UNIT-V

16-03-19

MULTITHREADED PROGRAMMING

Multi-tasking:-

when doing more than one task is called multi tasking. The multi-tasking is achieved based on the following two ways

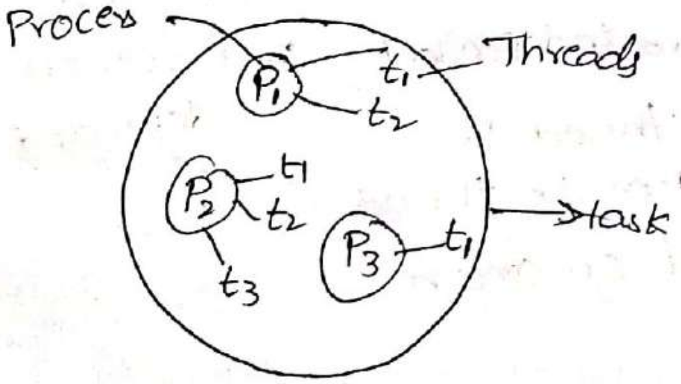
1) Thread based Multi tasking (Multi-threading)
2) Process based Multi Processing (Multi processing)

Multi-threading:-

A thread is a light weight Entity. It is a Part of a process. A Process can be devide multiple Parts. Each Part is called thread. Running more than one thread is called Multi-threading

Multiprocessing-

A Process is a heavy weight entity. A Program under Execution is called Processing. if you are using more than one Process complete a task then this is called Multi Processing.



Multi Programming:-

Executing more than one Program is called Multi Programming. A Program is a Multiple statements.

Differences between Multi-Processing & Multi-threading.

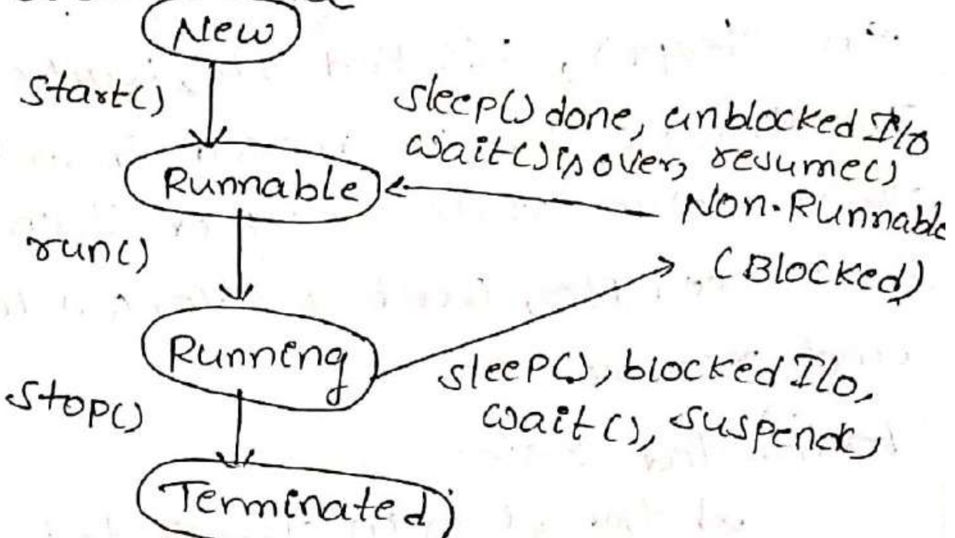
Multi-threading	Multiprocessing.
1. A thread is a Part of Process	1. A Process is defined as Program under Execution
2. It is a light weight Entity.	2. It is a heavy weight Entity.
3. Thread shares memory area	3. Processor doesn't share memory
4. Threads has individual addresses	4. Processor has only one address.
5. A cost of communication is very low	5. A cost of communication is high
6. A context switching between threads is fast	6. A context switching between Processors is high.
7. Threads doesn't block the user	7. Processor can block the user.
8. Threads are independent because if a thread is interrupted then the thread will not effect on another thread.	8. Processors are depended

Creating Threads:

In Java we can create the threads by using the Thread class and Runnable interface we can use Thread and Runnable interface as follows.

1. we can create a thread by ~~using~~ Extending Thread class.
2. we can create a thread by implements Runnable interface.

Life Cycle of a thread:-



A thread has the following 5 state in its life cycle.

1) New State
2) Runnable state
3) Running State
4) Terminated State
5) Non-Runnable state.

