File Handling: File handling in Java implies reading from and writing data to a file. The File class from the java.io package, allows us to work with different formats of files. In order to use the File class, you need to create an object of the class and specify the filename or directory name.

Create a new file:In this case, to create a file you can use the createNewFile() method. This method returns true if the file was successfully created, and false if the file already exists.

For Example:

```
import java.io.File;
import java.io.IOException;
class Object {
    public static void main(String[] args)
    {
        File f1=new File("color1.txt");
        try{
            f1.createNewFile();
        }
        catch(IOException e)
        {
            System.out.println("unable to create file");
           System.out.println(e);
        }
    }
}
```

Write in File: I have used the FileWriter class together with its write() method to write some text into the file. Let's understand this with the help of a code.

For Example:

```
import java.io.FileWriter;
import java.io.IOException;
class Object {
    public static void main(String[] args)
    {

        try{
            FileWriter w=new FileWriter("color.java");
           w.write("This is my first file using File handling\nOkay good bye");
```

```
            w.close();

             }
         catch(IOException e)
         {
             System.out.println("unable to create file");
            System.out.println(e);
         }
    }
}
```

Delete a file: Java provides methods to delete files using java programs. On the contrary to normal delete operations in any operating system, files being deleted using the java program are deleted permanently without being moved to the trash/recycle bin.Deletes the file or directory denoted by this abstract pathname.

Syntax:

public boolean delete();

For Example:

```
import java.io.File;
class Object {
    public static void main(String[] args)
    {
             File file=new File("color1.java");
             if(file.delete())
             {
                 System.out.println("I have delete this file"+file.getName());
             }
             else
             {
                 System.out.println("mi file is not delete");
        }
    }
}
```

Read a file: A simple text scanner that can parse primitive types and strings using regular expressions. A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods**.**

For Example:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
class read
{
    public static void main(String[] args) throws IOException
    {
       int i=0;
FileInputStream fis=new FileInputStream("D://java file -jr//hello.txt");
        i=fis.read();
        while(!(i==-1))
        {
            char c=(char)i;
            System.out.print(c);
        i=fis.read();
        }
            fis.close();
    }
}
```

For Example: (Part two)

```
import java.util.*;
import java.io.File;
class Object {

    public static void main(String[] args)
    {
            File file=new File("color.txt");
            try{
                Scanner sc=new Scanner(file);
                while(sc.hasNextLine())
```

```
                {
                    String line=sc.nextLine();
                    System.out.println(line);
                }
                sc.close();
            }
                catch(Exception e)
                {
                    System.out.println(e);
                }
        }
}
```

Program: Without try block using use read file handling

For Example:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Object
{
    public static void main(String[] args)  throws FileNotFoundException
    {
            File file=new File("color.txt");
            Scanner sc=new Scanner(file);
            sc.useDelimiter("\\Z");

    System.out.println(sc.next());
    }
}
```

Get File Information: The operation is performed to get the file information. We use several methods to get the information about the file like name, absolute path, is readable, is writable and length.

For Example:

```
import java.io.File;
class Fileinfo {
    public static void main(String[] args) {
```

```
        // Creating file object
        File f0 = new File("color2.txt");
        if (f0.exists()) {
            // Getting file name
            System.out.println("The name of the file is: " + f0.getName());

            // Getting path of the file
            System.out.println("The absolute path of the file is: " +
f0.getAbsolutePath());

            // Checking whether the file is writable or not
            System.out.println("Is file writeable?: " + f0.canWrite());

            // Checking whether the file is readable or not
            System.out.println("Is file readable " + f0.canRead());

            // Getting the length of the file in bytes
            System.out.println("The size of the file in bytes is: " +
f0.length());
        } else {
            System.out.println("The file does not exist.");
        }
    }
}
```

Use Delimeter: Java Scanner class provides the useDelimiter() method to split the string into tokens.

For Example:

```
import java.util.*;
import java.io.IOException;
import java.io.File;
class Objects {
    public static void main(String[] args) throws IOException
    {
        File file=new File("color2.txt");
            Scanner sc=new Scanner(file);
            sc.useDelimiter(" ");
           while(sc.hasNext()){
```

```
    System.out.println(sc.next());
         }}
}
```

Create a Directory:In Java, the mkdir() function is used to create a new directory. This method takes the abstract pathname as a parameter and is defined in the Java File class.

For Example:

```
import java.io.File;
import java.io.IOException;
class car {
    public static void main(String[] args)
    {
        File f=new File("E://rani");

            if (f.mkdir() == true) {
                System.out.println("Directory has been created successfully");
            }
            else {
                System.out.println("Directory cannot be created");
            }
        System.out.println("File is create");
    }
}
```

Delete a Directory: The delete() method of the File class deletes the files and empty directory represented by the current File object. If a directory is not empty or contain files then that cannot be deleted directly. First, empty the directory, then delete the folder.

