	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Jose Antonio Ayala Barbosa

Asignatura: Programación Orientada A Objetos

Grupo: 8

No de Práctica(s): 11

Integrante(s): Velarde Valencia Josue, N.L. 44

Mendoza Valdez Emily Isabela, N.L. 32

Semestre: 2024-1

Fecha de entrega: 17 de Noviembre de 2023

Observaciones:

CALIFICACIÓN: _____

Manejo De Archivos.

Mendoza, Emily.
Ingeniería en Computación, UNAM.
CDMX, México.
mendoza.valdez.ei@gmail.com

Velarde, Josue
Ingeniería en Computación, UNAM.
CDMX, México.
josue.velardevalencia@gmail.com

Cuestionario Previo 11

1. ¿Qué es un token?

En programación, un token es la unidad básica de un lenguaje de programación. Puede ser una palabra clave, un identificador, un operador, un delimitador o cualquier otra entidad reconocible que constituye la sintaxis de un lenguaje. Durante el análisis léxico (la fase inicial del proceso de compilación), el código fuente se divide en tokens para facilitar su procesamiento por parte del compilador o intérprete.

En el contexto de procesamiento de texto, un token puede ser una palabra, una frase, un número, o cualquier otra entidad que se desee identificar y manipular individualmente.

2. Investigar sobre StringTokenizer.

StringTokenizer es una clase en Java que se encuentra en el paquete java.util. Esta clase se utiliza para dividir o tokenizar una cadena en subcadenas más pequeñas, llamadas tokens, basándose en un delimitador específico. El delimitador puede ser especificado durante la creación de un objeto StringTokenizer, y por defecto, el espacio en blanco se utiliza como delimitador si no se proporciona uno explícitamente.

La clase StringTokenizer proporciona varios métodos útiles para trabajar con tokens, como hasMoreTokens() para verificar si hay más tokens en la cadena, y nextToken() para obtener el próximo token. Es comúnmente utilizado para procesar y analizar cadenas de texto en aplicaciones como la lectura de archivos CSV, análisis léxico simple, entre otros. Sin embargo, es importante tener en cuenta que StringTokenizer ha quedado obsoleto en versiones más recientes de Java, y se recomienda el uso de otras clases como Scanner o String.split() para tareas similares.

Resumen— En la práctica realizada se hizo uso del código en Java que maneja la lectura y escritura de datos de alumnos en archivos de texto, utilizando la clase Alumno para estructurar la información. Además, emplea StringTokenizer para procesar datos separados por comas en formato CSV.

Keywords—JAVA, clases, herencia, atributos, métodos,archivos .

I. INTRODUCCIÓN.

¡Claro! El manejo de datos es un proceso integral que involucra la adquisición, almacenamiento, procesamiento, análisis y visualización de conjuntos de información, ya sean grandes o pequeños. En un mundo cada vez más impulsado por la información, el manejo adecuado de datos se ha vuelto crucial en prácticamente todos los sectores: desde negocios y finanzas hasta la investigación científica y la atención médica.

El proceso comienza con la adquisición de datos, que puede provenir de diversas fuentes como encuestas, registros financieros, sensores, redes sociales, entre otros. Una vez recopilados, estos datos deben ser almacenados de manera estructurada en bases de datos o sistemas de almacenamiento para facilitar su acceso y manejo.

El paso siguiente implica la limpieza y preparación de los datos. A menudo, los conjuntos de información contienen errores, valores faltantes o duplicados, lo que puede afectar la calidad de los análisis posteriores. Por eso, es crucial realizar procesos de limpieza y preprocesamiento para asegurar que los datos estén listos para su análisis.

Luego, se procede con el análisis de los datos, donde se aplican diversas técnicas y herramientas, como el análisis estadístico, el aprendizaje automático y la minería de datos, para identificar patrones, tendencias o relaciones dentro de los datos. Este análisis ayuda a extraer información valiosa y a tomar decisiones fundamentadas en evidencias.

Finalmente, la visualización de datos juega un papel crucial al presentar los resultados de manera comprensible y

efectiva. Gráficos, tablas y otros métodos visuales permiten comunicar hallazgos de manera más accesible, facilitando la comprensión y la toma de decisiones por parte de los interesados.

