

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 IO 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 IO, API.

Core Concepts of Java IO.

Java IO revolves around two primary concepts: streams and readers/writers.

Streams: Streams represent a sequence of data. In Java there are two types of streams. Input streams and output streams. Input streams are used to read data from a source. While output streams are used to write data to A destination. Streams can be categorized into byte streams (InputStream and OutputStream) and character streams (Reader and Writer).

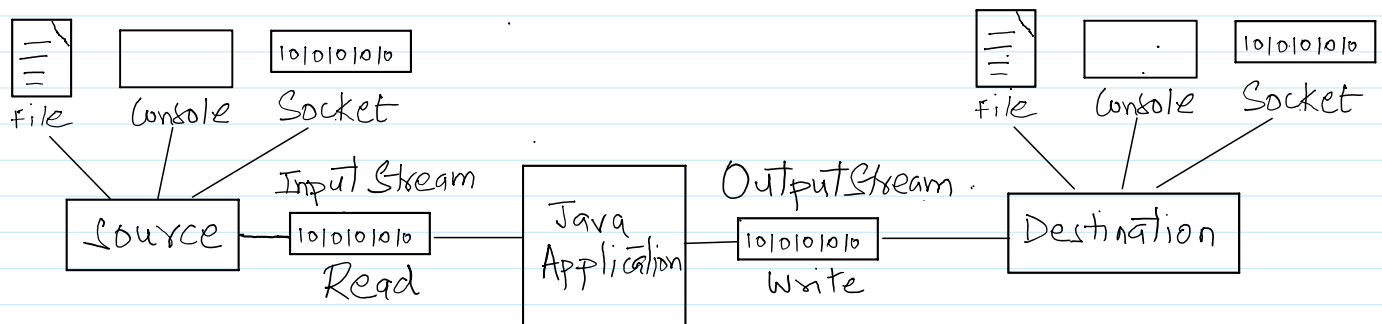
Readers/Writers: Readers and Writers are specialized stream classes designed for handling character data. They provide a convenient way to read from and write to character based data sources. Readers read character data from input streams, while Writer writes character data to output streams.

Streams: 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.

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.

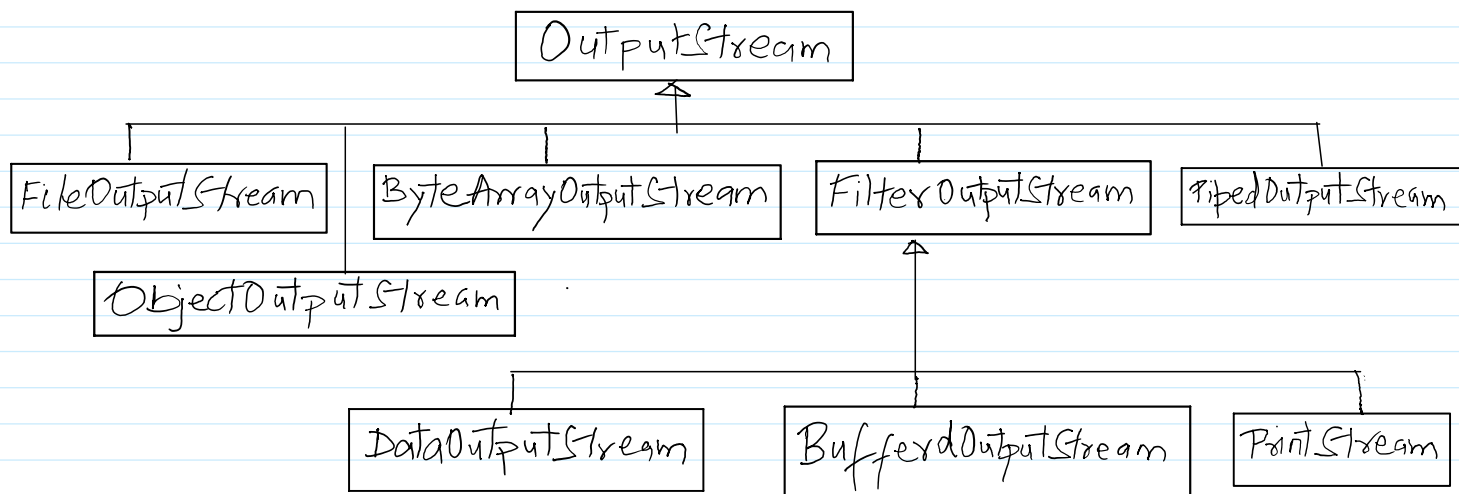
InputStream: Java application uses an input stream to read data from a source. It may be a file, an array, peripheral device or socket.

