Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream

Code to print **output and an error** message to the console:

```
1. System.out.println("simple message");
2. System.err.println("error message");
```

Code to get **input** from console.

```
1. int i=System.in.read();//returns ASCII code of 1st character
2. System.out.println((char)i);//will print the character
```

OutputStream vs InputStream

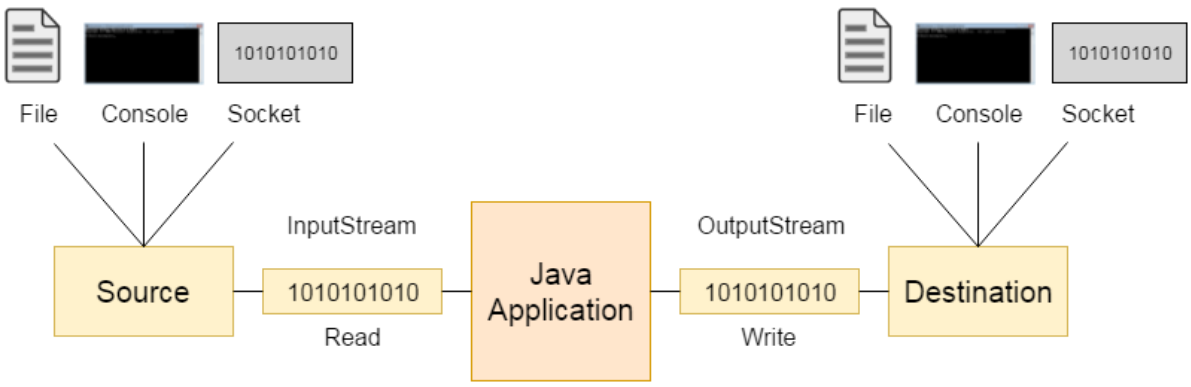
OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream by the figure given below.



9.1 Java.io package

The java.io package in Java provides classes for input and output operations. It is an essential part of Java's I/O (Input/Output) functionality. Some key classes and concepts in the java.io package include:

InputStream and OutputStream:
InputStream is the abstract class for all input streams, and OutputStream is the abstract class for all output streams. They serve as the foundation for reading and writing bytes of data, respectively.

File class:
The File class represents a file or directory in the file system. It provides methods for file manipulation, such as creating, deleting, and checking the existence of files and directories.

Reader and Writer:
Reader and Writer are abstract classes for reading and writing characters, respectively. They are built on top of the InputStream and OutputStream classes and provide methods for handling character-based I/O.

BufferedInputStream and BufferedOutputStream:
These classes provide buffering for input and output streams, improving efficiency by reducing the number of I/O operations.

FileReader and FileWriter:
These classes are used for reading and writing character data to files, respectively. They are built on top of the Reader and Writer classes.

BufferedReader and BufferedWriter:
These classes provide buffering for character input and output streams, enhancing performance by reducing the frequency of file system calls.

PrintStream and PrintWriter:
These classes provide formatted output for different data types. PrintStream is used for outputting bytes, while PrintWriter is used for outputting characters.

The java.io package is an integral part of Java programming, offering a wide range of classes to handle various I/O scenarios, from simple file operations to more complex input and output streams.

9.2 Byte Stream and Character Stream classes

Byte Stream reads and writes data byte by byte (8 bits). **InputStream** and **OutputStream** are two abstract classes provided by java that are based on Byte Stream.

Note that most of the applications today use Unicode(UTF-8 or UTF-16) to store text data. It may take one or more bytes to represent a single character in UTF-8. Therefore, when reading text data, a single byte in the data may not correspond to one character in UTF. If you just read one byte at a time of UTF-8 data via a Byte Stream and try to convert each byte into a char, you may not end up with the text you expected.

Character Stream reads and writes data character by character using unicode conventions. **Reader** and **Writer** are two abstract classes provided by java that are based on Character Stream.

9.3 Using FileInputStream and FileOutputStream classes

We can read data from file by using FileInputStream class and write data to files using FileOutputStream class.

Java FileInputStream Class

FileInputStream is a subclass of InputStream and is used for reading bytes from a file.

It opens an input stream to a specified file, allowing you to read data from the file as a stream of bytes.

FileInputStream fis = new FileInputStream("example.txt");

Java FileInputStream class methods

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
void close()	It is used to closes the stream.

Java FileInputStream Example: read all characters

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=0;
          while((i=fin.read())!=-1){
           System.out.print((char)i);
          }
          fin.close();
         }catch(Exception e){System.out.println(e);}
        }
       }
```

Output:
Welcome to Pokhara Engineering College

Java FileOutputStream Class
Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

FileOutputStream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
void close()	It is used to closes the file output stream.

Java FileOutputStream Example :write string

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
   public static void main(String args[]){
        try{
         FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
         String s="Welcome to Pokhara Engineering College";
         byte b[]=s.getBytes();//converting string into byte array
         fout.write(b);
         fout.close();
         System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
```

Output:
Success...

