Unit IV:

Input/output Programming: Basics Streams, Byte and Character Stream, predefined streams, Reading and writing from console and files. Using Standard Java Packages (Lang, util, io, net).

Event Handling: Different Mechanism, the Delegation Event Model, Event Classes, Event Listener Interfaces, Adapter and Inner Classes.

File Handling in Java

- File handling refers to working with the file in java. Reading files & writing into java files is known as file handling in java.
- The File is a container that can contain different types of information. The file can contain text, images, videos, tables, etc.
- In java, the File class enables us to work with different types of files. File class is a member of **the java.io packages**. Java provides various methods to read, write, update & delete files.

Types of Operation

Different types of operation which can be performed on a file is given below:

Different types of operation

- Creating a file
- Updating a file
- Deleting a file
- Uploading a file to a specific location
- Opening file
- Closing file

Streams

- Java programs perform I/O through streams. A stream is an abstraction that either produces or consumes information.
- A stream is linked to a physical device by the Java I/O system.
- All streams behave in the same manner, even if the actual physical devices to which they are linked differ.
- Thus, the same I/O classes and methods can be applied to different types of devices. This means that an input stream can abstract many different kinds of input: from a disk file, a keyboard, or a network socket.
- Likewise, an output stream may refer to the console, a disk file, or a network connection. Streams are a clean way to deal with input/ output without having every part of your code understand the difference between a keyboardand a network, for example.
- Java implements streams within class hierarchies defined in the **java.io package.**

Byte Streams and Character Streams

- Java defines two types of streams: **byte and character.**
- Byte streams provide a convenient means for handling input and output of bytes.
- Byte streams are used, for example, when reading or writing binary data.
- Character streams provide a convenient means for handling input and output of characters.

Byte Streams in Java

- Byte streams in Java are designed to provide a convenient way for handling the input and output of bytes (i.e., units of 8-bits data). We use them for reading or writing to **binary data I/O.**
- Byte streams are especially used when we are working **with binary files such as executable files, image files, and files in low-level file formats such as .zip, .class, .obj, and .exe.**
- **Binary files are those files that are machine readable. For example, a Java class file is an extension of ".class" and humans cannot read it.**

- Byte streams that are used for reading are called **input streams and for writing are called output stream**s. They are represented by the abstract classes of **InputStream and OutputStream in Java.**

Character Streams in Java

- Character streams in Java are designed for handling the input and output of characters. They use 16-bit Unicode characters.
- **Character streams are more efficient than byte streams. They are mainly used for reading or writing to character or text-based I/O such as text files, text documents, XML, and HTML files.**
- Text files are those files that are human readable. For example, a .txt file that contains human-readable text. This file is created with a text editor such as Notepad in Windows.
- Character streams that are used **for reading are called readers and for writing are called writers**. They are represented by the abstract classes of Reader and Writer in Java.