New state:-

The thread is in new state whenever the new thread is created

Thread t = new Thread();

Runnable State:

A thread is in Runnable state whenever the newly created thread started. The thread is started using start method. Ex: start()

Running state

After starting a thread a thread is in running state. The thread will be run by using run method

Ex; run();

Non-Runnable States:-

When thread is in Sleep mode, when a thread is blocked by I/O, waiting for a Loc, Suspended then the thread is in non-runnab and block mode.

Ex:- Sleep(), blocked I/O, wait(), suspend()

The thread can be moved from Non-runnable to runnable state whenever Sleep is done & un blocked I/O, wait is over(or) lock is aquired and resume.

Terminated state:-

A thread is in terminated state. when Ever the thread is stop (or) destroy.

19-03-19

We can create a new thread by using the above two methods. The Thread class have the following constructor..

1) Thread();

* 2) Thread(String str)

* 3) Thread (Runnable . Obj, String str)!

A Thread class have the following Methods

Method Name	Description.
1. setName()	It is used to set a name to the newly created thread.
2. getName()	To get a thread name.
3. get Priority ().	To get a Thread Priority.
4. setPriority()	To set Priority to the thre
5) isAlive ( )	used to check a thread is alive are not. the thread is alive it

will written true otherwise false.

6) join ( )	waiting for a thread to be finish
7) *run ( )	It is an entry point for a thread.
8) *Start ( )	Start a new thread by calling run method.
9) *sleep ( )	Suspend a thread for a period of time, the time will be mentioned interms of milliseconds.
10) stop ( )	To terminate the thread.

~~Creation of a thread using a runnable interface~~

Main Thread:-

Every Java Program has a thread it is running while the Programm is Executing. when Ever the Java Program is initiated its Execution then immediately a thread will be run. A thread is called main thread. Because Every ~~method~~ Program has a main method.

we can know about the main thread or current thread by using following method

Thread currentThread ( )

Examples:-

```
Class MainThread
{
Public static void main ( String args[])
{
Thread t = Thread. CurrentThread ( );
 S.o.p ("current Thread:" + t);
 t. setName ("myThread"),
S.O.P ("After name change:" + t);
try
{
 for(int i=1; i<=5; i++)
    S.O.P (i);
} Thread . Sleep (1000);
Catch(InterruptedException ie)
{
S.O.P("main Thread Interrupted");
}
S.O.P ("main Thread Exit");
}
}
```

The above Program displays following lines

```
Current Thread:[main, 5, main]
After name change: [MyThread, 5, main]
1
2
3
4
5
Main Thread Exit.
```

The above Program we have an object call 't' of Thread type when we display obj 't' it will display the following line [main, 5, main] the line has 3 Values 1st value specifies the thread name, 2nd value specifies thread Priority, 3rd value specifies Thread groups.

→ we can also change the thread name using set name method & we can also change Priority using set Priority method.

Creation of a thread using a Running Interface:-

we can create a thread by implementing a runnable Interface. It is a simplest method to crea a thread. All thread methods are declared in runnable interface but a class which implementin a runnable interface may implement only the following method.

Public void run()

we can create a runnable instance by creating a class object. we will pass the same instance to the thread constructor as follows.

Thread(Runnable obj , String str);

where str is a thread name

write a Java Programm to create a thread by implementing a runnable interface?

```
Class NewThread implements Runnable
 {
  Thread t;
  New Thread () ;
  {
 t = new Thread (this, "child Thread");
 System.out.Println ("Child Thread:" + t);
  t. Start ();
}
Public void run()
 {
 try
  {for(int i=5; i>=1 ; i--)
  system.out.Println ("Child Thread:" + i);
  Thread.sleep(500);
  }
   Catch(InterruptedException ie)
  {
   System.out.Println ("child thread interrupted");
  }
 S.o.p ("child thread Exit");
  }
}
Class RunnableDemo
 {
 Public static Void main(String args[])
 {
```

```
new NewThread();
 try
  {
  for (int i=5; i>=1; i--);
  S.O.P ("main Thread:" +i);
  Thread.sleep(1000);
 }
  Catch(InterruptedException ie)
   {
   S.O.P ("main thread interrupted");
   }
   S.O.P ("main thread Exit");
   }
  }
```

Creation of a thread by Extending a thread Class:-

It is another way to create a new thread. All new threads are child threads of thread class. Because Every new thread ~~class~~ is Extending a Thread class.

In this method we will call ~~use~~ the following Thread Constructor by using a Super method.

Thread (string str).

where str is a Thread name which will be get from Super method.

