenido completo del archivo, ahora almacenado en str_data, se imprime en la consola. Finalmente, el flujo de lectura es cerrado utilizando el método close. Si ocurre una excepción al intentar abrir el archivo, como que no exista el archivo especificado, se captura mediante un bloque catch y se imprime un mensaje de error que indica "Archivo no encontrado ...". Si la lectura del archivo falla por alguna otra razón, se lanza y maneja una excepción de tipo IOException con el mensaje "No es posible leer el archivo ...".

`Ejercicio2`	
Planteamiento	**Solución**
Programa que almacene los contenidos de un archivo de texto línea por línea en un arreglo	Se implementa un bloque try-catch para manejar la lectura de un archivo utilizando un objeto de tipo BufferedReader. El archivo se especifica con la ruta ./test.txt. Dentro del bloque try, se define una cadena strLine para almacenar cada línea leída temporalmente, un StringBuilder para construir un texto acumulativo, y una lista list para almacenar cada línea del archivo como un elemento independiente. El programa utiliza un ciclo while que lee línea por línea del archivo. Cada línea leída se agrega al StringBuilder para formar el contenido acumulado. Después de leer una

	línea, se agrega a la lista list si no es nula. El ciclo se interrumpe al encontrar que strLine es igual a null, lo que indica que se llegó al final del archivo. Una vez terminado el proceso de lectura, la lista que contiene las líneas individuales del archivo es impresa en consola utilizando Arrays.toString(). En caso de que el archivo no se encuentre, se lanza y captura una excepción FileNotFoundException, mostrando el mensaje "Archivo no encontrado...". Si ocurre cualquier otra excepción, se captura con un bloque catch genérico, mostrando el mensaje "Error al abrir el archivo". Esto asegura que el programa maneje de manera robusta posibles errores durante la operación.

`Ejercicio3`	
Planteamiento	**Solución**
Programa para escribir y leer en un archivo de texto plano	En este código, se implementa un bloque try-catch para manejar tanto la escritura como la lectura de un archivo. Primero, se utiliza la clase FileWriter para crear o sobrescribir el archivo especificado con la ruta ./hola.txt. Dentro del bloque try, el programa escribe una línea de texto en el archivo utilizando el método write. Posteriormente, el flujo se cierra con el método close para asegurarse de que los datos se guarden correctamente. Luego, el archivo recién creado se lee utilizando un objeto de tipo BufferedReader, que se inicializa con la misma ruta del

	archivo. El programa utiliza un ciclo while para leer el archivo línea por línea, almacenando cada línea en un StringBuilder para construir el contenido completo del archivo. Al mismo tiempo, cada línea leída se imprime en la consola. El ciclo finaliza al alcanzar el final del archivo, identificado por un valor nulo en la lectura. En caso de que ocurra una excepción relacionada con la entrada o salida de datos, como problemas al abrir, escribir o leer el archivo, esta se captura en el bloque catch y se imprime un mensaje con la descripción del error utilizando el método getMessage.

