File Operations in Java

Description

- File handling in java enables to read data from and write data to files along with other file manipulation tasks.
- File operations are present in "java.io" package streams.
- A stream symbolizes series of data and holds different kind of processes to perform computations upon those data.
- Streams can maintain variety of data formats such as bytes, primitive data types, characters, and objects.
- Streams may just pass on data or work on them and convert into useful ways.
- In general java program utilizes an inputstream to read source data, one item at a time and an outputstream to write data to a target place, one item at time.

Java streams are categorized into two major options:

- Byte Oriented – permits input and output of 8-bit bytes
- Character Oriented – permits input and output for 16-bit Unicode characters.

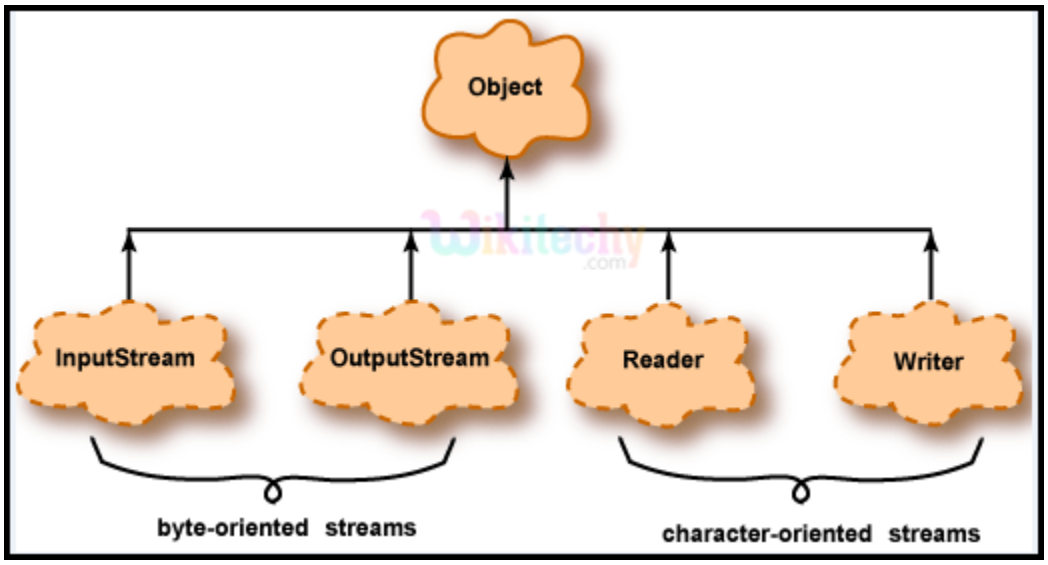


Fig1. Java io streams

Character Oriented Stream:

- Implements Unicode characters for data transfer.
- Routinely accepts the limited / wide set of characters and hence preferable for globalization.
- FileReader and FileWriter are frequently used classes for file operations.

