UNIT III – INPUT / OUTPUT

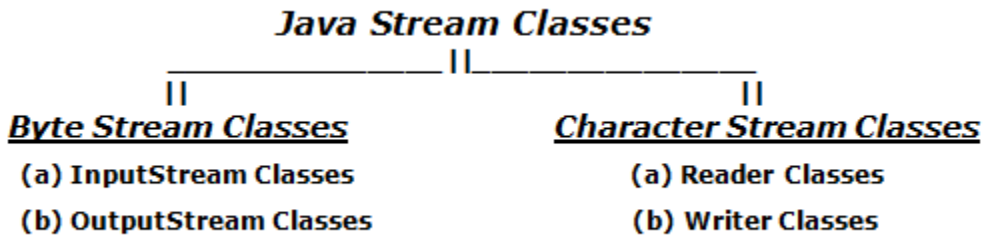
Streams classes: Byte – Character - File class - File operations - Console class – Serialization. Multithreading: Java thread model – Creating thread – Creating multi thread - Thread priorities – Synchronization - Inter thread communication.

STREAMS CLASSES

The Java Input/Output (I/O) is a part of **java.io** package. This package contains a relatively large number of classes that support input and output operations. These classes may be categorized into groups based on the data type on which they operate.

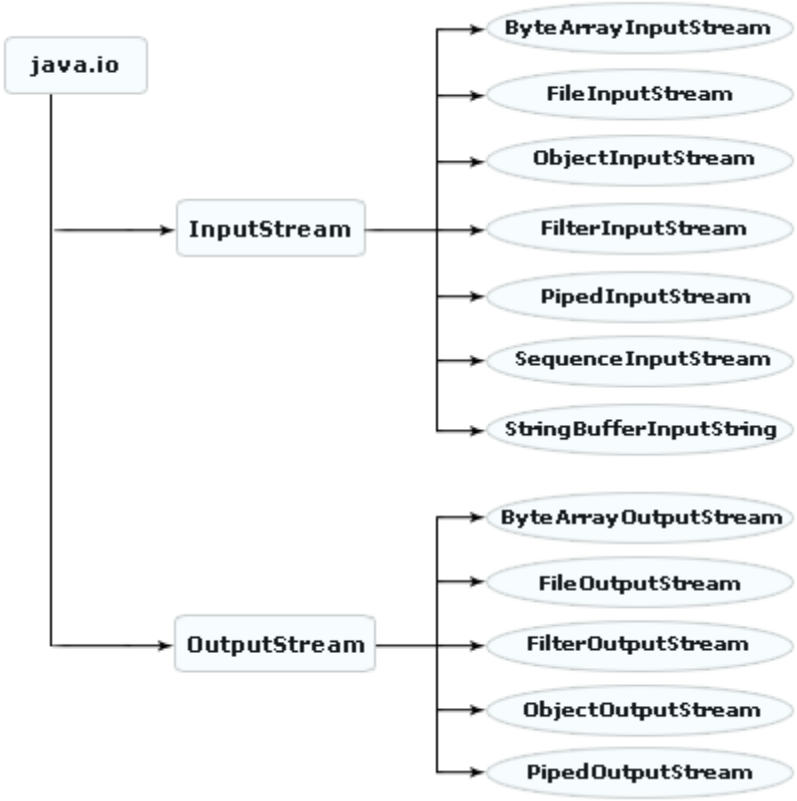
1. **Byte Stream Classes** that provide support for handling I/O operations on **bytes**.
2. **Character Stream Classes** that provide support for managing I/O operations on **characters**.

These two groups may further be classified based on their purpose. Figure 3 below shows how stream classes are grouped based on their functions. Byte stream and Character stream classes contain specialized input and output stream classes to deal with input and output operations independently on various types of streams.



