Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

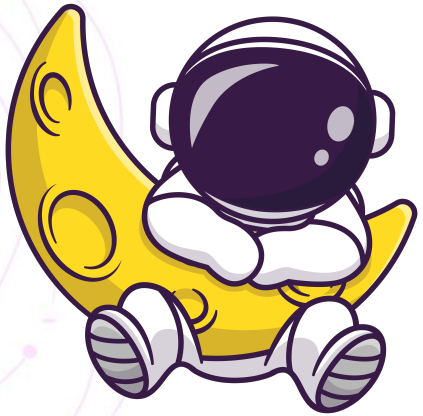
Unidad 4

10 - Entrada/Salida en Java

Hechos QUE CONECTAN

Persistencia en Java

- Se llama “persistencia” de los objetos a su capacidad para guardarse y recuperarse desde un medio de almacenamiento.
  - **Archivos**
  - Bases de Datos Relacionales
    - JDBC - Java DataBase Connectivity
    - JPA - Java Persistence API



Los flujos de datos de Java

- Prácticamente todos los programas deben leer datos del exterior para procesarlos y después presentar los resultados.
- La información que necesita un programa normalmente se obtiene mediante la entrada de datos por el teclado o leyendo un fichero.
- Los resultados de la ejecución de un programa se pueden presentar por la consola, la impresora o en un fichero.
- El tipo de información que se utiliza tanto en las entradas como en las salidas puede tener diversos formatos: texto, imagen, sonido, binario, etc.

Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

Java IO

La forma estándar de gestionar flujos de datos

https://docs.oracle.com/javase/tutorial/essential/io/streams.html

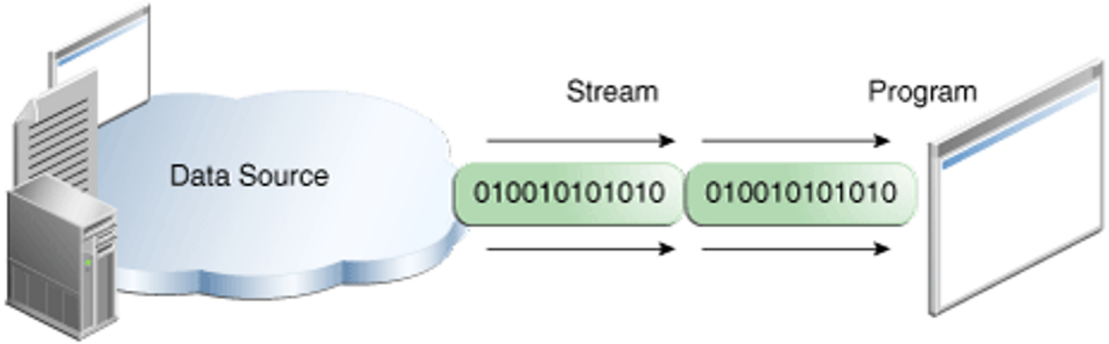
Hechos QUE CONECTAN

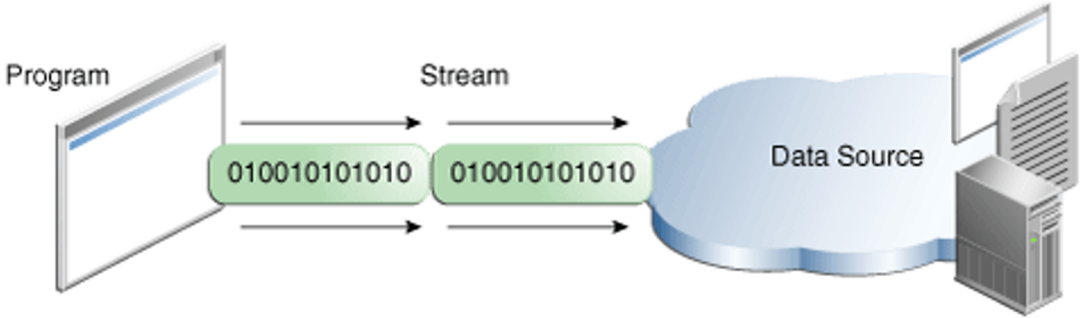
Streams

- Los **streams** son flujos secuenciales de bytes.
- Para que un programa pueda leer datos de alguna fuente, debe crear un ***stream de entrada*** conectado a ésta; una fuente típica puede ser el teclado o un fichero.
- Para escribir datos hacia un destino, debe crear un ***stream de salida*** conectado a éste; un destino típico puede ser la pantalla o un fichero.
- Java proporciona distintas clases para el manejo de estos flujos de información, todas ellas contenidas en el paquete `java.io` .

