Java I/O

System is a predefined class within java.lang that encapsulates different aspects of the JVM and provides access to the underlying system and out is the *output stream* connected to the console. The println() method displays the message passed to it on a new line. So when joined together a message is passed to the console and appears on a new line.

But what do we mean by *output stream*? To investigate this we need to take a closer look at the System class which contains three predefined fields named err, in and the field we are most familiar with out. Further inspection of these fields shows that they are public static and so can be used without an instance and are of the following types:

Name	Type	Description	Default I/O Device
err	PrintStream	The "standard" error output stream.	Console
in	InputStream	The "standard" input stream.	Keyboard
out	PrintStream	The "standard" output stream.	Console

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

Let's see the code to print **output and an error** message to the console.

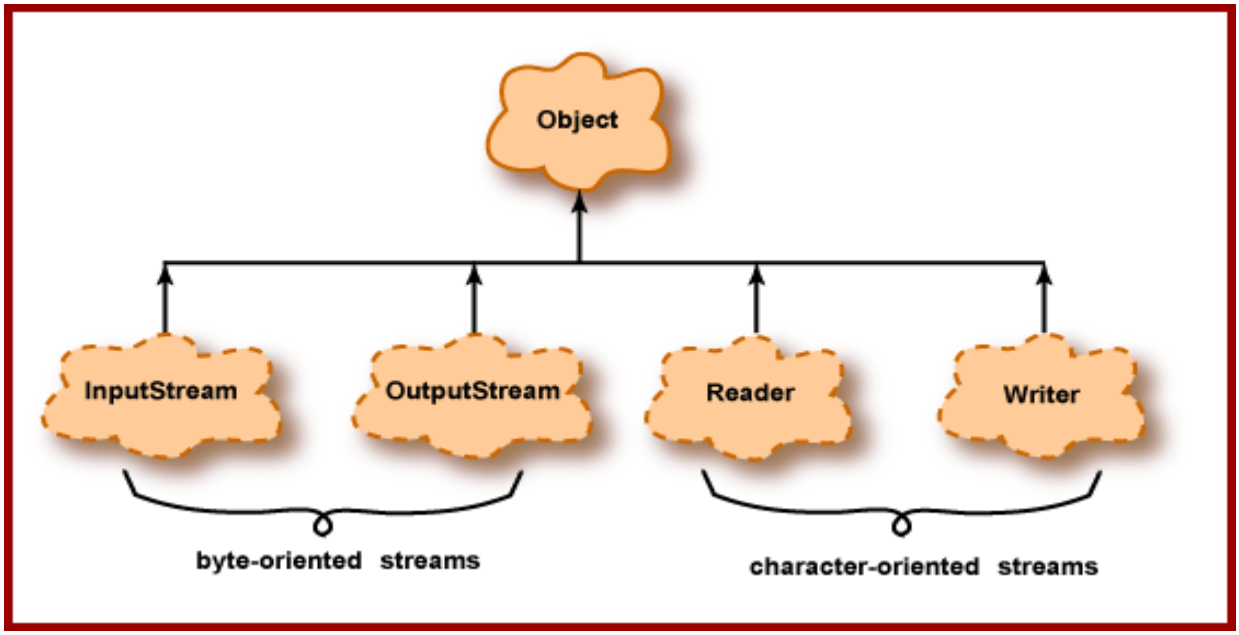
```
1. System.out.println("simple message");
2. System.err.println("error message");
```

So we can now see that these fields are of types InputStream and PrintStream and this is what Java I/O constitutes, class hierarchies of *streams*.

Java is made up of two types of streams, these being *byte streams* used for handling byte input and and output and *character streams* used for handling character input and and output.

The primary reason Java uses streams for I/O is to make Java code independent of the input and output devices involved, thus making the peripheral device used and

thus the coding to use it, transparent.these streams are represented as classes of the java.io package



