**Input and Output**

* [Java](https://www.geeksforgeeks.org/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.

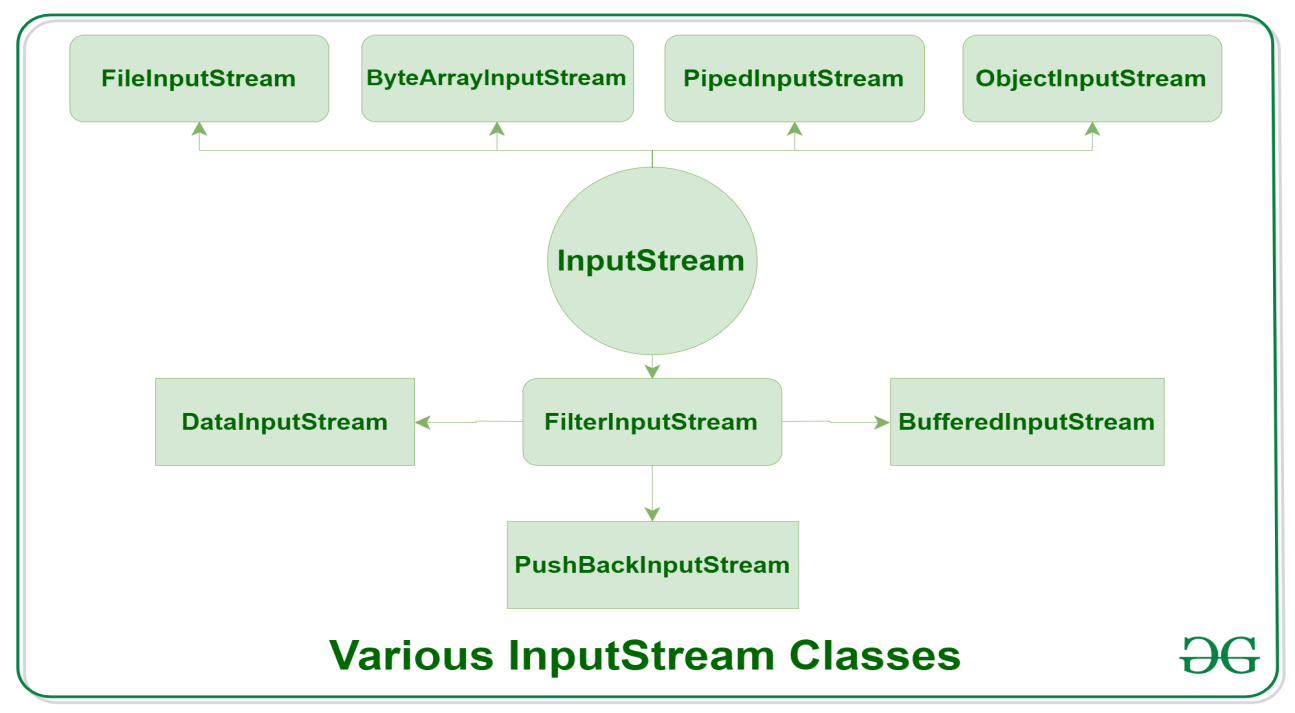


**Standard or Default Streams**

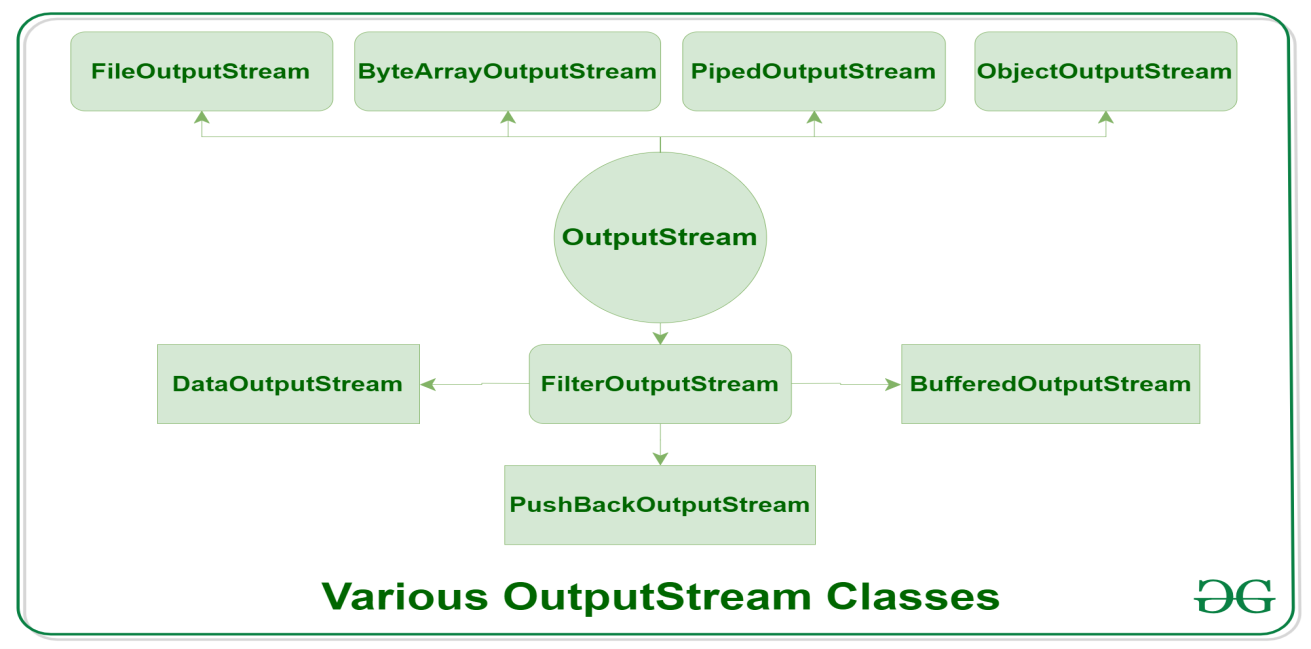
* Java provides **3** most common **standard or default streams,** that are:
  + [System.in](https://www.geeksforgeeks.org/java-lang-system-class-java/)
  + [System.out](https://www.geeksforgeeks.org/java-lang-system-class-java/)
  + [System.err](https://www.geeksforgeeks.org/java-lang-system-class-java/)
* [**System.in**](https://www.geeksforgeeks.org/java-lang-system-class-java/): This is the standard input stream that is used to read characters from the keyboard or any other standard input device.
* [**System.out**](https://www.geeksforgeeks.org/java-lang-system-class-java/): This is the standard output stream 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()**](https://www.geeksforgeeks.org/difference-between-print-and-println-in-java/)**:** 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.
    - [**println()**](https://www.geeksforgeeks.org/difference-between-print-and-println-in-java/)**:** 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.
    - [**printf()**](https://www.geeksforgeeks.org/formatted-output-in-java/)**:** 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.
* [**System.err**](https://www.geeksforgeeks.org/java-lang-system-class-java/)**:** This is the standard error stream that is used to output all the error data that a program might throw, on a computer screen or any standard output device. This stream also uses all the 3 above-mentioned functions to output the error data:
  + print()
  + println()
  + printf()