Ejercicios de Tarea

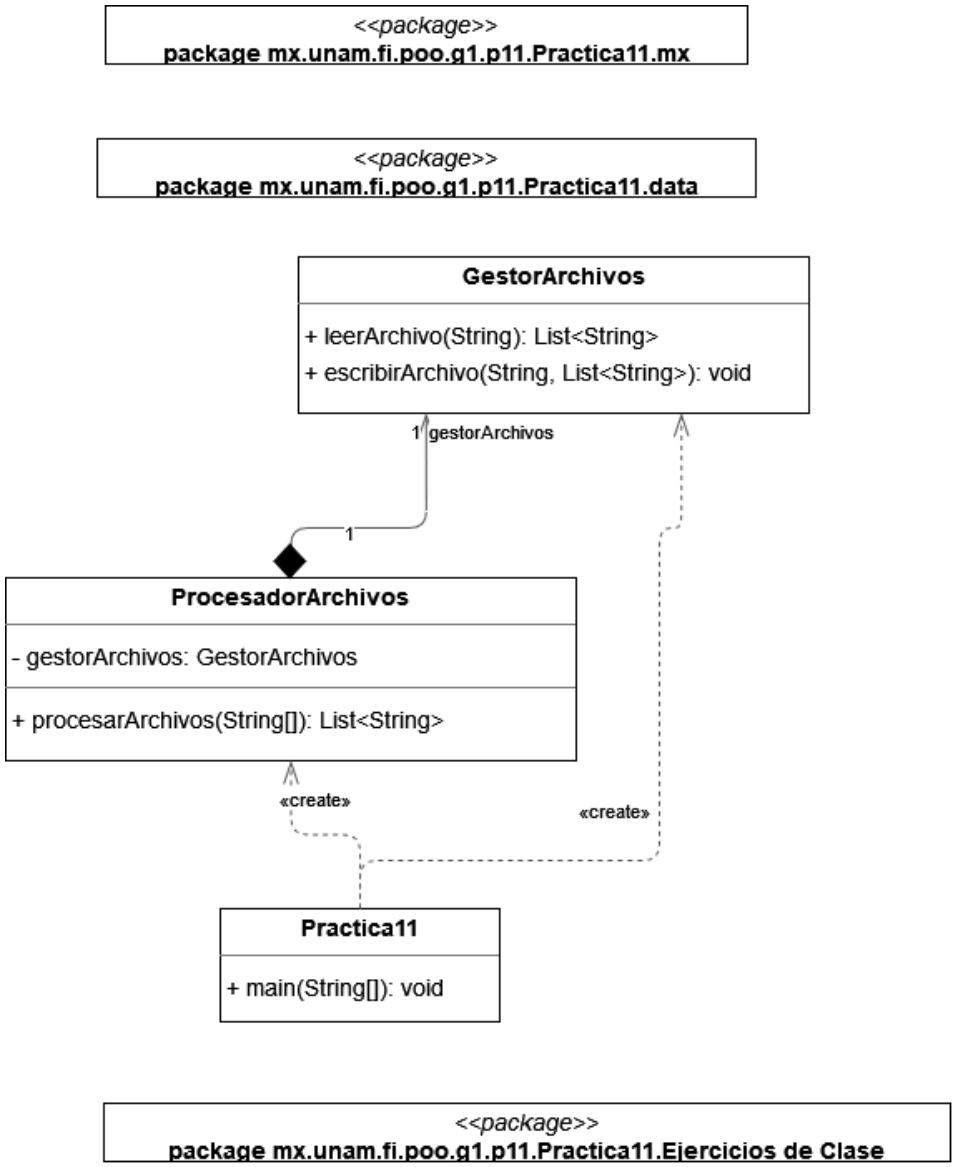
`Consideraciones`

Se crearon 2 clases aparte de la principal

GestorArchivos	**ProcesadorArchivos**
Cuenta con 2 métodos, leerArchivo y escribirArchivo, cada una usa su respectivo objeto para hacer su acción principal, manejando mediante try-catch excepciones. BufferedReader se ocupa en bucle while hasta encontrar null, para guardar en una cadena todo el contenido. BufferedWriter para escribir en bucle for mientras haya contenido. El contenido se almacena en una lista de Strings, siendo cada elemento lo leído de un solo archivo.	Utiliza una instancia de GestorArchivos, y a través de su método procesarArchivo, maneja la lectura de los 3 archivos solicitados.[4] La lista resultante con el contenido de los 3 archivos será devuelta para que desde main se llame al método escribirArchivo del gestor, mandandole la lista con lo leído en este método

[4] Se recuerda que los archivos son rutas dispuestas en un arreglo de strings, por lo que se iterará sobre este arreglo para hacer la lectura

Diagramas de clases UML



Se puede apreciar como ProcesadorArchivos tiene una relación 1 a 1 con GestorArchivos

