UNIT IV – FILE I/O

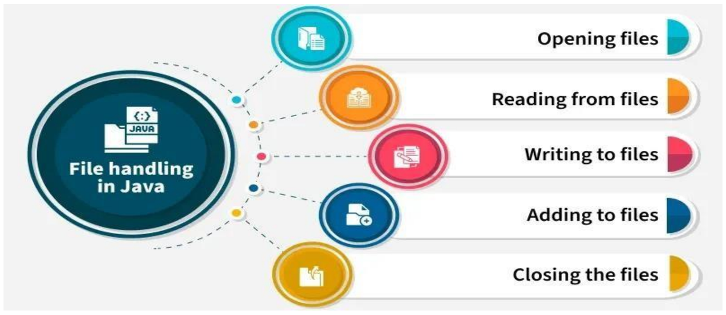
Introduction to File I/O:

- In Java, file handling means working with files like creating them, reading data, writing data or deleting them.
- It helps a program save and use information permanently on the computer.

Why File Handling is Required?

- To store data permanently instead of keeping it only in memory.
- To read and write data from/to files for later use.
- To share data between different programs or systems.
- To organize and manage large data efficiently.

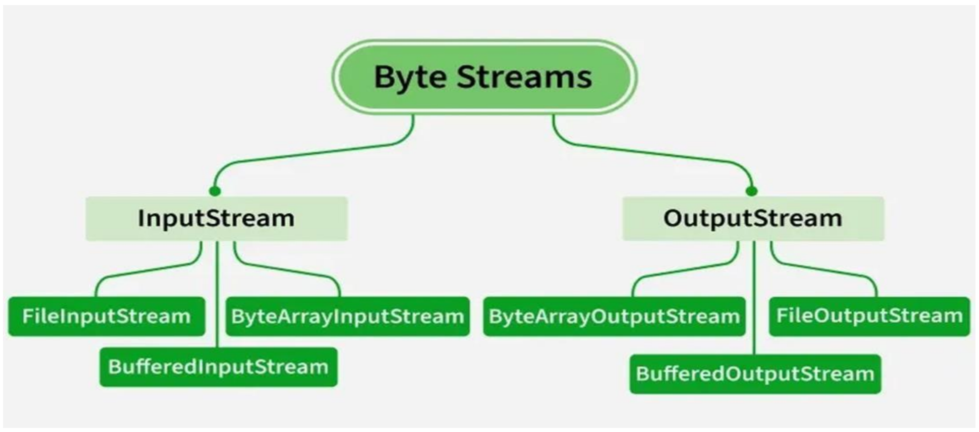
To support file handling, Java provides the File class in the java.io package.



I/O Streams in Java:

- In Java, I/O streams are the fundamental mechanism for handling input and output operations.
- They provide a uniform way to read data from various sources (files, network, memory) and write data to different destinations.
- Java I/O streams are categorized into two main types based on the type of data they handle:
  - ✓ Byte Streams
  - ✓ Character Streams

Byte Streams:



- In Java, Byte Streams are used to handle raw binary data such as images, audio files, videos or any non-text file.They work with data in the form of 8-bit bytes.

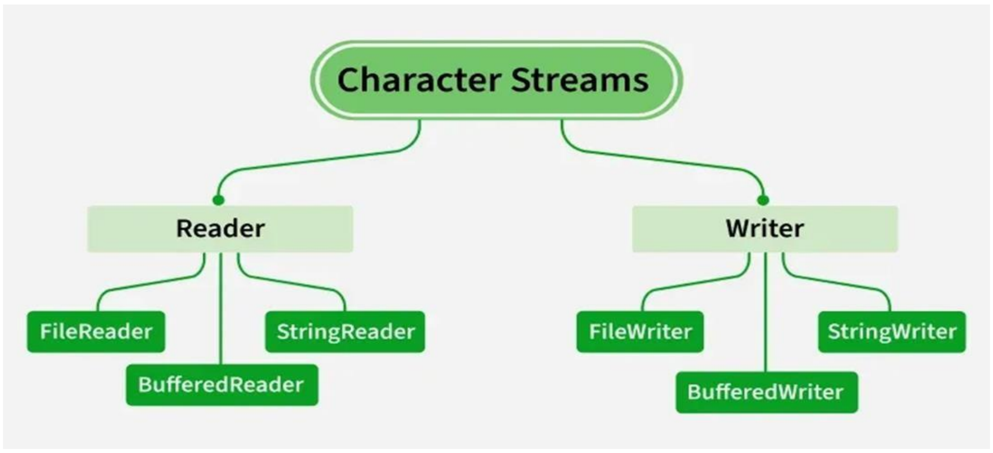
The two main abstract classes for byte streams are:

- **InputStream:** for reading data (input)
- **OutputStream:** for writing data (output)

- Since abstract classes cannot be used directly, we use their implementation classes to perform actual I/O operations.
- **FileInputStream:** reads raw bytes from a file.
- **FileOutputStream:** writes raw bytes to a file.
- **BufferedInputStream / BufferedOutputStream:** use buffering for faster performance.
- **ByteArrayInputStream:** reads data from a byte array as if it were an input stream.
- **ByteArrayOutputStream:** writes data into a byte array, which grows automatically.

Character Streams:

- In Java, Character Streams are used to handle text data.
- They work with 16-bit Unicode characters, making them suitable for international text and language support.



