FILES

# 

[STREAM:](#_sqnuhdr4ydxm)

[BYTE STREAM:](#_nedeuxitr3f)

[CHARACTER STREAM:](#_qia31nnmteqx)

[FILE CREATION:](#_43er70cw2fih)

[READING AND WRITING FILES:](#_wr19hublmx9)

[FILE INPUT STREAM](#_x5jbcdfp70p4)

[FILE OUTPUT STREAM:](#_ui98eig9vf4j)

[BUFFERED INPUT STREAM:](#_7kq44tufscpo)

[BUFFERED OUTPUT STREAM:](#_xj6bxfqu8y8)

[PROGRAM 1:](#_m7uysr3c70oy)

[SOLVED PROGRAM 1:](#_oqb8c5f9edrv)

[PROGRAM 2:](#_jbr3qgihdsbj)

[SOLVED PROGRAM 2:](#_273s3iipugab)

[PROGRAM 3:](#_i7oggze53ylg)

[SOLVED PROGRAM 3:](#_l2cbker3ljgu)

# STREAM:

A stream can be defined as a sequence of data.

* InPutStream − The InputStream is used to read data from a source.
* OutPutStream − The OutputStream is used for writing data to a destination.

## BYTE STREAM:

Java byte streams are used to perform input and output of 8-bit bytes.

FileInputStream and FileOutputStream comes under byte stream since they read and write byte by byte.

## CHARACTER STREAM:

Character streams are used to perform input and output for 16-bit unicode.(i.e) they read and write character by character.

FileReader and FileWriter comes under character stream

# FILE CREATION:

we create a file object using File() method as follows

File f = new File("C:/java/hello");

# READING AND WRITING FILES:

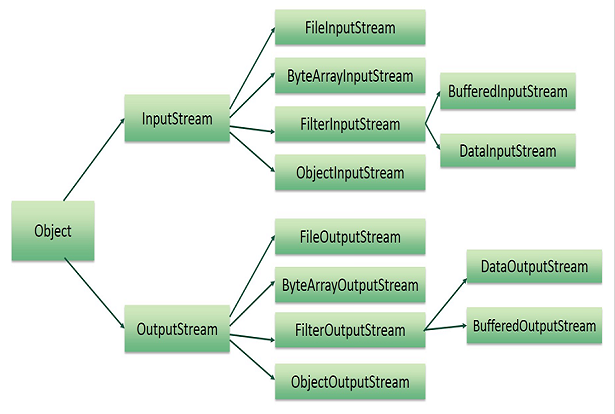
The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

## FILE INPUT STREAM

Just use new FileInputStream to create FileInputStream object and use this object to read the file.

With the help of this FileInputStream object any helper methods inside FileInputStream can be used.

File f = new File("C:/java/hello");  
InputStream f = new FileInputStream(f);



## FILE OUTPUT STREAM:

Similarly ,Just use new FileOutputStream to create FileOutputStream object and use this object to write into a file.

With the help of this FileOutputStream object any helper methods inside FileOutputStream can be used.

## BUFFERED INPUT STREAM:

This is normally used in cases where a file needs to accessed line by line.

## BUFFERED OUTPUT STREAM:

Similarly used in cases where files are written line by line.

While using buffered input and output stream objects object.flush() is done.

# PROGRAM 1:

Create 2 files, test1.txt and test2.txt , write content into them separately, copy content of both files in a new file newtest.txt and delete the files test1.txt and test2.txt.

## SOLVED PROGRAM 1:

<https://github.com/sruthiviswanathan/Files/blob/master/src/com/zilker/file/BasicFile.java>

* Create a two files.
* Use BufferedWriter or any other writer objects for writing input to the files.
* Create a new file to copy contents.
* Now read the contents from first file and write it to the new file.
* Again read contents from second file and write it to third file.
* Now delete first two files.

# PROGRAM 2:

Read contents of a file test.txt and write it in another file sample.txt with the condition: Alternate characters capitalized.

Do exception handling and create a custom reader/writer to implement the scenario.

## SOLVED PROGRAM 2:

<https://github.com/sruthiviswanathan/Files/blob/master/src/com/zilker/file/CapitalizeCharacters.java>

* Create a new file and write contents into it.
* Have a separate class that extends BufferedReader and override the method readLine().
* Apply the business logic of implementing the capitalizing alternate characters.
* Write the output to new file.

# PROGRAM 3:

Read in an xml file , parse it and convert to JSON

## SOLVED PROGRAM 3:

<https://github.com/sruthiviswanathan/Files/blob/master/src/com/zilker/file/XmlToJson.java>

* Create a buffered reader and read the xml file.
* Store the contents of file line by line into a String variable.
* Pass this string variable to an inbuilt function provided by java.
  + JSONObject obj = XML.toJSONObject(result);
* To use the above inbuilt function you will need a jar file that is responsible for converting the xml file to json.
* Find the jar file in the below link:

<https://jar-download.com/artifacts/org.json>