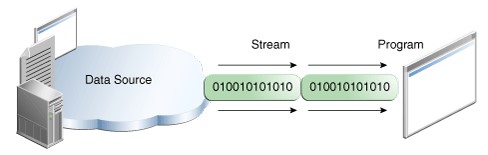
### Java .io package

* Java.io package contains classes to perform input and output operations.
* By using java.io package we are preforming file handling java. It possible to work with only text files but not word & excel files.
* To work with excel files & word file required extra jar file like **poi.jar jxl.jar** files.

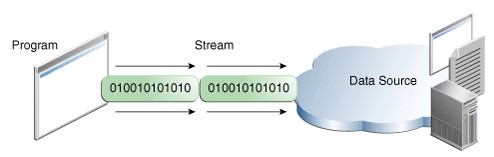
**I/O Streams:-**

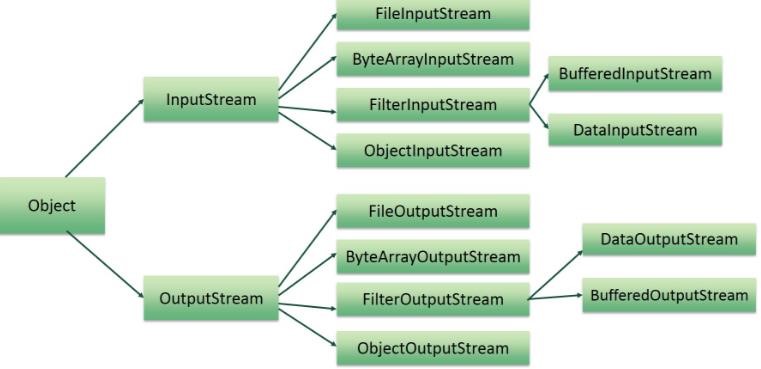
* + Byte Streams handle I/O of raw binary data.
  + Character Streams handle I/O of character data, automatically handling translation to and from the local character set.
  + Buffered Streams optimize input and output by reducing the number of calls to the native API.

**Input stream:-** Program uses Input stream to read the data from a source one item at a time.



**Output stream:-**Program uses output stream to write the data to a destination one item at a time.





**Example :- creation of physical File** import java.io.\*;

class Test

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

{ File f = new File("anu.txt"); boolean b = f.createNewFile();

if (b)

{ System.out.println("File is created successfully");

}

else

{ System.out.println("File is already existed in location");

}

}

}

**Example : creation of directory** import java.io.\*;

class Test

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

{ **//creation of File**

File f = new File("anu.txt");

System.out.println(f.exists()); f.createNewFile();

System.out.println(f.exists());

**//creation of directory**

File f1 = new File("durga"); System.out.println(f1.exists());

f1.mkdir();

System.out.println(f1.exists());

**//creation of file inside the directory (directory must present)**

File f2 = new File("durga","durga.txt"); f2.createNewFile();

}

}

**There are two types of streams:-**

1. Byte oriented channel
2. Character oriented channel

#### Byte streams:-

* Program uses byte stream to perform input & output of byte data. All byte stream classes developed based on InputStream & OutputStream.
* In byte channel the data is transferred in the from of bytes.
* Generally to transfer the images use byte streams.
* Program uses byte stream to perform input and output of 8-bit format.

To demonstrate how the byte stream works file I/O provided two main classes

* FileInputStream
  + It is used to read the data from source one item at a time.
  + To read the data from source use read() method of FileInputStream class. **public int read() throws java.io.IOException;**

read() method returns first character Unicode value in the form of integer value.

* FileOutputStream
  + It is used to write the data to destination one item at a time.
  + To write the data to destination use write() method of FileOutputStream class. **public void write(int unicode) throws java.io.IOException;**

write() method is taking Unicode value of the character as a parameter.

#### Character streams:-

* Program uses character stream to perform input & output of character data. All character stream classes developed based on Reader & Writer classes.
* In character channel the data is transferred in the form of characters.
* Generally to transfer text file data use character channel.
* Here the data is transferred in 16 bit format.

To work with character channel required two java classes

FileReader o It is used to read the data from source one item at a time.

* + To read the data from source use read() method of FileInputStream class. **public int read() throws java.io.IOException;**

read() method returns first character Unicode value in the form of integer value.

FileWriter o It is used to write the data to destination one item at a time.

* + To write the data to destination use write() method of FileOutputStream class. **public void write(int unicode) throws java.io.IOException;** write() method is taking Unicode value of the character as a parameter.