The two main abstract classes for character streams are:

- **Reader:** Base class for all character-based input streams (reading).
- **Writer:** Base class for all character-based output streams (writing).
- Since abstract classes cannot be used directly, we use their implementation classes to perform actual I/O operations.

- Unlike byte streams (FileInputStream, FileOutputStream), these work with character data (Unicode support), making them suitable for text files.

FileReader:

- FileReader is a character stream class for reading data from files.
- It reads text data (characters) from files and decodes it into Unicode characters.

Key Points:

- Belongs to java.io package.
- Reads one character at a time or into a character array.
- Used mainly for text files (not binary files).
- Inherits from InputStreamReader.

Constructors:

- FileReader(String fileName)
- FileReader(File file)
- Both constructors throw FileNotFoundException if the file doesn’t exist.

Common Methods:

Method	**Description**
int read()	Reads a single character, returns -1 if end of file (EOF).
int read(char[] cbuf)	Reads characters into an array.
void close()	Closes the stream to release resources.

Example: Reading from a file

```
import java.io.*;
public class FileReaderExample
{
    public static void main(String[] args)
    {
        try
         {
            FileReader fr = new FileReader("sample.txt");
            int i;
            while ((i = fr.read()) != -1)
            { // Read till EOF
```

```
                System.out.print((char) i);
            }
            fr.close();
        }
        catch (IOException e)
        {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

FileWriter:

FileWriter is a **character stream class** for writing character data to files.

Key Points:

- Belongs to java.io package.
- Writes text data (characters) to files.
- Can **overwrite** the existing file or **append** to it.
- Inherits from OutputStreamWriter.

Constructors:

- FileWriter(String fileName)   // overwrites file
- FileWriter(String fileName, boolean append) // append if true
- FileWriter(File file)
- FileWriter(File file, boolean append)

Common Methods:

Method	**Description**
void write(int c)	Writes a single character.
void write(String str)	Writes a string.
void write(char[] cbuf)	Writes an array of characters.
void flush()	Flushes the stream (forces data to file).
void close()	Closes the stream.

Example: Writing to a file

```
import java.io.*;
public class FileWriterExample
{
        public static void main(String[] args)
        {
            try
            {
                FileWriter fw = new FileWriter("sample.txt");
                fw.write("Hello, this is a FileWriter example.\n");
                fw.write("FileWriter writes character data
                easily!"); fw.close();
                System.out.println("Data written successfully.");
            }
            catch (IOException e)
            {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }
```

Comparison: FileReader vs FileWriter

Feature	**FileReader**	**FileWriter**
Purpose	Reading characters	Writing characters
Package	java.io	java.io
Inheritance	Extends InputStreamReader	Extends OutputStreamWriter
Input/Output	Reads text from files	Writes text to files
Usage	Reading text-based files	Writing/modifying text files

BufferedReader:

- A class that reads text from a character input stream efficiently by buffering characters.
- It also provides the handy method readLine() to read a line of text at once.

Package: java.io

Key Features:

- Reads characters, arrays, and lines.
- Stores input temporarily in a buffer, reducing disk access.
- Provides the **readLine()** method (not available in FileReader).

  - Must be wrapped around a Reader object (e.g., FileReader).

Constructor:

- BufferedReader(Reader in)
- BufferedReader(Reader in, int size) // custom buffer size

Common Methods:

Method	**Description**
int read()	Reads a single character.
int read(char[] cbuf, int off, int len)	Reads characters into part of an
array. String readLine()	Reads a line of text, returns null at
EOF.	
void close()	Closes the stream.

```
import
java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class BufferedReaderExample
{
    public static void main(String[] args)
    {
        try (BufferedReader reader = new BufferedReader(new
             FileReader("example.txt")))
        {
            String line;
            while ((line = reader.readLine()) != null)
            {
                System.out.println(line);
            }
        }
        catch (IOException e)
        {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

This approach reads the file line by line, which is more efficient and often more practical for text file processing.

BufferedWriter:

- A class that writes text to a character output stream efficiently by buffering characters.
- It reduces the number of I/O operations by collecting characters into a buffer before writing them to the file.

Package: java.io

Key Features:

- Writes characters, arrays, and strings.
- Improves efficiency by buffering data before writing.
- Provides the **newLine()** method to write system-dependent line separators.
- Must be wrapped around a Writer object (e.g., FileWriter).

Constructor:

- BufferedWriter(Writer out)
- BufferedWriter(Writer out, int size) // custom buffer size

Common Methods:

Method	**Description**
void write(int c)	Writes a single character.
void write(String s)	Writes a string.
void write(char[] cbuf, int off, int len)	Writes a portion of a character array.
void newLine()	Writes a new line.
void flush()	Forces data in buffer to be written immediately.
void close()	Closes the stream.

```
import java.io.BufferedWriter;
import java.io.FileWriter; import java.io.IOException;
public class BufferedWriterExample
{
    public static void main(String[] args)
    {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt")))
```

```
        {
            writer.write("Hello, Buffered I/O!"); writer.newLine();
            writer.write("This is another line.");
        }
        catch (IOException e)
        {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

BufferedWriter provides methods like newLine() for adding line breaks, which can make your code more readable.

Comparison: BufferedReader vs BufferedWriter

Feature	**BufferedReader**	**BufferedWriter**
Purpose	Reading text efficiently	Writing text efficiently
Special Method	readLine()	newLine()
Input/Output	Reads text from a file	Writes text to a file
Efficiency	Uses buffer to reduce disk reads	Uses buffer to reduce disk writes