índice

A partir, del clase de análisis para flujos de datos, IMPORTANTE para examen lo demás tipo test

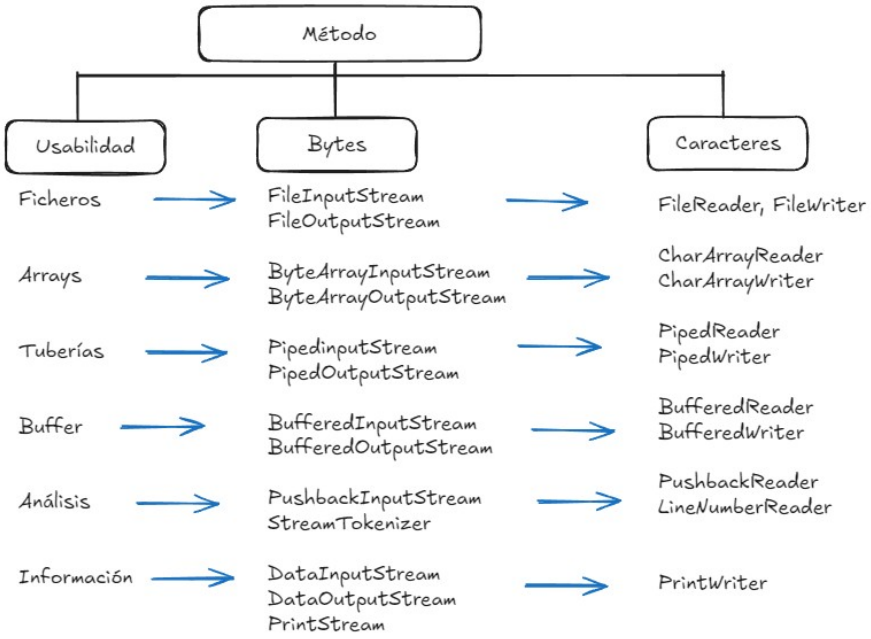
Tema 2 Flujos

Definición y tipos de ficheros

Flujo de datos o Streams: Un flujo de datos es una secuencia ordenada de información que se divide en dos direcciones:

- **Flujo de entrada:** Para leer datos.
- **Flujo de salida:** Para escribir datos.

Los streams son unidireccionales; es decir, un flujo puede ser utilizado únicamente para leer o escribir, pero no para ambas operaciones simultáneamente.



Clases de flujos basados en tuberías

Las tuberías en Java: Tuberías en Java

Las tuberías permiten la comunicación entre **dos hilos (threads)** en la misma máquina virtual. Pueden actuar como orígenes o destinos de datos, facilitando la comunicación entre procesos que se ejecutan en diferentes ubicaciones.

- **Acceso basado en bytes:**
  - **`PipedInputStream`** en Java leer los datos de una tubería (pipe)
  - **`PipedOutputStream`** `escribir`
- **Acceso basado en caracteres:**
  - **`PipedReader`**
  - **`PipedWriter`**

Conectar la entrada y la salida PipedInputStream y PipedOutputStream

se usa el método y se usa con un try catch

- connect()

Thread son los hilos, las tuberías permiten la comunicación de dos hilos. Uno puede actuar de origen y otro de destino.

- **Métodos Thread**
  - write()
  - read()
  - start()
  - getBytes()

Ejemplo:

Thread1 podría escribir utilizando PipedOutputStream y try catch y entrada.**write()**

Threas2 leer con PipedInputStream try catch y salida.**read()**

```
//Crea un programa Java que contenga dos hilos: Thread1 y Thread2.
        /*La clase PipedInputStream en Java  leer los datos de una tubería (pipe).
       establecer comunicación entre dos threads a través de una tubería. */
        final PipedInputStream entrada = new PipedInputStream();//leer
        final PipedOutputStream salida = new PipedOutputStream();//escribir

        /*conectar la entrada y la salida */
        try {
            entrada.connect(salida);

        } catch (Exception e) {
            e.printStackTrace();
        }


        /*Thread1 debe escribir una cadena de texto en una tubería utilizando PipedOutputStream.
        La cadena de texto que debes escribir es: "¡Hola, desde el hilo Thread1!". */

        Thread thread1 = new Thread(new Runnable() { //creamos un hilo

            public void run(){

                try {
                    salida.write("¡Hola, desde el hilo Thread1!".getBytes());

                } catch (Exception e) {

                }
            }
        });


        /*Thread2 debe leer los datos de la tubería utilizando PipedInputStream
        y los imprimirá en la consola hasta que alcance el final de la tubería (-1 indica el final). Asegúrate de que Thread2 imprima los datos como caracteres en la consola. */
        Thread thread2 = new Thread(new Runnable() { //creamos otro hilo

            public void run(){

                try {

                    int unByte = entrada.read();
                    while (unByte != -1) {
                        System.out.print((char) unByte);
                        unByte = entrada.read();
                    }


                } catch (Exception e) {

                }
            }
        });

        //Inicia la ejecución de ambos hilos para poder comunicarse a través de la tubería.
        thread1.start();
        thread2.start();


    }
}
```

