FILE HANDLING IN JAVA

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 to create an object of the class and then specifying the name of the file.

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.

Example: Importing File Class

```
import java.io.File;


class Geeks
{
   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. Below are the 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:

InputStream obj = new FileInputStream();

Common Methods of InputStream:

Method	Description
read()	Reads one byte of data from the input stream.
read(byte[] array)()	Reads byte from the stream and stores that byte in the specified array.
mark()	It marks the position in the input stream until the data has been read.
available()	Returns the number of bytes available in the input stream.

Method	Description
markSupported()	It checks if the mark() method and the reset() method is supported in the stream.
reset()	Returns the control to the point where the mark was set inside the stream.
skips()	Skips and removes a particular number of bytes from the input stream.
close()	Closes the input stream.

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:

OutputStream obj = new FileOutputStream();

Common Methods of OutputStream:

Method	Description
write()	Writes the specified byte to the output stream.
write(byte[] array)	Writes the bytes which are inside a specific array to the output stream.
close()	Closes the output stream.
flush()	Forces to write all the data present in an output stream to the destination.

Java File Class Methods

The following table depicts several File Class methods:

Method Name	Description	Return Type
canRead()	It tests whether the file is readable or not.	Boolean
canWrite()	It tests whether the file is writable or not.	Boolean
createNewFile()	It creates an empty file.	Boolean
delete()	It deletes a file.	Boolean
exists()	It tests whether the file exists or not.	Boolean
length()	Returns the size of the file in bytes.	Long

Method Name	Description	Return Type
getName()	Returns the name of the file.	String
list()	Returns an array of the files in the directory.	String[]
mkdir()	Creates a new directory.	Boolean
getAbsolutePath()	Returns the absolute pathname of the file.	String

File Operations

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

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.

Example: Creating File using Java Program

```
// Import the File class

import java.io.File;

import java.io.IOException;


public class CreateFile

{

   public static void main(String[] args)

   {
```

```
        // Creating the File also
        // Handling Exception
       try {
          File Obj = new File("myfile.txt");

           // Creating File
           if (Obj.createNewFile()) {
             System.out.println("File created: " + Obj.getName());
          }
          else {
             System.out.println("File already exists.");
          }
       }

        // Exception Thrown
       catch (IOException e) {
          System.out.println("An error has occurred.");
          e.printStackTrace();
       }
    }
}
```

2. Write to a File

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

Example: Writing Files using Java Program

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

```
public class WriteFile
{
    public static void main(String[] args)
    {
        // Writing Text File also
        // Exception Handling
        try {

            FileWriter Writer = new FileWriter("myfile.txt");

            // Writing File
            Writer.write("Files in Java are seriously good!!");
            Writer.close();

            System.out.println("Successfully written.");
        }

        // Exception Thrown
        catch (IOException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

3. Read from a File

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

Example: Reading File using Java Program

```
// Import the File class
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadFile
{
   public static void main(String[] args)
   {
      // Reading File also
      // Handling Exception
      try {
         File Obj = new File("myfile.txt");
         Scanner Reader = new Scanner(Obj);

         // Traversing File Data
          while (Reader.hasNextLine()) {
            String data = Reader.nextLine();
            System.out.println(data);
         }

         Reader.close();
      }

      // Exception Cases
      catch (FileNotFoundException e) {
```

```
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

4. Delete a File

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

Example: Deleting File using Java Program

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