**Types of Streams:**

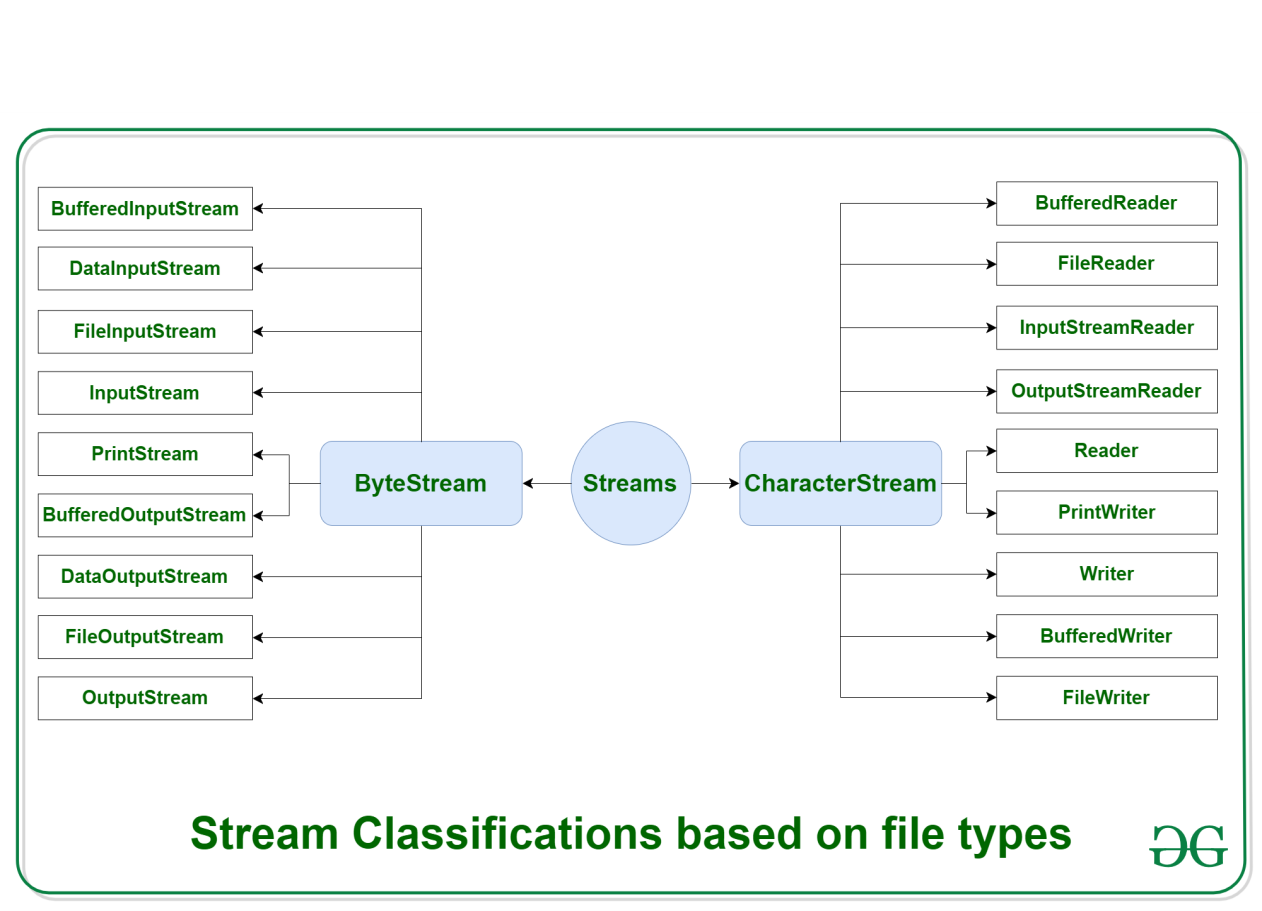
* The types of stream broadly divided into 2 types
  + Depending on the type of operations
  + Depending on the types of file
* **Depending on the type of operations**, the streams can be divided into two primary classes:
  + Input Stream
  + Output Stream
* [**Input Stream:**](https://www.geeksforgeeks.org/java-io-inputstream-class-in-java/) 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 e.g., FileInputStream, BufferedInputStream, ByteArrayInputStream etc.



