

# FILE HANDLING

In **Java**, with the help of File Class, we can work with files. This File Class is inside the **java.io** package. The File class can be used by creating an object of the class and then specifying the name of the file.

## ❖ Why File Handling is Required?

**File Handling** is an integral part of any programming language as file handling 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.

```
// Importing File Class
import java.io.File;

class TW {


    public static void main(String[] args)
    {

        // File name specified
        File obj = new File("myfile.txt");
        System.out.println("File Created!");

    }

}
```

Output:



File Created!

## Streams in Java

- 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 :

### 1.Input Stream:

The Java `InputStream` class is the superclass of all input streams. The input stream is used to read data from numerous input devices like the keyboard, network, etc. `InputStream` is an abstract class, and because of this, it is not useful by itself. However, its subclasses are used to read data.

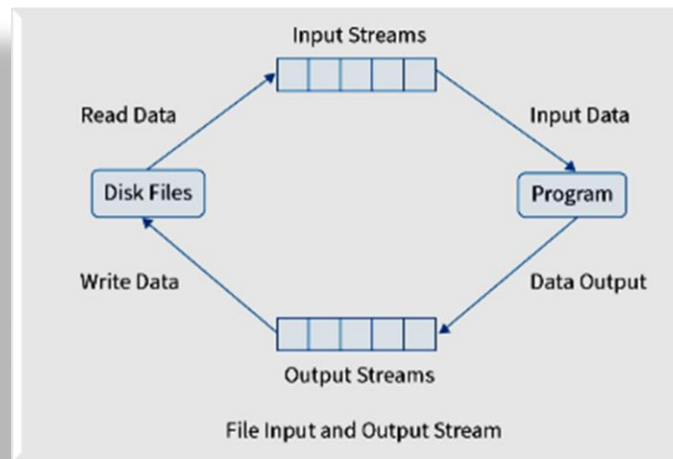
There are several subclasses of the `InputStream` class, which are as follows:

- `AudioInputStream`
- `ByteArrayInputStream`
- `FileInputStream`
- `FilterInputStream`
- `StringBufferInputStream`
- `ObjectInputStream`

### Creating an InputStream

```
// Creating an InputStream  
InputStream obj = new FileInputStream();
```

Here, an input stream is created using **`FileInputStream`**.



## 2. Output Stream:

The output stream is used to write data to numerous output devices like the monitor, file, etc. OutputStream is an abstract superclass that represents an output stream. OutputStream is an abstract class and because of this, it is not useful by itself. However, its subclasses are used to write data.

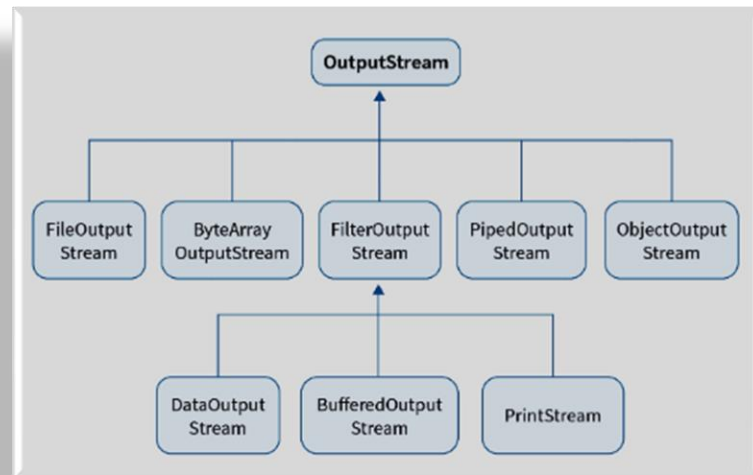
There are several subclasses of the OutputStream class which are as follows:

- **ByteArrayOutputStream**
- **FileOutputStream**
- **StringBufferOutputStream**
- **ObjectOutputStream**
- **DataOutputStream**
- **PrintStream**

### Creating an OutputStream

```
// Creating an OutputStream  
OutputStream obj = new  
FileOutputStream();
```

Here, an output stream is created using FileOutputStream.



