



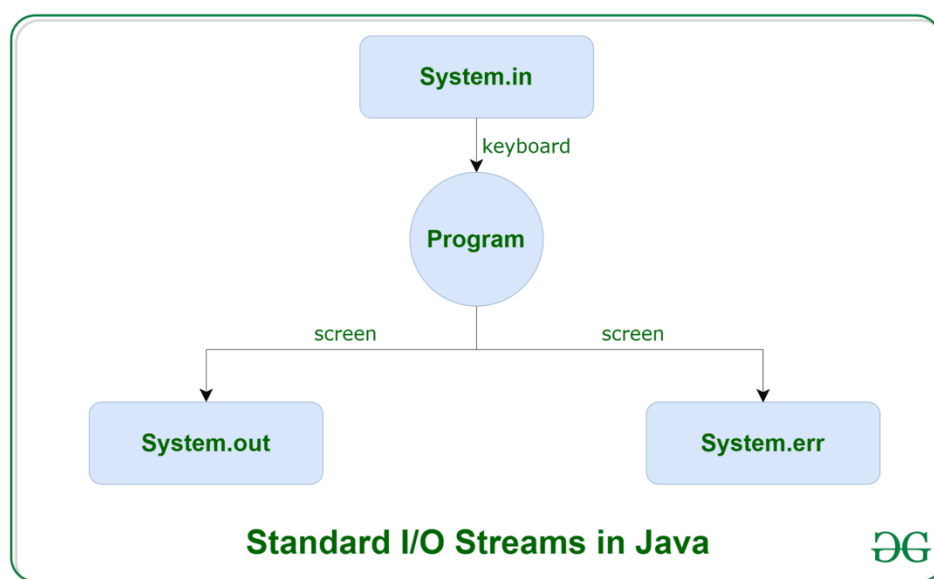
## Java IO : Input-output in Java with Examples

Last Updated : 16 Jan, 2025

[Java](#) brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all the types of objects, data-types, characters, files etc. to fully execute the I/O operations.



Before exploring various input and output streams let's look at **3 standard or default streams** that Java has to provide which are also most common in use:



### System.in

This is the **standard input stream**([System.in](#)) that is used to read characters from the keyboard or any other standard input device.

## System.out

This is the **standard output stream**([System.out](#)) that is used to produce the result of a program on an output device like the computer screen. Here is a list of the various print functions that we use to output statements:

– **print()**: This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.

### Syntax:

```
System.out.print(parameter);
```

### Example:

```
// Java code to illustrate print()  
import java.io.*;  
  
class Demo_print {  
    public static void main(String[] args)  
    {  
  
        // using print()  
        // all are printed in the  
        // same line  
        System.out.print("GfG! ");  
        System.out.print("GfG! ");  
        System.out.print("GfG! ");  
    }  
}
```

### Output

GfG! GfG! GfG!

– **println():** This method in Java is also used to display a text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.

### Syntax:

```
System.out.println(parameter);
```

### Example:

```
// Java code to illustrate println()
import java.io.*;

class Demo_print {
    public static void main(String[] args)
    {
        // using println()
        // all are printed in the
        // different line
        System.out.println("GfG! ");
        System.out.println("GfG! ");
        System.out.println("GfG! ");
    }
}
```

### Output

```
GfG!
GfG!
GfG!
```

– **printf():** This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments. This is used to format the output in Java.

### Example:

```
// A Java program to demonstrate
// working of printf() in Java
class JavaFormatter1 {

    public static void main(String[] args) {
        int x = 100;
```

```

// Printing a simple integer
System.out.printf("Printing simple integer:
                  x = %d\n", x);

// Printing a floating-point
// number with precision
System.out.printf("Formatted with precision:
                  PI = %.2f\n", Math.PI);

float n = 5.2f;

// Formatting a float to 4 decimal places
System.out.printf("Formatted to specific width:
                  n = %.4f\n", n);

n = 2324435.3f;

// Right-aligning and formatting a
// float to 20-character width
System.out.printf("Formatted to right margin:
                  n = %20.4f\n", n);
}
}

```

## Output

```

Printing simple integer: x = 100
Formatted with precision: PI = 3.14
Formatted to specific width: n = 5.2000
Formatted to right margin: n =                2324435.2500

```

## System.err Example

To demonstrate the usage of System.err with print():

```

// Java Program demonstrating System.err
public class Main {

    public static void main(String[] args) {

        // Using print()
        System.err.print("This is an error message
                        using print().\n");

        // Using println()
        System.err.println("This is another error
                           message using println().");

        // Using printf()
        System.err.printf("Error code: %d, Message:
                          %s\n", 404, "Not Found");
    }
}