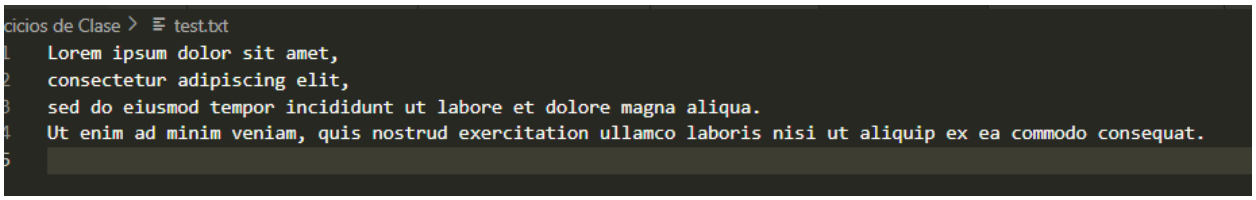
Pruebas

Al ser cortas, se disponen directamente en los Resultados

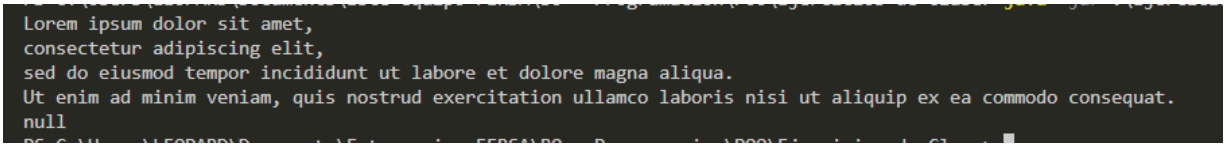
Resultados

Ejercicios en Clase

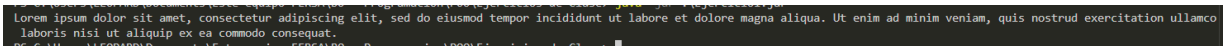
Se dispone en general con el archivo de texto:



Ejercicio0

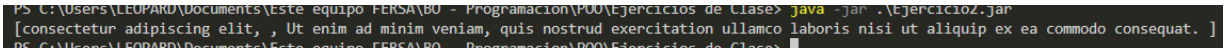


Ejercicio1



Note la separación por comas

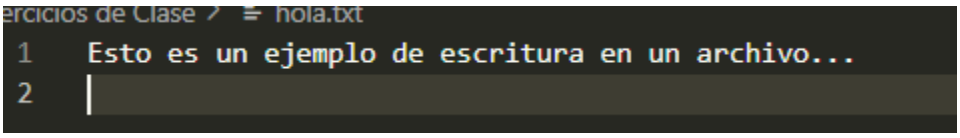
Ejercicio2



Note la forma de lista

Ejercicio3

Escritura en archivo:



Ejercicios de tarea

Se tienen 3 archivos:
Resumen.txt

Resumen.txt

```
El resumen del programa es:
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. (Si)
```

Programa.txt

Programa.txt

````
Este es el ejercicio0:
```java
package mx.unam.fi.poo.g1.p11.ejClase.Ej0;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileReader;
import java.io.FileNotFoundException;

public class Ejercicio0{
    public static void main(String[] args){
        StringBuilder sb = new StringBuilder();
        String strLine = "";
        try {
            BufferedReader br = new BufferedReader(new FileReader("./test.txt"));
            while(strLine != null){
                sb.append(strLine);
                sb.append(System.lineSeparator());
                strLine = br.readLine();
                System.out.println(strLine);
            }
            br.close();
        } catch (FileNotFoundException e) {
            System.err.println("Archivo no encontrado...");
        } catch (IOException e){
            System.err.println("No es posible leer el archivo...");
        }

    }
}

```
````

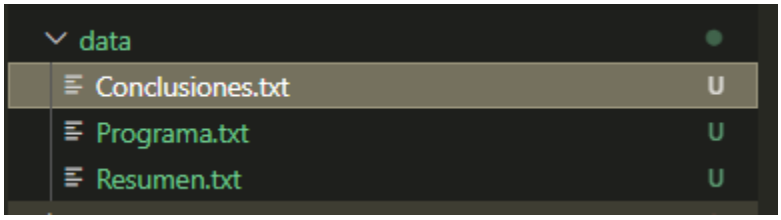
Conclusiones.txt

Conclusiones.txt

```
En conclusion, se puede concluir de manera contundente que al finalizar el proyecto
podemos dar cierre a la elaboración finalmente convergiendo en la idea de que es unanime la decision
de poder cerrar con el analisis y dar paso a la conclusion la cual resulta en una clausura al proyecto en si.
```

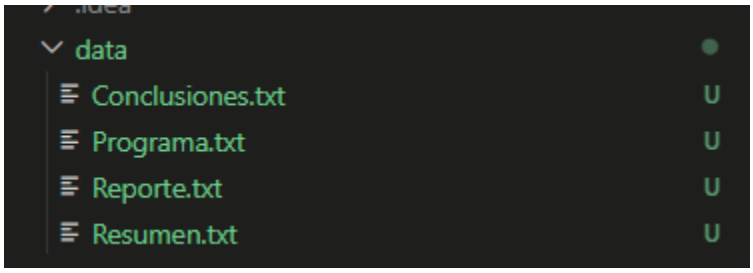
Los 3 en un directorio a nivel de raíz que se llama data.

