Bienvenidos

Manejo de Datos con Archivos

David R. Luna G.
davidrlunag@gmail.com
0412-3111011

Temario del Curso

- ✓ Introducción al Lenguaje Java
- ✓ Estructura del Lenguaje Java
- ✓ Herramientas de Desarrollo Java
- ✓ Programación Orientada a Objetos con Java
- ✓ **Manejo de Datos con Archivos**
- ✓ Trabajando con Base de Datos Relaciones
- ✓ Desarrollando GUIs

Manejo de Datos con Archivos

- ✓ Java streams
- ✓ Streams, Readers and Writers
- ✓ Catching and throwing exceptions
- ✓ Formatting text output
- ✓ Files and directories
- ✓ Reading and writing files
- ✓ Creating, deleting and renaming files
- ✓ Obtaining directory and file information

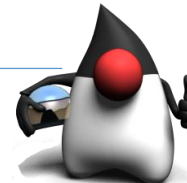
Manejo de Datos con Archivos

Manejo de Streams

Cualquier programa realizado en Java que necesite llevar a cabo una operación de I/O lo hará a través de un stream. Un stream, cuya traducción literal es "flujo", es una abstracción de todo aquello que produzca o consuma información.

Un Stream I/O representa una fuente de entrada o un destino de salida. Un Stream puede representar diversas clases de fuentes y de destinos, incluyendo archivos en disco, dispositivos, otros programas, y arreglos en memoria.

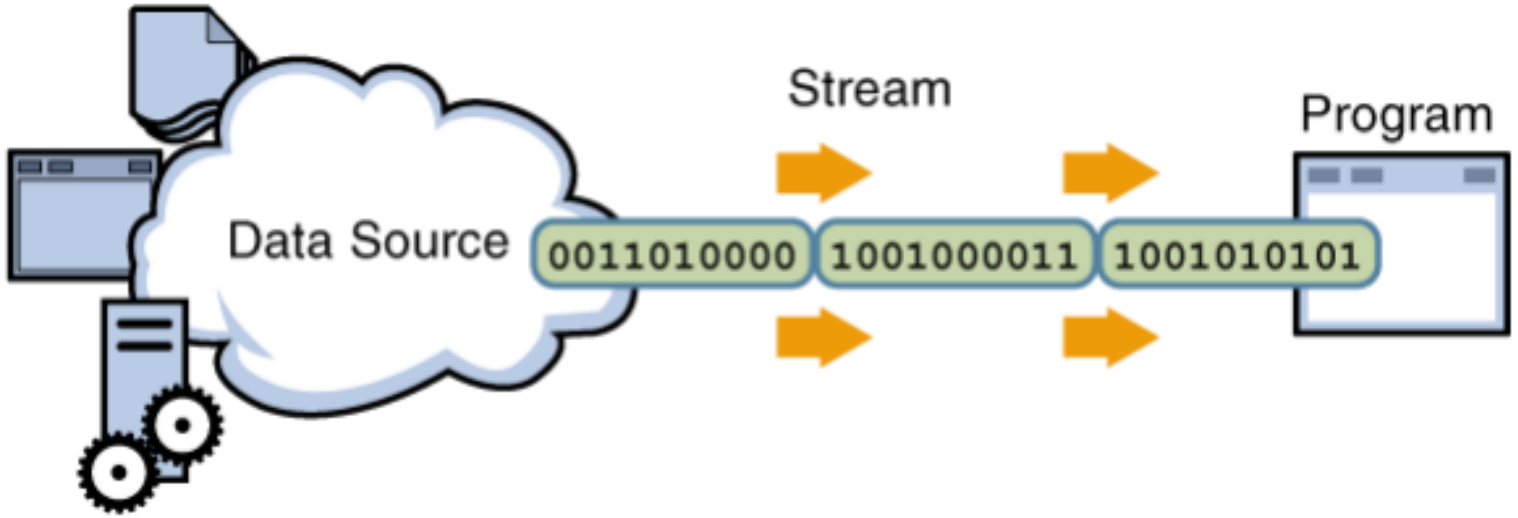
Los Stream soportan diversas clases de datos, incluyendo bytes simples, tipos de datos primitivos, caracteres localizados, y objetos. Algunos Stream simplemente pasan datos; otros manipulan y transforman los datos de forma útil.