OutputStream Class: Output stream class is an abstract class. It is a super class of all classes representing an output stream of bytes. An output stream accepts output bytes and send them to some sink.

Useful methods of OutputStream Class

Method	Description
--------	-------------

<code>public void write(int) throws IOException</code>	It is used to write a byte to a current output stream.
<code>public void write(byte[]) throws IOException</code>	It is used to write an array of byte to the current output string.
<code>public void flush() throws IOException</code>	It flushes the current output stream.
<code>public void close() throws IOException</code>	It is used to close the current output stream.



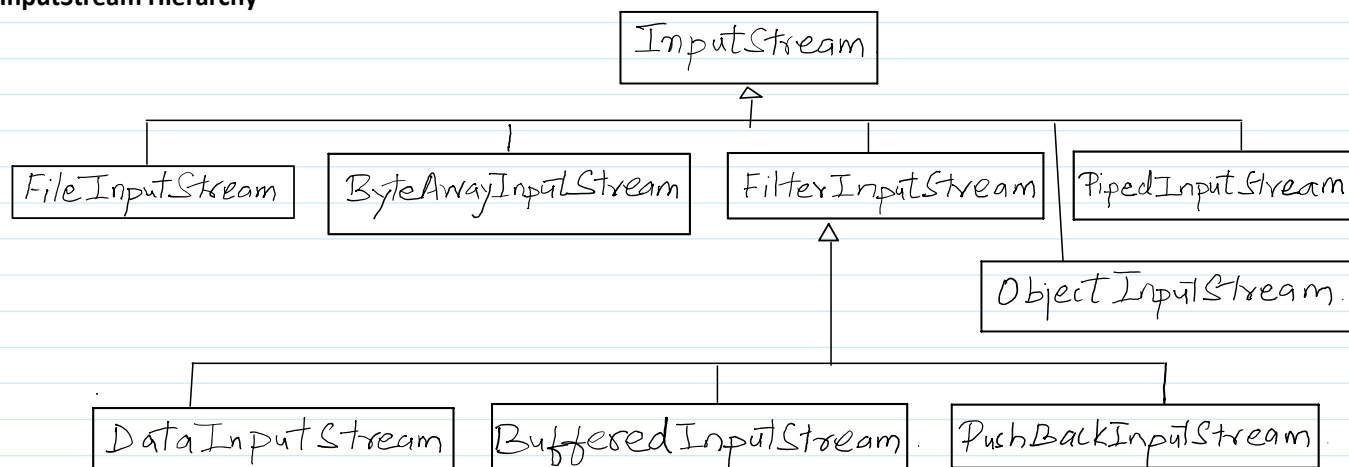
InputStream Class

InputStream class is an abstract class. It is the Super class of all classes representing an input stream of bytes.

Useful methods of InputStream class

Method	Description
<code>public abstract int read() throws IOException</code>	It reads the next byte of data from the input stream. It returns '-1' at the end of the file.
<code>public int available() throws IOException</code>	It returns an estimate of the number of bytes that can be read from the current inputs stream.
<code>public void close() throws IOException</code>	It is used to close the current input stream.