```

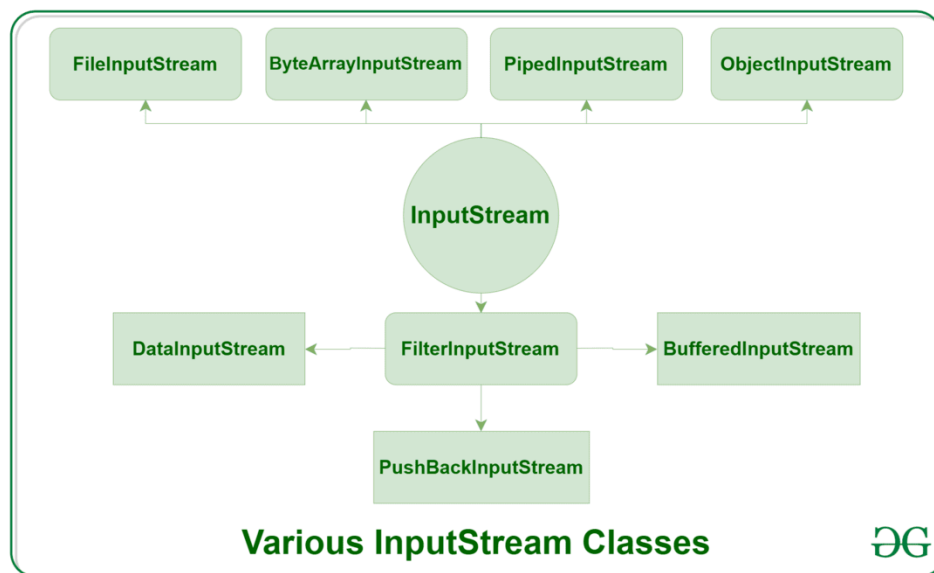
## Output:

```
This is an error message using print().  
This is another error message using println().  
Error code: 404, Message: Not Found
```

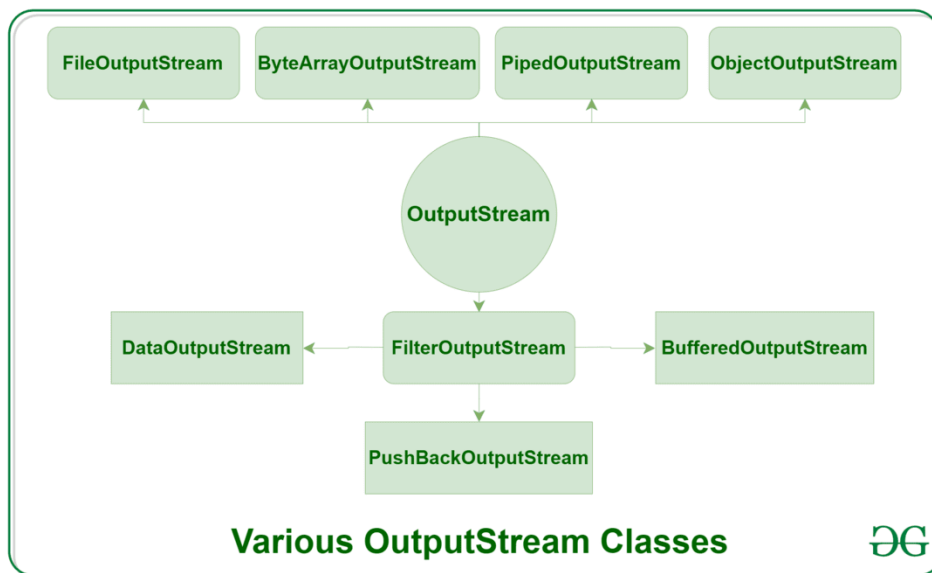
## Types of Streams

Depending on the type of operations, streams can be divided into two primary classes:

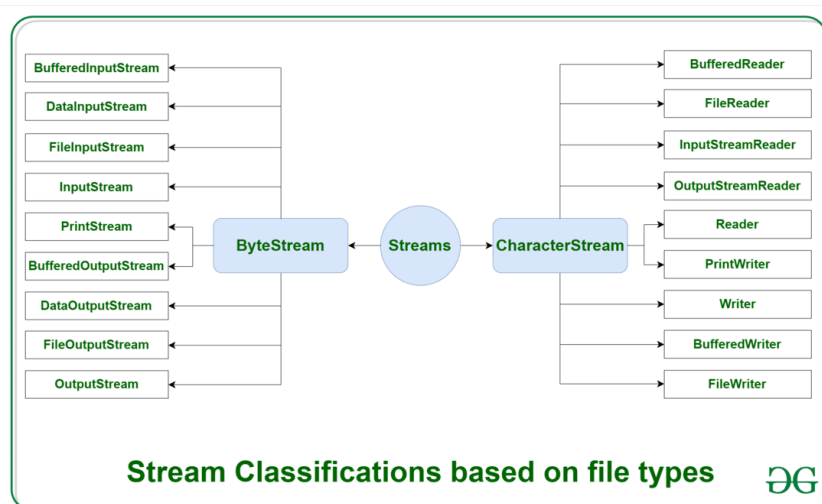
**Input Stream:** These streams are used to read data that must be taken as an input from a source array or file or any peripheral device. For eg., `FileInputStream`, `BufferedInputStream`, `ByteArrayInputStream` etc.