Originalmente solo estos 3 archivos existen



Pero al momento de ejecutar el programa, los 3 contenidos se juntan:

```
PS C:\Users\LEOPARD\Documents\Este equipo FERSA\BO - Programacion\POO> java -jar Practica11.jar
El contenido de los archivos ha sido combinado en: ./data/Reporte.txt
```



Reporte.txt

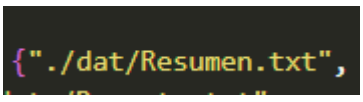
El resumen del programa es:
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. (Si)
Este es el ejercicio0:

```java
package mx.unam.fi.poo.g1.p11.ejClase.Ej0;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileReader;
import java.io.FileNotFoundException;

public class Ejercicio0{
    public static void main(String[] args){
        StringBuilder sb = new StringBuilder();
        String strLine = "";
        try {
            BufferedReader br = new BufferedReader(new FileReader("./test.txt"));
            while(strLine != null){
                sb.append(strLine);
                sb.append(System.lineSeparator());
                strLine = br.readLine();
                System.out.println(strLine);
            }
            br.close();
        } catch (FileNotFoundException e) {
            System.err.println("Archivo no encontrado...");
        } catch (IOException e){
            System.err.println("No es posible leer el archivo...");
        }

    }
}
```

En conclusion, se puede concluir de manera contundente que al finalizar el proyecto
podemos dar cierre a la elaboración finalmente convergiendo en la idea de que es unanime la decision
de poder cerrar con el analisis y dar paso a la conclusion la cual resulta en una clausura al proyecto en si.

Escritura de los 3 archivos originales en 1 solo

Excepción con archivo no existente

Si se ingresa el archivo: 

En vez de data, la ruta no existe, por lo que el programa maneja una excepción:

```
PS C:\Users\LEOPARD\Documents\Este equipo FERSA\BO - Programacion\POO> ja
Archivo no encontrado: ./dat/Resumen.txt
El contenido de los archivos ha sido combinado en: ./data/Reporte.txt
```

Conclusiones

El manejo de archivos es esencial para sistemas complejos, ya que ellos requieren almacenar grandes cantidades de información, misma que puede ser consultada posteriormente por el propio usuario, dicha acción requiere una gestión de archivos que no solo sea eficiente, sino sea capaz de manejar excepciones.

La escritura y lectura de archivos se da en todo momento, el poder tener el conocimiento para construir sistemas que permitan implementar acción tan primordial resulta necesario para cualquier ingeniero en computación ocupando cualquier lenguaje realmente.

Dado que se cubren los requerimientos, el objetivo de la práctica fue cubierto.

Referencias

[1] P. Deitel y H. Deitel. *CÓMO PROGRAMAR EN JAVA*. Sétima edición. México: Pearson Educación México, 2008. ISBN: 978-970-26-1190-5.

[2] "BufferedReader (Java SE 11 & JDK 11)". Java API. Accedido el 4 de noviembre de 2024. [En línea]. Disponible: https://javadoc.scijava.org/Java11/java.base/java/io/BufferedReader.html

[3] "FileWriter (Java SE 11 & JDK 11)". Java API. Accedido el 4 de noviembre de 2024. [En línea]. Disponible: https://javadoc.scijava.org/Java11/java.base/java/io/FileWriter.html

[4] "StringBuilder (Java SE 11 & JDK 11)". Java API. Accedido el 3 de noviembre de 2024. [En línea]. Disponible: https://javadoc.scijava.org/Java11/java.base/java/lang/StringBuilder.html

[5] "System (Java SE 11 & JDK 11)". Java API. Accedido el 4 de noviembre de 2024. [En línea]. Disponible: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#lineSeparator()

[6] "FileNotFoundException (Java SE 11 & JDK 11)". Java API. Accedido el 4 de noviembre de 2024. [En línea]. Disponible: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/FileNotFoundException.html