The content of a text file **testout.txt** is set with the data **Welcome to Pokhara Engineering College**

Output File:
In D:\testout.txt
Welcome to Pokhara Engineering College

9.4 Using FileReader and FileWriter Classes

Java FileWriter and FileReader classes are used to write and read data from text files (they are Character Stream classes). It is recommended **not** to use the FileInputStream and FileOutputStream classes if you have to read and write any textual information as these are Byte stream classes.

FileWriter

FileWriter is useful to create a file writing characters into it.

- This class inherits from the OutputStream class.
- The constructors of this class assume that the default character encoding and the default byte-buffer size are acceptable. To specify these values yourself, construct an OutputStreamWriter on a FileOutputStream.

- FileWriter is meant for writing streams of characters. For writing streams of raw bytes, consider using a FileOutputStream.
- FileWriter creates the output file if it is not present already.

Constructors:

- **FileWriter(File file) –** Constructs a FileWriter object given a File object.
- **FileWriter (File file, boolean append) –** constructs a FileWriter object given a File object.
- **FileWriter (FileDescriptor fd) –** constructs a FileWriter object associated with a file descriptor.
- **FileWriter (String fileName) –** constructs a FileWriter object given a file name.

- **FileWriter (String fileName, Boolean append) –** Constructs a FileWriter object given a file name with a Boolean indicating whether or not to append the data written.

Methods:

- **public void write (int c) throws IOException –** Writes a single character.
- **public void write (char [] stir) throws IOException –** Writes an array of characters.
- **public void write(String str)throws IOException –** Writes a string.
- **public void write(String str, int off, int len)throws IOException –** Writes a portion of a string. Here off is offset from which to start writing characters and len is the number of characters to write.
- **public void flush() throws IOException** flushes the stream
- **public void close() throws IOException** flushes the stream first and then closes the writer.

Reading and writing take place character by character, which increases the number of I/O operations and affects the performance of the system.**BufferedWriter** can be used along with FileWriter to improve the speed of execution. The following program depicts how to create a text file using FileWriter

```
// Creating a text File using FileWriter

import java.io.FileWriter;
import java.io.IOException;
class CreateFile
{
   public static void main(String[] args) throws IOException
   {
       String str = "File Handling in Java using FileWriter and FileReader";

      // attach a file to FileWriter
      FileWriter fw=new FileWriter("output.txt");

      // read character wise from string and write
      // into FileWriter
      for (int i = 0; i < str.length(); i++)
         fw.write(str.charAt(i));

      System.out.println("Writing successful");
      //close the file
      fw.close();
   }
}
```

FileReader

FileReader is useful to read data in the form of characters from a 'text' file.

- This class inherited from the InputStreamReader Class.
- The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate. To specify these values yourself, construct an InputStreamReader on a FileInputStream.

- FileReader is meant for reading streams of characters. For reading streams of raw bytes, consider using a FileInputStream.

Constructors:

- **FileReader(File file) –** Creates a FileReader , given the File to read from
- **FileReader(FileDescripter fd) –** Creates a new FileReader , given the FileDescripter to read from
- **FileReader(String fileName) –** Creates a new FileReader , given the name of the file to read from

Methods:

- **public int read () throws IOException –** Reads a single character. This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.
- **public int read(char[] cbuff) throws IOException –** Reads characters into an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.
- **public abstract int read(char[] buff, int off, int len) throws IOException –**Reads characters into a portion of an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.
  Parameters:
  cbuf – Destination buffer
  off – Offset at which to start storing characters
  len – Maximum number of characters to read

- **public void close() throws IOException** closes the reader.
- **public long skip(long n) throws IOException –**Skips characters. This method will block until some characters are available, an I/O error occurs, or the end of the stream is reached.
  Parameters:
  n – The number of characters to skip

The following program depicts how to read from the 'text' file using FileReader

```
// Reading data from a file using FileReader
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
class ReadFile
{
   public static void main(String[] args) throws IOException
   {
      // variable declaration
      int ch;

      // check if File exists or not
      FileReader fr=null;
      try
      {
         fr = new FileReader("text");
      }
      catch (FileNotFoundException fe)
      {
```

```
            System.out.println("File not found");
        }

        // read from FileReader till the end of file
        while ((ch=fr.read())!=-1)
            System.out.print((char)ch);

        // close the file
        fr.close();
    }
}
```