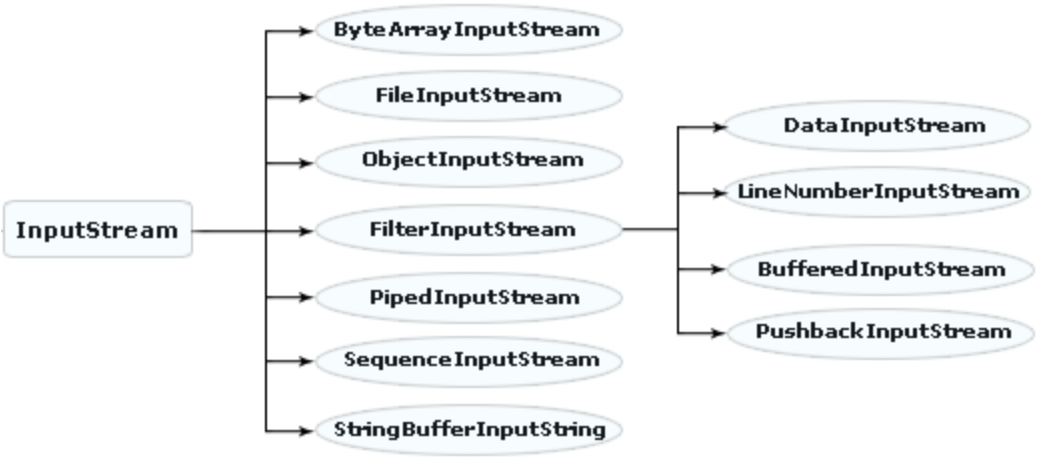
BYTE STREAM CLASSES

Byte streams provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary (8-bit) data. Byte streams are defined by using two class hierarchies. At the top there are two abstract classes: **java.io.InputStream** and **java.io.OutputStream**. Each of these abstract classes has several concrete subclasses of each of these. The abstract classes InputStream and OutputStream define several key methods that the other stream classes implement. Two of the most important are read() and write(), which, respectively, read and write bytes of data. Both methods are declared as abstract inside InputStream and OutputStream. They are overridden by derived stream classes. The java.io package can be categorized along with its stream classes in a hierarchy structure shown below:



InputStream Classes

The InputStream class is used for reading the data such as a byte and array of bytes from an input source. An input source can be a file, a string, or memory that may contain the data. It is an abstract class that defines the programming interface for all input streams that are inherited from it. An input stream is automatically opened when we create it. We cans explicitly close a stream with the close() method, or let it be closed implicitly when the object is found as a garbage. InputStream is inherited from the **java.lang.Object** class. Each subclass of the InputStream provided by the **java.io** package is intended for a different purpose. The subclasses inherited from the InputStream class can be seen in a hierarchy manner shown below:

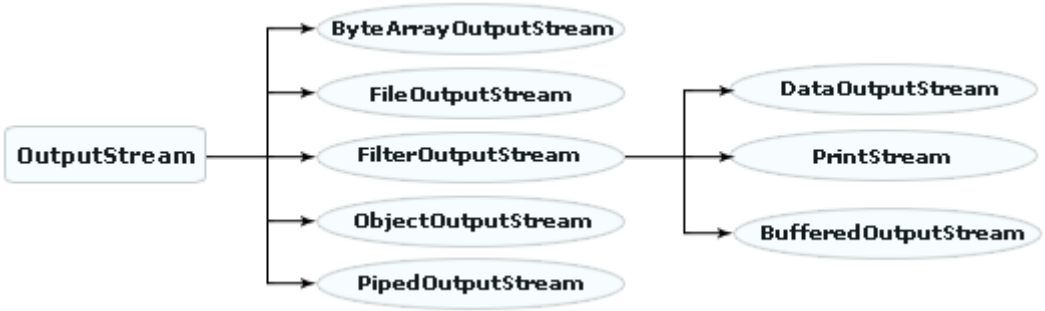


Summary of InputStream Methods	
int available()	Returns the number of bytes that can be read from the input stream
void close()	Closes this input stream and releases any system resources associated with the stream.
void mark(int readlimit)	Marks the current position in this input stream.
boolean markSupported()	Tests if this input stream supports the mark and reset methods.
abstract int read()	Reads the next byte of data from the input stream.
int read(byte[ ] array)	Reads some number of bytes from the input stream and stores them into the buffer array named array.
int read(byte[ ] array, int off, int len)	Reads up to len bytes of data from the input stream into an array of bytes.

void reset()	Repositions this stream to the position at the time the mark method was last called on this input stream.
long skip(long n)	Skips over and discards n bytes of data from the input stream.

OutputStream Classes

The OutputStream class is a sibling to InputStream that is used for writing bytes and array of bytes to an output source. Similar to input sources, an output source can be anything such as a file, a string, or memory containing the data. Like an input stream, an output stream is automatically opened when we create it. We can explicitly close an output stream with the close() method, or let it be closed implicitly when the object is garbage collected. OutputStream is also inherited from the **java.lang.Object** class. Each subclass of the OutputStream provided by the **java.io** package is intended for a different purpose. The classes inherited from the OutputStream class can be seen in a hierarchy structure shown below:



Summary of OutputStream Methods	
void close()	Closes this output stream and releases any system resources associated with this stream.
void flush()	Flushes this output stream and forces any buffered output bytes to be written out.
void write(byte[ ] array)	Writes array.length bytes from the specified byte array to this output stream.

void write(byte[ ] array, int off, int len)	Writes len bytes from the specified byte array starting at offset off to this output stream.
abstract void write(int array)	Writes the specified byte to this output stream.