Streams

InputStream





OutputStream

Entrada y salida básica

Los atributos ***in***, ***out*** y ***err*** son variables estáticas de la clase **System**, y se han inicializado previamente con flujos de entrada y salida, respectivamente.

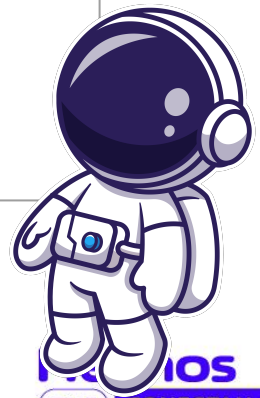
Campo de clase System	Descripción
`static PrintStream err`	El flujo de salida de error estándar.
`static InputStream in`	El flujo de entrada estándar.
`static PrintStream out`	El flujo de salida estándar.

Flujos con bytes	Flujos con caracteres
InputStream ByteArrayInputStream FileInputStream FilterInputStream BufferedInputStream DataInputStream LineNumberInputStream PushBackInputStream ObjectInputStream PipedInputStream SequenceInputStream StringBufferInputStream	Reader BufferedReader LineNumberReader CharArrayReader FilterReader PushBackReader InputStreamReader FileReader PipedReader StringReader

Flujos de entradas - `java.io`.

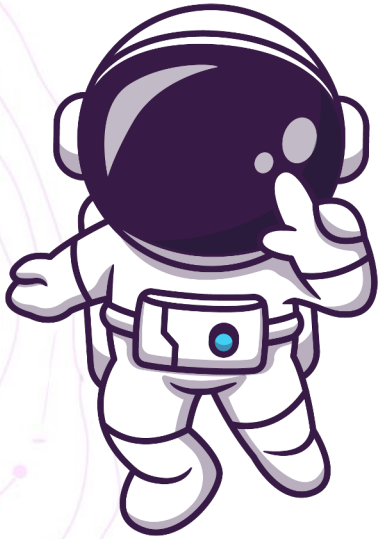
Flujos con bytes	Flujos con caracteres
OutputStream ByteArrayOutputStream FileOutputStream FilterOutputStream BufferedOutputStream DataOutputStream PrintStream ObjectOutputStream PipedOutputStream	Writer BufferedWriter CharArrayWriter FilterWriter OutputStreamWriter FileWriter PipedWriter PrintWriter StringWriter

Flujos de salidas - `java.io`.



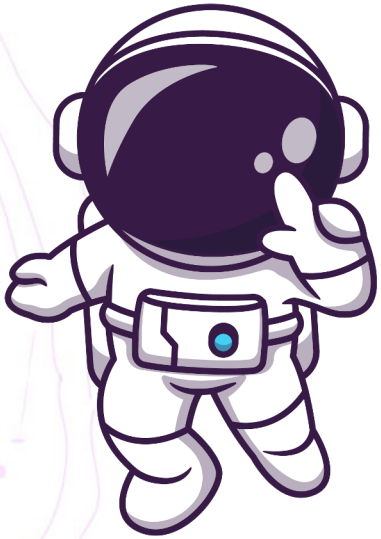
Entrada y salida básica

```
import java.io.*;

public class IOEstandar1 {

  public static void main(String args[]) {
    int numBytes = 0;
    char caracter;
    System.out.println("\nEscribe el texto: ");
    try {
      do {
        caracter = (char)System.in.read();
        System.out.print(caracter);
        numBytes++;
      } while (caracter != '\n');
      System.err.printf("%d bytes leidos %n", numBytes);
    } catch (IOException e) {
      System.err.println(e);
    }
  }
}
```

