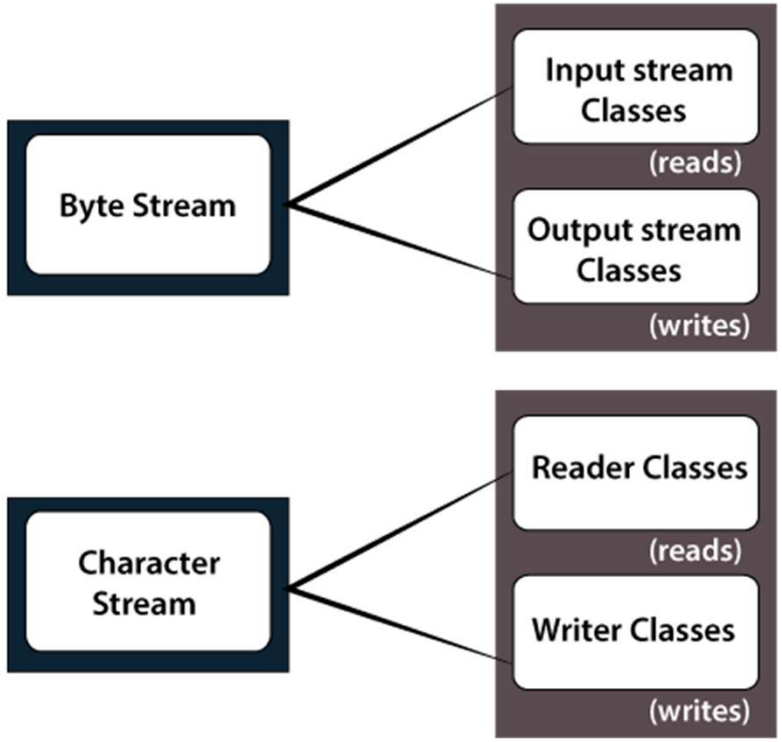
FILE HANDLING

INTRODUCTION

- File Handling is abstract type and integral part of JAVA programming language which enables us to store the output of any particular program in a file and allows us to perform certain operations on it.
- In simple words, file handling means reading and writing data to 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.

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.
- In Java, a sequence of data is known as a stream. This concept is used to perform I/O operations on a file.
- There are two types of streams :



Brief classification of I/O streams

Byte Stream

- This stream is used to read or write byte data. The byte stream is again subdivided into two types which are as follows:
    1. **Byte Input Stream**: Used to read byte data from different devices.
    2. **Byte Output Stream**: Used to write byte data to different devices.

Byte Stream

Input Stream Classes

SN	Class	Description
1	BufferedInputStream	This class provides methods to read bytes from the buffer.
2	ByteArrayInputStream	This class provides methods to read bytes from the byte array.
3	DataInputStream	This class provides methods to read Java primitive data types.
4	FileInputStream	This class provides methods to read bytes from a file.
5	FilterInputStream	This class contains methods to read bytes from the other input streams, which are used as the primary source of data.
6	ObjectInputStream	This class provides methods to read objects.
7	PipedInputStream	This class provides methods to read from a piped output stream to which the piped input stream must be connected.
8	SequenceInputStream	This class provides methods to connect multiple Input Stream and read data from them.

Byte Stream

SN	Method	Description
1	int read()	This method returns an integer, an integral representation of the next available byte of the input. The integer -1 is returned once the end of the input is encountered.
2	int read (byte buffer [])	This method is used to read the specified buffer length bytes from the input and returns the total number of bytes successfully read. It returns -1 once the end of the input is encountered.
3	int read (byte buffer [], int loc, int nBytes)	This method is used to read the 'nBytes' bytes from the buffer starting at a specified location, 'loc'. It returns the total number of bytes successfully read from the input. It returns -1 once the end of the input is encountered.
4	int available ()	This method returns the number of bytes that are available to read.
5	Void mark(int nBytes)	This method is used to mark the current position in the input stream until the specified nBytes are read.
6	void reset ()	This method is used to reset the input pointer to the previously set mark.
7	long skip (long nBytes)	This method is used to skip the nBytes of the input stream and returns the total number of bytes that are skipped.
8	void close ()	This method is used to close the input source. If an attempt is made to read even after the closing, IOException is thrown by the method.

Byte Stream

Output Stream Classes

SN	Class	Description
1	BufferedOutputStream	This class provides methods to write the bytes to the buffer.
2	ByteArrayOutputStream	This class provides methods to write bytes to the byte array.
3	DataOutputStream	This class provides methods to write the java primitive data types.
4	FileOutputStream	This class provides methods to write bytes to a file.
5	FilterOutputStream	This class provides methods to write to other output streams.
6	ObjectOutputStream	This class provides methods to write objects.
7	PipedOutputStream	It provides methods to write bytes to a piped output stream.
8	PrintStream	It provides methods to print Java primitive data types.