**Steps to design application:-** Step1:- create the channel.

Step 2:- read the data & store the data in temporary variable.

Step 3:- check the stream is ended or data (data flow completed or not).

Step 4:- write the data to destination file.

Step 5:- close the streams.

**Example :-** import java.io.\*; class Test

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

{ **//Byte channel creation**

FileInputStream fis = new FileInputStream("abc.txt"); FileOutputStream fos = new FileOutputStream("xyz.txt");

int c;

while((c=fis.read())!=-1) { System.out.print((char)c); fos.write(c);

}

System.out.println("read() & write operatoins are completed");

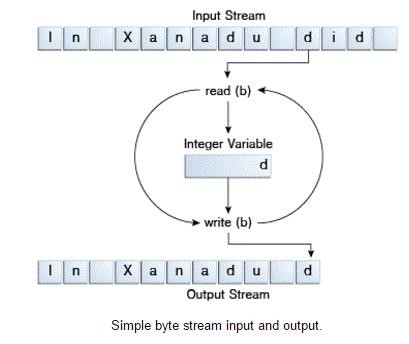
**//stream closing operations**  fis.close();

fos.close();

}

}

* While working with streams we will get two exceptions mainly FileNotFoundException , IOException & these two exceptions are checked exceptions so must handle these exception by using try-catch blocks or throws keyword.
* The above example is not recommended because when the exception raised in the middle there may be chance of resources are not released.
* To overcome above problem use try-catch-finally blocks to release the resources.



**Steps to design the application :-** 1. Declare the resources.

* 1. Open the try block do the operations
  2. Catch block handle the exception 4. Finally block release the resources.

**Example :- application with try-catch-finally** import java.io.\*;

class Test

{ public static void main(String[] args)

{ FileInputStream fis=null;

FileOutputStream fos=null;

try{

fis = new FileInputStream("abc.txt");

fos = new FileOutputStream("xyz.txt");

int c;

while((c=fis.read())!=-1) { System.out.println((char)c);

fos.write(c);

}

}

catch(IOException e){ e.printStackTrace(); }

finally{

try{ if(fis!=null)fis.close();

if(fos!=null)fos.close();

}

catch(IOException io){io.printStackTrace();}

}

}

}

**Example** : **try-with resources**

when we declare the resource by using try block once the try block is completed resources are released.

import java.io.\*;

class Test

{ public static void main(String[] args)

{ try(FileInputStream fis = new FileInputStream("abc.txt");

FileOutputStream fos = new FileOutputStream("xyz.txt");)

{ int c;

while((c=fis.read())!=-1) { System.out.println((char)c);

fos.write(c);

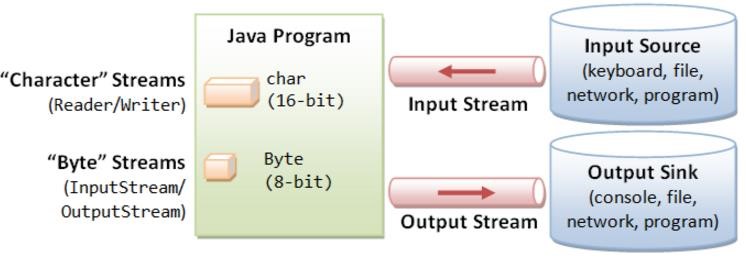
}

}

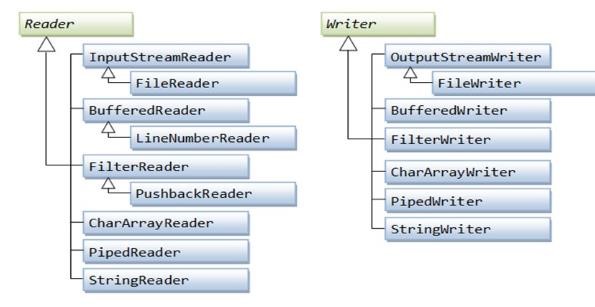
catch(IOException e){ e.printStackTrace();}

}

}



Note : In CopyCharacters, the int variable holds a character value in its last 16 bits; in CopyBytes, the int variable holds a byte value in its last 8 bits.