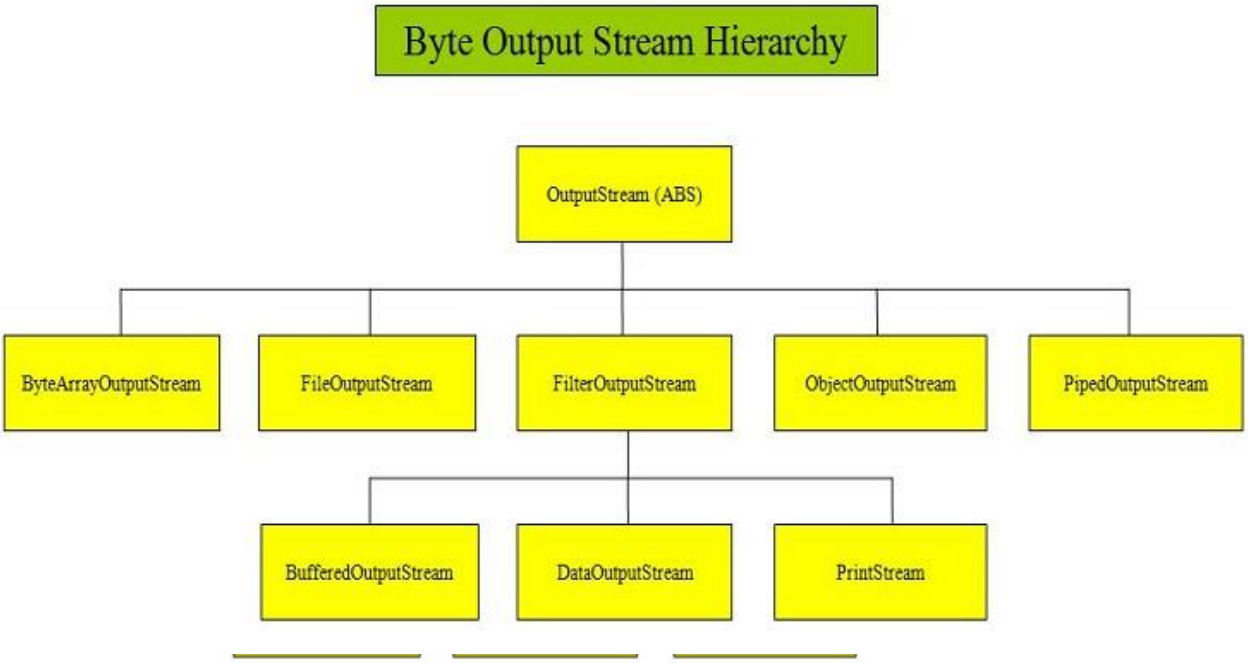
1. Byte Streams

Byte streams are defined within two class hierarchies, one for input and one for output and represent byte stream classes which provide the tools to read and write binary data as a sequence of *bytes*.

- The OutputStream class is the *abstract superclass* of all byte output streams
- The InputStream class is the *abstract superclass* of all byte input streams

Byte Output Stream Hierarchy

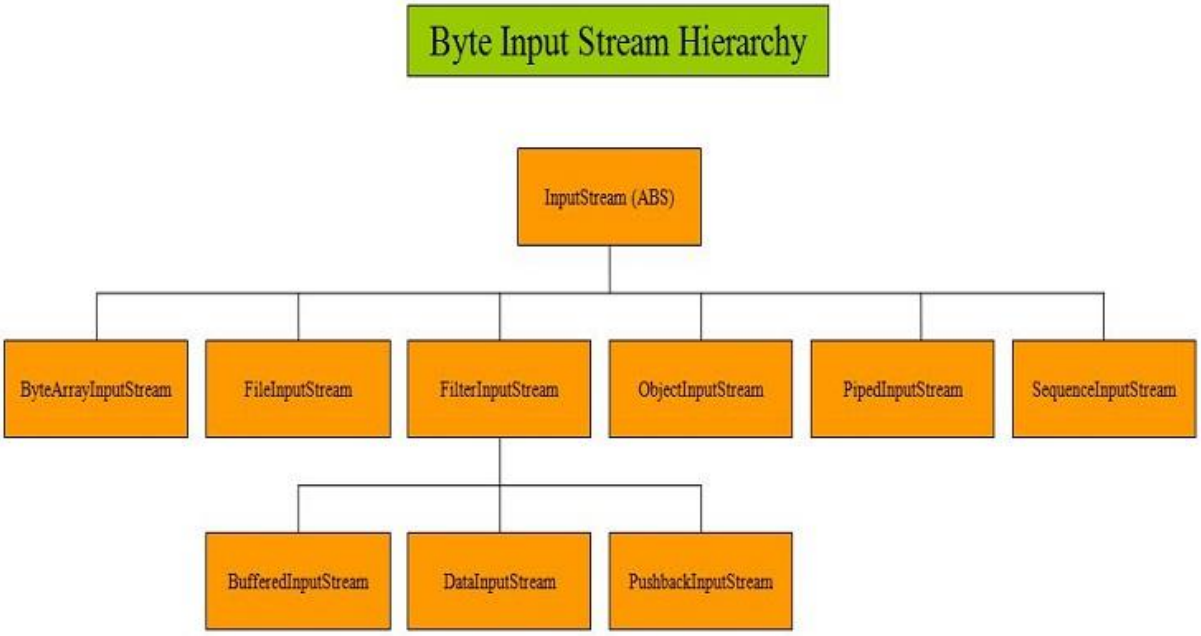
The diagram below shows most of the classes in the *byte output stream hierarchy* of which OutputStream class is the *abstract superclass*. Some subclasses of the FilterOutputStream class are not shown.



