Manejo de archivos de texto en Java

Programación Interactiva

LANZAR EXCEPCIONES

- Los métodos usan la instrucción ***throws*** para lanzar una excepción. La instrucción ***throws*** requiere un único argumento: un objeto lanzable (throwable).

```
public static double division(int x, int y) throws ArithmeticException{


    if (y==0){
        throw new ArithmeticException("Division por cero");
    }

    return x/y;

}
```

LANZAR EXCEPCIONES

- Otro Ejemplo:

```java
public double div(double num1, double num2) throws ArithmeticException
{
    double resultado;

    resultado = num1/num2;

    if(Double.POSITIVE_INFINITY == resultado)
    {
        throw new ArithmeticException("Error infinito en operación de división");
    }

   return resultado;
}
```

```java
if(e.getSource() == bDiv)
{
    try{
        tfResul.setText(""+op.div(num1, num2));

    }catch(ArithmeticException ae)
    {
        tfResul.setText(ae.getMessage());
    }


}
```

CREAR EXCEPCIONES

- Crear una clase que derive de la clase *Exception.*

- Esta **clase normalmente contiene un único constructor que tiene como parámetro un *String*** con un mensaje que se puede recuperar con el método getMessage() de la clase Exception.

- Esta clase **llama por medio de *super()*** al constructor de la clase padre.

CREAR EXCEPCIONES

```
public class MyException extends Exception {

    public MyException(String mensaje){
        super(mensaje);
    }
}
```

```
public void calificar (String curso, String cod,  double nota)
                        throws MyException {

        if (nota < 0.0  &&  nota > 5.0){
            throw  new MyException("La nota debe estar entre 0.0 y 5.0");
        }

        //Calificar....
    }
```

ARCHIVOS DE TEXTO EN JAVA

INTRODUCCION

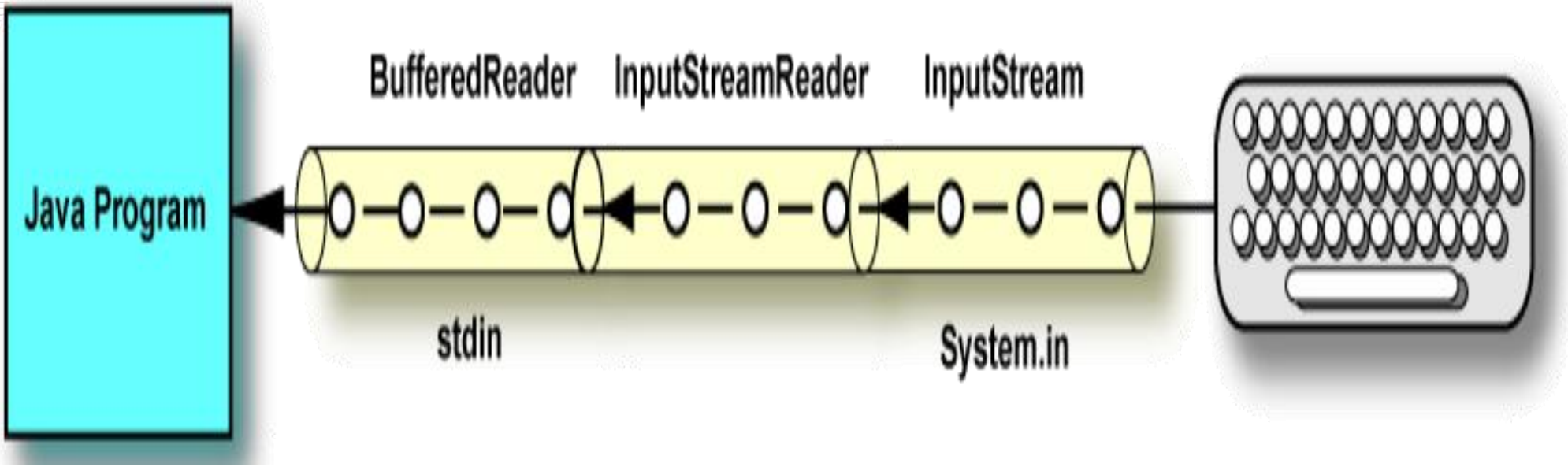
Los datos almacenados en variables y arreglos son temporales, estos se pierden cuando el programa finaliza. Para mantener estos datos de manera persistente se usan los ***archivos.***

Los archivos se almacenan en dispositivos de almacenamiento secundario como los discos duros, discos ópticos, memorias usb, etc.

