OBJECT ORIENTED PROGRAMMING USING JAVA
Paper Code: CSCS231

MODULE – IV

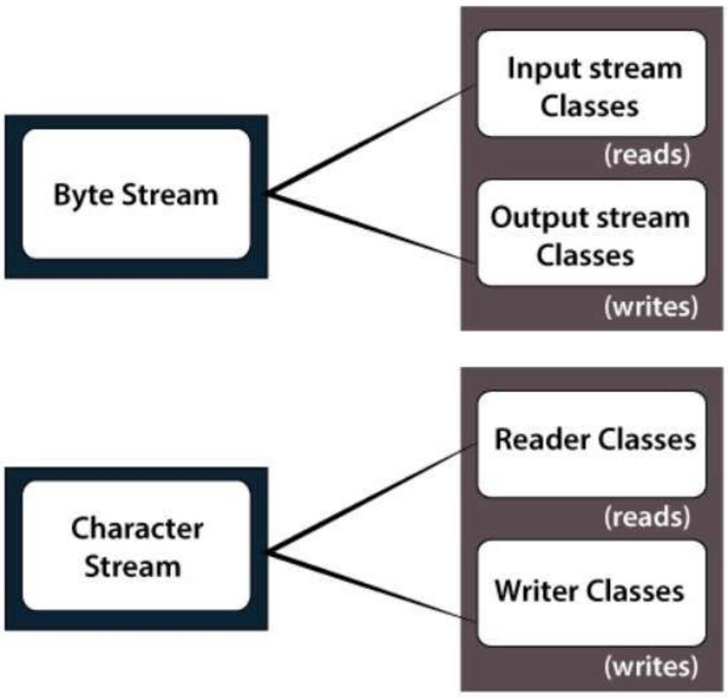
Files, Streams & I/O – Introduction – Files & Streams – Sequential Access Text Files

Introduction

File handling in Java is defined as reading and writing data to a file. The particular file class from the package called java.io allows us to handle and work with different formats of files. Thus, if we want to use a file class, we need to create an object of that particular class and should specify the filename or directory name.

Stream

A series of data is referred to as a stream. In Java, Stream is classified into two types, i.e., Byte Stream and Character Stream. This package that supports stream input/output is java.io, and it is vast. This package defines over seventy classes and interfaces, many of which have a large number of methods.



The stream is the only representation of input or output that is maybe a source or destination of the data. We can also write the data in the stream or read the particular data from the stream. We can also visualize the stream of data as a sequence of bytes that flow out of the program.

Input and Output Streams

Writing data in the stream or to a stream is called an output stream. The particular output stream can go to any device which can connect through a hard disk or maybe any stream which can contain the sequence of bytes. An output stream also can be displayed on any output screen, which has its true capability. Stream output to your display is output to your command line. When you start writing something for your display screen, it can only display the characters but not the graphical content. Graphical output requires more specialized support.

You read data from an input stream. In principle, this can only be any source of serial data but is typically a disk file, the keyboard, or a remote computer.

Further, under normal circumstances, the file input and output you write in a machine can only be through a java application. It's not available to java apples except strictly to a limited extent.

This has two advantages: First, you don't have to worry about the detailed mechanics of each device, which are taken care of behind the scenes.

The streams which you write under the input and output can only have a small amount of data like a single character, but not more than that. Transferring data to or from a stream like this may be extremely inefficient, so a stream often equipped with a buffer in memory, in which case it is called a buffered stream.