* [**Output Stream:**](https://www.geeksforgeeks.org/java-io-outputstream-class-java/) These streams are used to write data as outputs into an array or file or any output peripheral device. For e.g., 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**
  + **CharacterStream**



* **ByteStream:** This is used to process a 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 are the list of various ByteStream Classes:

| **Stream class** | **Description** |
| --- | --- |
| [**BufferedInputStream**](https://www.geeksforgeeks.org/java-io-bufferedinputstream-class-java/) | **It is used for Buffered Input Stream.** |
| [**DataInputStream**](https://www.geeksforgeeks.org/java-io-datainputstream-class-java-set-1/) | **It contains a method for reading java standard datatypes.** |
| [**FileInputStream**](https://www.geeksforgeeks.org/java-io-fileinputstream-class-java/) | **This is used to read from a file** |
| [**InputStream**](https://www.geeksforgeeks.org/java-io-inputstream-class-in-java/) | **This is an abstract class that describes stream input.** |
| [**PrintStream**](https://www.geeksforgeeks.org/java-io-printstream-class-java-set-1/) | **This contains the most used print() and println() method** |
| [**BufferedOutputStream**](https://www.geeksforgeeks.org/java-io-bufferedoutputstream-class-java/) | **This is used for Buffered Output Stream.** |
| [**DataOutputStream**](https://www.geeksforgeeks.org/dataoutputstream-in-java/) | **This contains method for writing java standard data types.** |
| [**FileOutputStream**](https://www.geeksforgeeks.org/creating-a-file-using-fileoutputstream/) | **This is used to write to a file.** |
| [**OutputStream**](https://www.geeksforgeeks.org/java-io-outputstream-class-java/) | **This is an abstract class that describes stream output.** |

* **CharacterStream:** In Java, the characters are stored using Unicode conventions. 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 are the list of various CharacterStream Classes:

| **Stream class** | **Description** |
| --- | --- |
| [**BufferedReader**](https://www.geeksforgeeks.org/java-io-bufferedreader-class-java/) | **It is used to handle buffered input stream.** |
| [**FileReader**](https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/) | **This is an input stream that reads from file.** |
| [**InputStreamReader**](https://www.geeksforgeeks.org/java-io-inputstreamreader-class/) | **This input stream is used to translate byte to character.** |
| **OutputStreamReader** | **This output stream is used to translate character to byte.** |
| [**Reader**](https://www.geeksforgeeks.org/java-io-reader-class-java/) | **This is an abstract class that define character stream input.** |
| [**PrintWriter**](https://www.geeksforgeeks.org/java-io-printwriter-class-java-set-1/) | **This contains the most used print() and println() method** |
| [**Writer**](https://www.geeksforgeeks.org/java-io-writer-class-java/) | **This is an abstract class that define character stream output.** |
| [**BufferedWriter**](https://www.geeksforgeeks.org/io-bufferedwriter-class-methods-java/) | **This is used to handle buffered output stream.** |
| [**FileWriter**](https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/) | **This is used to output stream that writes to file.** |

**Reading Standard Input**

In Java, there are three different ways of reading input from the user in the command line environment(console).

* Using Buffered Reader Class
* Using Scanner Class
* Using Console Class

**Using Buffered Reader Class**

* This is the Java classical method to take input, Introduced in JDK1.0.
* We can read input from the user in the command line.
* This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader.
* InputStreamReader class can be used to read data from the keyboard. It performs two tasks:
  + Connects to the input stream of keyboard.
  + Converts the byte-oriented stream into character-oriented stream.
* BufferedReader class can be used to read the data line by line by readLine() method.
* **Advantages:** The input buffers for efficient reading.
* **Drawback:** The wrapping code is hard to remember.

**Example:** Java program to demonstrate BufferedReader.

**import** java.io.\*;

**public** **class** BufferRederEX {

**public** **static** **void** main(String args[])**throws** Exception{

//reading input

BufferedReader br = **new** BufferedReader(**new**

InputStreamReader(System.***in***));

//alternate method for reading input

// InputStreamReader r=new InputStreamReader(System.in);

// BufferedReader br=new BufferedReader(r);

System.***out***.println("Enter your name : ");

String name=br.readLine();

System.***out***.println("Welcome " + name);

}

}

**Output:**

Enter your name :

ABC

Welcome ABC

**Example:** Program to read an Integer from the standard input.

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**public** **class** ReadStdinInt {

**public** **static** **void** main(String[] ap) {

String line = **null**;

**int** val = 0;

**try** {

BufferedReader is = **new** BufferedReader( **new**

InputStreamReader(System.***in***));

System.***out***.println("Enter the string");

line = is.readLine();

val = Integer.*parseInt*(line);

System.***out***.println("I read this number: " + val);

} **catch** (NumberFormatException ne) {

System.***err***.println("Not a valid number: " + line);

} **catch** (IOException e) {

System.***err***.println("Unexpected IO ERROR: " + e);

