**File Handling in Java:**

**File handling define how we can read and write data on file .**

**Java Io package contains all the clases through which we can perform all input & input operations in the file**

Stream:

Straeam ia sequence of data,on the basis of java.io.package all the classes are divide into two stream:

i)Byte

ii)Character

File handling Methods:

canRead()

its return a boolean value.

If read data in file then return true.

canWrite()

its return a boolean value.

If write data in file then return true.

createNewFile()

this method is used to craete new file.

delete()

this method is used to delete the file

exists()

This method is used to file exist or not.

length()

Find the length value of input datatypes.

getName()

this method provide the file name.

getAbsolutePath()

this method is provide the full path of the file.

Mkdir()

It is used to create directory.

List()

This method is used to how much file available in this directory.

Read()

Read the data in file

Write()

Write the data in file

renameTol()

this method is used to rename the filename.

File handling classes:

File

It is the superclasses of all classes.

FileReader

Read the data in file.

FileWriter

Write the data in file.

FileInputStream

Read the data in byte form.

FileOutputStream

Write the data in file

BufferedInputStream

Buffer is the temprory memory

It is used for Buffered Input Stream.

BufferedOutputStream

This is used for Buffered Output Stream.

1.Create File:

import java.io.File;  
  
public class Creates {  
 public static void main(String[] args) {  
 File file = new File("D:\\Java Program\\Myfile\\abc1.txt");  
 try {  
 if (file.createNewFile()) {  
 System.*out*.println("File Created:" + file.getName());  
 } else {  
 System.*out*.println("file already exists:");  
 }  
 }catch(Exception e)  
 {  
 System.*out*.println(e);  
 }finally {  
 System.*out*.println("Finnaly block");  
 }  
 File newfile = new File("D:\\Java Program\\Myfile\\");  
 }  
}

Output: File Created

import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
  
public class Createfile {  
 public static void main(String[] args) {  
 File file = new File("D:\\Java Program\\File handling\\Myfile.txt");  
 try {  
 if (file.createNewFile()) {  
 System.*out*.println("File created:");  
 } else {  
 System.*out*.println("Fil already exist ");  
 }  
 }catch(IOException e)  
 {  
 System.*out*.println(e);  
 }  
 }  
}

Example of methods:

exist(),getName(),getAbsolutePath(),canWrite(),canRead(),length():

import java.io.File;  
  
public class Fileinfo {  
 public static void main(String[] args) {  
 File file = new File("D:\\Java Program\\File handling\\Myfile.txt");  
 if(file.exists())  
 {  
 System.*out*.println("File name:"+file.getName());  
 System.*out*.println("File Path:"+file.getAbsolutePath());  
 System.*out*.println("File Read:"+file.canWrite());  
 System.*out*.println("File Write:"+file.canRead());  
 System.*out*.println("File length:"+file.length());  
 // System.out.println("File delete:"+file.delete());  
 }  
 else{  
 System.*out*.println("File not found:");  
 }  
 }  
}

Output:

File name:Myfile.txt

File Path:D:\Java Program\File handling\Myfile.txt

File Read:true

File Write:true

File length:18

Delete() method:

import java.io.File;  
  
public class Fileinfo {  
 public static void main(String[] args) {  
 File file = new File("D:\\Java Program\\File handling\\Myfile.txt");  
 if(file.exists())  
 {  
 System.*out*.println("File name:"+file.getName());  
 System.*out*.println("File Path:"+file.getAbsolutePath());  
 System.*out*.println("File Read:"+file.canWrite());  
 System.*out*.println("File Write:"+file.canRead());  
 System.*out*.println("File length:"+file.length());  
 System.*out*.println("File delete:"+file.delete());  
 }  
 else{  
 System.*out*.println("File not found:");  
 }  
 }  
}

Output:

File not found:

2.Read file:

import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Scanner;  
  
public class Reading {  
 public static void main(String[] args) {  
 try{  
 File file = new File("D:\\Java Program\\Myfile\\abc1.txt");  
 Scanner sc = new Scanner(file);  
  
 while (sc.hasNextLine()) {  
 String data = sc.nextLine();  
 System.*out*.println(data);  
 }  
 sc.close();  
 }catch(FileNotFoundException e)  
 {  
 System.*out*.println("An error has occurred.");  
 e.printStackTrace();  
 }  
 }  
}

Output: My name is kaushik

2.FileReader:

import java.io.\*;  
import java.util.Scanner;  
  
public class Writefile {  
 public static void main(String[] args) throws IOException {  
 FileReader file = new FileReader("D:\\Java Program\\File handling\\Myfile1.txt");  
 Scanner name = new Scanner(file);  
 while (name.hasNextLine()) {  
 String name1 = name.nextLine();  
 System.*out*.println(name1);  
 }  
 file.close();  
 }  
}