Flujos basados en arrays

Clases para Flujos de Arrays

Estas clases permiten manejar flujos de datos que residen en arrays en memoria. **El objetivo es trabajar con flujos de caracteres sin utilizar archivos**

- **Acceso basado en bytes:**
  - `ByteArrayInputStream`
  - `ByteArrayOutputStream`
- **Acceso basado en caracteres:**
  - `CharArrayReader`
  - `CharArrayWriter`

```java
/* Ejemplo 2.2: Manipulación de Texto con CharArrayWriter y CharArrayReader

Objetivo:
Aprender a usar las clases CharArrayWriter y CharArrayReader en Java para escribir, modificar, y leer datos de texto en memoria.
El objetivo es trabajar con flujos de caracteres sin utilizar archivos, simulando operaciones de entrada y salida.

Instrucciones:
Crea un programa en Java que utilice las clases CharArrayWriter y CharArrayReader.
Escribe una cadena de texto en el CharArrayWriter, conviértela a un array de caracteres y modifícala. Por ejemplo, escribe la cadena "Hola amigo",
 modifica el array cambiando "amigo" por "amiga". Deberás cambiar el carácter en la posición 9 del array ('o') por 'a'.
Usa el CharArrayReader para leer los caracteres modificados y mostrar el resultado en pantalla. */

import java.io.CharArrayReader;
import java.io.CharArrayWriter;

public class Ej_2_2_CharArrayWriter_reader {
    public static void main(String[] args) {

        // Crear un CharArrayWriter y escribir la cadena "Hola amigo"
        CharArrayWriter writer = new CharArrayWriter();

        //Escribir
        try {
            writer.write("Hola amigo");
        } catch (Exception e) {

        }

        // Convertir el contenido de CharArrayWriter a un array de caracteres
        char[] caracteres = writer.toCharArray();

        // Modificar el array cambiando 'o' por 'a' en la posición 9
        caracteres[9] = 'a'; // Cambiamos 'o' por 'a'

        // Crear un CharArrayReader para leer los caracteres modificados
        CharArrayReader reader = new CharArrayReader(caracteres);

        // Leer y mostrar el resultado en pantalla
        int data = 0;
        try {
            data = reader.read();
            while (data != -1) {
                System.out.print((char) data); // Mostrar cada carácter
                data = reader.read();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // Cerrar los streams
            try {
                reader.close();
                writer.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Clases de análisis para flujos de datos I

`PushbackInputStream y PushbackReader` se usan para el análisis de datos previo de un InputStream. A veces se necesita leer algunos bytes de forma anticipada para ver que se aproxima. **Para leer caracteres de un archivo y devolverlos al flujo para ser leídos nuevamente con la posibilidad de "desempujar" caracteres para volver a procesarlos.**

- **Acceso basado en bytes:**
  - `PushbackInputStream`
  - `StreamTokenizer`
- **Acceso basado en caracteres:**
  - `PushbackReader`
  - `LineNumberReader`

```java
/* Ejemplo 2.3: Manipulación de Flujos de Entrada con PushbackReader

Objetivo:
Aprender a utilizar la clase PushbackReader en Java para leer caracteres de un archivo y devolverlos al flujo para ser leídos nuevamente.
Esta práctica permitirá entender cómo funcionan los flujos de entrada
con la posibilidad de "desempujar" caracteres para volver a procesarlos.