For Example:

```
import java.io.File;
class tel
{
     public static void main(String[] args)
    {
        File file=new File("D:\\jyoti\\rani");
        if(file.delete()==true)
        {
```

```
            System.out.println("Directory is deleted");
        }
        else{
            System.out.println("Folder is not deleted");
        }


        System.out.println("Program is running");
    }
}
```

Delete a non-empty directory: In Java, to delete a non-empty directory, we must first delete all the files present in the directory. Then, we can delete the directory.

For Example:

```
import java.io.File;
class gel
{
     public static void main(String[] args)
    {
        try{
            File dir=new File("D:\\jyoti\\rani\\hut");
            File[] files=dir.listFiles();
            for(File file:files)
            {
                System.out.println(file + " delete ");
                file.delete();
            }
            if(dir.delete())
            {
                System.out.println("Directory is deleted");
            }
            else{
                System.out.println("Directory is not deleted");
            }


        }
```

```
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Listing Directories: You can use list( ) method provided by File object to list down all the files and directories available in a directory.

For Example:

```
import java.io.File;
import java.io.IOException;
class bike {
    public static void main(String[] args)
    {
        File f=null;
      String[] path;
      try{
          f=new File("D://jyoti//rani");
          path=f.list();
          for(String p:path)
          {
              System.out.println(p);
          }
      }
      catch(Exception e)

      {
System.out.println(e);
      }

      System.out.println("----------------------------");

        System.out.println("Program is running stage");
    }
}
```

Print File Name: The java.lang.Class.getName() returns the name of the entity (class, interface, array class, primitive type, or void) represented by this Class object, as a String.

Syntax

java.lang.Class.getName() method

public String getName()

For Example:

```
import java.lang.*;
class fit
{
    public static void main(String[] args)
    {
        fit f=new fit();
        Class c=f.getClass();
        String name=c.getName();
        System.out.println("Class Name=" +name);
    }
}
```

Creating Directories: There are two useful File utility methods, which can be used to create directories – The mkdirs() method creates both a directory and all the parents of the directory.

For Example:

```
import java.io.File;
class fund
{
    public static void main(String[] args)
    {
        String dir="D://jyoti//rani//home//hut//gun//root";
        File d=new File(dir);
      d.mkdirs();

        System.out.println("this is running program");
    }
}
```

For Example: Note(In which one file create and write and read method is used and print the output).

```
import java.io.*;
//import java.io.FileReader;

import java.io.IOException;
class tile{
public static void main(String[] args) throws IOException
{
    File f=new File("D://jyoti//well.txt");
    f.createNewFile();

    FileWriter wt=new FileWriter(f);
    wt.write("this is my first file\n this is well file\n okay good luck\n
bye\n java");
    wt.flush();
    wt.close();

    FileReader rd=new FileReader(f);
    char[] a=new char[88];
    rd.read(a);
    for(char c:a)
System.out.print(c);
rd.close();
}
}
```

Byte Oriented Stream: ByteStream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that ByteStream classes read/write the data of 8-bits. We can store video, audio, characters, etc., by using ByteStream classes. These classes are part of the java.io package.

InputStream Class: The InputStream class provides methods to read bytes from a file, console or memory. It is an abstract class and can't be instantiated; however, various classes inherit the InputStream class and override its methods.

Available():The available() method of FileInputStream class is used to return the estimated number of remaining bytes that can be read from the input stream without blocking. This method returns the number of bytes remaining to read from the file.

For Example:

```
import java.io.*;


class Main {
  public static void main(String args[]) {

    byte[] array = new byte[100];

    try {
      InputStream input = new FileInputStream("D://jyoti//hut.txt");

      System.out.println("Available bytes in the file: " + input.available());
      input.read();
      input.read();
      System.out.println("Data read from the file: ");
      System.out.println("----------------: ");
      System.out.println("Available bytes in the file: " + input.available());

    } catch (Exception e) {
      e.getStackTrace();
    }
  }
}
```

For Example: Note(Part:two)

```
import java.io.*;

class Main {
  public static void main(String args[]) {

    byte[] array = new byte[100];
```

```
    try {
      InputStream input = new FileInputStream("D://jyoti//hut.txt");

      System.out.println("Available bytes in the file: " + input.available());

      // Read byte from the input stream
      input.read(array);
      System.out.println("Data read from the file: ");

      // Convert byte array into string
      String data = new String(array);
      System.out.println(data);

      // Close the input stream
      input.close();
    } catch (Exception e) {
      e.getStackTrace();
    }
  }
}
```

For Example:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
class read
{
    public static void main(String[] args) throws IOException
    {
        FileInputStream fis=null;
       int i=0;
     fis=new FileInputStream("D://java file -jr//hello.txt");
        i=fis.read();
        while(!(i==-1))
        {
            char c=(char)i;
```

```
            System.out.print(c);
        i=fis.read();
        }
            fis.close();
    }
}
```

OutputStream Class: The OutputStream is an abstract class that is used to write 8-bit bytes to the stream. It is the superclass of all the output stream classes.