En resumen, el manejo de datos es un proceso integral que abarca desde la recolección hasta la presentación de la información, siendo una herramienta poderosa para obtener insights significativos y fundamentar decisiones en múltiples áreas y sectores.

II. DESARROLLO.

Como primer paso se abre el proyecto en el cual vamos a trabajar, en este caso le asignaremos el nombre de POOP11 y crearemos la clase principal con el mismo nombre.

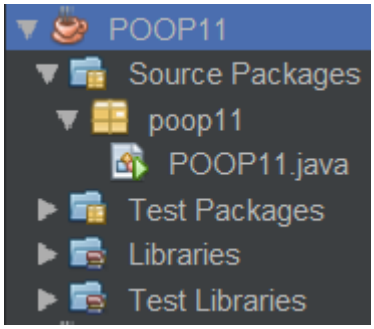

Figura 1. Creación del proyecto POOP11.

Este código en Java se centra en el manejo de archivos y la manipulación de datos de alumnos a través de la lectura y escritura en archivos de texto.

A. Clases

La clase 'Alumno' define un modelo de datos para representar información sobre un estudiante. Contiene atributos como nombre, apellidos, edad y número de cuenta. Además, tiene un constructor para inicializar estos atributos y un método `toString()` para imprimir los datos del alumno.

```
public static class Alumno {
    String nombre;
    String apellidoPaterno;
    String apellidoMaterno;
    int edad;
    int numCuenta;

    public Alumno(String nombre, String apellidoPaterno, String apellidoMaterno, int edad, int numCuenta) {
        this.nombre = nombre;
        this.apellidoPaterno = apellidoPaterno;
        this.apellidoMaterno = apellidoMaterno;
        this.edad = edad;
        this.numCuenta = numCuenta;
    }

    @Override
    public String toString() {
        return "Nombre: " + nombre + ", Apellido Paterno: " + apellidoPaterno +
               ", Apellido Materno: " + apellidoMaterno + ", Edad: " + edad +
               ", Número de Cuenta: " + numCuenta;
    }
}
```

Figura 2. Creación de la clase Alumno.

B. Main.

Primero, se crea un archivo llamado "archivo.txt" si no existe.Luego, se solicita al usuario que ingrese texto desde el teclado, se escribe ese texto en el archivo "archivo.txt" y se lee para mostrarlo por consola.

```
System.out.println(x: "###########File###########");
File archivo = new File(pathname: "archivo.txt");
System.out.println("Archivo existe: " + archivo.exists());
if(!archivo.exists()){
    try {
        boolean seCreo = archivo.createNewFile();
        System.out.println("Se creó un archivo: " + seCreo);
        System.out.println("Archivo existe: " + archivo.exists());
    } catch (IOException ex) {
        System.out.println(x: ex.getMessage());
    }
}
```

Figura 2. Creación del archivo.

En esta imagen se muestra la lectura y escritura en el archivo anteriormente creado.

```
System.out.println(x: "##########FileWriter/Reader##########");

try{
    System.out.println(x: "#####Lectura desde teclado#####");
    BufferedReader br = new BufferedReader(new InputStreamReader(in: System.in));
    System.out.println(x: "Escriba el texto para el archivo.");
    String texto = br.readLine();
    System.out.println("El usuario escribió: " + texto);

    System.out.println(x: "#####Escritura del archivo#####");
    FileWriter fw = new FileWriter(fileName: "archivo.txt");
    BufferedWriter bw = new BufferedWriter(out:fw);
    PrintWriter impresoraDeArchivos = new PrintWriter(out:bw);
    impresoraDeArchivos.println(x: texto);

    impresoraDeArchivos.close();

    System.out.println(x: "#####Lectura del archivo#####");
    FileReader fr = new FileReader(fileName: "archivo.txt");
    br = new BufferedReader(in: fr);
    System.out.println(x: "El texto del archivo es: ");
    String linea = br.readLine();
    while(linea != null){
        System.out.println(x: linea);
        linea = br.readLine();
    }
    br.close();
```

Figura 3. Lectura y escritura del archivo.