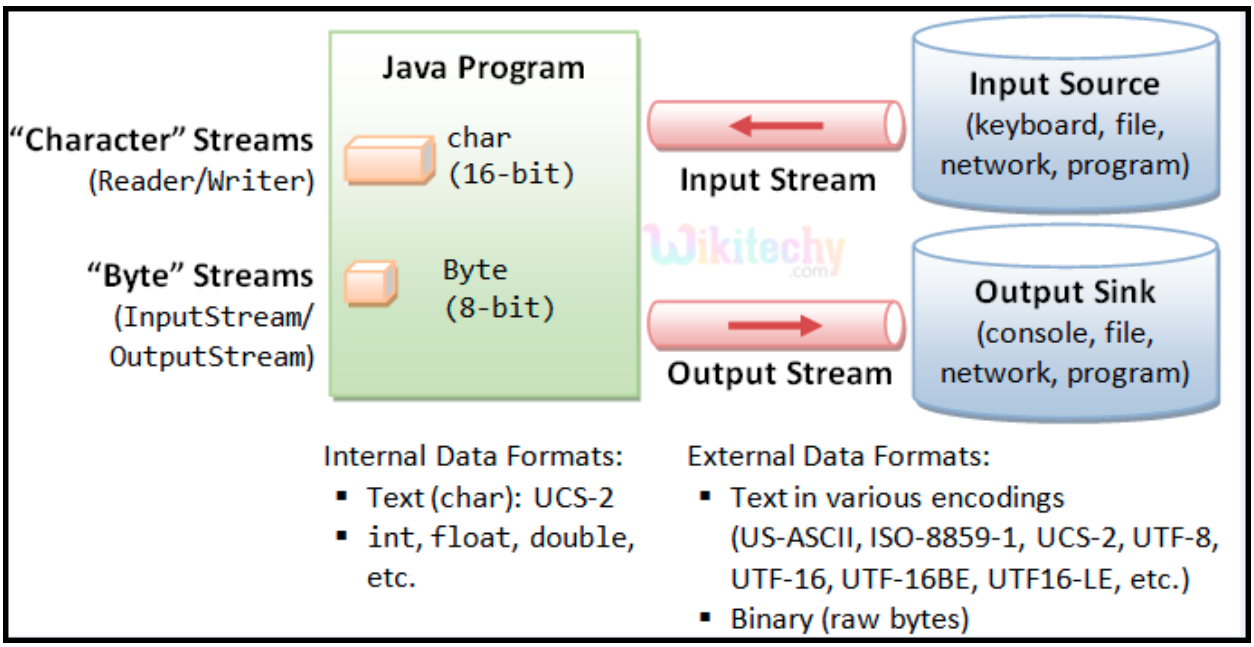


Fig2. Java Stream Types

Byte Oriented Stream:

- All byte stream classes are descended from InputStream and OutputStream.
- They work on with 8-bit bytes of input / output operations.
- Byte stream is considered as a low-level I/O processing and hence not preferable often.
- FileInputStream and FileOutputStream are frequently used classes for file operations.

Sample Code:

```
import java.io.*;
public class JavaFileOperations      {
   public static void main(String[] args) throws IOException        {
      System.out.println("\n\nWikiTechy - Java File Operations\n");
      FileInputStream file1 = null; FileOutputStream file2 = null;
      FileReader rdr = null; FileWriter wrtr = null;
      try {
         file1 = new FileInputStream("myfile1.txt");
         file2 = new FileOutputStream("myfile2.txt");
         int c;
         while ((c = file1.read()) != -1) {
            file2.write(c);
              }
              System.out.println("\nByteStream File Operations
              completed\n");                    }
              catch(Exception ex)      {       System.out.println(ex);          }
              finally {
                       if (file1 != null) {          file1.close();          }
                      if (file2 != null) {           file2.close();          }
              }
```



```
        try {
                rdr = new FileReader("myfile3.txt");
                wrtr = new FileWriter("myfile4.txt");
                int c;
                while ((c = rdr.read())!= -1) {
                wrtr.write(c);
                }
                System.out.println("\n\nCharacterStream File Operations
                completed\n");              }
                catch(Exception ex)       {
                        System.out.println(ex);
                }
                finally {
        if (rdr != null) {
                rdr.close();
        }
        if (wrtr != null) {
                wrtr.close();
        }
          }
    }
}
```


Code Explanation:

```
import java.io.*;
public class JavaFileOperations      {
    public static void main(String[] args) throws IOException    {
        System.out.println("\n\nWikiTechy - Java File Operations\n");
        FileInputStream file1 = null; FileOutputStream file2 = null;
        FileReader rdr = null; FileWriter wrtr = null;
        try {
            file1 = new FileInputStream("myfile1.txt");
            file2 = new FileOutputStream("myfile2.txt");
            int c;
            while ((c = file1.read()) != -1) {
                file2.write(c);
            }
            System.out.println("\nByteStream File Operations completed\n");
        }
        catch(Exception ex) {
            System.out.println(ex);
        }
        finally {
            if (file1 != null) {
                file1.close();
            }
            if (file2 != null) {
                file2.close();
            }
        }
```

1. Declare objects for class FileInputStream and FileOutputStream classes with the code

   FileInputStream file1 = null; FileOutputStream file2 = null;

