File handling

* file handling means reading and writing data to a file.
* In Java, the concept Stream is used in order to perform I/O operations on a file

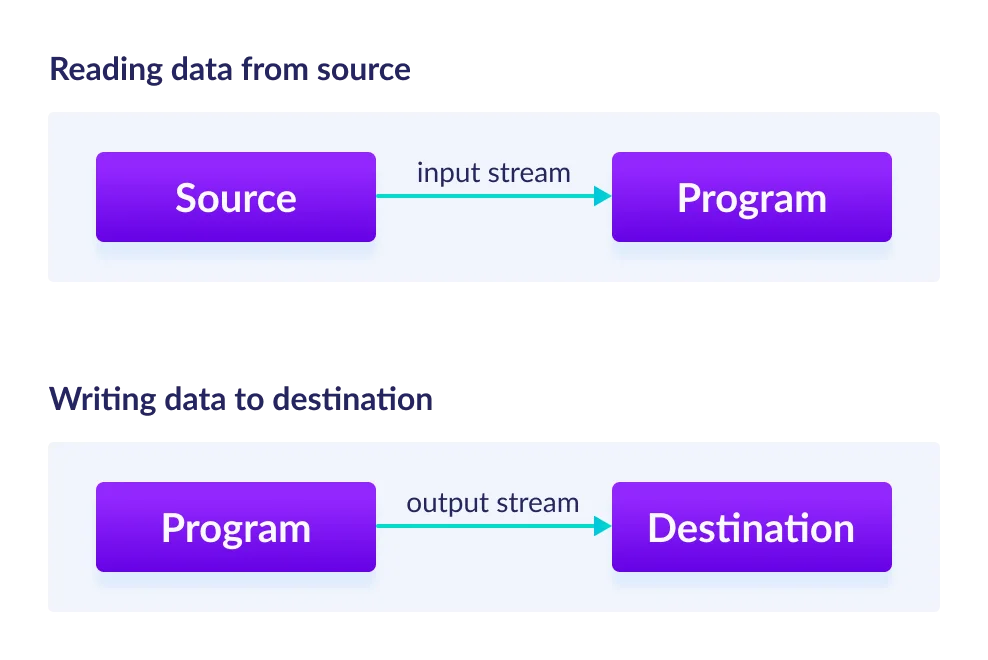
# **Streams in Java**

* A sequence of data is known as a stream.
* This concept is used to perform I/O operations on a file.

There are two types of streams :

1.Input stream

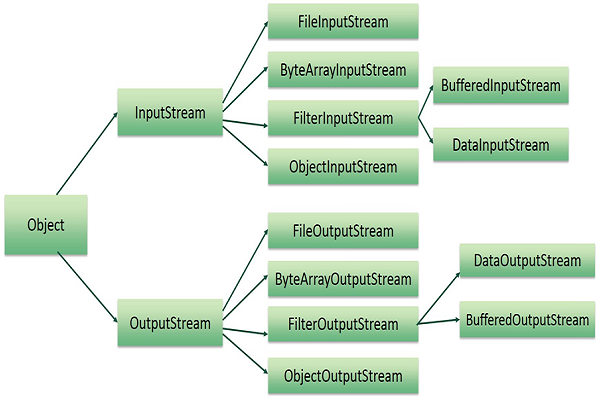
2. Output stream



#### 

## **Input Stream:**

* InputStream class is the superclass of all input streams
* Read data from numerous input devices like the keyboard, network, etc.
* InputStream is an abstract class,so it is not useful directly. Hence its subclasses are used to read data.
* Sub-classes are-
  + AudioInputStream
  + ByteArrayInputStream
  + FileInputStream
  + FilterInputStream
  + StringBufferInputStream
  + ObjectInputStream



### Syntax for creating input stream-

InputStream obj = new FileInputStream();

| read() | Reads one byte of data from the input stream. |  |
| --- | --- | --- |
| 2 | read(byte[] array)() | Reads byte from the stream and stores that byte in the specified array. |
| 3 | mark() | It marks the position in the input stream until the data has been read. |
| 4 | available() | Returns the number of bytes available in the input stream. |
| 5 | markSupported() | It checks if the mark() method and the reset() method is supported in the stream. |
| 6 | reset() | Returns the control to the point where the mark was set inside the stream. |
| 7 | skips() | Skips and removes a particular number of bytes from the input stream. |
| 8 | close() | Closes the input stream. |

## **2. Output Stream**

Sub classes of output stream -

* ByteArrayOutputStream
* FileOutputStream
* StringBufferOutputStream
* ObjectOutputStream
* DataOutputStream
* PrintStream

Syntax- // Creating an OutputStream

OutputStream obj = new FileOutputStream();

1. write() Writes the specified byte to the output stream.

2. write(byte[] array) Writes the bytes which are inside a specific array to the output stream.

3. close() Closes the output stream.

4. flush() Forces to write all the data present in an output stream to the destination.

Based on data type- there are 2 types of streams

1.Byte Stream

2. Character Stream

## **1. Byte Stream:**

* Byte Input Stream: Used to read byte data from different devices.
* Byte Output Stream: Used to write byte data to different devices.