InputStream Hierarchy

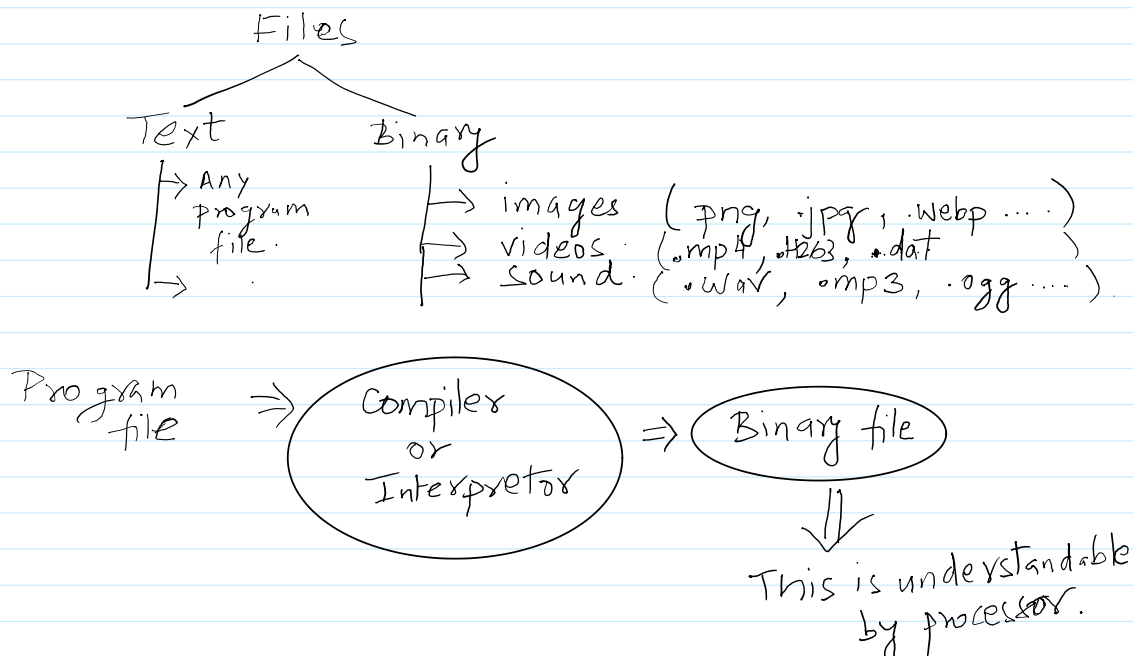


Java I/O Classes

Java provides a rich set of classes for performing IO operation. Some of the key classes include:

1. **InputStream and OutputStream:** These abstract classes form the foundation of byte oriented IO operations. They provide methods for reading and writing bytes from or to various sources and destinations.

2. **Reader and Writer:** These abstract classes are used for character based IO operations. They provide methods for reading and writing characters from or to character-based streams.
3. **FileInputStream and FileOutputStream:** These classes allow reading from and writing to files in a byte oriented manner.
4. **FileReader and FileWriter:** These classes enable reading from and writing to file using character oriented operations.
5. **BufferedInputStream and BufferedOutputStream:** These classes provide buffering capabilities which can significantly improve IO performance by reducing the number of system calls.
6. **BufferedReader and BufferedWriter:** These classes offer buffered reading and writing of character data, enhancing IO efficiency when working with character based streams.



Practical Applications of Java IO

Java IO is employed in various real world scenarios including:

1. **File Handling**
2. **Network Communication**
3. **Serialization**
4. **Data Processing**

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 file output stream class. But for character oriented data it is preferred to use FileWriter than FileOutputStream.

```
public class FileOutputStream extends OutputStream
```

Official Documentation

Output ← write

pixels
1920 x 1080

```
public class FileOutputStream extends OutputStream
```

Official Documentation

```
public class FileOutputStream extends OutputStream
```

A file output stream is an output stream for writing data to a File or to a FileDescriptor. Whether or not a file is available or may be created depends upon the underlying platform. Some platforms, in particular, allow a file to be opened for writing by only one FileOutputStream (or other file-writing object) at a time. In such situations the constructors in this class will fail if the file involved is already open.

