File Handling in Java

In general terms, a file is a collection of information. In Java, a file is an abstract type that stores related information together. This article will let you know about the file and its various useful operations. But before getting to know about it, you must have a clear idea about Stream and File Methods in Java.

Standard streams in Java:

A stream is a series of data. It helps us perform I/O operations with a file. The operating system will define three standard streams. The need for standard Input and Output us necessary. This is where a user can take input from a keyboard and result in producing an output on the monitor. Though the input and output devices may vary, the process is the same. In C and C++, there are three standard devices STDIN, STDOUT and STDERR.

It ensures that these standard streams are accessible via the members of the system class in Java:

1. Standard Input: It provides input data to the program. This task is mostly accomplished by a keyboard that is used as a standard input stream and is denoted as System.in.

2. Standard Output: It provides the result of the data that that program produces. this output is usually displayed on the computer screen which is the standard output stream and is denoted as System.out.

3. Standard Error: It displays the error data that the program produces. This is usually displayed on the computer screen that is used as the standard error stream and is represented as System.err.

Input Stream in Java:

The superclass of all input streams is the Java InputStream class. We use input devices like the keyboard, network, etc., to read the input. This InputStream is also an abstract class.

The various subclasses of the InputStream class are:

- AudioInputStream
- ByteArrayInputStream
- FileInputStream
- FilterInputStream
- StringBufferUnputStream
- ObjectInputStream

Creating an Input Stream in java:

```
InputStream obj = new FileInputStream();
```

S.no	Method	Description
1	read()	We can use this to read a byte of date from the input stream.
2	read(byte[] array)()	It reads a byte from the stream and lets us store the byte in the mentioned array.
3	mark()	It marks the position in the input stream till the data is read.

4	available()	It returns the number of bytes from the input stream.
5	markSupported()	It checks if the mark() method and result() method in the system.
6	reset()	It returns the control to the point where the mark was set.
7	skips()	It skips and removes the mentioned number of bytes from the input stream.
8	close()	It closes the input stream.
9	finalize()	It cleans the connection to the file. It also ensures that the close method

Sample program to create InputStreamReader to read the standard input stream until the code word for quit, ‘q’ is entered by the user:

```
import java.io.*;

public class FirstCodeReadConsole {

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

InputStreamReader cin = null;

try {

cin = new InputStreamReader(System.in);

System.out.println("Enter the character 'q' to quit.");

char c;

do {

c = (char) cin.read();

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

```
} while(c != 'q');

}finally {

if (cin != null) {

cin.close();

}

}

}

}
```

Output:

1

2

r

q

q

Output stream in Java:

The output stream gives the ability to write data into various output devices. These devices include the monitor, printer and even file. OutputStream is an abstract superclass to represent the output stream.

The various subclasses under the output stream class are:

- ByteArrayOutputStream
- FileOutputStream
- StringBufferOutputStream
- DataOutputStream
- PrintStream

Creating an OutputStream:

```
OutputStream obj = new FileOutputStream();
```

S.No	Method	Description
1	write()	It writes the specified byte to the output stream.
2	write(byte[] array)	It writes the bytes which are given in a specific array to the output stream
3	close()	It closes the output stream
4	flush()	It forces to write all the data present in an output stream to its destination
5	finalize()	It cleans the connection to the file. It ensures that the close method if the file outstream is invoked in the absence of other references to the stream.

Output Streams Operations in Java:

There are three methods to write the data into a stream. These methods are available in the Independent output stream. These mirror the read() methods present in the Input Stream Class, which is also an abstract class.

Output Streams Operations	Description
FileOutputStream	To write a file

ObjecyInputStream	To write objects to a stream
ByteArrayOutputStream	To write an array of bytes
PipeOutputStream	To write to a piped stream
FilterOutputStream	To filter the output from an end stream

Based on the data type, the stream is divided into two categories:

1. Byte Stream

2. Character Stream

Classification of I/O streams in java:

1. Byte Stream:

Byte Stream takes place with byte data. In this type of file handling method, the byte stream process allows execution with byte data.

2. Character Stream:

It takes place with character data. In this type of file handling method, the character stream process allows execution with character data.

Java File Class Methods:

S.No	Method	Return Type	Description

1	canRead()	Boolean	Using this method, we can check if the file can be read or not.
2	createNewFile()	Boolean	We can use this method to create a new file.
3	canWrite()	Boolean	It lets us check if we can write in the file or not.
4	exists()	Boolean	It is used to check if the mentioned file is available or not.
5	delete()	Boolean	Using this method, we can delete a file.
6	getName()	String	We can use this method to find the name of the file.
7	getAbsolutePath()	String	This method helps us in knowing the absolute pathname of the file.
8	length()	Long	It lets us attain the size of the files in bytes
9	list()	String[]	Using this method, we can get the array of files in the directory
10	mkdir()	Boolean	We can use this method to create a new directory

What is File handling in Java?

File handling is the process of reading and writing data into a file. The File class in Java is present in the java.io package. We can simply use the file class by creating an object for the class and specifying the name of the file or directory.

Why is File handling required?

To begin with, let us first understand the importance of this concept.

File handling is an integral part of every programming language. Storing the output of a program in a file format makes it easily retrievable. Therefore, file handling is an essential part of programming.

Java File Operations:

The various operations that we can perform in a file in java are listed below:

- Creating a file
- Getting file information
- Writing into a file
- Reading from a file
- Deleting a file

1. File creation in Java:

The File Creation operation lets us create a new file. The method that performs this action is the createNewFile() method. It returns true after successfully creating a new file. If the field name already exists, this method returns false.

Sample program to create a file in Java:

```
import java.io.File;

import java.IOException;

class FirstCode{

public static void main(String args[]){

try{
```

```
File f() = new File("D:CreatingFileExample.txt");

if(f().createNewFile()){

System.out.println("The file" + f().getName() + "is created succefully.");

}else{

System.out.println("The file name already exists in the directory");

}

}catch(IOException exception){

System.out.println("Error!!!");

exception.printStackTrace();

}

}

}
```

2. Read from a file / Get File Information:

We perform this operation to retrieve the data from the file. Some of the methods that let us achieve this are name, absolute path, is readable, is writeable and length.

Sample program to read information from a file:

```
import java.io.File;

class FirstCode{

public static void main(String args[]){

File f() = new File("D:ReadingFileExample.txt");

if(f().exists()){

System.out.println("The file name is: "+f()getName()); // to get file name
```

```
System.out.println("The absolute path is: "+f().getAbsolutepath()); // to get file path

// to check it the file is writeable or not

System.out.println("Is the file writeable?" +f().canWrite());

// to check if the file is readable or not

System.out.println("Is the file readable?" +f().canRead());

// to get the length of the file

System.out.println("The size of the file in bytes is: "+f().length());

}else{

System.out.println("The file does not exist");

}

}

}
```

3. Write into a file:

To write into a file, we use the FileWriter class and the write() method. The stream should be closed using the close() method to retrieve the resources.

Sample program to write into a file:

```
import java.io.FileWriter;

import java.io.IOException;

class FirstCode{

public static void main(String args[]){

try{
```

```
FileWriter fwrite = new FileWriter("D:FileWritingExample.txt");

fwrite.write("Learn Java with FirstCode");

fwrite.close();

System.out.println("Content is successfully written in the file");

}

catch(IOException e){

System.out.println("Error!!!");

e.printStackTerrace();

}

}

}
```

Java FileWriter:

This class in Java helps the programmers to create new file writing characters. It inherits from the OutputStream class.

The constructors present in this class generally assume that the byte-buffer size and default character encoding are acceptable. To declare these constructors, we must construct OutputStreamWriter on a FileOutputStream.

The Java FileWriter is very much useful for writing streams of characters.constructors

4. Read from a file in java:

To read the content present in the file, we need to use the Scanner class. The stream is closed using the close() method. We can create an object for the Scanner

class to implement the hasNextLine() and nextLine() methods. These methods retrieve the information from the file.

Java FileReader:

The FileReader in Java plays a vital role in reading the data that is present in the form of characters. This is done in the form of a ‘text’ file. The FileReader inherits from the InputStreamReader class.

The constructors in this class assume the default character encoding and the default byte are appropriate. As the Java FileReader is used only for particularly reading streams of character, you can use the FileInputStream to read streams of raw bytes.

Methods:

Methods	Description
public int read() throws IOException	This method read a single character and blocks one until another one is available. That is, an input/ output error occurs.
public int read(char[] cbuff[]) throws IOException	It reads characters into an array and blocks until a character is available.
public abstract int read(char[] buff, int off, int len) which throws an IOException	It read characters into a portion of an array. It blocks until the input is available or an error occurs in the input and output, or the end of the stream is reached.

Parameters:

- cbuff – Destination buffer.
- off – It sets the offset at which to start storing characters.
- len – it uses to see the maximum number of characters to read.

- public long skip(long n) throws IOException: It skips the characters and blocks until some characters are available, an I/O error occurs or the stream end is reached.

Sample program to read from a file:

```
import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

class FirstCode{

public static void main(String args[]){

try{

File f = new File("D:ReadFromFile.txt");

Scanner s =new Scanner(f);

while(s.hasNextLine()){

String fileData = s.nextLine();

System.out.println(fileData);

}

s.close();

}catch(FileNotFoundException exception){

System.out.println("Error!!!");

exception.printStackTrace();

}

}

}
```

Deleting a file:

Knowing to delete a file is as important as creating one. The delete() method performs this task. We need not close the stream with the close() method or use the FileWriter and Scanner class here.

Sample program to delete a file:

```
import java.io.File;

class FirstCode{

public static void main(String args[]){

File f() = new File("D:DeletingFile.txt");

if(f().delete()){

System.out.println(f().getname()+ "The file is deleted");

}else{

System.out.println("Error!!!");

}

}

}
```

Java Directories:

Methods to create and modify files and directories in java:

Methods	Description

renameTo(File path)	The file that the current object represents will be renamed to the path that the file object passes as an argument to the method.
setreadonly()	It sets the files that represent the current object as read-only and returns true when the task is succeeded.
mkdir()	It creates a directory with the path specified by the current file object.
mkdirs()	Creates the directory represented by the current file object, including parent directories that are required.
createanewfile()	It creates a new empty file with a pathname defined by the current file object as long as the file exists.
delete()	This deletes the file in a directory represented by the current file object and returns true if the delete is successful.
	createDirectory(Path, FileAttribute</>)

Creating Directories in java:

Java provides a couple of beneficial File utility methods that are used to create directories.

- The mkdir() method creates a directory. It returns true for success and false for failure. The failure denotes that the path that is mentioned in the File object exists already. It also denotes that the directory cannot be created if the path does not exist.
- The mkdirs() methods create both a directory and also parents of the directory.

Sample program to create a directory:

```
import java.io.File;
```

```
public class CreateDirFirstCode{

public static void main(String args[]) {

String dirname = "/tmp/user/java/bin";

File d = new File(dirname);

d.mkdirs();

}

}
```

Listing directories in Java

The File object provides the list() method to list down all the files and directories that are present.

Sample program to list the directories:

```
import java.io.File;

public class ReadDirFirstCode {

public static void main(String[] args) {

File file = null;

String[] paths;

try {

file = new File("/tmp");

paths = file.list();

for(String path:paths) {

System.out.println(path);

}
```

```
} catch (Exception e) {

e.printStackTrace();

}

}

}
```

Output:

test1.txttest2.txt

ReadDir.java

ReadDir.class

Querying Files and Directories in Java:

Method	Description
exists()	It returns if the file or directory referred to by the file object exists and false otherwise
isfile()	It returns if the file object refers to an existing file and false otherwise.
canread()	It returns true if it allows reading the sile that is referred by the file object.
canwrite()	It returns true if you are permitted to write the file referred by the file object.

Constructors:

Constructors	Description
FileWriter(File file)	This constructor constructs a FileWriter object when a file object is given.
FileWriter(File file, boolean append)	It constructs an object for Filewriter.
FileWriter(FileDescriptor fd)	Constructs a FileWriter object associated using a file descriptor.
FileWriter(String fileName)	It constructs a FileWriter object when a file name is given.
public void write(int c) throws IOException	It writes a single character.
public void write(char[] stir) throws IOException	It writes an array of character.
public void write(String str) throws IOException	It writes a string in Java.
FileWriter(String filename, Boolean append)	It constructs a FileWriter object when a file name is present.
public void write(String str, int off, int len) throws IOException	It writes a portion of a String.

Conclusion:

This article would have helped you in knowing various details about file handling in Java. You can now create your own file and perform various file handling operations to it. This lets you store the data and retrieve them easily.