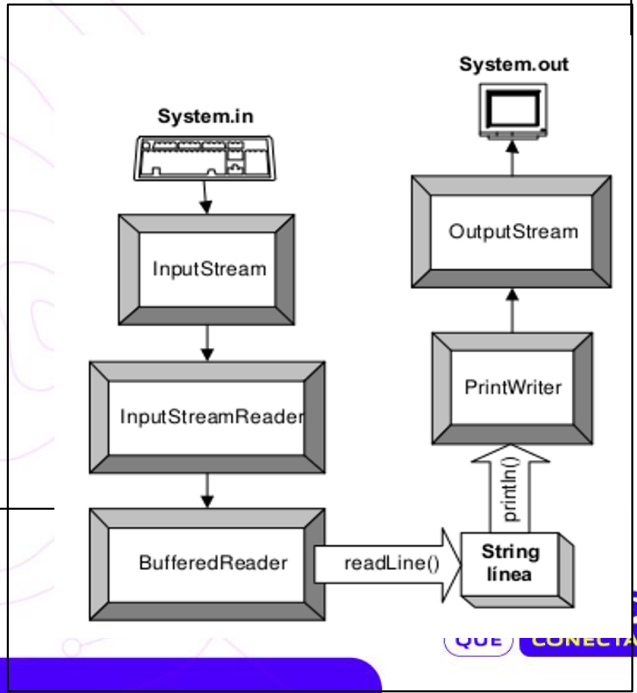


Entrada y salida básica

```
import java.io.*;

public class IOEstandar2 {

  public static void main(String args[]) {
    byte[] buffer = new byte[255];
    System.out.println("\nEscribe el texto: ");
    try {
      System.in.read(buffer, 0, 255);
    } catch (IOException e) {
      System.err.println(e);
    }
    System.out.println("\nLa línea escrita es: ");
    System.out.println(new String(buffer));
  }
}
```



```
import java.io.*;
public class EntradaSalida1 {
  public static void main(String arg[]) {
    String línea=null;
    var entrada = new BufferedReader(new InputStreamReader(System.in));
    var salida = new PrintWriter(System.out, true);

    salida.println("\nEscribe el texto: ");
    try {
      línea = entrada.readLine();
    } catch (Exception e) {
      System.err.println(e);
    }
    salida.println("\nLa línea escrita es: ");
    salida.println(línea);
  }
}
```



Entrada por teclado y salida por pantalla

Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

Gestión de Archivos

Hechos QUE CONECTAN

Gestión de archivos

- La clase **File** sirve para representar ficheros o directorios en el sistema de ficheros de la plataforma específica.
- Mediante esta clase pueden abstraerse las particularidades de cada sistema de ficheros y proporcionar los métodos necesarios para obtener información sobre los mismos.

```
var f = new File("sub\\prueba.txt");
System.out.println("pathSeparator: "+File.pathSeparator);
System.out.println("separator: " + File.separator);
System.out.println("separatorChar: "+File.separatorChar);
try {
  System.out.println("canRead():" + f.canRead());
  System.out.println("canWrite():" + f.canWrite());
  System.out.println("exists():" + f.exists());
  System.out.println("getName():" + f.getName());
  System.out.println("getParent():" + f.getParent());
  System.out.println("isDirectory():" + f.isDirectory());
  System.out.println("isFile():" + f.isFile());
  System.out.println("length():" + f.length());
} catch (IOException e) {
  System.out.println(e);
}
```

```java
int[][] numeros = { { 1, 2, 3, 4, 5},
                    { 6, 7, 8, 9, 10},
                    {11, 12, 13, 14, 15},
                    {16, 17, 18, 19, 20},
                    {21, 22, 23, 24, 25}};


var archivo = "c:\\Numeros.txt";
var salida = new PrintWriter(archivo);

for (int i=0; i< numeros.length; i++) {
  for (int j=0; j< numeros[i].length; j++)
{
    salida.print(numeros[i][j] + ",");
  }
  salida.println("");
}
salida.close();
```

```
1,2,3,4,5,
6,7,8,9,10,
11,12,13,14,15,
16,17,18,19,20,
21,22,23,24,25,
```

```java
String nombre = "c:\\Numeros.txt";
var archivo = new File(nombre);

if (archivo.exists()) {
  var lector = new Scanner(archivo);
  System.out.println("Números del archivo");
  while (lector.hasNext()) {
    var numeros = new StringTokenizer(lector.next(),",");
    while (números.hasMoreTokens()) {
      System.out.print(numeros.nextToken() + "\t");
    }
    System.out.println("");
  }
  lector.close();
} else {
  System.out.println("¡El fichero no existe!");
}
```

```
Números del fichero
1           2           3
            4           5
6           7           8
            9           10
11          12          13
            14          15
16          17          18
```