FileOutputStream is meant for writing streams of raw bytes such as image data. For writing streams of characters, consider using FileWriter.

API Note:

The close() method should be called to release resources used by this stream, either directly, or with the try-with-resources statement.

Implementation Requirements:

Subclasses are responsible for the cleanup of resources acquired by the subclass. Subclasses requiring that resource cleanup take place after a stream becomes unreachable should use Cleaner or some other mechanism.

Since:

1.0

From <https://docs.oracle.com/en/java/javase/24/docs/api/java.base/java/io/FileOutputStream.html>

Constructors

Constructor	Description
<code>FileOutputStream(File file)</code>	Creates a file output stream to write to the file represented by the specified <code>File</code> object.
<code>FileOutputStream(FileDescriptor fdObj)</code>	Creates a file output stream to write to the specified file descriptor, which represents an existing connection to an actual file in the file system.
<code>FileOutputStream(File file, boolean append)</code>	Creates a file output stream to write to the file represented by the specified <code>File</code> object.
<code>FileOutputStream(String name)</code>	Creates a file output stream to write to the file with the specified name.
<code>FileOutputStream(String name, boolean append)</code>	Creates a file output stream to write to the file with the specified name.

```
package FileHandling;  
import java.io.*;  
import java.util.*;
```

```
/**  
 * Write a description of class FileDemoOne here.  
 *  
 * @author (your name)  
 * @version (a version number or a date)  
 */  
public class FileDemoOne  
{  
    public static void main(String args[]) throws IOException {  
        //Absolute path  
        //String path = "D:/Students/Students/LishaNishat/Coding/JavaSem2/DataFiles/";
```

Output ← write
Input ← read

pixels
1920 x 1080
RGB
0 0 0 - Black
255 0 0 - RED

color codes
R G B
0-255 0-255 0-255

depends on O/S

```
//Relative Path
String path = "../JavaSem2/DataFiles/"; ←
String fileName;
Scanner sc = new Scanner(System.in);
System.out.print("\nEnter file name: ");
fileName = sc.nextLine();
File fobj = new File(path+fileName);
if(fobj.createNewFile() || fobj.exists()){
    System.out.print("\nFile created successfully");
    FileOutputStream fis = new FileOutputStream(fobj, true);
    fis.write(65);
    fis.close();
}
else{
    System.out.print("\nFile is not created.");
}
}
```

It does depend on your project directory structure.

inefficient.

void write (byte[] arr)
void write (int b)
void write (byte[] arr, int off, int len)

It is used to write len bytes from the byte array starting at offset off to the file output stream.

Creating file and writing data into file.

```
package FileHandling;
import java.io.*;
import java.util.*;

/**
 * Write a description of class FileDemoOne here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class FileDemoOne
{
    public static void main(String args[]) throws IOException {
        //Absolute path
        //String path = "D:/Students/Students/LishaNishat/Coding/JavaSem2/DataFiles/";
        //Relative Path
        String path = "../JavaSem2/DataFiles/";
        String fileName;
        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter file name: ");
        fileName = sc.nextLine();
        File fobj = new File(path+fileName);
        if(fobj.createNewFile() || fobj.exists()){
            System.out.print("\nFile created successfully");
            FileOutputStream fis = new FileOutputStream(fobj, true);
            DataInputStream dis = new DataInputStream(System.in);
            System.out.print("Enter your Sentence, end with 0: ");
            while(true){
                int x = dis.read();
                if(x == '0'){
                    break;
                }
                fis.write(x);
            }
        }
    }
}
```

```

        fis.close();
    }
    else{
        System.out.print("\nFile is not created.");
    }
}
}

```

```

package FileHandling;
import java.util.*;
import java.io.*;