Los programas necesitan comunicarse con su entorno, tanto para recoger datos e información que deben procesar, como para devolver los resultados obtenidos.

STREAMS (FLUJOS)

Un ***Streams*** es un medio utilizado para l**eer datos de una fuente y para escribir datos en un destino**. Tanto la fuente como el destino pueden ser archivos, sockets, memoria, cadena de caracteres, y también procesos.



JAVA STREAMS (FLUJO)

Los Streams se caracterizan por ser ***unidireccionales***, es decir que un Stream se utilizará solo para leer, solo para escribir, pero no ambas acciones al mismo tiempo.

Para utilizar un Stream, el programa deberá construir el Stream relacionándolo directamente con una fuente o con un destino, dependiendo si se necesita leer o escribir información.

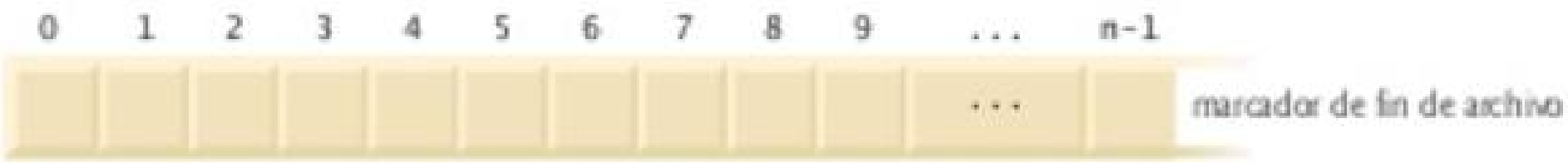
JAVA STREAMS (FLUJO)

La acción de leer información de una fuente es conocida también como *input*, y la acción de escribir información en un destino es conocida como *output*.

Dentro de Java, todas las clases utilizadas tanto para el input como para el output están incluidas en el paquete *Java.io*

JAVA STREAMS (FLUJO)

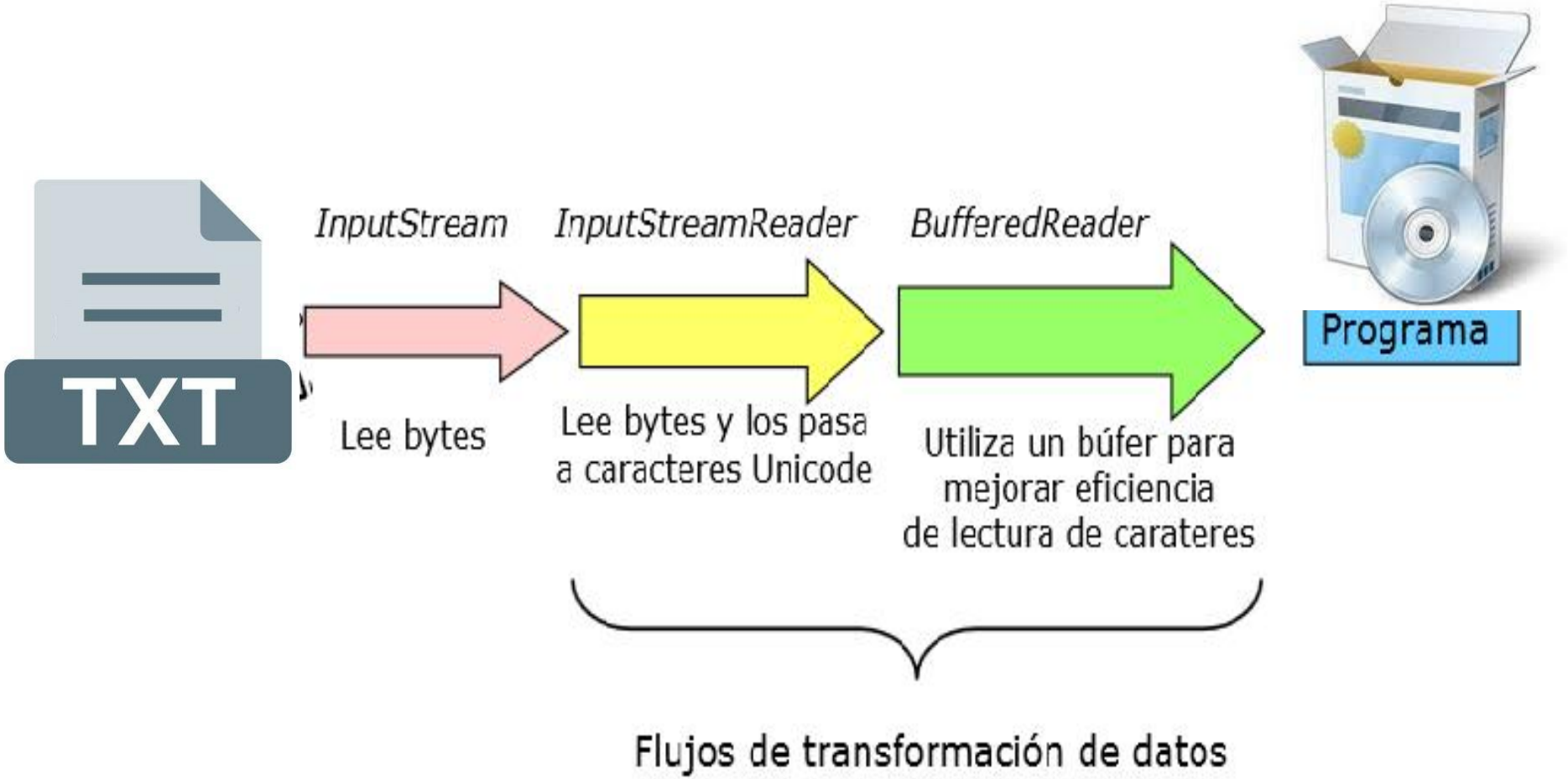
Java considera cada archivo como un **flujo de bytes secuencial**, en el que cada Sistema Operativo proporciona un mecanismo para determinar el fin del archivo como un **marcador de fin de archivo**.