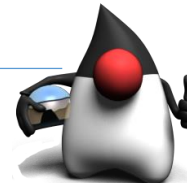
Manejo de Datos con Archivos

Manejo de Streams

Un Stream es una secuencia de datos. Un programa utiliza input stream (un flujo de entrada) a los datos leídos de una fuente, leyendo un ítem a la vez:



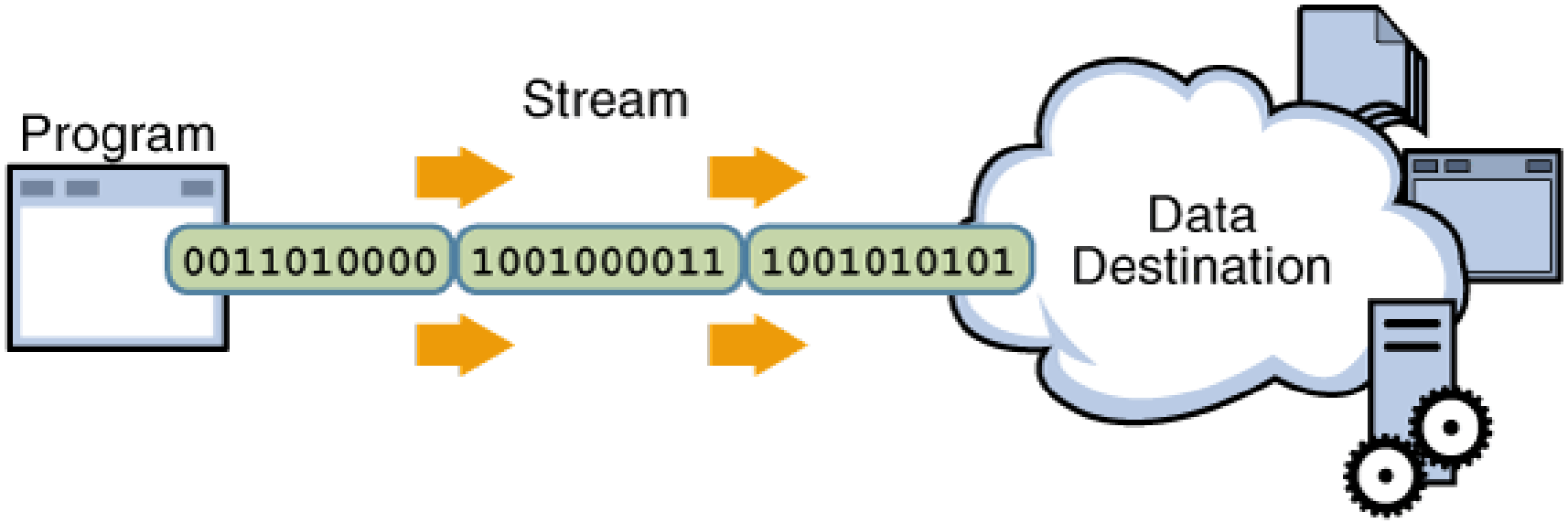
Información de la lectura en un programa.



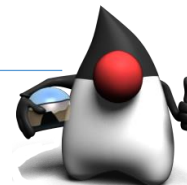
Manejo de Datos con Archivos

Manejo de Streams

Un programa utiliza un output stream (flujo de salida) para escribir los datos a un destino, un dato a la vez:



Información de escritura en un programa.



Manejo de Datos con Archivos

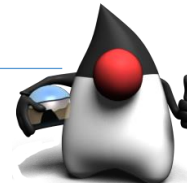
Manejo de Streams

Java define 2 tipos de Streams:

ByteStreams

Proporciona un medio adecuado para el manejo de entradas y salidas de bytes y su uso lógicamente está orientado a la lectura y escritura de datos binarios. El tratamiento del flujo de bytes viene gobernado por dos clases abstractas que son InputStream y OutputStream.
Los flujos de bytes se usan para manipular datos binarios, legibles solo por la maquina (por ejemplo un fichero .exe)

Cada una de estas clases abstractas tienen varias subclases concretas que controlan las diferencias entre los distintos dispositivos de I/O que se pueden utilizar. Así mismo, estas dos clases son las que definen los métodos que sus subclases tendrán implementados y, de entre todas, destacan los métodos read() y write() que leen y escriben bytes de datos respectivamente.