3.BufferedReader:

import java.io.\*;  
import java.util.Scanner;  
  
public class Writefile {  
 public static void main(String[] args) throws IOException {  
 BufferedReader file = new BufferedReader(new FileReader("D:\\Java Program\\File handling\\Myfile1.txt"));  
 Scanner name = new Scanner(file);  
 while (name.hasNextLine()) {  
 String name1 = name.nextLine();  
 System.*out*.println(name1);  
 }  
 file.close();  
 }  
}

4.Scanner class:

import java.io.\*;  
import java.util.Scanner;  
  
public class Writefile {  
 public static void main(String[] args) throws IOException {  
 try {  
 File file = new File("D:\\Java Program\\File handling\\Myfile1.txt");  
  
  
 Scanner name = new Scanner(file);  
 while (name.hasNextLine()) {  
 String name1 = name.nextLine();  
 System.*out*.println(name1);  
 }  
 file.close();  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
  
   
 }  
}

5.FileInputStream:

import java.io.\*;  
import java.util.Scanner;  
  
public class Writefile {  
 public static void main(String[] args) throws IOException {  
 try {  
 FileInputStream file = new FileInputStream("D:\\Java Program\\File handling\\Myfile1.txt");  
 Scanner sc = new Scanner(file);  
 while (sc.hasNextLine()) {  
 String n = sc.nextLine();  
 System.*out*.println(n);  
 }  
 sc.close();  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
}

3.Write File:

There are four ways to write in a file.

1.FileWriter class:

import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class Write {  
 public static void main(String[] args) {  
  
 try  
 {  
 FileWriter file = new FileWriter("D:\\Java Program\\Myfile\\abc1.txt");  
 file.write("My name is kaushik");  
 file.close();  
 System.*out*.println("Successfully written:");  
 }catch(IOException e)  
 {  
 System.*out*.println("Error Occured:");  
 e.printStackTrace();  
 }  
  
 }  
}

Output: Successfully written

import java.io.FileNotFoundException;  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class Writefile {  
 public static void main(String[] args) {  
 try {  
 FileWriter file = new FileWriter("D:\\Java Program\\File handling\\Myfile.txt");  
 try {  
 file.write("My name is kaushik prasad,I m from ranchi jharkhand,My favourite game is cricket.");  
 } finally {  
 file.close();  
 }  
 }  
 catch (IOException e)  
 {  
 System.*out*.println(e);  
 }  
  
 }  
}

2.BufferReader:

import java.io.BufferedWriter;  
import java.io.FileNotFoundException;  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class Writefile {  
 public static void main(String[] args) {  
 try {  
 BufferedWriter file = new BufferedWriter( new FileWriter ("D:\\Java Program\\File handling\\Myfile.txt"));  
 try {  
 file.write("My name is kaushik prasad,I live in Ranchi,I m Graduated student .");  
 } finally {  
 file.close();  
 }  
 }  
 catch (IOException e)  
 {  
 System.*out*.println(e);  
 }  
  
 }  
}

3.FileOutputStream:

import java.io.\*;  
  
public class Writefile {  
 public static void main(String[] args) {  
 try {  
 FileOutputStream file = new FileOutputStream( "D:\\Java Program\\File handling\\Myfile1.txt");  
 try {  
  
 String content ="My name is kaushik prasad,I have completed Bca from Gossner college";  
 file.write(content.getBytes());  
 } finally {  
 file.close();  
 }  
 }  
 catch (IOException e)  
 {  
 System.*out*.println(e);  
 }  
  
 }  
}

4.PrintWriter:

import java.io.\*;  
  
public class Writefile {  
 public static void main(String[] args) {  
 try {  
 PrintWriter file = new PrintWriter( "D:\\Java Program\\File handling\\Myfile1.txt");  
 try {  
   
 file.write("My name is kaushik prasad,I have completed Mca from Gossner college");  
  
 } finally {  
 file.close();  
 }  
 }  
 catch (IOException e)  
 {  
 System.*out*.println(e);  
 }  
  
 }  
}

4.Delete File:

import java.io.File;  
  
public class Delete {  
 public static void main(String[] args) {  
 File deletes = new File("D:\\Java Program\\Myfile\\abc1.txt");  
 if(deletes.delete())  
 {  
 System.*out*.println("File deleted successfully:"+deletes.getName());  
 }  
 else {  
 System.*out*.println("Deleted file failed");  
 }  
 }  
}

Output: File deleted successfully:abc1.txt

Rename :

Rename the file name.

import java.io.File;  
import java.util.Scanner;  
  
public class Rename {  
 public static void main(String[] args) {  
 File file = new File("D:\\Java Program\\File handling\\names.txt");  
 File files = new File("D:\\Java Program\\File handling\\kAUSHIK.txt");  
 if(file.exists())  
 {  
 System.*out*.println(file.renameTo(files));  
 }  
 else{  
 System.*out*.println("not rename");  
 }  
  
 }  
}

Questions:

Give the hierarchy of InputStream and OutputStream classes.