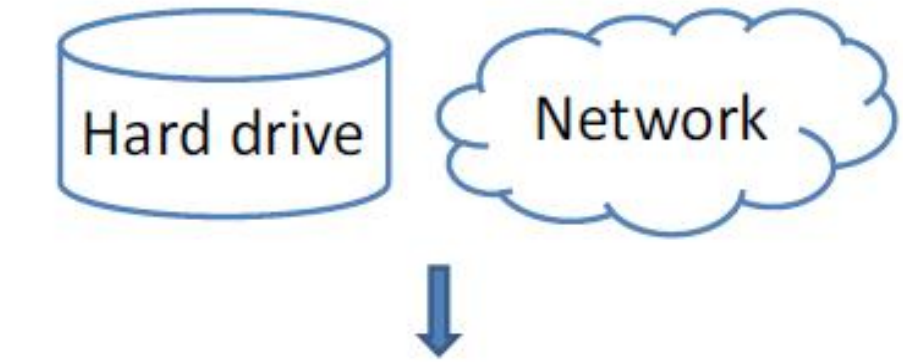
Java distingue 2 tipos de flujos:

- Flujos basados en bytes - (5 – 101)
- Flujos basados en caracteres. - (5 – 00000000 00110101 -- (53))

FLUJO DE DATOS EN JAVA



FLUJO DE DATOS EN JAVA



1001010101010001010 ...