}

}

}

**Output:**

**Case 1:**

Enter the string

12

I read this number: 12

**Case 2:**

Enter the string

abc 12

Not a valid number: abc 12

**Using Scanner Class**

* This is probably the most preferred method to take input.
* The main purpose of the Scanner class is to parse primitive types and strings using regular expressions.
* It is also can be used to read input from the user in the command line.
* The following table explains the methods for reading different types of input data:

|  |  |
| --- | --- |
| **Method** | **Description** |
| nextBoolean() | Reads a boolean value from the user |
| nextByte() | Reads a byte value from the user |
| nextDouble() | Reads a double value from the user |
| nextFloat() | Reads a float value from the user |
| nextInt() | Reads an int value from the user |
| nextLine() | Reads a String value from the user |
| nextLong() | Reads a long value from the user |
| nextShort() | Reads a short value from the user |

* **Advantages:** Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input and Regular expressions can be used to find tokens.
* **Drawback:** The reading methods are not synchronized

**Example:** Java program to demonstrate the working of Scanner in Java.

**import** java.util.Scanner;

**public** **class** ScannerEx {

**public** **static** **void** main(String[] args) {

Scanner myObj = **new** Scanner(System.***in***);

System.***out***.println("Enter name, age and salary:");

// String input

String name = myObj.nextLine();

// Numerical input

**int** age = myObj.nextInt();

**double** salary = myObj.nextDouble();

// Output input by user

System.***out***.println("Name: " + name);

System.***out***.println("Age: " + age);

System.***out***.println("Salary: " + salary);

}

}

**Output:**

Enter name, age and salary:

ABC

25

5000

Name: ABC

Age: 25

Salary: 5000.0

**Using Console Class**

* The Java 6 System.console() method to obtain a Console object, and use its methods.
* It has become a preferred way for reading user’s input from the command line.
* It can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).
* **Advantages:**
  + Reading password without echoing the entered characters.
  + Reading methods are synchronized.
  + Format string syntax can be used.
* **Drawback:** Does not work in a non-interactive environment (such as in an IDE).

**Example:** Java program to demonstrate the working of System.console().

public class ConsoleRead {

    public static void main(String[] args)

    {

        //Using Console to input data from user

        String name = System.console().readLine();

        System.out.println(name);

    }

}

**Note** that this program does not work on IDEs as System.console() may require console.

**Example:** The Console class is quite useful for is reading a password without having it echo.

**import** java.io.Console;

**public** **class** ReadPassword {

**public** **static** **void** main(String[] args) {

Console cons;

**if** ((cons = System.*console*()) != **null**) {

**char**[] passwd = **null**;

**try** {

// readPassword() method that takes a prompt

//argument

passwd = cons.readPassword("Password: ");

// In real life you would send the password into

//authentication code

System.***out***.println("Your password was: " + **new**

String(passwd));

}

**finally** {

// Shred this in-memory copy for security reasons

**if** (passwd != **null**) {

java.util.Arrays.*fill*(passwd, ' ');

}

}

}

**else** {

**throw** **new** RuntimeException("No console, can't get

password");

}

}

}

**Writing Standard Output or Standard Error**

**Problem**

Suppose, You want your program to write to the standard output or the standard error stream.

**Solution**

Use System.out or System.err as appropriate.

**Example:** Java program to illustrate standard System.out or System.err

**public** **class** DemoPrint {

**public** **static** **void** main(String[] args)

{

System.***out***.println("Java code to illustrate print() ");

// println() print all in

// the same line

System.***out***.print("Hello");

System.***out***.print(" "); //print blank space

System.***out***.print("to");

System.***out***.print(" "); //print blank space

System.***out***.print("JAVA");

//Print the blank line

System.***out***.println(" ");

**int** Val = 10;

System.***out***.print("The Value: " + Val);

//Print the blank line

System.***out***.println(" ");

System.***out***.println("Java code to illustrate println() ");

// println() print all in

// the different line

System.***out***.println("Hello");

System.***out***.println("to");

System.***out***.println("JAVA");

**int** Data = 20;

System.***out***.println("The Value: " + Data);

//Java code to illustrate System.err

System.***out***.println("Java code to illustrate System.err ");

**int** a=14, b=0;

**try** {

**int** c= a/b;

System.***out***.println("The result = " + c);

} **catch**(Exception e) {

//else

//error stream print in red with error

System.***err***.println("Catch Error " + e);

}

}

}

**Output:**

Java code to illustrate print()

Hello to JAVA

The Value: 10

Java code to illustrate println()

Hello

to

JAVA

The Value: 20

Java code to illustrate System.err

Catch Error java.lang.ArithmeticException: / by zero

**Printing with Formatter and printf**

* **Problem**
  + You want the ease of use that the java.util.Formatter class brings to simple printing tasks.
* **Solution**
  + Use Formatter for printing values with fine-grained control over the formatting.
* **Formatting output using System.out.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.
* **Formatting output using java.util.Formatter**
  + The **java.util.Formatter** class provides support for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output.
* The format code consists of a percent sign (%), optionally an argument number followed by a dollar sign ($), optionally a field width or precision, and a format type (d for decimal integer, that is, an integer with no decimal point, f for floating point, and so on).
* A simple use might look like the following:



|  |  |
| --- | --- |
| ***Table : Formatter format codes*** | |
| **Code** | **Meaning** |
| c | Character (argument must be char or integral type containing valid character value). |
| d | “decimal int”—integer to be printed as a decimal (radix 10) with no decimal point (argument must be integral type). |
| f | Floating-point value with decimal fraction (must be numeric); field width may be followed by decimal point and fractional digit field width; e.g., 7.2f. |
| e | Floating-point value in scientific notation. |
| g | Floating-point value, as per f or e, depending on magnitude. |
| s | General format; if value is null, prints “null,” else if arg implements Formattable, format as per arg.format-To(); else format as per arg.toString(). |
| t | Date codes; follow with secondary code. Argument must be long, Long, Calendar, or Date. |
| n | Newline; insert the platform-dependent line ending character. |
| % | Insert a literal % character. |