Programm:

```
Class NewThread extends Thread
  {
   Thread t;
   NewThread ()
  {
  Super ("child Thread");
 Sys:o.p ("child thread :" +this);
```

```
    start ();
  }
  public void run ()
  {
  try
    {
     for (int i=5; i>=1; i--)
     S.o.p ("child Thread:"+i);
    Thread.sleep(500);
    }
  catch (InterruptedException ie)
   {
  S.o.p ("child thread interrupted");
   }
  S.o.p("child thread exit");
   }
  }
  Class ThreadDemo
   {
  public static void main (String args[ ])
   {
   new NewThread ();
   try
    {
     for(int i=5; i>=1; i--)
      S.o.p ("main Thread: "+i);
    Thread.sleep (100);
     }
    catch (InterruptedException ie)
     {
     S.o.p ("mainThread interrupted");
     }
    S.o.P ("main thread exit");
     }
    }
```

6

Creation of Multiple threads:-

We can also create multiple threads by using Either runnable or thread class. following Programm creates three threads those thread nam are EEE, ECE, IT. The following Program using runnable interface to create a new thread.

Programm:-

```
class NewThread implements Runnable
  {
  Thread t;
  String name;
  NewThread (String tname)
   {
   name = tname;
   t = new Thread (this, name);
    S.O.P ("NewThread :" + t);
     t.start ();
   }
   Public void run ()
  {
   try
    {
     for (int i=5; i>=1; i--)
      S.O.P (name + " : " + i);
     Thread. sleep (1000);
    }
   catch (InterruptedException ie)
    {
    S.O.P (name + "Interrupted");
   }
   S.O.P (name + "Exit");
   }
    Class MultiDemo
    {
```

```
public static void main(String args[])
   {
   NewThread ob1 = new NewThread("EEE");
   NewThread ob2 = new NewThread ("ECE");
   NewThread ob3 = new NewThread ("IT");
   try
     {
      Thread.sleep (10000);
     }
    catch (InterruptedException ie)
     {
      S.O.P ("main thread interrupted");
     }
    S.O.P ("main thread exit");
      }
    }
```

Using isAlive() & join() methods

Generally we use sleep() method to suspend a thread upto a period of time but using this method based on Period of time the threads will be terminated. But we have a Question about thread termination. i.e., How can the thread know the other thread has Ended? It is possible with isAlive() & join() methods.

→ isAlive() method is used to check wheather the thread is alive are not. If the thread is alive it will written true otherwise false. waiting for a thread to be finished is known by the join method because the threads are joining to be finish.

```
         Class NewThread implements Runnable
         {
         Thread t;
         String name;
         NewThread (string tname)
```

```
{
name=tname;
t=new Thread (this, name);
S.O.P(" NewThread :"+t);
 t.start();
 }
 Public void run()t
 {
 try
  {
   for(int i=5; i>=1; i--);
   S.O.P(name+":"+i);
   Thread .sleep(1000);
  }
 Catch(Interrupted Exception ie)
  {
  S.O.P(name+"Interrupted");
  }
 S.O.P(name+ "Exit");
 }
}
  Class DemoThread
   {
 Public static void main (string args[])
   {
 NewThread ob1 = new NewThread ("EEE");
 NewThread ob2 = new NewThread ("ECE");
 NewThread ob3 = new NewThread ("IT");
 S.O.P ("Thread EEE is alive:" ob1.t.isAlive())
S.O.P("Thread ECE is alive:" ob2.t.isAlive());
S.O.P ("Thread IT is alive:" ob3.t.isAlive());
try
{
S.O.P ("waiting for a thread to finish");
```

```
      ob1.t.join();
      Ob2.t.join();
      Ob3.t.join();
    }
    catch (InterruptedException ie)
    {
    S.O.P ("Interrupt");
    }
  S.O.P("Thread EEE is alive:" ob1.t.isAlive());
  S.O.P ("Thread ECE is alive:" ob2.t.isAlive());
  S.O.P ("Thread IT is alive:" obj3.t.isAlive());
  S.O.P ("main thread Exit");
    }
  }
```

21-03-19

APPLET PROGRAMMING:-

Applet:-

Applet is a special Programm which was Embedded in webpage. It will be run in a web browser, and work at client side. It has the following rules

1) we must import java.applet.*; & java.awt.*
2) The appleet class should be declared as Public.
3) An applet class should be Extends an Applet class
4) In applet Programming we never use main meth
5) Every applet uses its life cycles methods.
6) we can compile applets same as java applicati
7) we can run an applets using either html file or using applet viewer tool (for testing)
8) we can create something in a applet window by using Paint method.