Escribir y leer archivos de texto

Serialización de objetos

- Si se desea guardar permanentemente el estado de un objeto puede utilizar las clases vistas hasta el momento para ir almacenando todos los valores de los atributos como valores char, int, byte, etc.
- Pero esto puede ser muy molesto y complicado, ya que un objeto puede tener atributos que sean otros objetos, que a su vez tengan atributos que hagan referencia a otras clases y así sucesivamente.
- Pero existe una forma más cómoda de enviar objetos a través de un stream como una secuencia de bytes para ser almacenados en disco, y también para reconstruir objetos a partir de streams de entrada de bytes. Esto puede conseguirse mediante la "***serialización***" de objetos.

La serialización consiste en la transformación de un objeto Java en una secuencia de bytes para ser enviados a un stream.

Objetos serializables

- Sólo los objetos de clases que implementen la interface **java.io.Serializable** o aquellos que pertenezcan a subclases de clases serializables pueden ser serializados.
- La interface Serializable no posee ningún método. Sólo sirve para “marcar” las clases que pueden ser serializadas.
- Cuando un objeto es serializado, también lo son todos los objetos alcanzables desde éste (los atributos que son objetos), ignorándose todos los atributos **static**, **transient** y los no serializables.

```
public class Persona implements Serializable {
  private String dni;
  private String nombre;
  private String apellidos;

  public Persona(String dni, String nombre,
                 String apellidos) {
    this.dni = dni;
    this.nombre = nombre;
    this.apellidos = apellidos;
  }
  public String getDNI() {
    return this.dni;
  }
  public String getNombre() {
    return this.nombre;
  }
  public String getApellidos() {
    return this.apellidos;
  }
  public String getAtributos() {
    return this.getDNI() + " "
           + this.getApellidos() + ", "
           + this.getNombre();
  }
}
```

```
var nombre = "c:\\Objetos.dat";
try {
  var archivo = new FileOutputStream(nombre);
  var oos = new ObjectOutputStream(archivo);

  oos.writeObject(new Persona("552871883",
                              "María", "Ruiz Ramos"));
  oos.writeObject(new Persona("403020104",
                              "Juan", "González López"));
  oos.close();
} catch (FileNotFoundException e) {
  System.out.println("¡El fichero no existe!");
} catch (IOException e) {
  System.out.println(e.getMessage());
} catch (Exception e) {
  System.out.println(e.getMessage());
};
```

Escribir archivos de objetos

```
public class Persona implements Serializable {
  private String cedula;
  private String nombre;
  private String apellidos;

  public Persona(String cedula, String nombre,
                 String apellidos) {
    this.cedula = cedula;
    this.nombre = nombre;
    this.apellidos = apellidos;
  }
  public String getCedula() {
    return this.cedula;
  }
  public String getNombre() {
    return this.nombre;
  }
  public String getApellidos() {
    return this.apellidos;
  }
  public String getAtributos() {
    return this.getCedula() + " "
           + this.getApellidos() + ", "
           + this.getNombre();
  }
}
```