|  |  |
| --- | --- |
| ***Table : Formatting codes for dates and times*** | |
| **Code** | **Meaning** |
| Y | Year (at least four digits). |
| m | Month as 2-digit (leading zeros) number |
| B | Locale-specific month name (b for abbreviated) |
| d | Day of month (2 digits, leading zeros) |
| e | Day of month (1 or 2 digits) |
| A | Locale-specific day of week (a for abbreviated) |
| H or I | Hour in 24-hour (H) or 12-hour (I) format (2 digits, leading zeros) |
| M | Minute (2 digits) |
| S | Second (2 digits) |
| P/p | Locale-specific AM or PM in uppercase (P) or lowercase (p) |
| R or T | 24-hour time combination: %tH:%tM ® or %tH:%tM:%tS (T) |
| D | Date formatted as “%tm/%td/%ty” |

**Example:** Some examples of using a Formatter

**import** java.util.Formatter;

**public** **class** FormatterDemo {

**public** **static** **void** main(String[] args) {

Formatter fmtr = **new** Formatter();

Object result = fmtr.format("%1$04d - the year of %2$f",

1956, Math.***PI***);

System.***out***.println(result);

// Shorter way using static String.format(), and

// default parameter numbering.

Object stringResult = String.*format*("%04d - the year of %f",

1956, Math.***PI***);

System.***out***.println(stringResult);

// A shorter way using PrintStream/PrintWriter.format, more

in line with other languages. But this way you must provide

the newline delimiter using %n (do NOT use \n as that is

platform-dependent!).

System.***out***.printf("%04d - the year of %f%n", 1956, Math.***PI***);

// Format doubles with more control

System.***out***.printf("PI is approximately %4.4f%n", Math.***PI***);

// Format doubles with more control

System.***out***.format("PI is approximately %4.2f%n", Math.***PI***);

}

}

**Output:**

1956 - the year of 3.141593

1956 - the year of 3.141593

1956 - the year of 3.141593

PI is approximately 3.1416

PI is approximately 3.14

**Example:** JAVA program for some date examples

**import** java.util.Calendar;

**import** java.util.Date;

**public** **class** FormatterDates {

**public** **static** **void** main(String[] args) {

// Format number as dates e.g., 2014-06-28

System.***out***.printf("%4d-%02d-%2d%n", 2014, 6, 28);

// Format fields directly from a Date object: multiple fields

from "1$"

Date today = Calendar.*getInstance*().getTime();

// Might print e.g., July 4, 2015:

System.***out***.printf("Today is %1$tB %1$td, %1$tY%n", today);

}

}

**Output:**

2014-06-28

Today is May 20, 2020

**Scanning Input with the Scanner Class**

**Problem**

* You want the ease of use that the java.util.Scanner class brings to *simple* reading tasks.

**Solution**

* Use Scanner’s next() methods for reading.
* Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
* The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.
* The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Table: Scanner methods*** | | | |
| **Returned type** | **“has” method** | **“next” method** | **Comment** |
| String | hasNext() | next() | The next complete token from this scanner |
| String | hasNext(Pattern) | next(Pattern) | The next string that matches the given regular expression (regex) |
| String | hasNext(String) | next(String) | The next token that matches the regex pattern constructed from the specified string |
| BigDecimal | hasNextBigDecimal() | nextBigDecimal() | The next token of the input as a BigDecimal |
| BigInteger | hasNextBigInteger() | nextBigInteger() | The next token of the input as a BigInteger |
| boolean | hasNextBoolean() | nextBoolean() | The next token of the input as a boolean |
| byte | hasNextByte() | nextByte() | The next token of the input as a byte |
| double | hasNextDouble() | nextDouble() | The next token of the input as a double |
| float | hasNextFloat() | nextFloat() | The next token of the input as a float |
| int | hasNextInt() | nextInt() | The next token of the input as an int |
| String | N/A | nextLine() | Reads up to the end-of-line, including the line ending |
| long | hasNextLong() | nextLong() | The next token of the input as a long |
| short | hasNextShort() | nextShort() | The next token of the input as a short |

**Example:** Java program to read data of string types using Scanner class.

**public** **class** ScannerEx {

**public** **static** **void** main(String[] args) {

//Create Scanner object

Scanner scan = **new** Scanner("Hello World!");

//Print the Strings

**while** (scan.hasNext())

System.***out***.println(scan.next());

}

}

**Output:**

Hello

World!

**Example:** Java program to read data of various types using Scanner class.

**public** **class** ScannerEx {

**public** **static** **void** main(String[] args) {

String sampleDate = "25 Dec 1988";

**try** (Scanner sDate = **new** Scanner(sampleDate)) {

//Scans the next token of the input as an int.

**int** dayOfMonth = sDate.nextInt();

//next() method finds and returns the next complete token.

String month = sDate.next();

**int** year = sDate.nextInt();

System.***out***.printf("%d-%s-%02d%n", year, month, dayOfMonth);

}

}

}

**Output:**

1988-Dec-25

**Opening a File by Name**

**Problem**

* The Java documentation doesn’t have methods for opening files. How do I connect a filename on disk with a Reader, Writer, or Stream?

**Solution**

* Construct a FileReader, FileWriter, FileInputStream, or FileOutputStream.