2. Declare objects for class FileReader and FileWriter classes as

   FileReader rdr = null; FileWriter wrtr = null;

3. Initialize the FileInputStream object to point to the file "myfile1.txt" and FileOutputStream to "myfile2.txt".

   file1 = new FileInputStream("myfile1.txt");
   file2 = new FileOutputStream("myfile2.txt");

4 Read contents from file1 till the end and write them into file2.

```
while ((c = file1.read()) != -1) {
    file2.write(c);
    }
```

5 Finally close the inputstream objects. This will destroy the pointers to the files.

```
finally {
        if (file1 != null) {    file1.close();    }
        if (file2 != null) {    file2.close();    }
        }
```

```
        try {
            rdr = new FileReader("myfile3.txt");
            wrtr = new FileWriter("myfile4.txt");
            int c;
            while ((c = rdr.read()) != -1) {
            wrtr.write(c);
         }
          System.out.println("\n\nCharacterStream File Operations completed\n");
        }
        catch(Exception ex) {
            System.out.println(ex);
        }
        finally {
            if (rdr != null) {
            rdr.close();
            }
            if (wrtr != null) {
            wrtr.close();
            }
        }
    }
}
```

6 Initialize the FileReader object to point to the file "myfile3.txt" and FileWriter to "myfile4.txt".

```
rdr = new FileReader("myfile3.txt");
wrtr = new FileWriter("myfile4.txt");
```

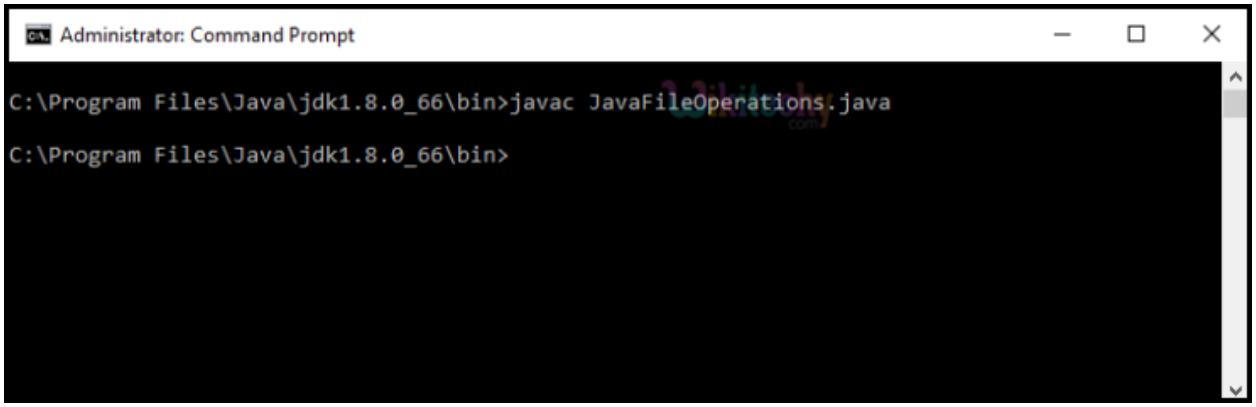
7 Read contents from file3 till the end and write them into file4.