A continuación se muestra la ejecución del mismo:

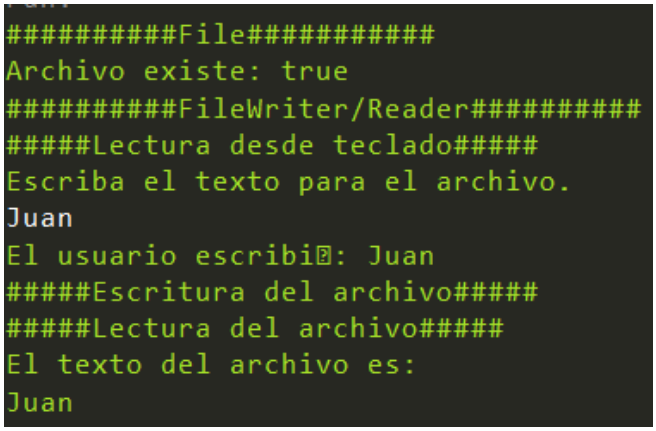

Figura 4. Ejecución del programa creación de "archivo.txt", escritura y lectura del mismo.

Como siguiente actividad, se utiliza 'StringTokenizer' para dividir una cadena de texto que representa información de un alumno en campos separados por comas.

```
System.out.println("##########StringTokenizer##########");
String alumno = "Hector Juan,Jimenez,Maya,320342330,20,8.3";
System.out.println("Cadena a tokenizar: " + alumno);
StringTokenizer tokenizador = new StringTokenizer(alumno, ",");
while(tokenizador.hasMoreTokens()){
    System.out.println(tokenizador.nextToken());
}
```

Figura 5. Código de StringTokenizer.

La ejecución del código anterior sería la siguiente:

```
##########StringTokenizer##########
Cadena a tokenizar: Hector Juan,Jimenez,Maya,320342330,20,8.3
Hector Juan
Jimenez
Maya
320342330
20
8.3
```

Figura 6. Ejecución del código de StringTokenizer.

C. Actividad extra.

Posteriormente, se realiza una actividad extra donde se escribe información ficticia de cinco alumnos en un archivo CSV ("alumnos.csv") y se lee este archivo, se tokeniza cada línea para crear objetos `Alumno`, los cuales se almacenan en una lista y se imprimen por consola.

```
System.out.println("##########Actividad Extra##########");
try {
    System.out.println("#####Escritura del archivo#####");
    FileWriter fw = new FileWriter("alumnos.csv");
    BufferedWriter bw = new BufferedWriter(fw);
    PrintWriter impresoraDeArchivos = new PrintWriter(bw);

    // Escribir 5 objetos Alumno en el archivo
    impresoraDeArchivos.println("Rodrigo,Sanchez,Perez,20,1234656,9.9,Copilco 300 Coyoacan");
    impresoraDeArchivos.println("Kevin Uriel,Jimenez,Tejeda,20,1234656,7.9,Copilco 300 Coyoacan");
    impresoraDeArchivos.println("Dhanesh Gael,Cruz,Manilla,20,1234656,6.9,Copilco 300 Coyoacan");
    impresoraDeArchivos.println("Jose David,Peña,Arredondo,20,1234656,9.9,Copilco 300 Coyoacan");
    impresoraDeArchivos.println("Tyler,Durden,Gonzalez,25,1234656,10,Copilco 300 Coyoacan");

    impresoraDeArchivos.close();

    System.out.println("#####Lectura del archivo#####");
    FileReader fr = new FileReader("alumnos.csv");
    BufferedReader br = new BufferedReader(fr);
    System.out.println("El texto del archivo es: ");
```