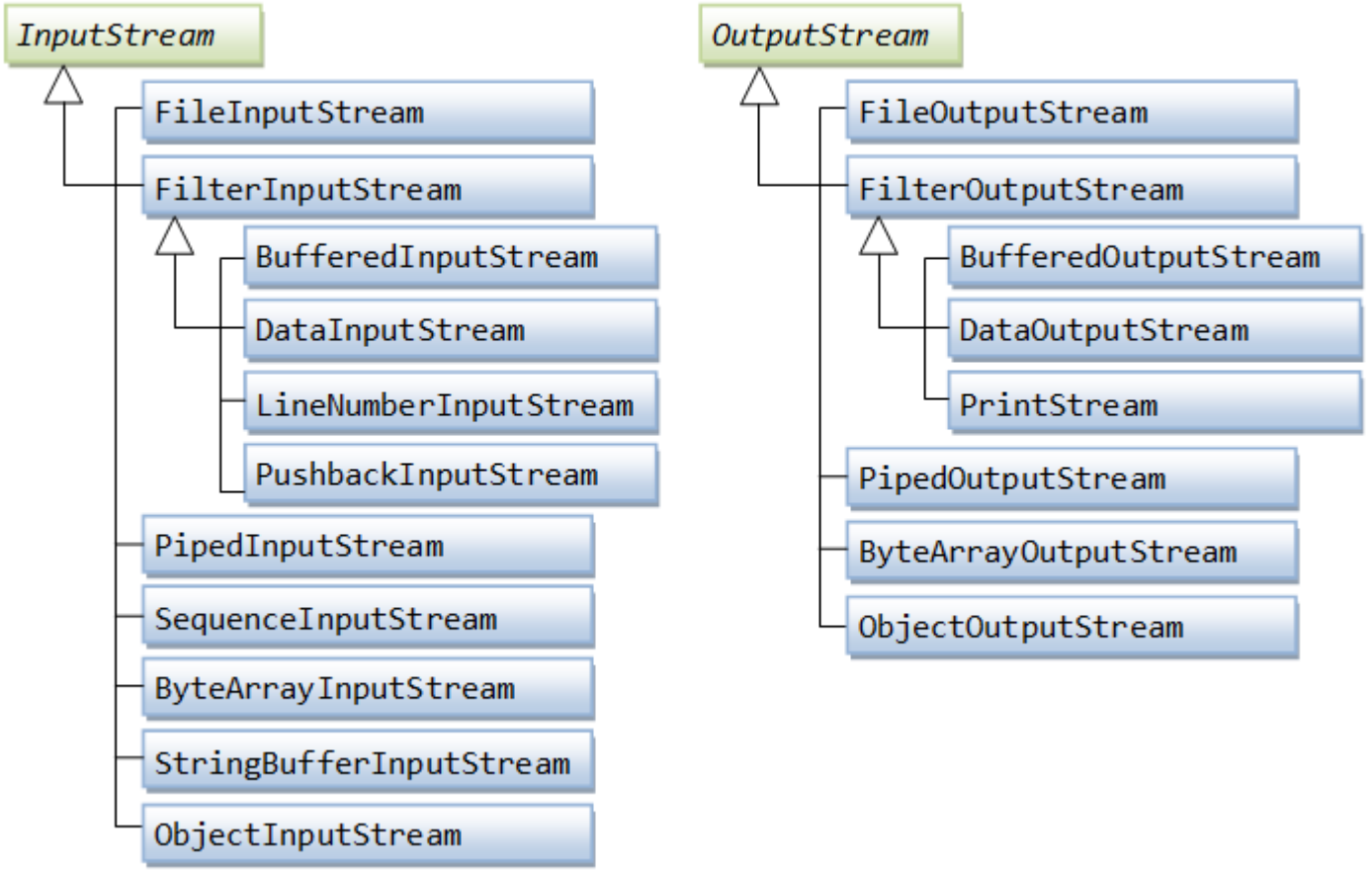


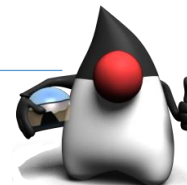
Manejo de Datos con Archivos

Manejo de Streams

Java define 2 tipos de Streams:

ByteStreams



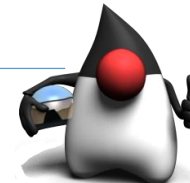


Manejo de Datos con Archivos

Manejo de Streams

```
public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream(“nombre_archivo.ext");
            out = new FileOutputStream("nombre_archivo.ext");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                n.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

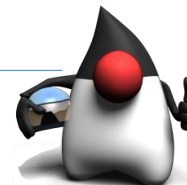
this.getClass().getResource("")



Manejo de Datos con Archivos

Manejo de Streams

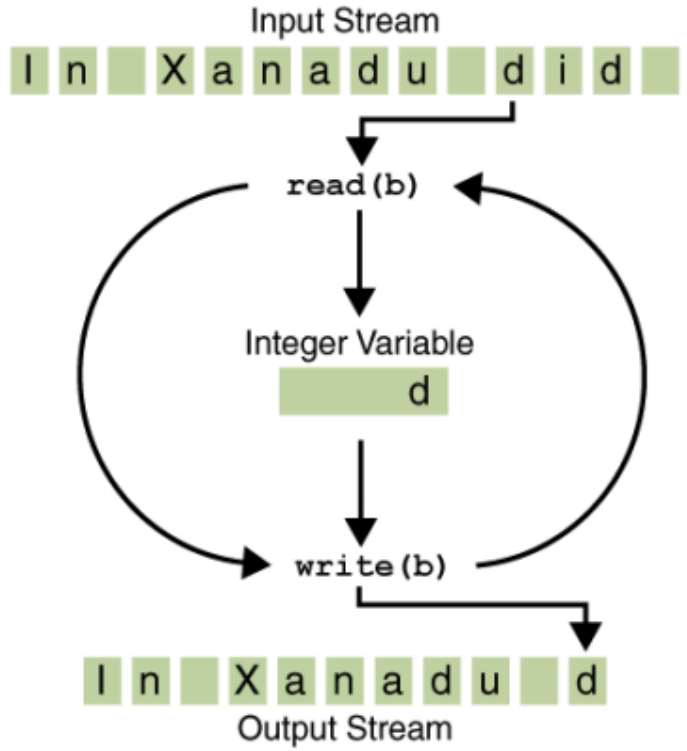
```
try(BufferedWriter writer=new BufferedWriter(new FileReader(path))){
        return br.readLine();
}catch(IOException ex){
        ex.printStackTrace();
}
```



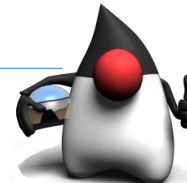
Manejo de Datos con Archivos

Manejo de Streams

El método read() retorna un dato tipo int. Si el stream de entrada de bytes no se puede leer, el método retorna el valor de -1. Retorna -1 para indicar que llego al final del stream o que no se puede leer.