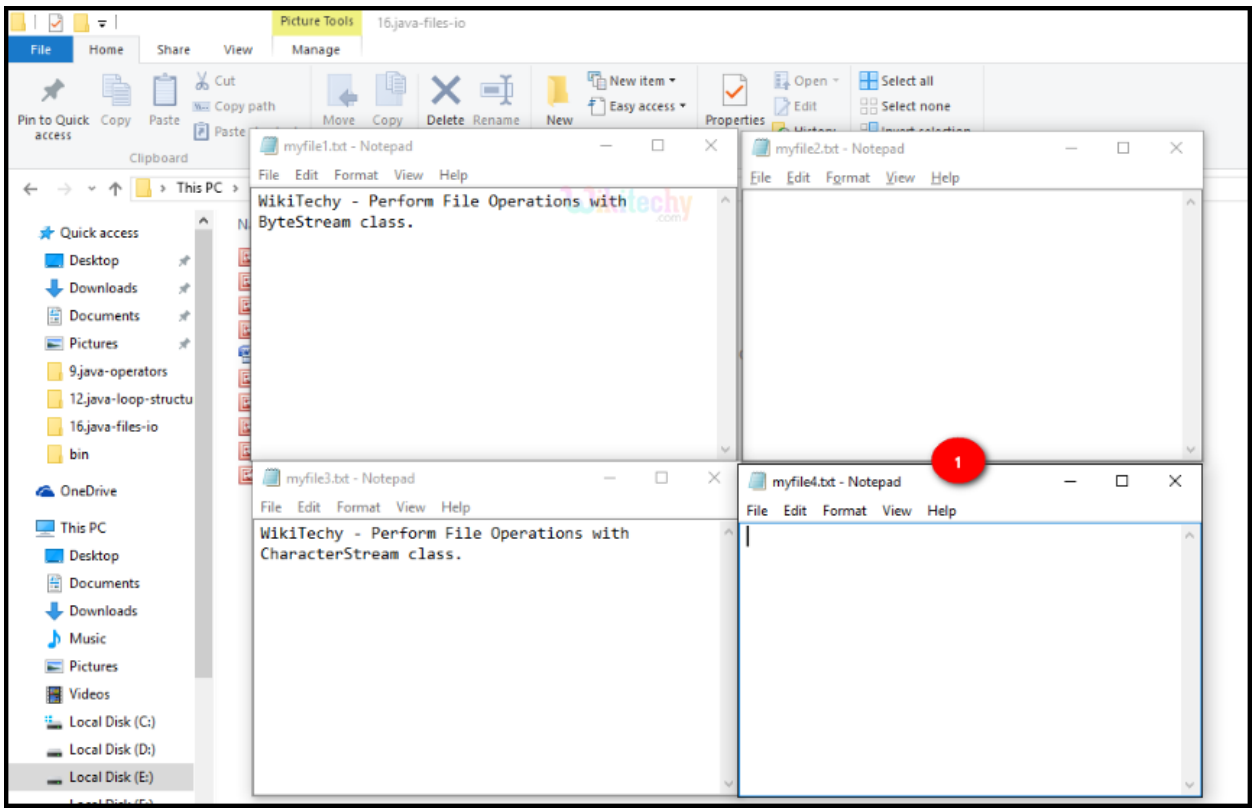
```
while ((c = rdr.read())!= -1)
{    wrtr.write(c);    }
```


8 Finally close the reader / writer objects. This will destroy the pointers to the files.

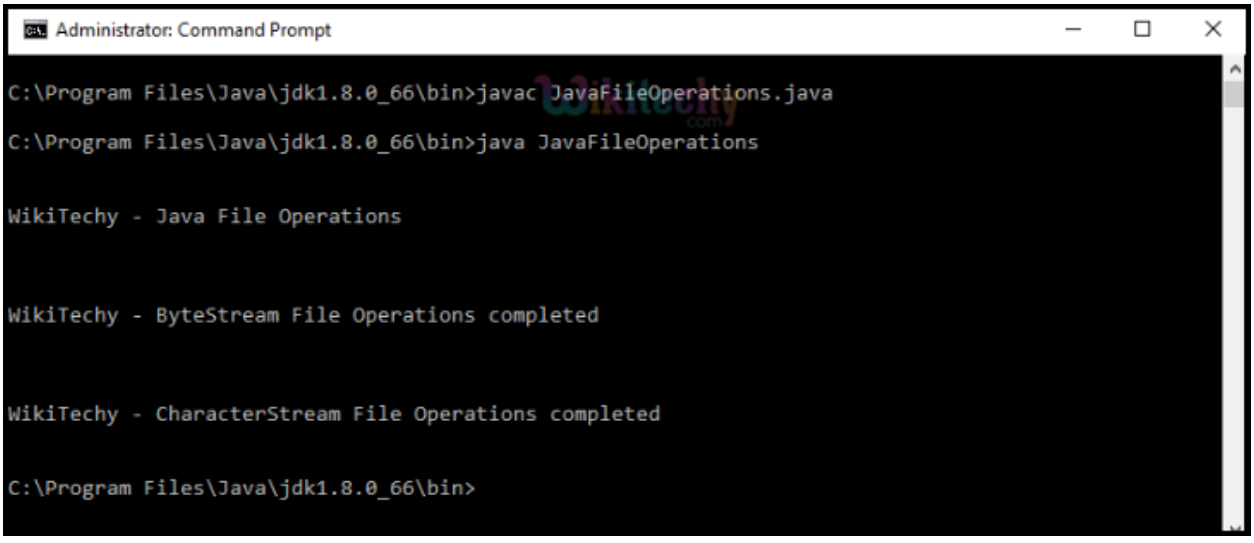
```
if (rdr != null) {
        rdr.close();
}
if (wrtr != null) {
        wrtr.close();
}
```