## **2. Character Stream:**

* Character Input Stream: Used to read character data from different devices.
* Character Output Stream: Used to write character data to different devices.

File class

* The File class in Java is part of the java.io package
* It is used to create, delete, and get information about files and directories.
* It does not provide methods for reading or writing file content directly.

Methods-

* boolean createNewFile(): Atomically creates a new, empty file if and only if a file with this name does not yet exist.
* boolean delete(): Deletes the file or directory denoted by this abstract pathname.
* void deleteOnExit(): Requests that the file or directory be deleted when the virtual machine terminates.
* boolean exists(): Tests whether the file or directory denoted by this abstract pathname exists.
* String getName(): Returns the name of the file or directory denoted by this abstract pathname.
* String getPath(): Converts this abstract pathname into a pathname string.
* boolean mkdir(): Creates the directory named by this abstract pathname.
* boolean canRead(): Tests whether the application can read the file denoted by this abstract pathname.
* boolean canWrite(): Tests whether the application can modify the file denoted by this abstract pathname.
* boolean isFile(): Tests whether the file denoted by this abstract pathname is a normal file.

Example1-

import java.io.File;

import java.io.IOException;

public class FileExample {

public static void main(String[] args) {

File file = new File("example.txt");

// Check if the file exists

if (file.exists()) {

System.out.println("File exists.");

} else {

System.out.println("File does not exist.");

}

// Create a new file

try {

if (file.createNewFile()) {

System.out.println("File created.");

} else {

System.out.println("File already exists.");

}

} catch (IOException e) {

e.printStackTrace();

}

// Get file information

if (file.exists()) {

System.out.println("File name: " + file.getName());

System.out.println("Absolute path: " + file.getAbsolutePath());

System.out.println("Writable: " + file.canWrite());

System.out.println("Readable: " + file.canRead());

System.out.println("File size in bytes: " + file.length());

}

// Delete the file

if (file.delete()) {

System.out.println("File deleted.");

} else {

System.out.println("Failed to delete the file.");

}

}

}

Example 2- import java.io.\*;

public class FileReadWriteExample {

public static void main(String[] args) {

String fileName = "example.txt";

String fileContent = "Hello, this is a test file.\nWelcome to file handling in Java!";

// Create and write to the file

try (FileWriter writer = new FileWriter(fileName)) {

writer.write(fileContent);

System.out.println("File written successfully.");

} catch (IOException e) {

System.err.println("An error occurred while writing to the file.");

e.printStackTrace();

}

// Read from the file

try (FileReader reader = new FileReader(fileName);

BufferedReader bufferedReader = new BufferedReader(reader)) {

String line;

System.out.println("File content:");

while ((line = bufferedReader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

System.err.println("An error occurred while reading from the file.");

e.printStackTrace();

}

}

}

Or

import java.io.IOException;

import java.nio.file.\*;

import java.util.List;

public class FilesWriteExample {

public static void main(String[] args) {

String fileName = "example.txt";

String fileContent = "Hello, this is a test file.\nWelcome to file handling in Java!";

// Create and write to the file using Files.write()

try {

Files.write(Paths.get(fileName), fileContent, StandardOpenOption.CREATE);

System.out.println("File written successfully using Files.write().");

} catch (IOException e) {

System.err.println("An error occurred while writing to the file.");

e.printStackTrace();

}

}

}

## **Reading data from a file**

**1.Using FileReader and BufferedReader**

* FileReader is used to read character files.
* BufferedReader provides buffering for FileReader, which improves efficiency by reducing the number of I/O operations.

import java.io.\*;

public class BufferedReaderExample {

public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new FileReader("example.txt")))

//ortry{ FileReader fileReader = new FileReader("example.txt");

//BufferedReader bufferedReader = new BufferedReader(fileReader);

{

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**2. Using FileInputStream and InputStreamReader**

* FileInputStream reads bytes from a file.
* InputStreamReader converts byte streams to character streams.

import java.io.\*;

public class FileInputStreamExample {

public static void main(String[] args) {

try (InputStreamReader isr = new InputStreamReader(new FileInputStream("example.txt"));

BufferedReader reader = new BufferedReader(isr)) {

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**3. Using Scanner**

* Scanner can be used to read from a file, and it provides parsing capabilities for primitive types and strings using regular expressions.

import java.io.\*;

import java.util.Scanner;

public class ScannerExample {

public static void main(String[] args) {

try (Scanner scanner = new Scanner(new File("example.txt"))) {

while (scanner.hasNextLine()) {

System.out.println(scanner.nextLine());