Un stream simple de entrada y salida



Manejo de Datos con Archivos

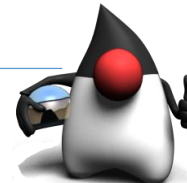
Manejo de Streams

Java define 2 tipos de Streams:

Character streams

Proporciona un medio conveniente para el manejo de entradas y salidas de caracteres. Dichos flujos usan codificación Unicode y, por tanto, se pueden internacionalizar.

Al igual que la anterior el flujo de caracteres también viene gobernado por dos clases abstractas: Reader y Writer. Dichas clases manejan flujos de caracteres Unicode. Y también de ellas derivan subclases concretas que implementan los métodos definidos en ellas siendo los más destacados los métodos read() y write() que, en este caso, leen y escriben caracteres de datos respectivamente.

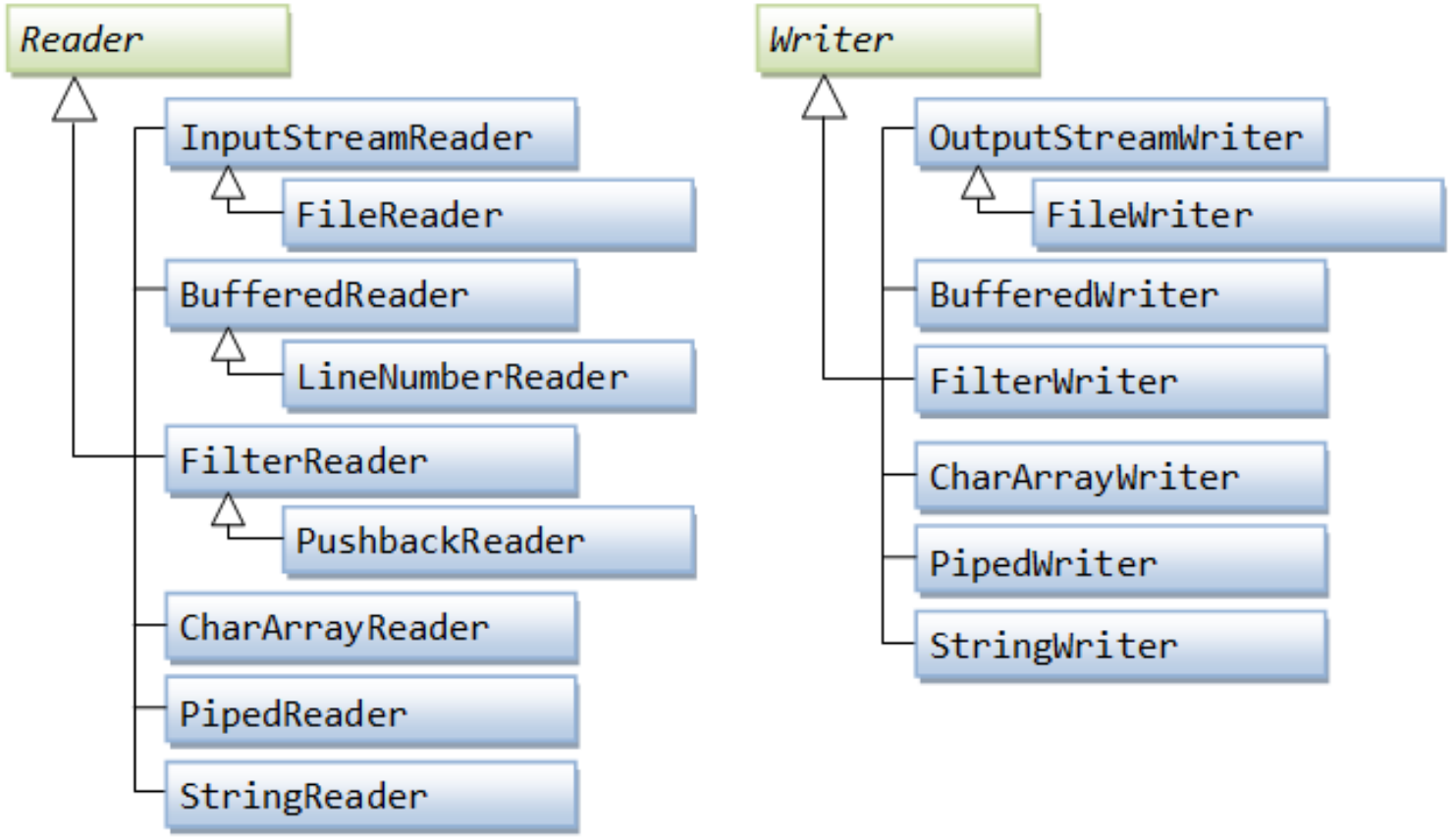


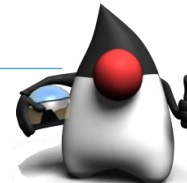
Manejo de Datos con Archivos

Manejo de Streams

Java define 2 tipos de Streams:

Character streams

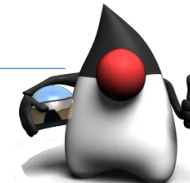




Manejo de Datos con Archivos

Manejo de Streams