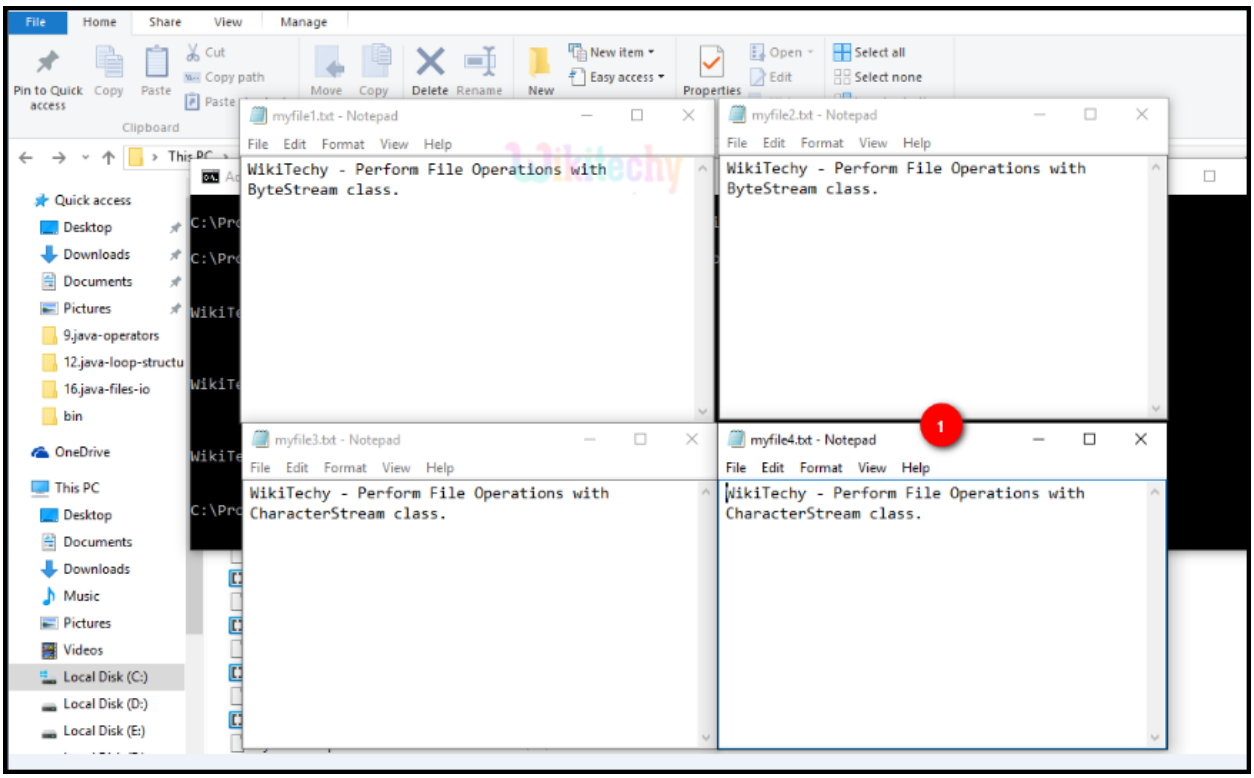
Output:





1 Files myfile2.txt and myfile4.txt are empty before executing file operations.





1 File content copied from source to destination after completing the file execution processes.