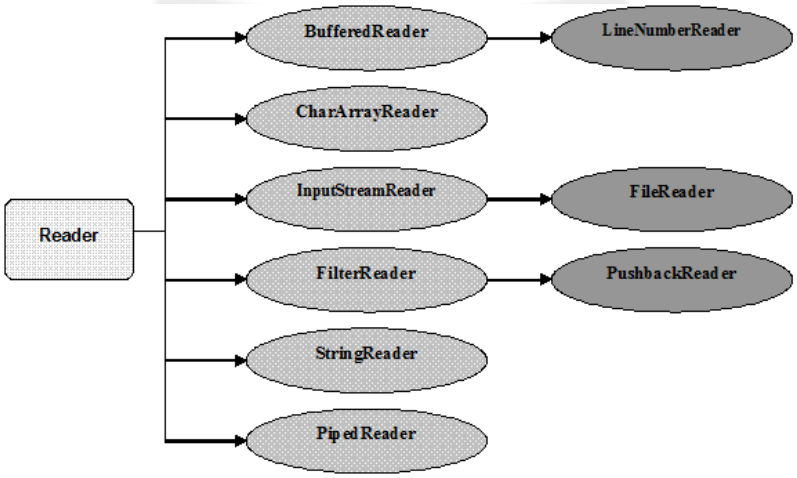
CHARACTER STREAM CLASSES

Java offers another type of streams called Character Streams, which are used to read from the input device and write to the output device in units of 16-bit (Unicode) characters. In some cases, character streams are more efficient than byte streams. The oldest version of Java (Java 1.0) did not include character streams and, thus, all I/O was byte-oriented. Character streams were added by Java 1.1, and certain byte-oriented classes and methods were deprecated.

Character streams are defined by using two class hierarchies. At the top there are two abstract classes, **Reader** and **Writer**. These abstract classes handle **Unicode character** (16-bit) streams. Java has several concrete subclasses of each of these. The abstract classes Reader and Writer define several key methods that the other stream classes implement. Two of the most important methods are read() and write(), which read and write characters of data, respectively. These methods are overridden by derived stream classes.

Reader Stream Classes

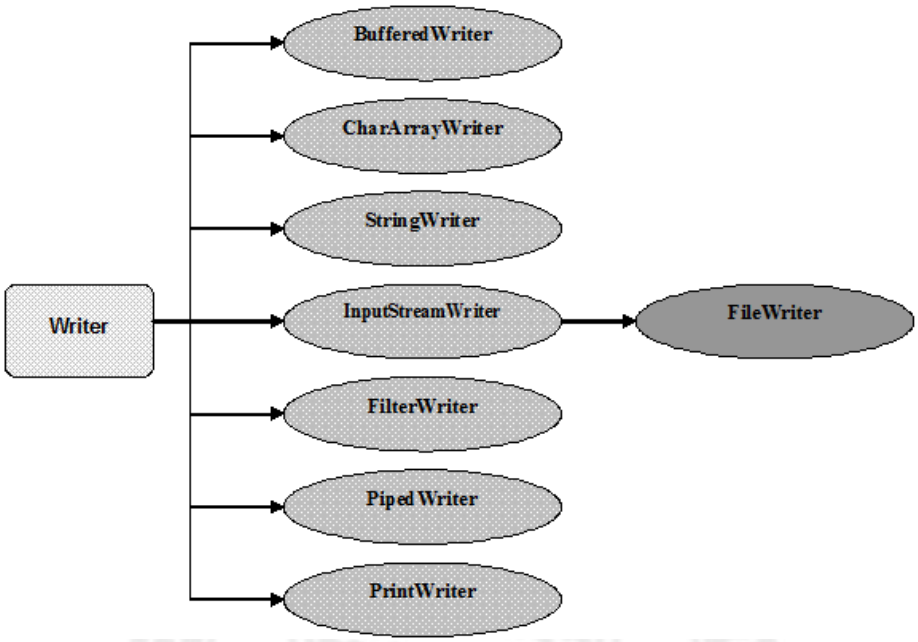
The **Reader** class contains methods that are identical to those available in the InputStream class, except Reader is designed to handle characters. Therefore, Reader classes can perform all the functions implemented by the InputStream classes. The hierarchy of Reader character stream classes is shown below:



Writer Stream Classes

The **Writer** class contains methods that are identical to those available in the OutputStream class, except Writer is designed to handle characters. Therefore, Writer

classes can perform all the functions implemented by the OutputStream classes. The hierarchy of Writer character stream classes is shown below:



FILE CLASSES

Java File class is a representation of a file or directory pathname. Because file and directory names have different formats on different platforms, a simple string is not adequate to name them. Java File class contains several methods for working with the pathname, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.

Features:

It is an abstract representation of files and directory pathnames.

- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the getParent() method of this class.
- First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.
- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

How to Create a File Object?

