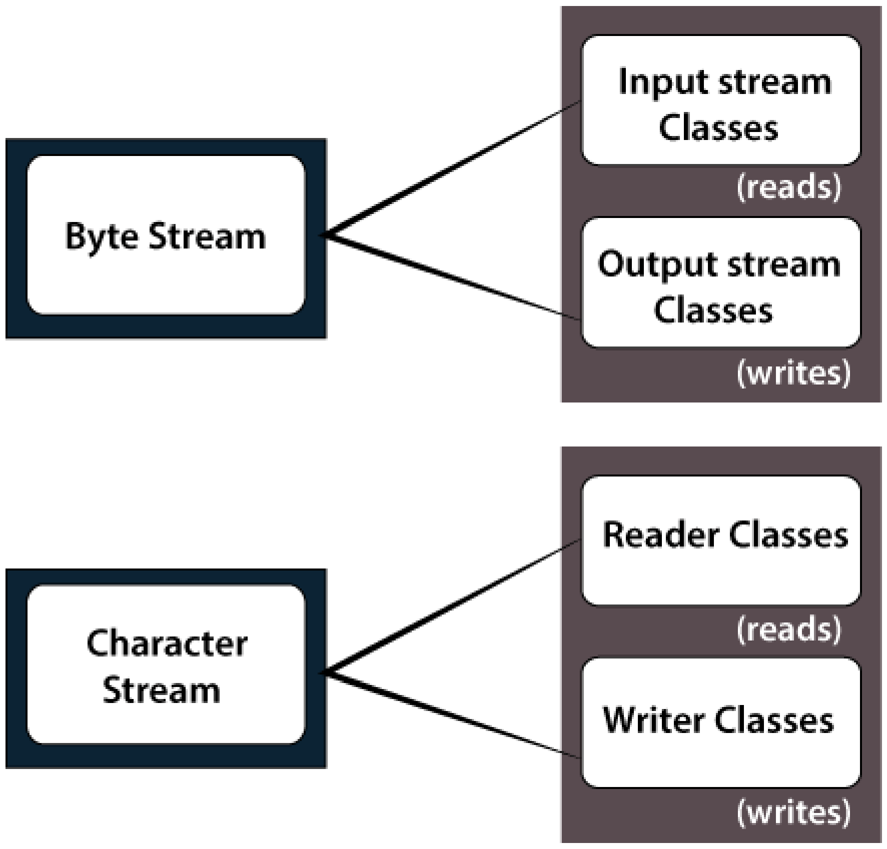
Stream

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**.



Brief classification of I/O streams

Byte Stream

Byte Stream is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.

Character Stream

Character Stream is mainly involved with character data. A file handling process with a character stream is a process in which an input is provided and executed with the character data.

ByteStream Classes in Java

ByteStream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that ByteStream classes read/write the data of 8-bits. We can store video, audio, characters, etc., by using ByteStream classes. These classes are part of the java.io package.

The ByteStream classes are divided into two types of classes, i.e., InputStream and OutputStream. These classes are abstract and the super classes of all the Input/Output stream classes.

InputStream Class

The InputStream class provides methods to read bytes from a file, console or memory. It is an abstract class and can't be instantiated; however, various classes inherit the InputStream class and override its methods. The subclasses of InputStream class are given in the following table.

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.

The InputStream class contains various methods to read the data from an input stream. These methods are overridden by the classes that inherit the InputStream class. However, the methods are given in the following table.

OutputStream Class

The OutputStream is an abstract class that is used to write 8-bit bytes to the stream. It is the superclass of all the output stream classes. This class can't be instantiated; however, it is inherited by various subclasses that are given in the following table.

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.

CharacterStream Classes in Java

The java.io package provides CharacterStream classes to overcome the limitations of ByteStream classes, which can only handle the 8-bit bytes and is not compatible to work directly with the Unicode characters. CharacterStream classes are used to work with 16-bit Unicode characters. They can perform operations on characters, char arrays and Strings.

However, the CharacterStream classes are mainly used to read characters from the source and write them to the destination. For this purpose, the CharacterStream classes are divided into two types of classes, I.e., Reader class and Writer class.

Reader Class

Reader class is used to read the 16-bit characters from the input stream. However, it is an abstract class and can't be instantiated, but there are various subclasses that inherit the Reader class and override the methods of the Reader class. All methods of the Reader class throw an IOException. The subclasses of the Reader class are given in the following table.

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.

Writer Class

Writer class is used to write 16-bit Unicode characters to the output stream. The methods of the Writer class generate IOException. Like Reader class, Writer class is also an abstract class that cannot be instantiated; therefore, the subclasses of the Writer class are used to write the characters onto the output stream. The subclasses of the Writer class are given in the below table.