InputStream
System.in

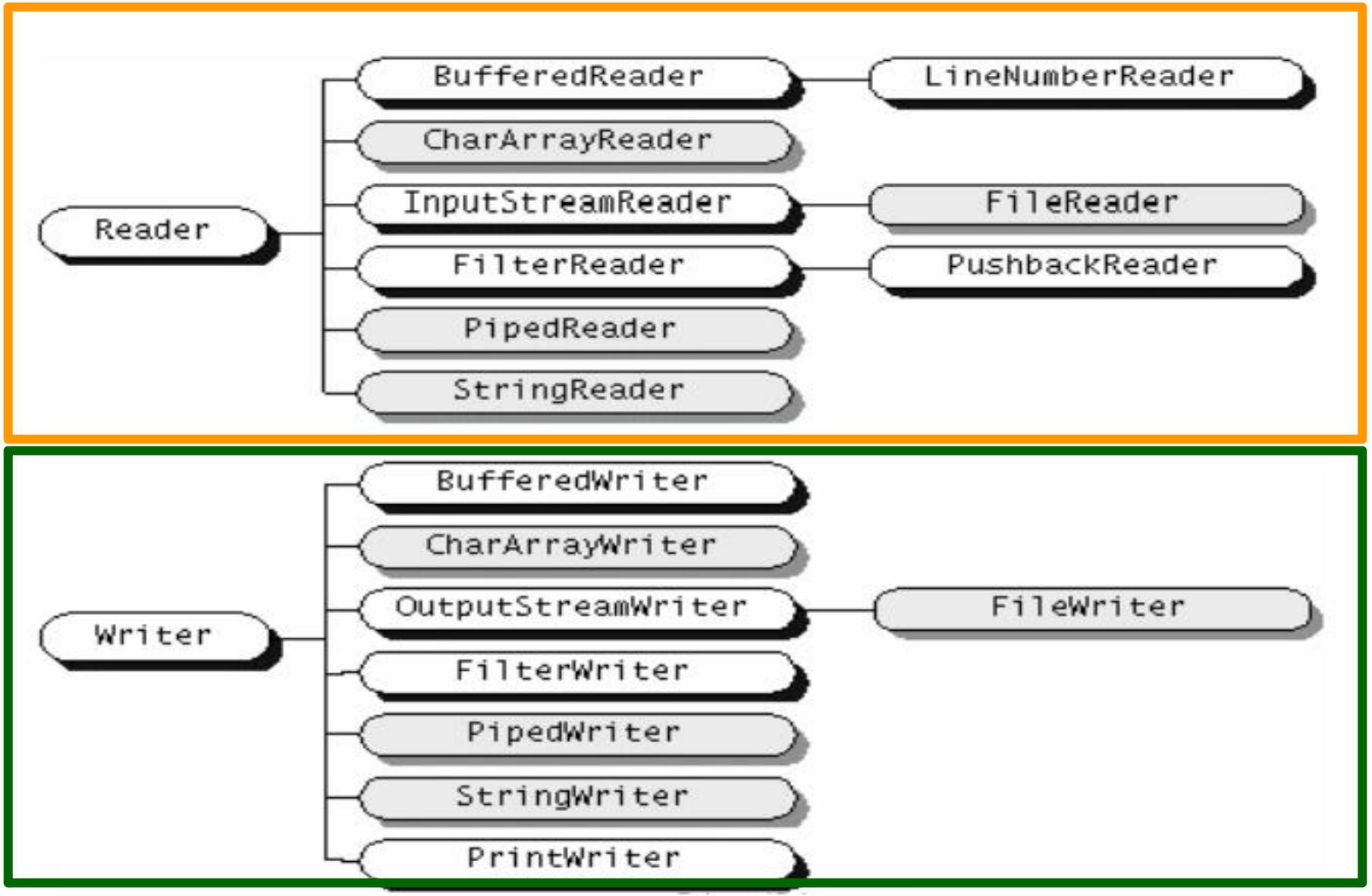
'O' 'k' 'a' 'y' ' ' 'a' 'w' 'e' ...

InputStreamReader

"Okay awesome, cool\n" ...

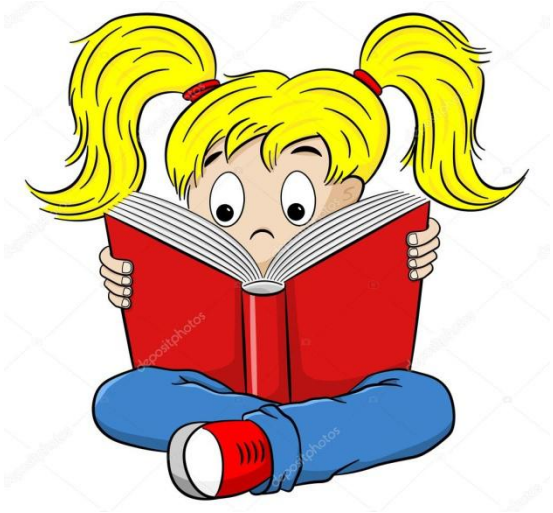
BufferedReader

PAQUETE JAVA IO* (Caracteres)