Figura 7. Código Parte1. de la actividad extra.

```
List<Alumno> listaAlumnos = new ArrayList<>();

String linea = br.readLine();
while (linea != null) {
    System.out.println(linea);
    // Tokenizar la línea y crear un objeto Alumno
    StringTokenizer tokenizador = new StringTokenizer(linea, ",");
    String nombre = tokenizador.nextToken();
    String apellidoPaterno = tokenizador.nextToken();
    String apellidoMaterno = tokenizador.nextToken();
    int edad = Integer.parseInt(tokenizador.nextToken());
    int numCuenta = Integer.parseInt(tokenizador.nextToken());
    double promedio = Double.parseDouble(tokenizador.nextToken());
    String direccion = tokenizador.nextToken();

    // Crear objeto Alumno y agregarlo a la lista
    Alumno alumno = new Alumno(nombre, apellidoPaterno, apellidoMaterno, edad, numCuenta, promedio, direccion);
    listaAlumnos.add(alumno);

    // Leer la siguiente línea
    linea = br.readLine();
}
br.close();

// Imprimir la información de los objetos Alumno en la consola
for (Alumno alumno : listaAlumnos) {
    System.out.println(alumno);
}
```

Figura 8. Código Parte 2. de la actividad extra.

La ejecución de esta actividad extra sería la siguiente:

```
##########Actividad Extra##########
#####Escritura del archivo#####
#####Lectura del archivo#####
El texto del archivo es:
Rodrigo,Sanchez,Perez,20,1234656,9.9,Copilco 300 Coyoacan
Kevin Uriel,Jimenez,Tejeda,20,1234656,7.9,Copilco 300 Coyoacan
Dhanesh Gael,Cruz,Manilla,20,1234656,6.9,Copilco 300 Coyoacan
Jose David,Pe?a,Arredondo,20,1234656,9.9,Copilco 300 Coyoacan
Tyler,Durden,Gonzalez,25,1234656,10,Copilco 300 Coyoacan
Nombre: Rodrigo, Apellido Paterno: Sanchez, Apellido Materno: Perez, Edad: 20, N?mero de Cuenta: 1234656
Nombre: Kevin Uriel, Apellido Paterno: Jimenez, Apellido Materno: Tejeda, Edad: 20, N?mero de Cuenta: 1234656
Nombre: Dhanesh Gael, Apellido Paterno: Cruz, Apellido Materno: Manilla, Edad: 20, N?mero de Cuenta: 1234656
Nombre: Jose David, Apellido Paterno: Pe?a, Apellido Materno: Arredondo, Edad: 20, N?mero de Cuenta: 1234656
Nombre: Tyler, Apellido Paterno: Durden, Apellido Materno: Gonzalez, Edad: 25, N?mero de Cuenta: 1234656
```

Figura 9. Ejecución del código de la actividad extra.

D. Resumen:

- El código interactúa con archivos de texto para escribir y leer información.
- Define una clase `Alumno` para representar los datos de un estudiante.
- Utiliza `StringTokenizer` para manejar datos separados por comas en formato CSV.
- Realiza la escritura y lectura de información de alumnos en archivos de texto.

III. CONCLUSIONES.

Mendoza Valdez Emily Isabela:

El manejo de archivos en Java es fundamental para la lectura, escritura y manipulación de datos. El código presentado demuestra cómo crear, escribir y leer información en archivos de texto. Utilizando clases como `FileWriter`, `BufferedWriter` y `FileReader`, se gestionan flujos de datos para almacenar y recuperar información, mientras que el uso de la clase `Alumno` permite estructurar datos complejos para su manipulación en archivos. La implementación de `StringTokenizer` facilita la separación y procesamiento de datos estructurados, como los registros de estudiantes en formato CSV, ofreciendo flexibilidad en la gestión de información.

Velarde Valencia Josue:

El manejo de archivos en Java es esencial para gestionar información de manera efectiva. El código proporcionado ilustra cómo crear, escribir y leer datos en archivos de texto, utilizando clases como `File`, `FileWriter`, `BufferedWriter`, `FileReader` y `BufferedReader`. Además, al estructurar la información de los alumnos mediante la clase `Alumno`, se facilita la manipulación y el almacenamiento de datos complejos. El uso de `StringTokenizer` permite procesar datos estructurados, como los registros de los estudiantes en formato CSV, mejorando la capacidad de trabajar con información organizada en campos. Esta combinación de herramientas ofrece flexibilidad y control en la gestión de datos en aplicaciones Java.

IV. LINKS DE GITHUB.

https://github.com/Josue-Velarde-Valencia/POOP11

https://josue-velarde-valencia.github.io/POOP11/