```
public class CopyCharacters {
        public static void main(String[] args) throws IOException {
            FileReader inputStream = null;
            FileWriter outputStream = null;
            try {
                inputStream = new FileReader("frase.txt");
                outputStream = new FileWriter("salidafrase.txt");
                int c;
                while ((c = inputStream.read()) != -1) {
                        outputStream.write(c);
                }
             } finally {
                        if (inputStream != null) {
                                inputStream.close();
                         }
                         if (outputStream != null) {
                                outputStream.close();
                         }
            }
        }
}
```



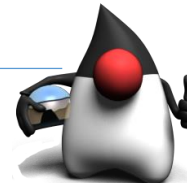
Manejo de Datos con Archivos

Manejo de Streams

Buffered streams

Hasta el momento se han colocado ejemplos de programas que leen o escriben directamente realizando una petición que es dirigida al OS subyacente. Esto hace que un programa sea menos eficiente, por que cada petición de lectura o escritura acciona el acceso de disco, o en acceso a la red lo que aumenta la actividad de la red, o acciona cierta operación que sea relativamente costosa.

Para reducir esta clase de consumos de recursos indirectos, la plataforma de Java implementa los buffered Streams de I/O. Que no es mas que leer datos en un área de memoria intermedia conocida como buffer ; el API de lectura - read() - nativo solamente es llamado cuando el buffer intermediario esta vacío. De forma similar, Los datos de salida se escriben a un buffer intermedio, y se llama al API de escritura - write() – nativo solamente cuando el buffer intermediario esta lleno.



Manejo de Datos con Archivos

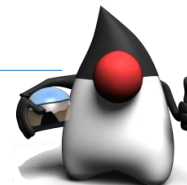
Manejo de Streams

Buffered streams

Un programa que usa streams sin buffer se puede convertir a un programa con buffer stream usando los wrapping (envolturas) que me brinda Java , donde el objeto unbuffered stream se pasa al constructor de una clase buffered stream.

Un ejemplo de la modificacion es cambiar el ejemplo CopyCharacters para utilizar I/O con Buffered Streams

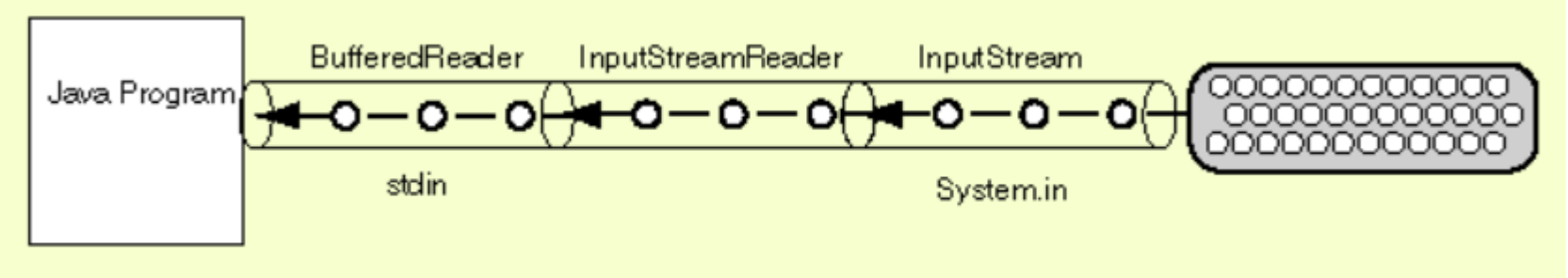
```
inputStream = new BufferedReader(new FileReader("frase.txt"));
outputStream = new BufferedWriter(new fileWriter("salidafrase.txt"));
```



Manejo de Datos con Archivos

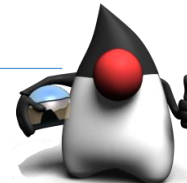
Manejo de Streams

Leer Flujo desde el Teclado



InputStream lee Bytes desde el flujo de entrada estándar
InputStreamReader lee Bytes y les descodifica como caracteres
BufferedReader lee caracteres y les agrupa en líneas.