Java output stream hierarchy

Java input stream hierarchy

* what are the super most classes for all the streams?The **SuperMost classes for all** the InputStream **classes** is java.io.InputStream and for **all** the output **stream classes** is java.io.OutPutStream.
* What do you understand by an IO stream?

An I/O Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

* What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

Key differences:

1. Data type: InputStream/OutputStream work with bytes, while Reader/Writer work with characters.

2. Encoding: Reader/Writer perform encoding and decoding automatically, while InputStream/OutputStream do not.

3. Purpose: InputStream/OutputStream are suitable for binary data, while Reader/Writer are suitable for character-based data.

* What are the FileInputStream and FileOutputStream?

To read and write data, Java offers I/O Streams. A Stream represents an input source or an output destination, which could be a file, an i/o device, another program, etc. **[FileInputStream](https://www.geeksforgeeks.org/java-io-fileinputstream-class-java/)** in Java is used to read data from a file as a stream of bytes. It is mostly used for reading binary data such as images, audio files, or serialized objects.

**Example:**

File file = new File("path\_of\_the\_file");  
FileInputStream inputStream = new FileInputStream(file);

In Java, the **[FileOutputStream](https://www.geeksforgeeks.org/fileoutputstream-in-java/)** function is used to write data byte by byte into a given file or file descriptor. Usually, raw byte data, such as pictures, is written into a file using FileOutputStream.

**Example:**

File file = new File("path\_of\_the\_file");  
FileOutputStream outputStream = new FileOutputStream(file);

* What is the purpose of using BufferedInputStream and BufferedOutputStream classes?

BufferedInputStream and BufferedOutputStream are classes in Java that provide buffering capabilities to input and output streams, respectively.

The purpose of using these classes is to improve the performance of I/O operations by reducing the number of disk accesses. Here's how they work:

• BufferedInputStream: When you read data from a file using a FileInputStream, the data is read one byte at a time. This can be slow, especially for large files. A BufferedInputStream reads a block of data (called a buffer) from the file and stores it in memory. When you read data from the stream, it is read from the buffer instead of the file, which is faster. When the buffer is empty, the next block of data is read from the file.

• BufferedOutputStream: Similarly, when you write data to a file using a FileOutputStream, the data is written one byte at a time. A BufferedOutputStream writes data to a buffer in memory, and when the buffer is full, the data is written to the file in a single operation. This reduces the number of disk writes, making the operation faster.

By using buffering, you can improve the performance of I/O operations, especially for large files.

o

o How to set the Permissions to a file in Java?

o In Java, you can set permissions to a file using the setReadable(), setWritable(), and setExecutable() methods of the File class. These methods allow you to set the read, write, and execute permissions for the file, respectively.

o Here's an example:

o java

o VerifyOpen In EditorEditCopy code

o 1File file = new File("example.txt");

o 2file.setReadable(true); // set read permission

o 3file.setWritable(true); // set write permission

o 4file.setExecutable(true); // set execute permission

o You can also use the setPermissions() method to set multiple permissions at once:

o java

o VerifyOpen In EditorEditCopy code

o 1file.setPermissions(0666); // set read, write, and execute permissions for owner, group, and others

o Note that the setPermissions() method takes an integer argument that represents the permissions in octal notation.

* What are FilterStreams?

FilterStreams are a type of stream in Java that allow you to perform additional processing on the data being read or written. They are used to filter, transform, or modify the data in some way.

FilterStreams are implemented as subclasses of FilterInputStream and FilterOutputStream. They wrap around an existing stream and provide additional functionality, such as:

• Compression and decompression

• Encryption and decryption

• Data transformation (e.g., converting between character sets)

• Error detection and correction

Examples of FilterStreams include BufferedInputStream, DataInputStream, and GZIPInputStream.

o

* What is an I/O filter?

An I/O filter is a component that performs additional processing on the data being read or written. It is a type of FilterStream that filters, transforms, or modifies the data in some way.

I/O filters are used to:

• Validate or sanitize input data

• Transform data formats (e.g., converting between character sets)

• Compress or decompress data

• Encrypt or decrypt data

• Detect and correct errors

I/O filters are typically implemented as subclasses of FilterInputStream and FilterOutputStream.

* In Java, How many ways you can take input from the console?

In Java, there are several ways to take input from the console:

o Using System.in: You can use the System.in stream to read input from the console. This stream is an InputStream that reads bytes from the console.

o Using BufferedReader: You can use a BufferedReader to read input from the console. This class provides a way to read text data from the console.

o Using Scanner: You can use a Scanner to read input from the console. This class provides a way to read primitive types, such as int, double, and String, from the console.

o Using Console: You can use the Console class to read input from the console. This class provides a way to read

Serialization:

* What is serialization?
* How can you make a class serializable in Java?
* How can you avoid serialization in child class if the base class is implementing the Serializable interface?
* Can a Serialized object be transferred via network?
* What is Deserialization?
* What is the transient keyword? Difference between transient and static variable.
* What is Externalizable?
* What is the difference between Serializable and Externalizable interface? Tell atleast 4-5 points