For Example:

```
import java.io.File;
import java.io.FileOutputStream;
 class fit
{
    public static void main(String[] args)
    {
        File f1=new File("D://jyoti//kolor.txt");

     FileOutputStream fileOut=null;
     try{

if(!f1.exists())
{
        f1.createNewFile();
}
   fileOut=new  FileOutputStream(f1);
    String t1="Hello,World";
    byte[] tt=t1.getBytes();
 fileOut.write(tt);
    fileOut.close();
}
  catch(Exception e)
  {
System.out.print("file not exists");
  }
}
```

```
}
```

For Example: Note(In which program use the integer value and print the character)

```
import java.io.*;
class out
{
    public static void main(String[] args) throws IOException
    {
FileOutputStream os=new FileOutputStream("D://jyoti//hut.txt");
        byte b[] ={65,66,67,68,69,70};
os.write(b);
os.flush();
for(int i=71; i<75;i++)
{
    os.write(i);

}
os.flush();
os.close();
System.out.println("------------------");
System.out.println("the program is running");
    }
}
```

For Example:Note(In example which makes use of these two classes to copy an input file into an output file)

```
import java.io.*;
import java.io.IOException; class Copy {

   public static void main(String args[]) throws IOException
    {
      FileInputStream in = null;
      FileOutputStream out = null;

      try {
         in = new FileInputStream("D:\\jyoti\\rani\\cello.txt");
         out = new FileOutputStream("D:\\jyoti\\rani\\hut.txt");
```

```
         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null)
         {
            in.close();
            System.out.println("program is running");
         }
         if (out != null)
         {
            out.close();
          System.out.println("program is runnning part-2");
         }
      }
   }
}
```

Character Streams: Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, FileReader and FileWriter. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

For Example:

```
import java.io.*;
import java.io.IOException;
 class toCopy {

   public static void main(String args[]) throws IOException
    {
      FileReader in = null;
      FileWriter out = null;

      try {
         in = new FileReader("D:\\jyoti\\rani\\cello.txt");
```

```
         out = new FileWriter("D:\\jyoti\\rani\\hello.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null)
         {
            in.close();
            System.out.println("program is running");
         }
         if (out != null)
         {
            out.close();
        System.out.println("program is runnning part-2");
         }
      }
   }
}
```

Standard Streams: All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen. If you are aware of C or C++ programming languages, then you must be aware of three standard devices STDIN, STDOUT and STDERR. Similarly, Java provides the following three standard streams –

Standard Input – This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as System.in.

For Example: Note(In which program rad the one by one character and stop the selected character in this program).

```
import java.io.*;
import java.io.IOException;
import java.io.FileInputStream;
 class send {

   public static void main(String args[]) throws IOException {
```

```
      InputStreamReader cin = null;

      try {
         cin = new InputStreamReader(System.in);
         System.out.print("Enter characters, 's' to quit.");
         char c;
         do {
            c = (char) cin.read();
            System.out.print(c);
         } while(c != 's');
      }finally {
         if (cin != null) {
            cin.close();
         }
      }
   }
}
```

Standard Output – This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as System.out.

For Example:

```
import java.io.*;
 class Test {

   public static void main(String args[])
    {

      try {
         byte bWrite [] = {11,21,3,40,65,69,78,89,41};
         OutputStream os = new FileOutputStream("D:\\jyoti\\rani\\cello.txt");
         for(int x = 0; x < bWrite.length ; x++) {
            os.write( bWrite[x] );   // writes the bytes
         }
         os.close();

         InputStream is = new FileInputStream("D:\\jyoti\\rani\\cello.txt");
```

```
         int size = is.available();

         System.out.println("this file is character length is: "+size);

         for(int i = 0; i < size; i++)
         {

            System.out.print((char)is.read() + "  ");
         }
         is.close();
      }
       catch (IOException e)
        {
         System.out.print("Exception");
         }
   }
}
```

Standard Error (System.err): This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as System.err. System.err is a PrintStream. System.err works like System.out except it is normally only used to output error texts. Some programs (like Eclipse) will show the output to System.err in red text, to make it more obvious that it is error text.

For Example:

```
import java.io.*;
import java.io.IOException;
class demo
{
    public static void main(String [] args)
    {

        try{
    InputStream input=new FileInputStream("D:\\Java_programs\\hello.txt");

            System.err.println("file opnen");
```

```
        }
        catch(IOException e)
        {
            System.err.println("File opening failed");
            e.printStackTrace();
        }
        System.out.println("hello world");
    }
}
```