```
InputStreamReader reader = new InputStreamReader(System.in);
BufferedReader in = new BufferedReader(reader);
```



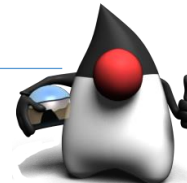
Manejo de Datos con Archivos

Manejo de Streams

La Clase File

Esta clase no es utilizada para leer y mucho menos para escribir , solo es para que entren otras operaciones:

- Obtener el tamaño del archivo.
- Obtener el nombre completo, incluida la ruta.
- Cambiar el nombre.
- Eliminar el nombre
- Saber si es un directorio o un archivó.
- Si en un directorio, obtener la lista de los archivos y directorios que contiene.
- Crear un directorio

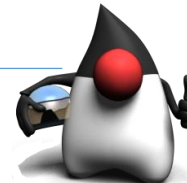


Manejo de Datos con Archivos

Manejo de Streams

La Clase File

```
public class MainClass {
    public static void main(String args[]) {
            File f1 = new File("MainClass.java");
            System.out.println("File Name:" + f1.getName());
            System.out.println("Path:" + f1.getPath());
            System.out.println("Abs Path:" + f1.getAbsolutePath());
            System.out.println("Parent:" + f1.getParent());
            System.out.println(f1.exists() ? "exists" : "does not exist");
            System.out.println(f1.canWrite() ? "is writeable" : "is not writeable");
            System.out.println(f1.canRead() ? "is readable" : "is not readable");
            System.out.println("is a directory" + f1.isDirectory() );
            System.out.println(f1.isFile() ? "is normal file" : "might be a named pipe");
            System.out.println(f1.isAbsolute() ? "is absolute" : "is not absolute");
            System.out.println("File last modified:" + f1.lastModified());
            System.out.println("File size:" + f1.length() + " Bytes");
    }
}
```

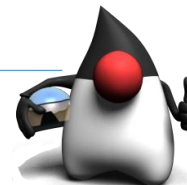


Manejo de Datos con Archivos

Manejo de Streams

La Clase File

```
import java.io.File;
public class MainClass {
 public static void main(String args[]) {
  String dirname = "c:\\";
  File f1 = new File(dirname);
  if (f1.isDirectory()) {
   System.out.println("Directorio de " + dirname);
   String s[] = f1.list();
   for (int i = 0; i < s.length; i++) {
    File f = new File(dirname + "/" + s[i]);
    if (f.isDirectory()) {
     System.out.println(s[i] + " es un directorio");
    } else {
     System.out.println(s[i] + " es a archivo");
    }
   }
  } else {
   System.out.println(dirname + " no es un directorio");
  }
 }
}
```



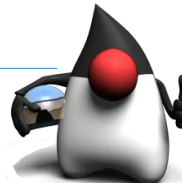
Manejo de Datos con Archivos

Manejo de Streams

Serialización

Esta clase no es utilizada para leer y mucho menos para escribir , solo es para que entren otras operaciones:

- Obtener el tamaño del archivo.
- Obtener el nombre completo, incluida la ruta.
- Cambiar el nombre.
- Eliminar el nombre
- Saber si es un directorio o un archivó.
- Si en un directorio, obtener la lista de los archivos y directorios que contiene.
- Crear un directorio



Manejo de Datos con Archivos

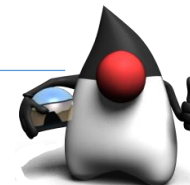
Manejo de Streams

Serialización

```
public class MainClass {
  public static void main(String[] args) {
    Empleado e1=new Empleado(¨Nombre Empleado¨,18);
    try {
      FileOutputStream fileOut = new FileOutputStream(“empleado.ser");
      ObjectOutputStream out = new ObjectOutputStream(fileOut);

      out.writeObject(e1);

      out.close();
      fileOut.close();
    } catch (IOException i) {
      i.printStackTrace();
    }
  }
}
```



Manejo de Datos con Archivos

Manejo de Streams

Serialización

```
public class MainClass {
  public static void main(String[] args) {
    Empleado e1=new Empleado(¨Nombre Empleado¨,18);
    try {
      ObjectInputStream in = new ObjectInputStream(
              new BufferedInputStream( new FileInputStream(“empleado.ser")));
Empleado e1 = (Empleado)in.readObject();
              in.close();

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