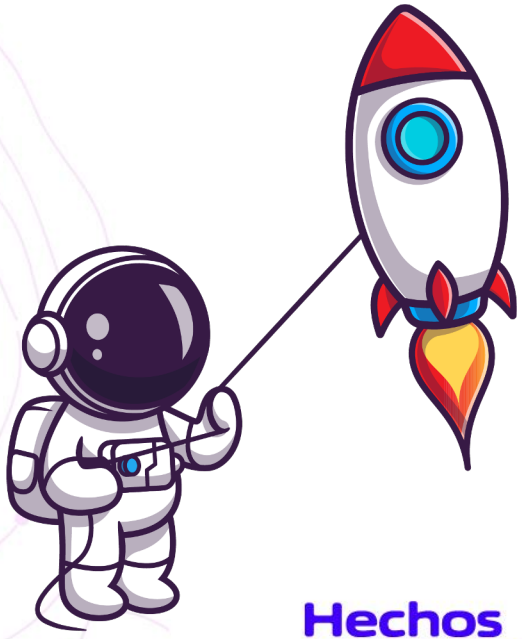
```
var nombre = "c:\\Objetos.dat";
try {
  var archivo = new FileInputStream(nombre);
  var ois = new ObjectInputStream(archivo);

  var p1 = (Persona) ois.readObject();
  var p2 = (Persona) ois.readObject();

  ois.close();

  System.out.println("Cedula\t Nombre");
  System.out.println(p1.getAtributos());
  System.out.println(p2.getAtributos());
} catch (FileNotFoundException e) {
  System.out.println("¡El fichero no existe!");
} catch (IOException e) {
  System.out.println(e.getMessage());
} catch (Exception e) {
  System.out.println(e.getMessage());
};
```

```
Cedula    Nombre
552871883 Ruiz Ramos, María
403020104 González López, Juan
```

Leer archivos de objetos

Vamos al código

- Crear un proyecto Maven para los ejercicios de la clase 10
  - Ctrl + Shift + P
  - *> Java: Create Java Project...*
  - *Maven*
  - *maven-archetype-quickstart*, *<versión más reciente>*
  - **group Id**: co.edu.utp.misiontic2022.c2
  - **artefact Id**: clase10-archivos
- Realizar las acciones propuestas en cada ejercicio.
- Crear una clase por ejercicio propuesto



Hechos QUE CONECTAN

Ejercicios

1. Crea un fichero de texto con el nombre y contenido que tu quieras. Ahora crea una aplicación que lea este fichero de texto carácter a carácter y muestre su contenido por pantalla sin espacios. Por ejemplo, si un fichero tiene el siguiente texto "**Esto es una prueba**", deberá mostrar "**Estoesunaprueba**".

   Captura las excepciones que veas necesario.

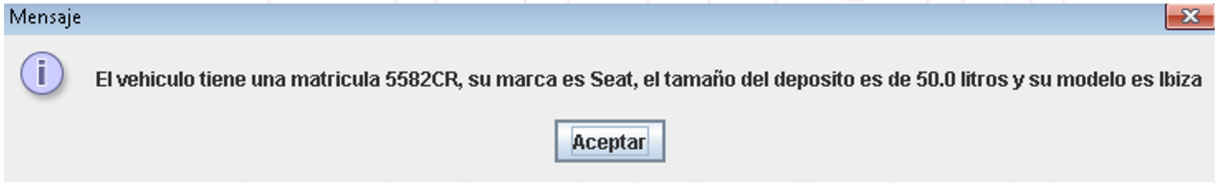
2. Crea una aplicación donde pidamos la ruta de un fichero por teclado y un texto que queramos a escribir en el fichero. Deberás mostrar por pantalla el mismo texto pero variando entre mayúsculas y minúsculas, es decir, si escribo "Bienvenido" deberá devolver "bIENVENIDO". Si se escribe cualquier otro carácter, se quedará tal y como se escribió. Deberás crear un método para escribir en el fichero el texto introducido y otro para mostrar el contenido en mayúsculas.

Ejercicios

3. Crea una aplicación que almacene los datos básicos de un vehículo como la matricula(String), marca (String), tamaño de depósito (double) y modelo (String) en ese orden y de uno en uno usando la clase DataInputStream.

   Los datos anteriores datos se pedirán por teclado y se irán añadiendo al fichero (no se sobrescriben los datos) cada vez que ejecutemos la aplicación.