Class	Description
`OutputStream`	Abstract byte stream superclass which describes this type of output stream.
`ByteArrayOutputStream`	Output byte stream that writes data to a byte array.
`FileOutputStream`	Output byte stream that writes bytes to a file in a file system.
`FilterOutputStream`	Output byte stream that implements `OutputStream`.
`BufferedOutputStream`	Output byte stream that writes bytes to a buffered output stream.
`DataOutputStream`	Output stream to write Java primitive data types.
`PrintStream`	Convenience output byte stream to add functionality to another stream, an example being to print to the console using `print()` and `println()`.
`ObjectOutputStream`	Output stream to write and serialize objects for reading using `ObjectInputStream`.
`PipedOutputStream`	Piped Output stream that is connected to a piped input stream to create a communication pipe.

Byte Input Stream Hierarchy

The diagram below shows most of the classes in the *byte input stream hierarchy* of which InputStream class is the *abstract superclass*. Some subclasses of the FilterInputStream class are not shown.



Class	Description
`InputStream`	Abstract byte stream superclass which describes this type of input stream.
`ByteArrayInputStream`	Input byte stream that reads bytes from an internal byte array.
`FileInputStream`	Input byte stream that reads bytes from a file in a file system.
`FilterInputStream`	Input byte stream that implements `InputStream`.
`BufferedInputStream`	Input byte stream that reads bytes into an internal buffer before use.
`DataInputStream`	Input stream to reads Java primitive data types.
`PushbackInputStream`	Input byte stream containing functionality to return bytes to the input stream.
`ObjectInputStream`	Input stream to read and deserialize objects output and serialized using `ObjectOutputStream`.
`PipedInputStream`	Piped input stream that is connected to a piped output stream to create a communication pipe.
`SequenceInputStream`	Concatenation of two or more input streams read sequentially.

DataInputStream Class:

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataInputStream class declaration

public class DataInputStream **extends** FilterInputStream **implements** DataInput

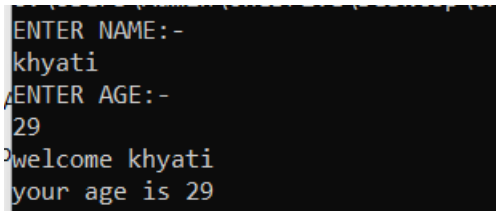
Java DataInputStream class Methods

Method	Description
int read(byte[] b)	It is used to read the number of bytes from the input stream.
int read(byte[] b, int off, int len)	It is used to read **len** bytes of data from the input stream.
int readInt()	It is used to read input bytes and return an int value.
byte readByte()	It is used to read and return the one input byte.
char readChar()	It is used to read two input bytes and returns a char value.
double readDouble()	It is used to read eight input bytes and returns a double value.
boolean readBoolean()	It is used to read one input byte and return true if byte is non zero, false if byte is zero.
int skipBytes(int x)	It is used to skip over x bytes of data from the input stream.
String readUTF()	It is used to read a string that has been encoded using the UTF-8 format.
void readFully(byte[] b)	It is used to read bytes from the input stream and store them into the buffer array.

void readFully(byte[] b, int off, int len)	It is used to read **len** bytes from the input stream.