A File object is created by passing in a string that represents the name of a file, a String, or another File object. For example,

File a = new File("Welcome.txt");

Example: Program to check if a file or directory physically exists or not

```
import java.io.File;
class CheckFileExist
{
	public static void main(String[] args)
	{
		String fname = args[0];
		File f = new File(fname);
		System.out.println("File name :" + f.getName());
		System.out.println("Path: " + f.getPath());
		System.out.println("Absolute path:" + f.getAbsolutePath());
		System.out.println("Parent:" + f.getParent());
		System.out.println("Exists :" + f.exists());
		if (f.exists())
		{
			System.out.println("Is writable:" + f.canWrite());
			System.out.println("Is readable" + f.canRead());
			System.out.println("Is a directory:" + f.isDirectory());
			System.out.println("File Size in bytes " + f.length());
		}
	}
}
```

Example: Program to display all the contents of a directory.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
class AllDir
{
	public static void main(String[] args)throws IOException
	{
		BufferedReader br = new BufferedReader(new InputStreamReader
(System.in));
		System.out.print("Enter directory path : ");
		String dirpath = br.readLine();
```

```
            System.out.print("Enter the directory name : ");
            String dname = br.readLine();
            File f = new File(dirpath, dname);
            if (f.exists())
            {
                  String arr[] = f.list();
                  int n = arr.length;
                  for (int i = 0; i < n; i++)
                  {
                        System.out.print(arr[i] + " ");
                        File f1 = new File(f,arr[i]);
                        if (f1.isFile())
                              System.out.println(": is a file");
                        if (f1.isDirectory())
                              System.out.println(": is a directory");
                  }
                  System.out.println("\nNo of entries in this directory : " + n);
            }
            else
                  System.out.println("Directory not found");
      }
}
```

Fields in File Class

Field	Type	Description
pathSeperator	String	the character or string used to separate individual paths in a list of file system paths.
pathSeperatorChar	Char	the character used to separate individual paths in a list of file system paths.
separator	String	default name separator character represented as a string.

separatorChar	Char	default name separator character.

Constructors of Java File Class

- File(File parent, String child): Creates a new File instance from a parent abstract pathname and a child pathname string.
- File(String pathname): Creates a new File instance by converting the given pathname string into an abstract pathname.
- File(String parent, String child): Creates a new File instance from a parent pathname string and a child pathname string.
- File(URI uri): Creates a new File instance by converting the given file: URI into an abstract pathname.

Methods of File Class in Java

Method	Description	Return Type
canExecute()	Tests whether the application can execute the file denoted by this abstract pathname.	boolean
canRead()	Tests whether the application can read the file denoted by this abstract pathname.	boolean
canWrite()	Tests whether the application can modify the file denoted by this abstract pathname.	boolean
compareTo(File pathname)	Compares two abstract pathnames lexicographically.	int
createNewFile()	Atomically creates a new, empty file named by this abstract pathname.	boolean
createTempFile(String prefix, String suffix)	Creates an empty file in the default temporary-file directory.	File
delete()	Deletes the file or directory denoted by this abstract pathname.	boolean

equals(Object obj)	Tests this abstract pathname for equality with the given object.	boolean
exists()	Tests whether the file or directory denoted by this abstract pathname exists.	boolean
getAbsolutePath()	Returns the absolute pathname string of this abstract pathname.	String
list()	Returns an array of strings naming the files and directories in the directory.	String[]
getFreeSpace()	Returns the number of unallocated bytes in the partition.	long
getName()	Returns the name of the file or directory denoted by this abstract pathname.	String
getParent()	Returns the pathname string of this abstract pathname’s parent.	String
getParentFile()	Returns the abstract pathname of this abstract pathname’s parent.	File
getPath()	Converts this abstract pathname into a pathname string.	String
setReadOnly()	Marks the file or directory named so that only read operations are allowed.	boolean
isDirectory()	Tests whether the file denoted by this pathname is a directory.	boolean
isFile()	Tests whether the file denoted by this abstract pathname is a normal file.	boolean

isHidden()	Tests whether the file named by this abstract pathname is a hidden file.	boolean
length()	Returns the length of the file denoted by this abstract pathname.	long
listFiles()	Returns an array of abstract pathnames denoting the files in the directory.	File[]
mkdir()	Creates the directory named by this abstract pathname.	boolean
renameTo(File dest)	Renames the file denoted by this abstract pathname.	boolean
setExecutable(boolean executable)	A convenience method to set the owner's execute permission.	boolean
setReadable(boolean readable)	A convenience method to set the owner's read permission.	boolean
setReadable(boolean readable, boolean ownerOnly)	Sets the owner's or everybody's read permission.	boolean
setWritable(boolean writable)	A convenience method to set the owner's write permission.	boolean
toString()	Returns the pathname string of this abstract pathname.	String
toURI()	Constructs a file URI that represents this abstract pathname.	URI

ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY
PALKULAM, KANYAKUMARI
OBSERVE OPTIMIZE OUTSPREAD