}

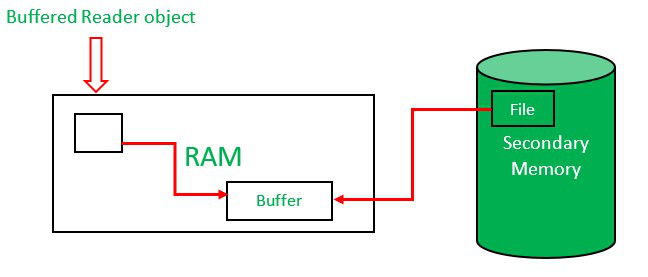
} catch (FileNotFoundException e) {

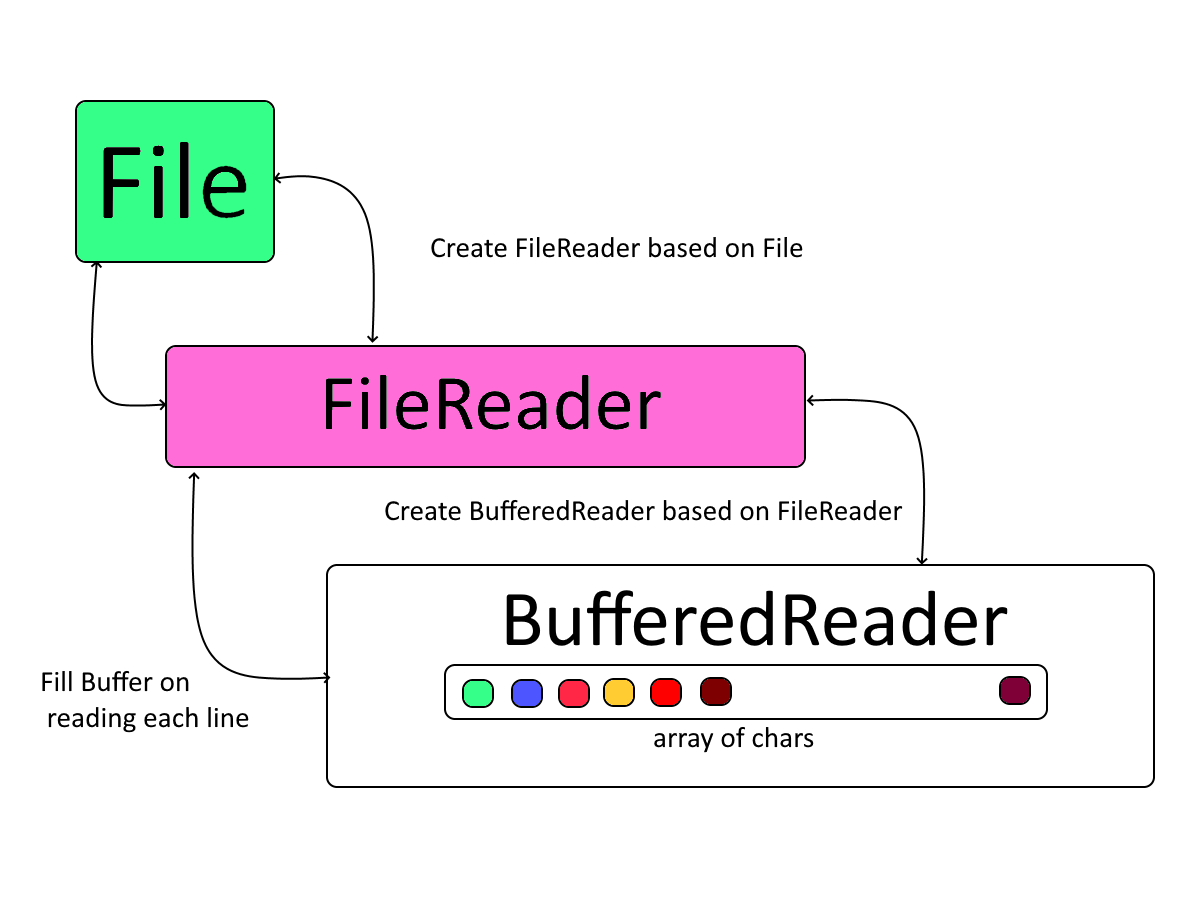
e.printStackTrace();

}

}

}

****

****

**Program without bufferReder**

import java.io.FileReader;

import java.io.IOException;

public class FileReaderExample {

public static void main(String[] args) {

FileReader fileReader = null;

try {

// Initialize FileReader

fileReader = new FileReader("example.txt");

// Read and print each character from the file

int character;

while ((character = fileReader.read()) != -1) {

System.out.print((char) character);

}

} catch (IOException e) {

e.printStackTrace();

} finally {

// Close FileReader

if (fileReader != null) {

try {

fileReader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

**Writing a CSV**

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

public class CsvWriteExample {

public static void main(String[] args) {

String csvFile = "example.csv";

try (PrintWriter writer = new PrintWriter(new FileWriter(csvFile))) {

// Write header

writer.println("Name, Age, Country");

// Write data

writer.println("John Doe, 30, USA");

writer.println("Jane Smith, 25, UK");

writer.println("Mike Brown, 35, Canada");

System.out.println("CSV file written successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Reading CSV**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class CsvReadExample {

public static void main(String[] args) {

String csvFile = "example.csv";

String line;

String csvSplitBy = ",";

try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {

while ((line = br.readLine()) != null) {

// Use comma as separator

String[] data = line.split(csvSplitBy);

System.out.println("Name: " + data[0] + " , Age: " + data[1] + " , Country: " + data[2]);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}