Byte Stream

SN	Method	Description
1	void write (int i)	This method is used to write the specified single byte to the output stream.
2	void write (byte buffer [])	It is used to write a byte array to the output stream.
3	Void write(bytes buffer[],int loc, int nBytes)	It is used to write nByte bytes to the output stream from the buffer starting at the specified location.
4	void flush ()	It is used to flush the output stream and writes the pending buffered bytes.
5	void close ()	It is used to close the output stream. However, if we try to close the already closed output stream, the IOException will be thrown by this method.

Character Stream

- This stream is used to read or write character data. Character stream is again subdivided into 2 types which are as follows:
  1. **Character Input Stream**: Used to read character data from different devices.
  2. **Character Output Stream**: Used to write character data to different devices.

Character Stream

Reader Classes

SN	Class	Description
1.	BufferedReader	This class provides methods to read characters from the buffer.
2.	CharArrayReader	This class provides methods to read characters from the char array.
3.	FileReader	This class provides methods to read characters from the file.
4.	FilterReader	This class provides methods to read characters from the underlying character input stream.
5	InputStreamReader	This class provides methods to convert bytes to characters.
6	PipedReader	This class provides methods to read characters from the connected piped output stream.
7	StringReader	This class provides methods to read characters from a string.

Character Stream

SN	Method	Description
1	int read()	This method returns the integral representation of the next character present in the input. It returns -1 if the end of the input is encountered.
2	int read(char buffer[])	This method is used to read from the specified buffer. It returns the total number of characters successfully read. It returns -1 if the end of the input is encountered.
3	int read(char buffer[], int loc, int nChars)	This method is used to read the specified nChars from the buffer at the specified location. It returns the total number of characters successfully read.
4	void mark(int nchars)	This method is used to mark the current position in the input stream until nChars characters are read.
5	void reset()	This method is used to reset the input pointer to the previous set mark.
6	long skip(long nChars)	This method is used to skip the specified nChars characters from the input stream and returns the number of characters skipped.
7	boolean ready()	This method returns a boolean value true if the next request of input is ready. Otherwise, it returns false.
8	void close()	This method is used to close the input stream. However, if the program attempts to access the input, it generates IOException.

Character Stream

Writer Classes

SN	Class	Description
1	BufferedWriter	This class provides methods to write characters to the buffer.
2	FileWriter	This class provides methods to write characters to the file.
3	CharArrayWriter	This class provides methods to write the characters to the character array.
4	OutpuStreamWriter	This class provides methods to convert from bytes to characters.
5	PipedWriter	This class provides methods to write the characters to the piped output stream.
6	StringWriter	This class provides methods to write the characters to the string.

Character Stream

SN	Method	Description
1	void write()	This method is used to write the data to the output stream.
2	void write(int i)	This method is used to write a single character to the output stream.
3	Void write(char buffer[])	This method is used to write the array of characters to the output stream.
4	void write(char buffer [],int loc, int nChars)	This method is used to write the nChars characters to the character array from the specified location.
5	void close ()	This method is used to close the output stream. However, this generates the IOException if an attempt is made to write to the output stream after closing the stream.
6	void flush ()	This method is used to flush the output stream and writes the waiting buffered characters.

Java File Class Method

S.N.	Method	Return Type	Description
1.	canRead()	Boolean	The **canRead()** method is used to check whether we can read the data of the file or not.
2.	createNewFile()	Boolean	The **createNewFile()** method is used to create a new empty file.
3.	canWrite()	Boolean	The **canWrite()** method is used to check whether we can write the data into the file or not.
4.	exists()	Boolean	The **exists()** method is used to check whether the specified file is present or not.
5.	delete()	Boolean	The **delete()** method is used to delete a file.
6.	getName()	String	The **getName()** method is used to find the file name.
7.	getAbsolutePath()	String	The **getAbsolutePath()** method is used to get the absolute pathname of the file.
8.	length()	Long	The **length()** method is used to get the size of the file in bytes.
9.	list()	String[]	The **list()** method is used to get an array of the files available in the directory.
10	mkdir()	Boolean	The **mkdir()** method is used for creating a new directory.

File Operation in Java

- The following are the several operations that can be performed on a file in Java :
  a. Create a File
  b. Read from a File
  c. Write to a File
  d. Delete a File

File Operation in Java

Create File

- Create a File operation is performed to create a new file. We use the createNewFile() method of file.
- The createNewFile() method returns true when it successfully creates a new file and returns false when the file already exists.

File Operation in Java

Read From File

- In order to read data from a file, we will use the Scanner class. we need to close the stream using the close() method.
- We will create an instance of the Scanner class and use the **hasNextLine**() method and **nextLine**() method to get data from the file.

File Operation in Java

Write to a File

- In order to write data into a file, we will use the **FileWriter** class and its **write**() method together. We need to close the stream using the **close**() method to retrieve the allocated resources.

File Operation in Java

Delete a File

- In order to delete a file, we will use the **delete**() method of the file. We don't need to close the stream using the **close**() method because for deleting a file, we neither use the **FileWriter** class nor the **Scanner** class.