Based on the **data type**, there are **two** types of streams :

### 1. **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:

- Byte Input Stream: Used to read byte data from different devices.
- Byte Output Stream: Used to write byte data to different devices.

### 2. **Character Stream:**

This stream is used to read or write character data. Character stream is again subdivided into 2 types which are as follows:

- Character Input Stream: Used to read character data from different devices.
- Character Output Stream: Used to write character data to different devices.

## Java File Class Methods

The following table depicts several File Class methods:

Method Name	Description	Return Type
<b>canRead()</b>	It tests whether the file is readable or not.	Boolean
<b>canWrite()</b>	It tests whether the file is writable or not.	Boolean
<b>createNewFile()</b>	It creates an empty file.	Boolean
<b>delete()</b>	It deletes a file.	Boolean
<b>exists()</b>	It tests whether the file exists or not.	Boolean
<b>length()</b>	Returns the size of the file in bytes.	Long
<b>getName()</b>	Returns the name of the file.	String
<b>list()</b>	Returns an array of the files in the directory.	String[]
<b>mkdir()</b>	Creates a new directory.	Boolean
<b>getAbsolutePath()</b>	Returns the absolute pathname of the file.	String

## File operations in Java

The following are the several operations that can be performed on a file in Java :

- Create a File
- Read from a File
- Write to a File
- Delete a File



Now let us study each of the above operations in detail.

## 1. Create a File

In order to create a file in Java, you can use the **createNewFile()** method.

If the file is successfully created, it will return a Boolean value true and false if the file already exists.

Following is a demonstration of how to create a file in Java :

```
import java.io.File;

// Import the IOException class to handle errors
import java.io.IOException;

public class GFG {
    public static void main(String[] args)
    {
        try {
            File Obj = new File("myfile.txt");
            if (Obj.createNewFile()) {
                System.out.println("File created: "
                                   + Obj.getName());
            }
            else {
                System.out.println("File already exists.");
            }
        }
        catch (IOException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Output:

```
An error has occurred.
```

## 2. Read from a File

We will use the Scanner class in order to **read()** contents from a file.

Following is a demonstration of how to read contents from a file in Java :

```
// Import the File class
import java.io.File;

// Import this class for handling errors
import java.io.FileNotFoundException;

// Import the Scanner class to read content from text files
import java.util.Scanner;

public class GFG {
    public static void main(String[] args)
    {
        try {
            File Obj = new File("myfile.txt");
            Scanner Reader = new Scanner(Obj);
            while (Reader.hasNextLine()) {
                String data = Reader.nextLine();
                System.out.println(data);
            }
            Reader.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Output:

An error has occurred.

### 3. Write to a File:

We use the `FileWriter` class along with its **`write()`** method in order to write some text to the file.

Following is a demonstration of how to write text to a file in Java :

```
// Import the FileWriter class
import java.io.FileWriter;

// Import the IOException class for handling errors
import java.io.IOException;

public class GFG {
    public static void main(String[] args)
    {
        try {
            FileWriter Writer
                = new FileWriter("myfile.txt");
            Writer.write(
                "Files in Java are seriously good!!");
            Writer.close();
            System.out.println("Successfully written.");
        }
        catch (IOException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Output:

An error has occurred.

#### 4.Delete a File:

We use the **delete()** method in order to delete a file.

Following is a demonstration of how to delete a file in Java :

```
// Import the File class
import java.io.File;

public class GFG {
    public static void main(String[] args)
    {
        File Obj = new File("myfile.txt");
        if (Obj.delete()) {
            System.out.println("The deleted file is : "
                               + Obj.getName());
        }
        else {
            System.out.println(
                "Failed in deleting the file.");
        }
    }
}
```



**Output:**

```
Failed in deleting the file.
```