Algoritmos y Estructura de Datos

ARCHIVOS

DOCENTE: MARX DANLY LEÓN TRUJILLO
FACULTAD DE ING. INDUSTRIAL Y SISTEMAS
E.P. INGENIERÍA DE SISTEMAS

ARCHIVOS

En Java, los archivos se manejan principalmente utilizando las clases y APIs proporcionadas por el **paquete java.io** y el **paquete java.nio** en versiones más recientes de Java. Estos paquetes permiten la ***creación, lectura, escritura y manipulación de archivos*** en el sistema de archivos local.

ARCHIVOS

Lectura y Escritura de Archivos en Java 8 y versiones posteriores:

Java 8 introdujo la API **java.nio** que ofrece una forma más moderna y eficiente de trabajar con archivos utilizando las clases ***Path, Files y BufferedReader / BufferedWriter.***

ARCHIVOS

```
import java.io.*;

public class ArchivoEjemplo {
    public static void main(String[] args) {
        // Ruta del archivo que quieres leer o escribir
        String rutaArchivo = "miarchivo.txt";

        try {
            // Escritura en un archivo
            FileWriter escritor = new FileWriter(rutaArchivo);
            escritor.write("Hola, mundo!");
            escritor.close();

            // Lectura de un archivo
            FileReader lector = new FileReader(rutaArchivo);
            int c;
            while ((c = lector.read()) != -1) {
                System.out.print((char) c);
            }
            lector.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

ARCHIVOS

```
import java.io.*;
import java.nio.file.*;

public class ArchivoEjemplo {
    public static void main(String[] args) {
        // Ruta del archivo que quieres leer o escribir
        String rutaArchivo = "miarchivo.txt";

        try {
            // Escritura en un archivo
            String contenido = "Hola, mundo!";
            Files.write(Paths.get(rutaArchivo), contenido.getBytes());

            // Lectura de un archivo
            String texto = new String(Files.readAllBytes(Paths.get(rutaArchivo)));
            System.out.println(texto);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ARCHIVOS

Operaciones avanzadas de archivos:

- Se puede definir si el archivo existe, se puede leer o escribir.

```
public class LeerArchivos {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        //MANEJO DE ARCHIVOS
        //1. Crear una instancia de la clase File
        File archivo = new File("archivo.txt");

        //Verificar si el archivo existe
        System.out.println("El archivo existe: " + archivo.exists());
        System.out.println("¿Se puede Leer? " + archivo.canRead());
        System.out.println("¿Se puede escribir? " + archivo.canWrite());

    }

}
```

ARCHIVOS

- Se puede definir si el directorio existe, se puede leer o escribir.

```
public class LeerArchivos {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        //MANEJO DE ARCHIVOS
        //MANEJO DE DIRECTORIOS
        File dir = new File("D:\\CARPETA DOCENTE\\UNHEVAL\\2023\\6. ALGORITMOS Y ESTRUCTURA DE DATOS\\Semana 5\\Ejercicios");
        //Verificar si el archivo existe
        System.out.println("El directtorio existe: " + dir.exists());
        System.out.println("¿Se puede Leer? " + dir.canRead());
        System.out.println("¿Se puede escribir? " + dir.canWrite());

    }

}
```

ARCHIVOS

- Crear directorios

```
//CREAR DE DIRECTORIOS
File dirl = new File("D:\\CARPETA DOCENTE\\UNHEVAL\\2023\\6. ALGORITMOS Y ESTRUCTURA DE DATOS\\Semana 5\\Ejercicios1");
boolean crearDir = dirl.mkdir();
System.out.println("El directorio :"+dirl+" ha sido creado con exito");
```

```
File archivo3 = new File("C:\\uskokrum2010\\carpeta1\\carpeta2\\carpeta3");

System.out.println(archivo3.mkdirs());
```

ARCHIVOS

- Crear Archivos y escribir archivos

```
import java.io.FileWriter;
import java.io.IOException;

public class Escribir {

    public static void main(String[] args) throws IOException {

        FileWriter fichero = new FileWriter ("C:/Users/Public/Documents/fichero.txt");
        for (int x= 0; x<10;x++){
            fichero.write("Fila numero " + x + "\n");
        }
        fichero.close();
    }

}
```

ARCHIVOS

- Copiar, mover o eliminar archivos: Utiliza las clases Files y métodos como copy, move y delete.

```
try {

    String ruta = "C:\\Users\\manuj\\Desktop\\hola.txt";
    Path fuente = Paths.get(ruta);
    Path destino = Paths.get("C:\\Users\\manuj\\Desktop\\LOLequisdejaja.txt");


    Files.copy(fuente, destino, StandardCopyOption.REPLACE_EXISTING);

} catch (Exception e) {
    System.out.println("LOOOL");
}
```

ARCHIVOS

BufferedReader y **BufferedWriter** son clases en Java que proporcionan una forma eficiente de leer y escribir datos en archivos, especialmente cuando se trata de archivos grandes. Estas clases funcionan en conjunto con otras clases de manejo de archivos para mejorar el rendimiento de las operaciones de lectura y escritura.

BufferedReader: Esta clase se utiliza para leer texto desde un archivo de manera eficiente. Al usar un búfer interno, reduce el número de operaciones de lectura en disco, lo que mejora el rendimiento. Aquí hay un ejemplo de cómo usar BufferedReader:

ARCHIVOS

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

public class EjemploBufferedReader {
    public static void main(String[] args) {
        String rutaArchivo = "archivo.txt";

        try (BufferedReader br = new BufferedReader(new
FileReader(rutaArchivo))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ARCHIVOS

BufferedWriter: Esta clase se utiliza para escribir texto en un archivo de manera eficiente. Al igual que BufferedReader, utiliza un búfer interno para reducir el número de operaciones de escritura en disco. Aquí hay un ejemplo de cómo usar BufferedWriter:

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class EjemploBufferedWriter {
    public static void main(String[] args) {
        String rutaArchivo = "archivo.txt";

        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(rutaArchivo))) {
            String texto = "Hola, mundo!\nEsta es otra
línea.";
            bw.write(texto);
            System.out.println("Datos escritos con éxito.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ARCHIVOS

En ambos ejemplos, se utiliza un bloque try-with-resources para garantizar que los recursos se cierren correctamente después de su uso. Esto es importante para evitar pérdida de datos o problemas de manejo de recursos.

Tanto **BufferedReader** como **BufferedWriter** son útiles al trabajar con archivos de texto en Java, ya que mejoran significativamente la eficiencia de las operaciones de lectura y escritura, especialmente cuando se trata de grandes volúmenes de datos.