Instrucciones:
Crea un programa en Java que utilice la clase PushbackReader para leer el contenido de un archivo.
Lee cada carácter del archivo, imprímelo, y utiliza la funcionalidad de "desempujar" el carácter para leerlo de nuevo.
Asegúrate de que los caracteres se leen e imprimen dos veces utilizando la función unread() de PushbackReader.
Sigue los pasos detallados en el temario. */

import java.io.*;

public class Ej_2_3_PushbackReader {

    public static void main(String[] args) {
        String rutaArchivo = "G:\\Mi unidad\\2º-DAM\\MODULOS\\Acceso a datos\\T1-T2\\Tema_2_Flujos\\texto2_3.txt";

        try (PushbackReader reader = new PushbackReader(new FileReader(rutaArchivo))) {
            int caracter;
            while ((caracter = reader.read()) != -1) {
                // Imprimir el carácter leído
                System.out.print((char) caracter);

                // "Desempujar" el carácter para leerlo nuevamente
                reader.unread(caracter);

                // Leer nuevamente el carácter
                caracter = reader.read();
                System.out.print((char) caracter); // Imprimir de nuevo el carácter
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Clases de análisis para flujos de datos II

`StreamTokenizer`

Para procesar el contenido de un archivo, identificando palabras y números en el texto

Esta clase permite descomponer un flujo de caracteres en tokens, utilizando un `StringReader`. Los tipos de tokens son:

- **TT_EOF:** final del fichero
- **TT_EOL:** final de linea
- **TT_WORD:** que es el token es de tipo de palabra
- **TT_NUMBER:** indica si el token es un numero

```java

/* Ejemplo 2.4: Procesamiento de Texto con StreamTokenizer en Java

Objetivo:
Aprender a utilizar la clase StreamTokenizer en Java para procesar el contenido de un archivo,
identificando palabras y números en el texto. Esta práctica permitirá a los estudiantes trabajar
con la tokenización de texto, reconociendo palabras, números y opcionalmente fin de línea.

Instrucciones:
Crea un programa en Java que utilice la clase StreamTokenizer para leer un archivo de texto y procesar su contenido.
El programa debe distinguir entre palabras y números en el archivo, imprimiendo cada token que se encuentra.
Opcionalmente, configura el programa para que también identifique el fin de línea y lo imprima. */
import java.io.FileReader;
import java.io.StreamTokenizer;
import java.io.FileReader;

public class Ej_2_4_StreamTokenizer {
    public static void main(String[] args) {


        try {
            StreamTokenizer st = new StreamTokenizer(new FileReader("G:\\Mi unidad\\2º-DAM\\MODULOS\\Acceso a datos\\Tema_2_Flujos\\datosE4.txt"));

            //Para que funcione el TT_EOL (fin de linea):
            st.eolIsSignificant(true);

            while(st.nextToken() != StreamTokenizer.TT_EOF){ //leyendo hasta el final del fichero

                //Muestra solo las palabras del archivo
                if(st.ttype == StreamTokenizer.TT_WORD){ //si el tipo es string (una palabra)
                    System.out.println("Palabra -" + st.sval);


                    //Muestra los numeros del archivo
                }else if(st.ttype == StreamTokenizer.TT_NUMBER){
                    System.out.println("Numero -" + st.nval);

                    //Fin de línea
                }else if(st.ttype == StreamTokenizer.TT_EOL){
                    System.out.println("_______FIN DE LINEA_______");

                }
            }


        }catch(

    Exception e)
    {

    }

}}

```

Clases de análisis para flujos de datos III

`LineNumberReader`

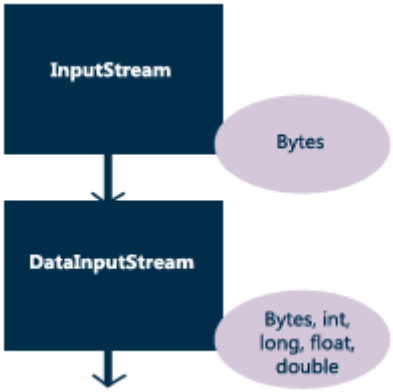
Para leer el contenido de un archivo línea por línea, mostrando el número de cada línea junto con su contenido.