There are following ways to open a file in Java:

* Java Desktop class
* ***Java FileInputStream class***
* ***Java FileReader class***
* ***Java BufferedReader class***
* ***Java Scanner class***
* Java nio package

## Java FileInputStream class

* The Java FileInputStream class is used to open and read a file.
* We can open and read a file by using the constructor of the FileInputStream class.
* It accepts a file as an argument.
* It throws **FileNotFoundException** if the file does not exist or file name is a directory.

**Example:** Java Program to illustrate to read the content from Text File using FileInputStream class.

**import** java.io.File;

**import** java.io.FileInputStream;

**public** **class** OpenFileExample {

**public** **static** **void** main(String args[]) {

**try** {

//constructor of file class having file as argument

File file = **new** File("D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\demofile.txt");

//opens a connection to an actual file

FileInputStream fis = **new** FileInputStream(file);

System.***out***.println("FILE CONTENT: ");

**int** r=0;

//read() method reads the next byte of the data from the

// inputstream and returns int in the range of 0 to 255.

**while**((r=fis.read())!=-1) {

System.***out***.print((**char**)r);

}

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Output:**

FILE CONTENT:

JAVA

PYTHON

C

C++

## Java FileReader class

* The Java **FileReader** class is also used for opening and reading a file.

* It belongs to a **java.io** package.
* It is a convenience for reading characters of the files.
* It is used for reading raw bytes using the FileInputStream class.
* We use the constructor of the FileInputStream class to open and read a file.
* It accepts a file as an argument.
* It throws **FileNotFoundException** if the file does not exist or file name is a directory.

**Example:** Java Program to illustrate to read the content from Text File using **FileReader** class.

**import** java.io.File;

**import** java.io.FileReader;

**public** **class** OpenFileExample {

**public** **static** **void** main(String args[]) {

**try** {

//constructor of File class having file as argument

File file=**new** File("D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\demofile.txt");

//creates a File reader input stream

FileReader fr=**new** FileReader(file);

System.***out***.println("FILE CONTENT: ");

**int** r=0;

//read() method reads the next byte of the data from the

// inputstream and returns int in the range of 0 to 255.

**while**((r=fr.read())! = -1) {

System.***out***.print((**char**)r);

}

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Output:**

FILE CONTENT:

JAVA

PYTHON

C

C++

## Java BufferedReader class

* The Java BufferedReader class reads text from a character input stream. It belongs to a java.io package.
* We use the constructor of the BufferedReader class to open or read a file.
* It creates a buffering character-input stream that uses a default sized input buffer.
* It accepts a file as an argument.
* It throws **FileNotFoundException** if the file does not exist or file name is a directory.

**Example:** Java Program to illustrate to read the content from Text File using BufferedReader class.

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.BufferedReader;

**public** **class** OpenFileExample {

**public** **static** **void** main(String args[]) {

**try** {

//constructor of File class having file as argument

File file=**new** File("D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\demofile.txt");

//creates a buffer reader input stream

BufferedReader br=**new** BufferedReader(**new** FileReader(file));

System.***out***.println("FILE CONTENT: ");

**int** r=0;

//read() method reads the next byte of the data from the

// inputstream and returns int in the range of 0 to 255.

**while**((r=br.read())!=-1) {

System.***out***.print((**char**)r);

}

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Output:**

FILE CONTENT:

JAVA

PYTHON

C

C++

## Java Scanner class

* The Java **Scanner** class is also used for opening and reading a file.
* The Scanner class belongs to **java.util** package. The constructor of Scanner class is used **for opening and reading a file.**
* **It accepts a file as an argument**.
* It throws FileNotFoundException if the file does not exist or file name is a directory.

**Example:** Java Program to illustrate to read the content from Text File using Java **Scanner** class.

**import** java.io.File;

**import** java.util.Scanner;

**public** **class** OpenFileExample {

**public** **static** **void** main(String args[]) {

**try** {

//constructor of File class having file as argument

File file=**new** File("D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\demofile.txt");

//file to be scanned

Scanner sc = **new** Scanner(file);

System.***out***.println("FILE CONTENT: ");

//The hasNextLine() is used to check if there is another line

//in the input of this scanner. It returns true if it finds another //line, otherwise returns false.

**while** (sc.hasNextLine())

System.***out***.println(sc.nextLine());

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Output:**

FILE CONTENT:

JAVA PYTHON

C

C++

## Write To a File

There are following ways to wirte a file in Java:

* Java FileWriter Class
* Java FileOutputStream Class
* Java BufferedWriter Class

**Java FileWriter Class**

* The Java FileWriter class is used to write character-oriented data to a [file](https://www.javatpoint.com/java-file-class).
* It is character-oriented class which is used for file handling in [java](https://www.javatpoint.com/java-tutorial).
* It provides method to write string directly.
* FileWriter class together with its write() method to write some text to the file.
* After writing to the file, close it with the close() method.

**Example:** Java Program to illustrate to write the content to Text File using Java FileWriter class.

**import** java.io.FileWriter;

**public** **class** WriteFileExample {

**public** **static** **void** main(String[] args) {

**try** {

//output file will be created

FileWriter fw = **new** FileWriter("D:\\CSW - 2\\CSW-1

September 2020\\DemoJava\\output.txt");

//write the content in the file

fw.write("Welcome to JAVA Programming Language.");

fw.close(); // close

} **catch**(Exception e){

System.***out***.println(e);

}

System.***out***.println("Success...");

}

}

**Output:**

Success...