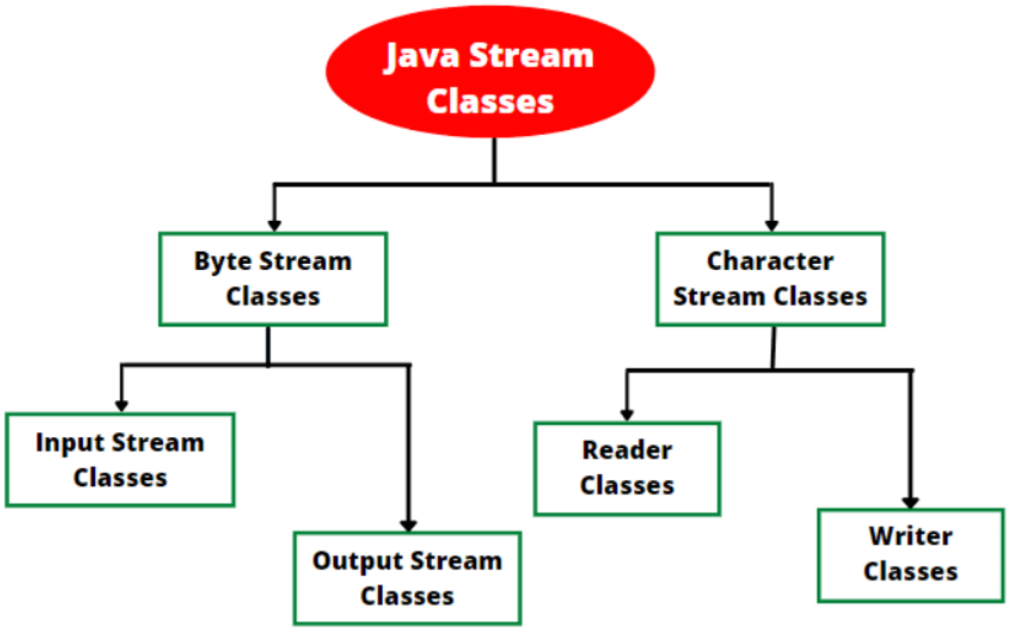


Fig: Classification of Java Stream Classes

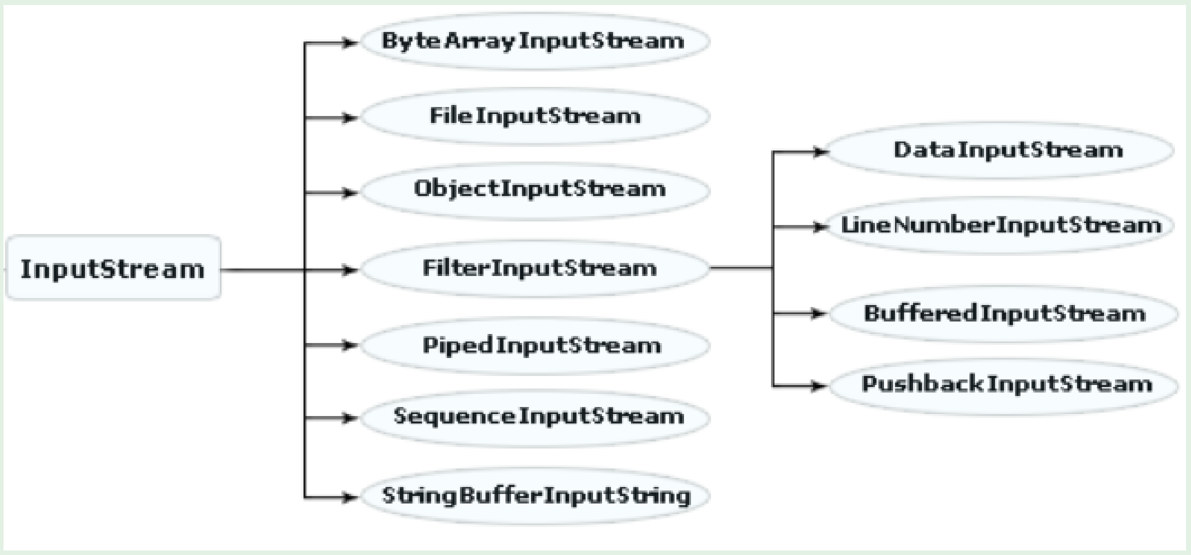


Figure 5: Hierarchy of InputStream Classes

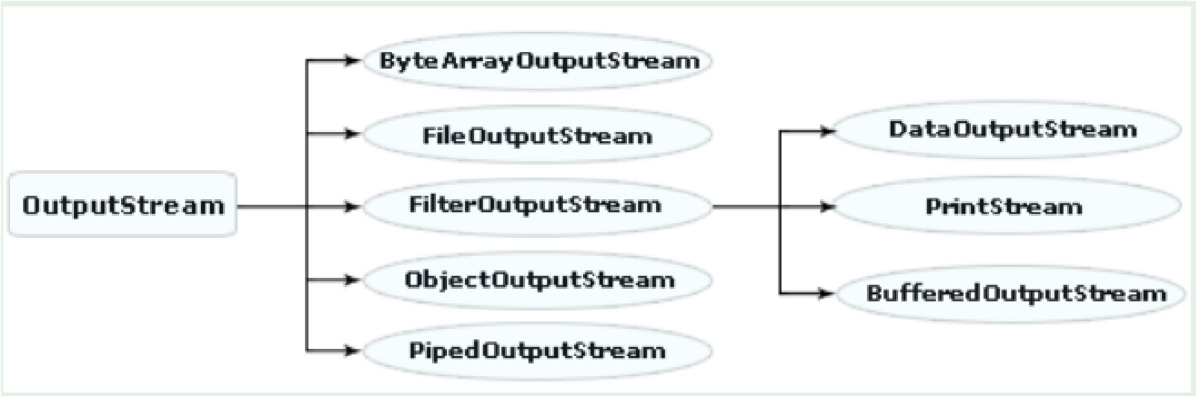
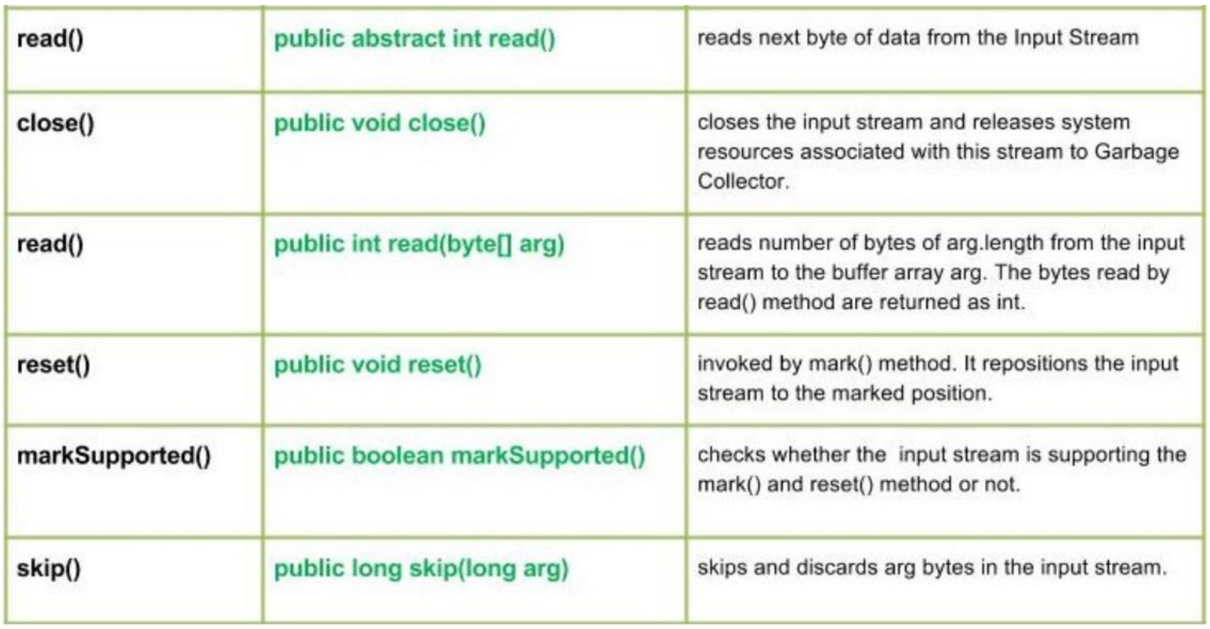