Esta clase permite contar y almacenar el número de líneas leídas en un flujo de caracteres. Métodos clave:

- `getLineNumber()`: Devuelve el número de la línea actual.
- `setLineNumber(int line)`: Establece el número de línea actual.

```java
/* Ejemplo 2.5: Lectura de Archivos con LineNumberReader en Java
Objetivo:
Aprender a utilizar la clase LineNumberReader en Java para leer el contenido de un archivo
línea por línea, mostrando el número de cada línea junto con su contenido.
Esta práctica permitirá a los estudiantes trabajar con la manipulación de archivos y
el control de líneas utilizando la clase LineNumberReader.

Instrucciones:
Crea un programa en Java que utilice la clase LineNumberReader para leer un archivo llamado entrada.txt.
Para cada línea del archivo, muestra el número de línea y el contenido correspondiente.
Asegúrate de cerrar los recursos después de la lectura del archivo. */

import java.io.LineNumberReader;
import java.io.*;

public class Ej_2_5_LineNumberReader {

    public static void main(String[] args) {

        //Leer por lineas
        try {
            LineNumberReader lnr = new LineNumberReader(new FileReader("Tema_2_Flujos/datosE5.txt"));
            String line = lnr.readLine();

            while(line != null){
                System.out.println("Contenido de la linea nº: " + lnr.getLineNumber());
                System.out.println(line);
                line = lnr.readLine();
            }
            lnr.close();

        } catch (Exception e) {
            // TODO: handle exception
        }
    }

}

```

Clases para el tratamiento de información I



Clases para el tratamiento de información II

Para leer ficheros utilizando `DataInputStream,` primero es necesario escribir en el fichero de forma ordenada **y** conocer los tipos de datos. Para esto, **se utiliza** `DataOutputStream.`

- **Instanciación:** Se crea un `DataOutputStream` asociado a un `FileOutputStream`, que recibe la ruta del fichero a escribir.
- **Métodos de escritura:**
  - `writeInt(int value)`: Escribe un número entero.
  - `writeFloat(float value)`: Escribe un número decimal.
  - `writeLong(long value)`: Escribe un número largo.

Ejercicio Ejemplo combinando todo

```java
import java.io.*;

/* Ejemplo 2.6: Extracción de datos usando clases de análisis para flujos de datos

Objetivo: El objetivo de esta actividad es que los estudiantes practiquen el análisis de flujo de datos en Java
utilizando las clases vistas hasta ahora ( LineNumberReader, StreamTokenizer y PushbackReader, etc). En este ejercicio,
analizarán un archivo de texto y contarán cuántas palabras y cuántos números hay en cada línea del archivo.

Instrucciones:

   1. Desde un archivo de texto llamado "datosE6.txt" que contiene líneas con una combinación de palabras y números. Por ejemplo:

   Hola 123 Mundo
   Prueba 45.67 de texto
   12345

   2. Cuenta cuántas palabras y cuántos números hay en cada línea. Muestra el resultado en la terminal con el siguiente formato de ejemplo:
   Línea 1: Hola 123 Mundo
   Palabras: 2, Números: 1
   Línea 2: Prueba 45.67 de texto
   Palabras: 3, Números: 1
   Línea 3: 12345
   Palabras: 0, Números: 1
 */

import java.io.LineNumberReader;

public class Ejercicio_2_6 {

  public static void main(String[] args) {

    //Variables
      String archivo = "G:\\Mi unidad\\2º-DAM\\MODULOS\\Acceso a datos\\Tema_2_Flujos\\datosE6.txt";
      String linea;

      //Bucle leer linea por linea
      try (LineNumberReader lineReader = new LineNumberReader(new FileReader(archivo))) {

          while ((linea = lineReader.readLine()) != null) {
              contarPalabrasYNumeros(linea, lineReader.getLineNumber()); //metodo dnde introducimos la linea
          }
      } catch (Exception e) {
          e.printStackTrace();
      }
  }

  //Método contar palabras
  private static void contarPalabrasYNumeros(String linea, int numeroLinea) {
      StringReader stringReader = new StringReader(linea);
      StreamTokenizer tokenizer = new StreamTokenizer(stringReader);

      int palabras = 0;
      int numeros = 0;

      try {
          while (tokenizer.nextToken() != StreamTokenizer.TT_EOF) {
              if (tokenizer.ttype == StreamTokenizer.TT_WORD) {
                  palabras++;
              } else if (tokenizer.ttype == StreamTokenizer.TT_NUMBER) {
                  numeros++;
              }
          }
      } catch (Exception e) {
          e.printStackTrace();
      }

      // Imprimir el resultado
      System.out.println("Línea " + numeroLinea + ": " + linea);
      System.out.println("Palabras: " + palabras + ", Números: " + numeros);
  }
}
```