**Note:** the content "Welcome to JAVA Programming Language.", has been written in the output.txt file.

# Java FileOutputStream Class

* Java FileOutputStream is an output stream used for writing data to a [file](https://www.javatpoint.com/java-file-class).
* If you have to write primitive values into a file, use FileOutputStream class.
* You can write byte-oriented as well as character-oriented data through FileOutputStream class.
* But, for character-oriented data, it is preferred to use [FileWriter](https://www.javatpoint.com/java-filterwriter-class) than FileOutputStream.

**Example:** Java Program to illustrate to write the content (write byte and write string) to Text File using Java FileOutputStream class.

**import** java.io.FileOutputStream;

**import** java.io.FileWriter;

**public** **class** WriteFileExample {

**public** **static** **void** main(String[] args) {

**try**{

//output file will be created

FileOutputStream fos = **new** FileOutputStream("D:\\CSW –

2\\CSW-1 September 2020\\DemoJava\\output1.txt");

//Writes the specified byte to this file output stream.

fos.write(65); // ASCII Code of 'A'

String s = " Welcome to JAVA Programming Language.";

//converting string into byte array

**byte** b[]=s.getBytes();

//Writes the specified byte to this file output stream.

fos.write(b);

fos.close(); // close

} **catch**(Exception e){

System.***out***.println(e);

}

System.***out***.println("Success...");

}

}

**Output:**

Success...

**Note:** the content "A Welcome to JAVA Programming Language." has been written in the output1.txt file.

# Java BufferedWriter Class

* The Java BufferedWriter class is used to provide buffering for Writer instances.
* It makes the performance fast.
* It inherits [Writer](https://www.javatpoint.com/java-writer-class) class.
* The buffering characters are used for providing the efficient writing of single [arrays](https://www.javatpoint.com/array-in-java), characters, and [strings](https://www.javatpoint.com/java-string).
* A buffer size needs to be specified, if not it takes Default value.
* An output is immediately set to the underlying character or byte stream by the Writer.

**Example:** Java Program to illustrate to write the content (write byte and write string) to Text File using BufferedWriter class.

**import** java.io.BufferedWriter;

**import** java.io.FileWriter;

**public** **class** WriteFileExample {

**public** **static** **void** main(String[] args) {

**try**{

//initializing FileWriter by creating output file

FileWriter fw = **new** FileWriter("D:\\CSW - 2\\CSW-1

September 2020\\DemoJava\\output2.txt");

//initializing BufferedWriter

BufferedWriter bw = **new** BufferedWriter(fw);

//Writes the specified byte to this file output stream.

bw.write(65); // ASCII Code of 'A'

String s = " Welcome to JAVA Programming Language.";

//Writes the specified string to the output file.

bw.write(s);

bw.close(); // close

} **catch**(Exception e){

System.***out***.println(e);

}

System.***out***.println("Success...");

}

}

**Output:**

Success...

**Note:** the content "A Welcome to JAVA Programming Language." has been written in the output2.txt file.

**Copying a File**

**Problem**

* You need to copy a file in its entirety.

**Solution**

* Use a pair of Streams for binary data, or a Reader and a Writer for text, and a while loop to copy until end-of-file is reached on the input.
* To copy the content of one file to another file in java
  + First, we can [read the file](https://beginnersbook.com/2014/01/how-to-read-file-in-java-bufferedinputstream/) using [FileInputStream](https://docs.oracle.com/javase/6/docs/api/java/io/FileInputStream.html) and
  + Then [write the read content to the output file](https://beginnersbook.com/2014/01/how-to-write-to-a-file-in-java-using-fileoutputstream/) using [FileOutputStream](https://docs.oracle.com/javase/6/docs/api/java/io/FileOutputStream.html).

**Example:** Java Program to illustrate to copy the content of one file to another by using [FileInputStream](https://docs.oracle.com/javase/6/docs/api/java/io/FileInputStream.html) class and [FileOutputStream](https://docs.oracle.com/javase/6/docs/api/java/io/FileOutputStream.html) class.

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**public** **class** CopyExample {

**public** **static** **void** main(String[] args) {

FileInputStream instream = **null**;

FileOutputStream outstream = **null**;

**try**{

//Creates a new File instance by converting the

//given pathname string into an abstract pathname.

File infile =**new** File("D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\MyInputFile.txt");

File outfile =**new** File("D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\MyOutputFile.txt");

//Creates a FileInputStream by opening a

//connection to an actual file

instream = **new** FileInputStream(infile);

//Creates a file output stream to write

outstream = **new** FileOutputStream(outfile);

**byte**[] buffer = **new** **byte**[1024];

**int** length;

/\*copying the contents from input stream to

\* output stream using read and write methods

\*/

//Reads up to 1024 bytes of data from this

//input stream into an array of bytes.

**while** ((length = instream.read(buffer)) > 0){

//Writes 1024 bytes from the specified byte array

// starting at offset off to this file output stream.

outstream.write(buffer, 0, length);

}

//Closing the input/output file streams

instream.close();

outstream.close();

System.***out***.println("File copied successfully!!");

} **catch**(IOException ioe) {

ioe.printStackTrace();

}

}

}

**Output:**

File copied successfully!!

**Note:** the contents “All are Welcomed to JAVA Programming Language.” of MyInputFile.txt have been copied to MyOutputFile.txt.

**Reading a File into a String**

**Problem**

* You need to read the entire contents of a file into a string.

**Solution**

* Use Files.*readAllBytes*(Paths.*get*()) method.