Figure 6: Hierarchy of OutputStream Classes

Some important Charcter stream classes

Stream class	Description
BufferedReader	Handles buffered input stream.
BufferedWriter	Handles buffered output stream.
FileReader	Input stream that reads from file.
FileWriter	Output stream that writes to file.
InputStreamReader	Input stream that translate byte to character
OutputStreamReader	Output stream that translate character to byte.
PrintWriter	Output Stream that contain `print()` and `println()` method.
Reader	Abstract class that define character stream input
Writer	Abstract class that define character stream output

Functions of InputStream



read()	public abstract int read()	reads next byte of data from the Input Stream
close()	public void close()	closes the input stream and releases system resources associated with this stream to Garbage Collector.
read()	public int read(byte[] arg)	reads number of bytes of arg.length from the input stream to the buffer array arg. The bytes read by read() method are returned as int.
reset()	public void reset()	invoked by mark() method. It repositions the input stream to the marked position.
markSupported()	public boolean markSupported()	checks whether the input stream is supporting the mark() and reset() method or not.
skip()	public long skip(long arg)	skips and discards arg bytes in the input stream.

Methods of OutputStream

The OutputStream class provides different methods that are implemented by its subclasses. Here are some of the methods:

write() - writes the specified byte to the output stream

write(byte[] array) - writes the bytes from the specified array to the output stream

flush() - forces to write all data present in output stream to the destination

close() - closes the output stream

Example- copy the content of onefile to another in java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

class FileTest {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating object of File class
        // Passing files from directory of local machine
        File file = new File(  "/Users/mayanksolanki/Desktop/demo.rtf");
        File oFile = new File(
    "/Users/mayanksolanki/Desktop/outputdemo.rtf");

        // Now creating object of FileInputStream
        // Here they are variables
        FileInputStream fis = null;
        FileOutputStream fos = null;

        try {
            // Now we make them as objects of both classes
            // and passed reference of file in directory
            fis = new FileInputStream(file);
            fos = new FileOutputStream(oFile);
        }

        // Catch block to handle exceptions
        catch (FileNotFoundException e) {

            // Display message if exception occurs
            // File Not Found or Path is Incorrect
            System.out.println(e.printStackTrace());
        }

        try {

            // Now let us check how many bytes are available
            // inside content of file
            fis.available();
        }

        catch (Exception e) {
            e.printStackTrace();
        }

        // Using while loop to
        // write over outputdemo file
        int i = 0;
        while (i = fis.read() != -1) {
            fos.write(i);
        }
        // It will execute no matter what to close connections which is always
good practice
        finally
        {
             if (fis != null😉 {
                fis.clsoe();
            }

            // For output stream
            if (fos != null) {
                fos.close();
            }
        }
    }
}
```