**Example :** import java.io.\*;

class Test

{ public static void main(String[] args)

{ FileReader fr=null; FileWriter fw=null;

try

{ fr=new FileReader("abc.txt");

fw=new FileWriter("balu.txt");

int c;

while ((c=fr.read())!=-1)

{ fw.write(c);

}

}

catch (IOException ie){ ie.printStackTrace(); }

finally

{ try{ if(fr!=null) fr.close(); if(fw!=null) fw.close(); }

catch(IOException e) { e.printStackTrace(); }

}

}

}

**CharArrayWriter:-**It is used to write the data to multiple files & this implements Appendable interface.

import java.io.\*; class Test

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

{ CharArrayWriter ch = new CharArrayWriter(); FileReader fr = new FileReader("abc.txt");

int a;

while((a=fr.read())!=-1)

{ ch.write(a);

}

FileWriter fw1 = new FileWriter("a.txt");

FileWriter fw2 = new FileWriter("b.txt");

ch.writeTo(fw1); ch.writeTo(fw2); fw1.close(); fw2.close();

fr.close();

System.out.println("operations are completed");

}

}

**Normal streams vs. Buffered Streams:-**

* Normal streams

FileInputStream

FileOutputStream

FileReader

FileWriter

* **Normal stream object creation** new FileInputStream("abc.txt"); new FileOutputStream("abc.txt"); new FileReader("abc.txt");

new FileWriter("abc.txt");

* In previous examples we are using un-buffered I/O (normal stream) .This means each read and write request is handled directly by the underlying OS.

* By using normal streams it is possible to read the data character by character ,In normal streams each request directly triggers disk access it is relatively expensive & performance is degraded.

* By reading the data character by character performance decreased.

**To overcome above limitations use buffered streams.**

* Buffered Streams BufferedInputStream

BufferedOutputStream

BufferedReader

BufferedWriter

* Buffered Stream Object creation new BufferedInputStream(new FileInputStream("abc.txt")); new BufferedOutputStream(new FileOutputStream("abc.txt")); new BufferedReader(new FileReader("abc.txt")); new BufferedWriter(new FileWriter("abc.txt"));

* The buffered streams are developed based on normal streams.

* Buffered input stream read the data from buffered memory and it interacting with hard disk only when buffered memory is empty.

* By using buffered streams it is possible to read the data line by line format.

* When we use buffered streams, performance increased.

**Example** :-  **To work with image data.** import java.io.\*;

class Test

{ public static void main(String[] args)

{ BufferedInputStream bis=null; BufferedOutputStream bos=null;

try{

bis=new BufferedInputStream(new FileInputStream("desert.jpg"));

bos=new BufferedOutputStream(new FileOutputStream("balu.jpg"));

int str;

while ((str=bis.read())!=-1)

{ bos.write(str);

}

}

catch(IOException e) { System.out.println(e);

System.out.println("getting Exception");

}

finally

{ try{ if (bis != null) bis.close();

if (bos != null) bos.close();

}

catch(IOException e) {e.printStackTrace();}

}

}

}

**Example:-** Buffered Character streams, to work with character data. import java.io.\*;

class Test

{ public static void main(String[] args)

{ try(BufferedReader br=new BufferedReader(new FileReader("abc.txt"));

BufferedWriter bw=new BufferedWriter(new FileWriter("balu.txt"));)

{

String str;

while ((str=br.readLine())!=null) { System.out.println(str); bw.write(str);

}

}

catch(Exception e)

{ System.out.println("getting Exception");

}

}

}

#### File IO

1. What is the purpose of java.io package?
2. What do you mean by stream?
3. What do you mean by channel and how many types of channels present in java?
4. What is the difference between normal stream & buffered Streams?
5. What is the difference between FileInputStream & BufferedReader?
6. What is the difference between FileOutputStream & printwriter?
7. Println() method present in which class?
8. Out is which type of variable(instance /static ) present in which class?
9. To create byte oriented channel we required two class what are those classes?
10. To create character oriented channel we required two class what are those classes?
11. What is the difference between byte oriented channel and character oriented channel?
12. What is the difference between read() & readLine() method?
13. What is the difference between normal Streams & bufferd streams?
14. Wat is the purpose of write() & println() ?
15. Example classes normal Streams & bufferd streams?
16. What do you mean by serialization?
17. What is the purpose of Serializable interface& it is marker interface or not ?
18. How to prevent serialization concept?
19. What do you mean deserialization?
20. To perform deserialization we required two classes what are those classes?
21. To perform serialization we required two classes what are those classes?
22. What is the purpose of transient modifier?
23. What are advantage of serialization?
24. Serializable interface present in which package?
25. When we will get IOException how many ways are there to handle the exceptions?
26. IOException is checked Exception or unchecked Exception?

**\*\*\*\*\*\*\*\* Thank you \*\*\*\*\*\*\*\***