**Output Stream:** These streams are used to write data as outputs into an array or file or any output peripheral device. For eg., `FileOutputStream`, `BufferedOutputStream`, `ByteArrayOutputStream` etc.



Depending on the types of file, Streams can be divided into two primary classes which can be further divided into other classes as can be seen through the diagram below followed by the explanations.



## ByteStream

This is used to process data byte by byte (8 bits). Though it has many classes, the FileInputStream and the FileOutputStream are the most popular ones. The FileInputStream is used to read from the source and FileOutputStream is used to write to the destination. Here is the list of various ByteStream Classes:

Stream class	Description
<a href="#">BufferedInputStream</a>	It is used for Buffered Input Stream.

Stream class	Description
<a href="#"><u>DataInputStream</u></a>	It contains method for reading java standard datatypes.
<a href="#"><u>FileInputStream</u></a>	This is used to reads from a file
<a href="#"><u>InputStream</u></a>	This is an abstract class that describes stream input.
<a href="#"><u>PrintStream</u></a>	This contains the most used print() and println() method
<a href="#"><u>BufferedOutputStream</u></a>	This is used for Buffered Output Stream.
<a href="#"><u>DataOutputStream</u></a>	This contains method for writing java standard data types.
<a href="#"><u>FileOutputStream</u></a>	This is used to write to a file.
<a href="#"><u>OutputStream</u></a>	This is an abstract class that describe stream output.

### Example:

```
// Java Program illustrating the
// Byte Stream to copy
// contents of one file to another file.
import java.io.*;
public class BStream {
    public static void main(
        String[] args) throws IOException
    {

        FileInputStream sourceStream = null;
        FileOutputStream targetStream = null;

        try {
            sourceStream
                = new FileInputStream("sourcefile.txt");
            targetStream
                = new FileOutputStream("targetfile.txt");

            // Reading source file and writing
            // content to target file byte by byte
```



```
        int temp;
        while ((
            temp = sourceStream.read())
            != -1)
            targetStream.write((byte)temp);
    }
    finally {
        if (sourceStream != null)
            sourceStream.close();
        if (targetStream != null)
            targetStream.close();
    }
}
```

### Output:

Shows contents of file test.txt

### CharacterStream

In Java, characters are stored using Unicode conventions (Refer this for details). Character stream automatically allows us to read/write data character by character. Though it has many classes, the `FileReader` and the `FileWriter` are the most popular ones. `FileReader` and `FileWriter` are character streams used to read from the source and write to the destination respectively. Here is the list of various `CharacterStream` Classes:

Stream class	Description
<a href="#">BufferedReader</a>	It is used to handle buffered input stream.
<a href="#">FileReader</a>	This is an input stream that reads from file.
<a href="#">InputStreamReader</a>	This input stream is used to translate byte to character.
<code>OutputStreamReader</code>	This output stream is used to translate character to byte.
<a href="#">Reader</a>	This is an abstract class that define character stream input.



Stream class	Description
<a href="#">PrintWriter</a>	This contains the most used print() and println() method
<a href="#">Writer</a>	This is an abstract class that define character stream output.
<a href="#">BufferedWriter</a>	This is used to handle buffered output stream.
<a href="#">FileWriter</a>	This is used to output stream that writes to file.

### Example:

```
// Java Program illustrating that
// we can read a file in a human-readable
// format using FileReader

// Accessing FileReader, FileWriter,
// and IOException
import java.io.*;

public class Geeks
{
    public static void main(String[] args) throws IOException
    {
        FileReader sourceStream = null;

        try {
            sourceStream = new FileReader("test.txt");

            // Reading sourcefile and
            // writing content to target file
            // character by character.
            int temp;

            while (( temp = sourceStream.read())!= -1 )
                System.out.println((char)temp);
        }
        finally {

            // Closing stream as no longer in use
            if (sourceStream != null)
                sourceStream.close();
        }
    }
}
```