Example: (taking input from user using DataInputStream Class)

```
import java.io.*;

class InputIo1
{
	public static void main(String args[]) throws IOException
	{

	DataInputStream dis=new DataInputStream(System.in);

	System.out.println("ENTER NAME:-");
	String st=dis.readLine();

	System.out.println("ENTER AGE:-");
	int age= Integer.parseInt(dis.readLine());


	System.out.println("welcome "+st);
	System.out.println("your age is "+age);

	}
}
```



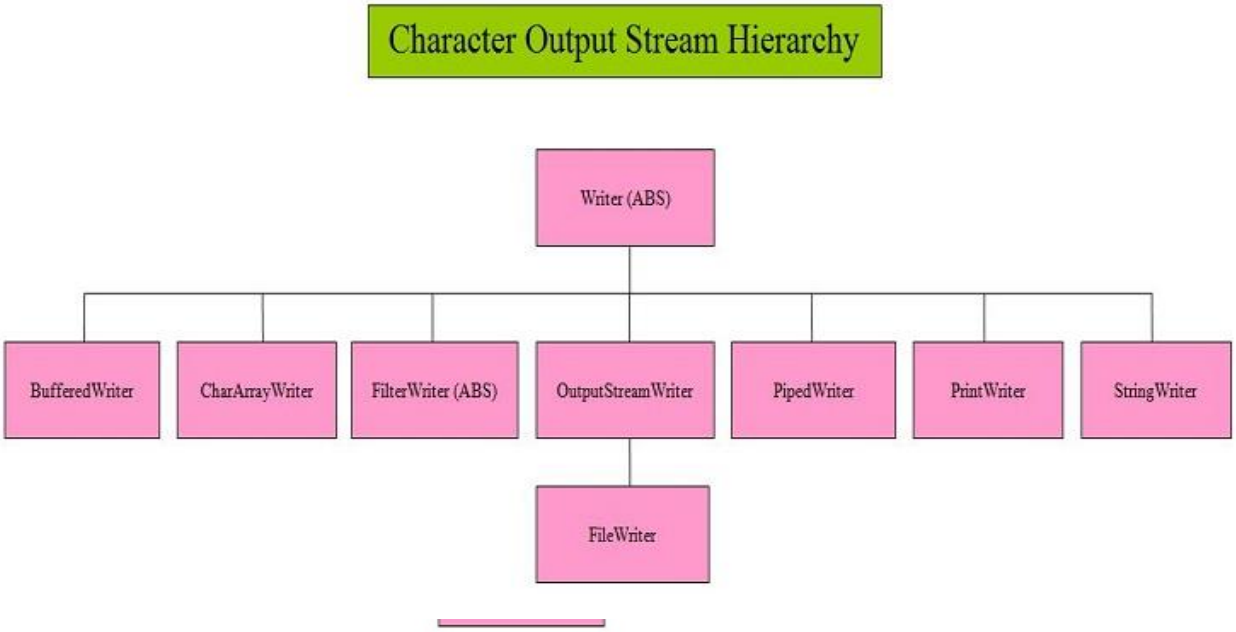
2. Character Stream

Character streams are defined within two class hierarchies, one for input and one for output:

- The Writer class is the *abstract superclass* of all character output streams
- The Reader class is the *abstract superclass* of all character input streams