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.

In Java the streams are used for input and output operations by allowing data to be read from or written to a source or destination.

Java offers two types of streams:

1. character streams
2. byte streams.

These streams can be different in how they are handling data and the type of data they are handling.

1. Character Streams:

Character streams are designed to address character based records, which includes textual records inclusive of letters, digits, symbols, and other characters. These streams are represented by way of training that quit with the phrase "Reader" or "Writer" of their names, inclusive of FileReader, BufferedReader, FileWriter, and BufferedWriter.

Character streams offer a convenient manner to read and write textual content-primarily based information due to the fact they mechanically manage character encoding and decoding. They convert the individual statistics to and from the underlying byte circulation the usage of a particular individual encoding, such as UTF-eight or ASCII.It makes person streams suitable for operating with textual content files, analyzing and writing strings, and processing human-readable statistics.

2. Byte Streams:

Byte streams are designed to deal with raw binary data, which includes all kinds of data, including characters, pictues, audio, and video. These streams are represented through cclasses that cease with the word "InputStream" or "OutputStream" of their names,along with FileInputStream,BufferedInputStream, FileOutputStream and BufferedOutputStream.

Byte streams offer a low-stage interface for studying and writing character bytes or blocks of bytes. They are normally used for coping with non-textual statistics, studying and writing files of their binary form, and running with network sockets. Byte streams don't perform any individual encoding or deciphering. They treat the data as a sequence of bytes and don't interpret it as characters.

Here are the some of the differences listed:

Aspect	Character Streams	Byte Streams
Data Handling	Handle character-based data	Handle raw binary data
Representation	Classes end with "Reader" or "Writer"	Classes end with "InputStream" or "OutputStream"
Suitable for	Textual data, strings, human-readable info	Non-textual data, binary files, multimedia
Character Encoding	Automatic encoding and decoding	No encoding or decoding
Text vs non-Text data	Text-based data, strings	Binary data, images, audio, video
Performance	Additional conversion may impact performance	Efficient for handling large binary data
Handle Large Text Files	May impact performance due to encoding	Efficient, no encoding overhead
String Operations	Convenient methods for string operations	Not specifically designed for string operations
Convenience Methods	Higher-level abstractions for text data	Low-level interface for byte data
Reading Line by Line	Convenient methods for reading lines	Byte-oriented, no built-in line-reading methods
File Handling	Read/write text files	Read/write binary files
Network Communication	Sending/receiving text data	Sending/receiving binary data
Handling Images/Audio/Video	Not designed for handling binary data directly	Suitable for handling binary multimedia data
Text Encoding	Supports various character encodings	No specific text encoding support

Example code for Character Stream:

FileName: CharacterStreamExample.java

```java
import java.io.CharArrayReader;
import java.io.IOException;

public class CharacterStreamExample
{
    public static void main(String[] args) {
        // Creates an array of characters
        char[] array = {'H','e','l','l','o'};
        try {
            CharArrayReader reader=new CharArrayReader(array);
            System.out.print("The characters read from the reader:");
            int charRead;
            while ((charRead=reader.read())!=-1) {
                System.out.print((char)charRead+",");
            }
            reader.close();
        } catch (IOException ex)
{
            ex.printStackTrace();
        }
    }
}
```

Output:

```
The characters read from the reader:H,e,l,l,o,
```

Example code for Byte Stream:

FileName: ByteStreamExample.java

```java
import java.io.ByteArrayInputStream;

public class ByteStreamExample
{
    public static void main(String[] args)
     {
      // Creates the array of bytes
      byte[] array = {10,20,30,40};
      try {
        ByteArrayInputStream input=new ByteArrayInputStream(array);
        System.out.println("The bytes read from the input stream:");
        for (int i=0;i<array.length;i++)
        {
          // Reads the bytes
          int data=input.read();
          System.out.print(data+",");
        }
        input.close();
      } catch (Exception ex)
{
        ex.getStackTrace();
      }
    }
}
```

Output:

```
The bytes read from the input stream:10,20,30,40.
```

Reading and Writing Files in Java

This is the java program to read a content and write it into a file.

```
import java.io.ByteArrayInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;


  public class InputOutputStreamExample {

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

// TODO Auto-generated method stub

byte content[] = "Jtp is the best website to learn new technologies".getBytes();

ByteArrayInputStream inputStream = new ByteArrayInputStream(content);

  inputStream.read(content);

  File newFile = new File("C:/Users/vinita sharma/Desktop/newfile.txt");

FileOutputStream outputStream = new FileOutputStream(newFile);

outputStream.write(content);

  }

  }
```