**Example:** Java Program to illustrate reading from text file as string in Java.

**import** java.nio.file.Files;

**import** java.nio.file.Paths;

**public** **class** ReadTextAsString {

**public** **static** String readFileAsString(String fileName)**throws** Exception{

String data = "";

//Files.readAllBytes():Reads all the bytes from a

//file. The method ensures that the file is closed

data = **new** String(Files.*readAllBytes*(Paths.*get*(fileName)));

**return** data;

}

**public** **static** **void** main(String[] args) **throws** Exception {

String data = *readFileAsString*("D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\MyInputFile.txt");

System.***out***.println(data);

}

}

**Output:**

All are Welcomed to JAVA Programming Language.

**Reassigning the Standard Streams**

**Problem**

* You need to reassign one or more of the standard streams System.in, System.out, or System.err.

**Solution**

* Construct an InputStream or PrintStream as appropriate, and pass it to the appropriate set method in the System class.
* **Standard Input:** This is used to feed the data to the user's program and usually a keyboard is used as a standard input stream and represented as System.in.
* **Standard Output:** This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as System.out.
* **Standard Error:** This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as System.err.

**Example:** Java Program to illustrate reassign standard streams.

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**public** **class** ReadTextAsString {

**public** **static** **void** main(String[] args) {

**try** {

//FileInputStream(): opening a connection to an actual file

InputStream input = **new** FileInputStream("c:\\Simple.java");

//InputStream input = new FileInputStream("D:\\CSW –

2\\CSW-1 September 2020\\DemoJava\\MyInputFile.txt");

System.***out***.println("File opened...");

} **catch** (IOException e){

System.***err***.println("File opening failed:");

}

}

}

**Output:**

File opening failed.

**Reading/Writing Binary Data**

**Problem**

* You need to read or write binary data, as opposed to text.

**Solution**

* Use a DataInputStream or DataOutputStream.
* The DataInputStream class reads primitive Java data types from an underlying input stream in a machine-independent way.
* The DataOutputStream class writes primitive Java data types to an output stream in a portable way.

**Example:** Java program to demonstrate DataInputStream and DataOutputStream.

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**public** **class** DataStreamExample {

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

//Creates a FileInputStream by opening a connection to an actual file

InputStream input = **new** FileInputStream("D:\\CSW - 2\\CSW-1

September 2020\\DemoJava\\BinaryInput.txt");

//Creates a DataInputStream that uses the

//specified underlying InputStream.

DataInputStream inst = **new** DataInputStream(input);

**byte**[] arr = **new** **byte**[1024];

//Reads some number of bytes from the input stream

// and stores them into the buffer array arr.

inst.read(arr);

//Creates a file output stream to write to the

//file with the specified name.

FileOutputStream fileOutputStream =

**new** FileOutputStream("D:\\CSW - 2\\CSW-1 September 2020\\DemoJava\\BinaryOutput.txt");

//Creates a new data output stream to write data

//to the specified underlying output stream.

DataOutputStream dataOutputStream =

**new** DataOutputStream(fileOutputStream);

**for** (**byte** bbt : arr) {

//Writes the specified byte (the low eight bits of

// the argument b) to the underlying output stream.

dataOutputStream.write(bbt);

}

//Print the data on computer screen

**for** (**byte** bt : arr) {

**char** k = (**char**) bt;

System.***out***.print(k);

}

}

}

**Output:**

001

101

110

**Seeking to a Position within a File**

**Problem**

* You need to read from or write to a particular location in a file, such as an indexed file.

**Solution**

* Use a RandomAccessFile.
* **Java.io.RandomAccessFile** class
  + This [class](https://www.javatpoint.com/object-class) is used for reading and writing to random access file.
  + The RandomAccessFile class also implements the DataInput and DataOutput interfaces.
  + A random access file behaves like a large [array](https://www.javatpoint.com/array-in-java) of bytes.
  + There is a cursor implied to the array called file [pointer](https://www.javatpoint.com/c-pointers), by moving the cursor we do the read write operations.
  + If end-of-file is reached before the desired number of byte has been read than EOFException is [thrown](https://www.javatpoint.com/throw-keyword). It is a type of IOException.

**Example:**

**import** java.io.IOException;

**import** java.io.RandomAccessFile;

**public** **class** ReadRandom {

**final** **static** String ***FILENAME*** = "D:\\CSW - 2\\CSW-1 September

2020\\DemoJava\\MyFile.txt";

**public** **static** **void** main(String[] args) {

**try** {

System.***out***.println(**new** String(*readFromFile*(***FILENAME***, 0, 18)));

*writeToFile*(***FILENAME***, "I love my country and my people", 31);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **static** **byte**[] readFromFile(String filePath, **int** position, **int** size)

**throws** IOException {

//Creates a random access file stream to

//read from a file with the specified name.

RandomAccessFile file = **new** RandomAccessFile(filePath, "r");

file.seek(position); //Sets the file-pointer offset

**byte**[] bytes = **new** **byte**[size];

file.read(bytes);

file.close();

**return** bytes; //Output: This class is used

}

**private** **static** **void** writeToFile(String filePath, String data, **int** position)

**throws** IOException {

//Creates a random access file stream to

//write to a file with the specified name.

RandomAccessFile file = **new** RandomAccessFile(filePath, "rw");

file.seek(position); //Sets the file-pointer offset

//getBytes(): Encodes this String into a sequence of bytes using the // platform's default charset, storing the result into a new byte array.

file.write(data.getBytes());

file.close();

}

}

**Output:**

This class is used

Before running the above program

The myFile.TXT contains text "This class is used for reading and writing to random access file."

After running the above program

The myFile.TXT contains text ” This class is used for reading I love my country and my peoplele.”