Character Output Stream Hierarchy

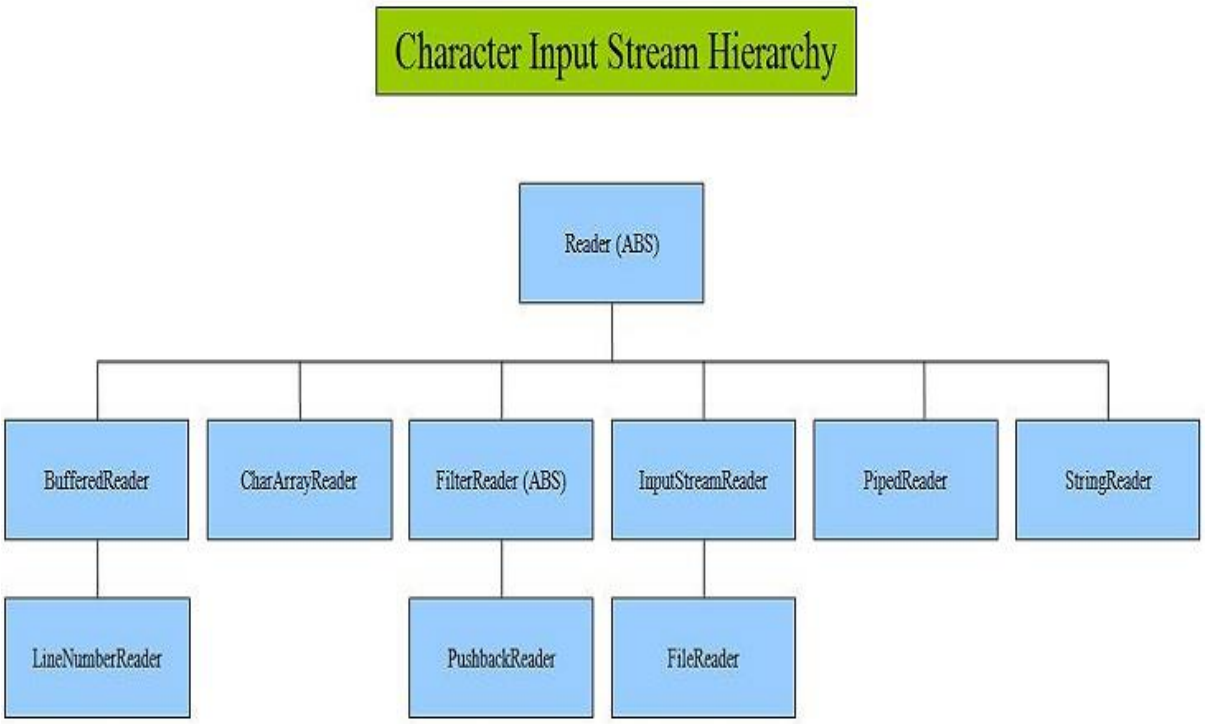
The diagram below shows the classes in the *character output stream hierarchy* of which the Writer class is the *abstract superclass*.:



Class	Description
`Writer`	Abstract character stream superclass which describes this type of output stream.
`BufferedWriter`	Buffered output character stream.
`CharArrayWriter`	Character buffer output stream.
`FilterWriter`	Abstract character stream for writing filtered streams.
`OuputStreamWriter`	Output Stream that acts as a bridge for encoding byte streams from character streams.
`FileWriter`	Output stream for writing characters to a file.
`PipedWriter`	Piped character output stream.
`PrintWriter`	Convenience output character stream to add functionality to another stream, an example being to print to the console using `print()` and `println()`.
`StringWriter`	Output stream for writing characters to a string.

Character Input Stream Hierarchy

The diagram below shows the classes in the *character input stream hierarchy* of which the Reader class is the *abstract superclass*.:



Class	Description
Reader	Abstract character stream superclass which describes this type of input stream.
BufferedReader	Buffered input character stream.
LineNumberReader	Input character stream that keeps a count of line numbers.
CharArrayReader	Character buffer input stream.
FilterReader	Abstract character stream for reading filtered streams.
PushbackReader	Character stream reader containing functionality to return characters to the input stream.
InputStreamReader	Input Stream that acts as a bridge for decoding byte streams into character streams.
FileReader	Input stream for reading characters from a file.
PipedReader	Piped character input stream.
StringReader	Input stream for reading characters from a string.

BufferedReader Class

Java **BufferedReader** class is used to read the text from a character-based input stream. It provides the method **readLine()** to read data line by line. It makes the performance fast. It inherits the Reader class. It is defined in the **java.io** package so, we must import the package at the starting of the program. The disadvantage to use this class is that it is difficult to remember.

To read a number, first, create a constructor of the **BufferedReader** class and parse a **Reader** as a parameter. We have parsed an object of the **InputStreamReader** class. After that, we have invoked the **parseInt()** method of the Integer class and parses the **readLine()** method of the BufferedReader class. The **readLine()** method reads a line of text.

java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

An InputStreamReader is a bridge from byte streams to character streams. It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Example: (taking input from user using BufferedReader Class)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Buffer1
{
   public static void main(String[] args)throws IOException
   {

//creating an object of InputStreamReader class

InputStreamReader read=new InputStreamReader(System.in);

//creating a constructor of the BufferedReader class

BufferedReader br=new BufferedReader(read);

      /*
      //Enter data using BufferReader

       BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
*/


   System.out.println("Enter your name");
```

```
        // Reading data using readLine
        String name = br.readLine();

System.out.print("Enter your age: ");

    // Reading data using readLine
    int age=Integer.parseInt(br.readLine());


        // Printing the read line
    System.out.println("Welcome  "+name);
    System.out.println("Age: "+age);

    }
}
```

```
Enter your name
khyati
Enter your age: 29
Welcome  khyati
Age: 29
```