Archvios secuenciales de texto

- **Leer Archivos**

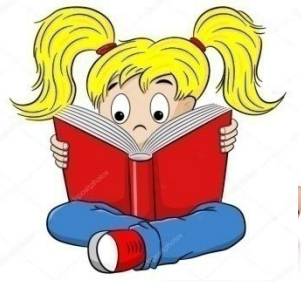


  - FileReader
  - BufferedReader
  - Scanner

- **Escribir archivos**

  - FileWriter
  - PrintWriter

LEER UN ARCHIVO

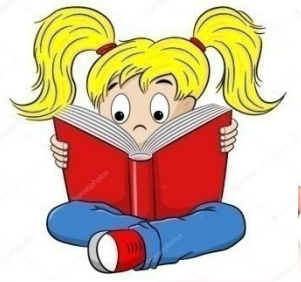


Class FileReader

java.lang.Object
    java.io.Reader
        java.io.InputStreamReader
            java.io.FileReader

```
FileReader fr = new FileReader("D:\\fichero1.txt");
```

LEER UN ARCHIVO



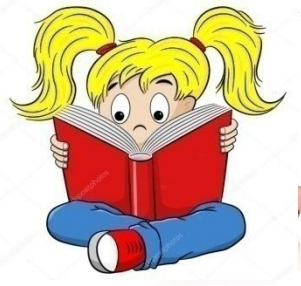
Class FileReader

java.lang.Object
    java.io.Reader
        java.io.InputStreamReader
            java.io.FileReader

```
FileReader fr = new FileReader("D:\\fichero1.txt");
```

```
File archivo = new File ("D:\\fichero1.txt");
FileReader fr = new FileReader (archivo);
```

BUFFER READER



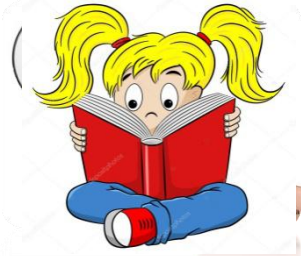
```
File archivo = new File (“prueba.txt");
FileReader  fr = new FileReader (archivo);
BufferedReader br = new BufferedReader(fr);

String linea;

while((linea = br.readLine())!=null){
   System.out.println(linea);
}
```

Ver LeerArchivoTextoFileRead.java

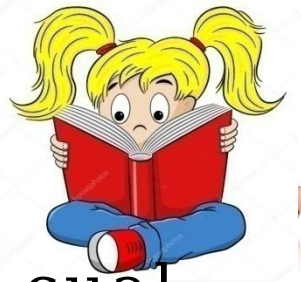
BUFFER READER



- Un buffer es un espacio de memoria intermedia que actúa de "colchón" de datos. Cuando se necesita un dato del disco se trae a memoria ese dato y sus datos contiguos, de modo que la siguiente vez que se necesite algo del disco la probabilidad de que esté ya en memoria sea muy alta.

- Algo similar se hace para escritura, intentando realizar en una sola operación de escritura física varias sentencias individuales de escritura.

La clase Scanner

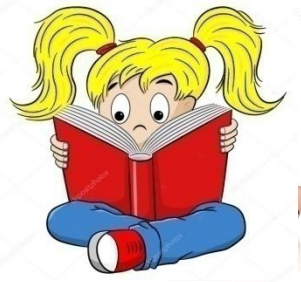


- Java tiene un método llamado **System.in**, el cual obtiene la información de usuario. Sin embargo, **Sytem.in** no es tan simple como **System.out**.

- La clase **Scanner** esta diseñada para **leer los bytes y convertirlos en valores primitivos** (int, double, bool, etc) o en valores String.

Método	Ejemplo
nextByte()	**byte** b = teclado.nextByte();
nextDouble()	**double** d = teclado.nextDouble();
nextFloat()	**float** f = teclado.nextFloat();
nextInt()	**int** i = teclado.nextInt();
nextLong()	**long** l = teclado.nextLong();
nextShort()	**short** s = teclado.nextShort();
next()	**String** p = teclado.next();
nextLine()	**String** o = teclado.nextLine();

La clase Scanner



```
public static void main(String[] args) {
    File f = new File("prueba.txt");
    String cadena;
    Scanner entrada = null;

    try {
        entrada = new Scanner(f);
        while (entrada.hasNext())
        {   cadena = entrada.nextLine();
            System.out.println(cadena);
        }
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    } finally{
        entrada.close();
    }
}
```

ESCRITURA DE ARCHIVOS

Class FileWriter

java.lang.Object
    java.io.Writer
        java.io.OutputStreamWriter
            java.io.FileWriter

Constructores

FileWriter (**File** file);
Construye un objeto para escribir en el archivo *file*

FileWriter (**File** file, **boolean** append);

Si ***append*** es ***false*** se abre el archivo como nuevo y si este ya existía ***se borra la información que tenga***. Si es ***true*** se abre el archivo de modo que **se pueda agregar información y mantener la actual**.