```

```

/**
 * Write a description of class WritingStringInFile here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class WritingStringInFile
{
    public static void main(String args[]){
        String path = "../JavaSem2/DataFiles/";
        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter file name: ");
        String fileName = sc.nextLine();
        try{
            FileOutputStream fos = new FileOutputStream(path+fileName, true);
            System.out.print("\nEnter any sentence: ");
            String sent = sc.nextLine();
            byte b[] = sent.getBytes();
            fos.write(b);
            fos.close();
            System.out.print("\nWritten data success...");
        }
        catch(FileNotFoundException fnfe){
            System.err.print("\nFile is not found at the specified path.");
        }
        catch(IOException ioe){
            System.err.print("\nData cannot transfer to the file.");
        }
    }
}

```

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte oriented data (Streams of raw bytes) such as image data, audio, video etc. You can also read character stream data. But for reading streams of characters it is recommended to use **FileReader** class.

Java FileInputStream Class declaration

```

public class FileInputStream extends InputStream

```

```

import java.util.*;
import java.io.*;

```

```

/**
 * Write a description of class DataInput here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class DataInput
{
    public static void main(String[] args)
    {
        String path = "./DataFiles/";
        String fileName;
        Scanner sc=new Scanner(System.in);
        System.out.print("\f Enter file name to read:");
        fileName=sc.nextLine();
        try {
            FileInputStream fis= new FileInputStream(path+fileName);
            int i=0;
            int count =0;

            while((i=fis.read())!=-1)
            {
                char a=(char)i;
                if(a==' ' || a=='.' || a==',' || a=='\n')
                {
                    count++;
                }

                System.out.print((char) i);
            }
            System.out.print("\n\n Number of words in the file: "+count);
            fis.close();
        }
        catch(FileNotFoundException fnfe)
        {
            System.err.print("\n Specified file is not found: "+path);
        }
        catch(IOException ioe)
        {
            System.err.print("\n Read error!");
        }
    }
}

```

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description
int	<u>available()</u> ✓	Returns an estimate of the number of remaining bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
void	close()	Closes this file input stream and releases any system resources associated with the stream.
FileChannel	getChannel() }	Returns the unique FileChannel object associated with this file input stream.

void	close()	Closes this file input stream and releases any system resources associated with the stream.
FileChannel	getChannel()	Returns the unique FileChannel object associated with this file input stream.
final FileDescriptor	getFD()	Returns the FileDescriptor object that represents the connection to the actual file in the file system being used by this FileInputStream.
int	read()	Reads a byte of data from this input stream.
int	read(byte[] b)	Reads up to <u>b.length</u> bytes of data from this input stream into an array of bytes.
int	read(byte[] b, int off, int len)	Reads up to <u>len</u> bytes of data from this input stream into an array of bytes.
byte[]	readNBytes(int len)	Reads up to a specified number of bytes from the input stream.
long	skip(long n)	Skips over and discards n bytes of data from the input stream.

how many
starting position

```

package FileHandling;
import java.util.*;
import java.io.*;

/**
 * Write a description of class DataInputFromFile here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class DataInputFromFile
{
    public static void main(String[] args)
    {
        String path = "../JavaSem2/DataFiles/";

        String fileName;

        Scanner sc=new Scanner(System.in);
        System.out.print("\f Enter file name to read:");
        fileName=sc.nextLine();

        try {

            FileInputStream fis= new FileInputStream(path+fileName);
            int numOfBytes = fis.available();
            byte b[] = new byte[numOfBytes];
            int x = fis.read(b);
            System.out.print("\n");
            for(int i = 0; i < numOfBytes; i++){
                System.out.print((char)b[i]);
            }
            fis.close();

        }

        catch(FileNotFoundException fnfe)

        {

```

```
        System.err.print("\n Specified file is not found: "+path);  
    }  
    catch(IOException ioe)  
    {  
        System.err.print("\n Read error!");  
    }  
}
```