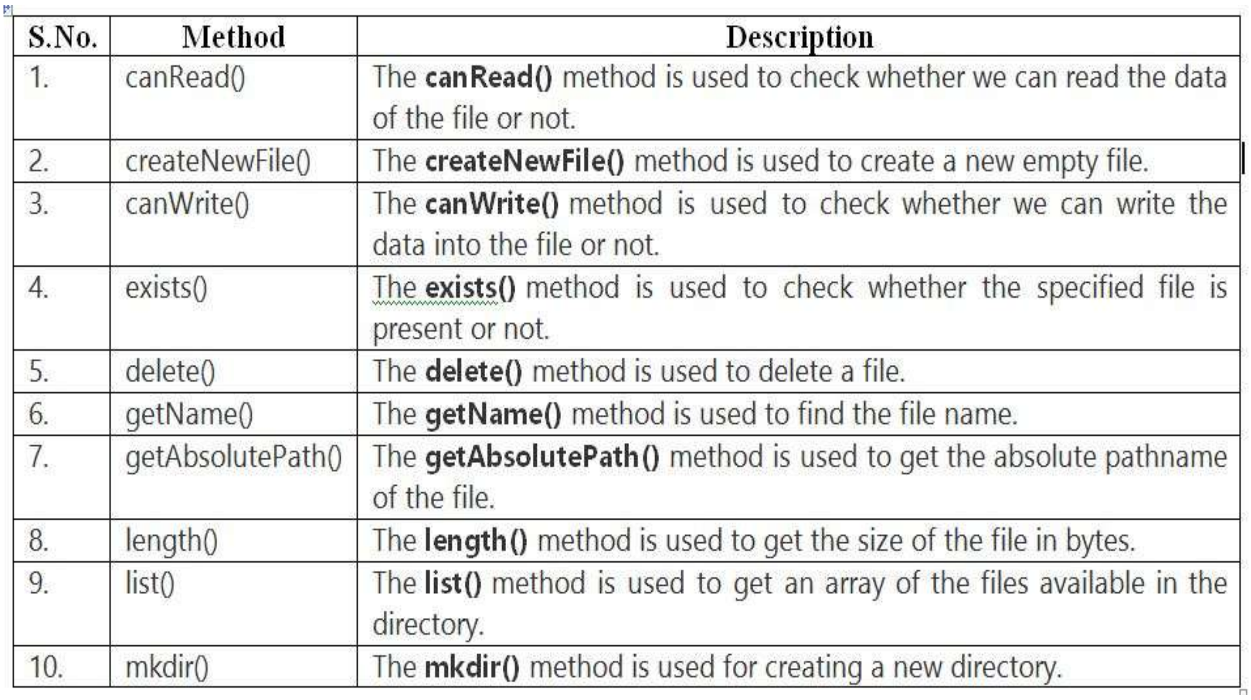
The Classes for Input and Output

The logical structure forms with the number of stream classes. Once you know how they are related, you shouldn't have much trouble using them. We will work through the class hierarchy from the top down, so you will be able to see how the classes hang together and how you can combine them in different ways to suit different situations.

java.io it contains all the classes for the support of the streams.

Class	**Description**
Input Stream	The base class for byte stream input operations.
Output Stream	The base class for byte stream output operations.

File Related Functions



S.No.	Method	Description
1.	canRead()	The **canRead()** method is used to check whether we can read the data of the file or not.
2.	createNewFile()	The **createNewFile()** method is used to create a new empty file.
3.	canWrite()	The **canWrite()** method is used to check whether we can write the data into the file or not.
4.	exists()	The **exists()** method is used to check whether the specified file is present or not.
5.	delete()	The **delete()** method is used to delete a file.
6.	getName()	The **getName()** method is used to find the file name.
7.	getAbsolutePath()	The **getAbsolutePath()** method is used to get the absolute pathname of the file.
8.	length()	The **length()** method is used to get the size of the file in bytes.
9.	list()	The **list()** method is used to get an array of the files available in the directory.
10.	mkdir()	The **mkdir()** method is used for creating a new directory.

Sequential Access Text Files

In Java, a File is an abstract data type. A named location used to store related information is known as a File. There are several File Operations like creating a new File, getting information about File, writing into a File, reading from a File and deleting a File. In simple words, file handling means reading and writing data to a file.

The process for writing a file is basically quite simple. For writing a file, you will load a file that you have created as one or more buffers and call a method for that particular object to write data to that file which is encapsulated by the file stream.

To start with, you will be using the simplest write() method for a file channel that writes the data contained in a single ByteBuffer object to a file. The number of bytes written to the file is determined by the buffers position and limit when the write() method executes.

The following example shows write data into a text file.

%%%%%%%%%%%%%% FW.java %%%%%%%%%%%%%%%

```
import java.io.FileWriter;
import java.io.IOException;
 public class FW {
   public static void main(String[] args)
   {
       try {
           FileWriter wr = new FileWriter("myfile.txt");
           wr.write("Files in Java are seriously good!!");
           wr.close();
           System.out.println("Successfully written.");
       }
       catch (IOException e) {
           System.out.println(e.getMessage());
       }
   }
}
```

Reading files

The process we use for reading a file is quite similar to writing a file. First, you will obtain a file channel of an object from a file stream and also use the same channel to read the data for one or more buffers. Initially, you will be using a channel object that you obtain from a FileInputStream object to read a file. Later you will be using a FileChannel object obtained from a RandomAccessFile object to read and write the same file.

Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

Constructors of FileWriter class

Constructor	Description
FileWriter(String file)	Creates a new file. It gets file name in string.
FileWriter(File file)	Creates a new file. It gets file name in File object.

Methods of FileWriter class

Method	Description
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

The following example shows read a file.

%%%%%%%%%%%%%%%%% FR.java %%%%%%%%%%%%%%%

```
import java.io.FileReader;
import java.io.IOException;
public class FR {
    public static void main(String[] args)
    {
        try {
            FileReader fr = new FileReader("myfile.txt");
             int i;
             while ((i = fr.read()) != -1) {
                System.out.print((char)i);
            }
            fr.close();
           System.out.println("\nFile reading both done");
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```