ESCRITURA DE ARCHIVOS

```
try
  {
            FileWriter archivo = new FileWriter("prueba.txt");
            PrintWriter pw = new PrintWriter(archivo);

           for (int i = 0; i < 10; i++){
               pw.println("Linea " + i);
           }

}
catch (IOException e)
{
           e.printStackTrace();
   }
```

Ver EscribirArchivoTexto.java

La clase Formatter

- Permite guardar archivos con cierto formato especificado por el programador

```
Formatter output;
output = new Formatter("file.txt")
```

```
output.format( "%d %s %s %2f\n" , field1.decimal,
field2.string, field3.string, field4.double)
```

GUARDANDO REGISTROS CON FORMAT

```
AccountRecord record = new AccountRecord();
   try
   {
      record.setAccount( 33 ); // read account number
      record.setFirstName( "Pedro" ); // read first name
      record.setLastName( "Perez" ); // read last name
      record.setBalance( 25 ); // read balance

      if ( record.getAccount() > 0 )
      {
         output.format( "%d %s %s %.2f\n", record.getAccount(),
                        record.getFirstName(), record.getLastName(),
                        record.getBalance() );
         //System.out.println(" format" + output.toString() + "\n" +

      } // end if
```

Entrada y Salida Estandar

La clase ***StringTokenizer*** se usa para dividir un String en *substrings* o *tokens*, con base a otro String (normalmente un carácter) separador entre ellos denominado **delimitador**.

```
import java.util.StringTokenizer;

Constructores:

StringTokenizer (String str)

StringTokenizer (String str, String delimitador)
```

Entrada y Salida Estandar

Métodos	Función que realizan
StringTokenizer(String)	Constructor a partir de la cadena que hay que separar
boolean hasMoreTokens()	¿Hay más palabras disponibles en la cadena?
String nextToken()	Devuelve el siguiente token de la cadena
int countTokens()	Devuelve el número de tokens que se pueden extraer de la frase

StringTokenizer: Ejemplo

```
public static void main(String[] args) {

    String str = "Este mensaje , se divide mediante  StringTokenizer, creado en PI, 2014";
    StringTokenizer st = new StringTokenizer(str);

    System.out.println("---- Divide por espacio ------");
    while (st.hasMoreTokens()) {
            System.out.println(st.nextElement());
    }

    System.out.println("\n---------- Divide por coma ',' ------");
    StringTokenizer st2 = new StringTokenizer(str, ",");

    while (st2.hasMoreTokens()) {
            System.out.println(st2.nextElement());
    }
  }//main
}//class
```

Ejercicio

- Construya un programa en Java que lea desde un archivo de texto y a través de un menú calcule y muestre lo siguiente:
  - La cantidad de palabras que tiene el archivo
  - La cantidad de líneas que tiene el archivo
  - La cantidad de veces que aparece una palabra específica introducida por el usuario en el archivo
  - Muestre las palabras que inicien con una letra específica introducida por el usuario

Clase JFileChooser

- La clase ***javax.swing.JFileChooser*** provee un mecanismo para usar un selector de archivos, que puede ser usado sólo o con un filtro para que sólo se visualicen los formatos definidos en el filtro.

```
JFileChooser fileChooser = new JFileChooser();
```

Ver FileChooserDemonstration.java

Clase JFileChooser

- Chooser de apertura de archivos

```
JFileChooser fileChooser = new JFileChooser();

fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES );

int result = fileChooser.showOpenDialog(frame);
```

- Chooser de apertura de archivos con filtro

```
JFileChooser fileChooser = new JFileChooser();
 FileNameExtensionFilter filtro = new FileNameExtensionFilter("JPG & GIF Images",
"jpg", "gif");
 fileChooser.setFileFilter(filtro);
 fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES );

int result = fileChooser.showOpenDialog( frame);
```

Ver FileFilterChooser.java