   Muestra todos los datos de cada vehículo en un cuadro de diálogo (JOptionPane), es decir, si tenemos 3 vehículos mostrará 3 cuadros de diálogo con sus respectivos datos. Un ejemplo de salida de información puede ser este:



Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

Try-with-resources

Gestionar IO streams de forma segura

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Hechos QUE CONECTAN

```
var nombre = "c:\\Objetos.dat";
try {
  var oos = new ObjectOutputStream(
                          new FileOutputStream(nombre));

  oos.writeObject(new Persona("552871883",
                          "María", "Ruiz Ramos"));
  oos.writeObject(new Persona("403020104",
                          "Juan", "González López"));
  oos.close();
} catch (FileNotFoundException e) {
  System.out.println("¡El fichero no existe!");
} catch (Exception e) {
  System.out.println(e.getMessage());
};
```

try-with-resources

```
var nombre = "c:\\Objetos.dat";
try (var oos = new ObjectOutputStream(
                          new FileOutputStream(nombre))) {
  oos.writeObject(new Persona("552871883",
                          "María", "Ruiz Ramos"));
  oos.writeObject(new Persona("403020104",
                          "Juan", "González López"));
} catch (FileNotFoundException e) {
  System.out.println("¡El fichero no existe!");
} catch (Exception e) {
  System.out.println(e.getMessage());
};
```

Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

Java NIO.2

Nueva forma de gestionar archivos

https://docs.oracle.com/javase/tutorial/essential/io/fileio.html

Hechos QUE CONECTAN

NIO.2 (java.nio.*)

- Entre las mejoras se incluyen permitir navegación de directorios sencillo, soporte para reconocer enlaces simbólicos, leer atributos de ficheros como permisos e información como última fecha de modificación, soporte de entrada/salida asíncrona y soporte para operaciones básicas sobre ficheros como copiar y mover ficheros.
- Las clases principales de esta nueva API para el manejo de rutas, ficheros y operaciones de entrada/salida son las siguientes:
  - **Path**: es una abstracción sobre una ruta de un sistema de ficheros. Puede usarse como reemplazo completo de ***java.io.File*** pero si fuera necesario con los métodos **File.toPath()** y **Path.toFile()** se ofrece compatibilidad.
  - **Files**: es una clase de utilidad con operaciones básicas sobre ficheros.
  - **FileSystems**: otra clase de utilidad como punto de entrada para obtener referencias a sistemas de archivos.

Java NIO por ejemplos

- Crea un archivo vacío si aún no existe

```
Path archivo = Paths.get("/examples/emptyFile.txt");
if (Files.notExists(archivo)) {
  archivo = Files.createFile(Paths.get("/examples/emptyFile.txt"));}

}
```

- Lee todo el contenido de un archivo de texto a una cadena

```
var contenido = new String(Files.readAllBytes(Paths.get("/examples/sampleText.txt")),
                                                    StandardCharsets.UTF_8);
```

- Lee el contenido de un archivo de texto linea por linea

```
var lineas = Files.readAllLines(Paths.get("/examples/sampleText.txt"),
                                                StandardCharsets.UTF_8);
```

Java NIO por ejemplos

- Escribe un String en un archivo de texto, sobreescribe si ya existe

```
var text = "Esto es una cadena de prueba";
Files.write(Paths.get("/examples/writeText.txt"), text.getBytes(StandardCharsets.UTF_8),
                    StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);
```

- Escribe una lista de String en un archivo, sobreescribe si ya existe

```
var textLines = Arrays.asList("Línea 1", "Línea 2", "Línea 3");
Files.write(Paths.get("/examples/writeText.txt"), textLines, StandardCharsets.UTF_8,
                    StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);
```

- Crear una estructura de directorios de forma recursiva

```
Files.createDirectories(Paths.get("/examples/level1/level2/level3"));
```

Java NIO por ejemplos

- Lista recursiva con los ficheros contenidos en un directorio

```
List<Path> files = Files.walk(Paths.get("/examples"))
                        .filter(Files::isRegularFile)
                        .map(x -> x.toAbsolutePath())
                        .collect(Collectors.toList());

for (Path file : files) {
  System.out.println("Ruta del fichero: ".concat(file.toString()));
}
```

- Mueve un directorio con todo su contenido

```
Files.move(Paths.get("/examples/source_dir"), Paths.get("/examples/dest_dir"),
           StandardCopyOption.REPLACE_EXISTING);
```

Para la próxima sesión...

- Terminar los ejercicios que no se terminaron… (si aplica)
- Revisar el funcionamiento de la aplicación que se encuenta en https://github.com/cesardiaz-utp/MisionTIC2022-Ciclo2-Unidad4-Archivos
- Ver videos: (Material complementario)
  - Lectura Archivos en Java
  - Escritura Archivos en Java