** Applet life cycles:

The applet has the following 5 life cycles.

1) Applet initialized
2) Applet Started
3) Applet Painted
4) Applet stopped
5) Applet destroyed.

$$\begin{tikzcd}
{} \arrow[d, "init()"] \\
\text{Initialized} \arrow[d, "Start()"] \\
\text{Started} \arrow[d, "Paint()"] \\
\text{Painted} \arrow[d, "stop()"] \\
\text{Stopped} \arrow[d, "destroy()"] \\
\text{destroyed}
\end{tikzcd}$$

Applet initialization:

The applet is initialized after invoking init(). It will be call only once

Applet started:

After initialization of an applet the JVM will call then immediately applet will be start.

JVM ↓ Java virtual Machine

Applet Painted:

After applet starting the JVM will be call Paint() in order to draw someting in a applet Panel.

Applet stopped:

The Applet will be stoped after invoking stop method whenever an applet is stopped will again restart the stop() using start()

Applet destroyed:-
The applet is destroyed after invoking destroy() it will be call only once after destroy we can't restart the same applet.

→ The init(), start(), stop() and destroy() are defined in Java.applet.* package so that we have to import this package into our applet Programm.

→ The paint() is defined in Java.awt.* package so that we have to import this package into our applet Programm.

Skeleton part of Applet Programm:-

```
import Java.applet.*;
import Java.awt.*;
Public class Simple extends Applet
{
Public void init()
 {
 - - -
 }
public void start()
 {
 - - -
 }
Public void Paint(Graphics g)
{
- - -
}
Public void stop()
{
- -
}
Public void display()
{
- - -
}
}
```

```
/**
<applet code ="simple.class" width="500" height="500">
    </applet>
      **/
```

21-03-19

Methods:

The applet class use the following methods

Method name		Description
1) *init( )	$\longrightarrow$	To initialize an applet
2) *start( )	$\longrightarrow$	To start an applet
3) *Paint ( )	$\longrightarrow$	To draw something on a applet Panel these method is taken from awt Package.
4) *stop ( )	$\longrightarrow$	To stop an applet.
5) *Destroy( )	$\longrightarrow$	To destroy an applet.
6) drawRect( )	$\longrightarrow$	used to draw a rectangle This method is in Graphics class.
7) fillRect ( )	$\longrightarrow$	To fill colour to the rectangle box.
8) drawOvel( )	$\longrightarrow$	To draw a ovelshape
9) fillOvel( )	$\longrightarrow$	fill ovelshape
10) drawline( )	$\longrightarrow$	To draw a line
11) drawArc( )	$\longrightarrow$	To draw an arc
12) *setColor ( )	$\longrightarrow$	To set aspecified color The colours will be taken as Color.red where Color is a class.
13) setSize( )	$\longrightarrow$	To set an image size

14) *drawString() → To display a string in a applet window.

15) *drawImage() → To display an image in a applet window.

16) *showStatus() → The specified string will be shown as status of applet.

17) *getCodeBase() → returns path of the specified file.

18) *getDocumentBase() → Specify the path along with the file name.

19) *resize() → To resize the applet panel.

20) play() → To play an audio or video clip.

21) *setBackground() → It is used to set the backgrou nd color

22) *setForeground() → It is used to set the foregro -und color

23) getParameter() → It is used to get Parameter values from the Param tag.

Applet tag:-

In applet Programing the applet tag will be Embedded in Java source file by using comment lines and we can also use applet tag by a sperate file those file is called HTML file. The Applet tag as follows.

```
<applet
  code = "filename.class"
 [code Base] = "url of class"
  [ALT] = "alternative text",
  [HSPACE] = "200"
 [VSPACE] = "500"}
  width = "500"
```

```
    height = "500"
    [align ]= "RIGHT"
    [Name] = "instance name";
    >
</applet>
```

Code:-
This attribute is used to specify the applet class

Code Base:-
It is used to specify the Path of an applet

ATT:-
These attribute is used to specify the alternati
text for an applet.

HSPace:-
It is Specify the Horizontal space

VSPACE:- It will specify the vertical Space.

width:- It will specify the width of an appl

Height:- It will Specify the height of an applet

align:- It will specify the alignment of an applet

Name:- It will specify the applet instance Name.

Note:

The Square brackets represents the optional attributes.

Param tag:-
This ~~s~~tag is used to Specify the Parameter name & value which are Passing t the applet.

Syntax:-

```
<Param name ="Param-name"
            value="param-value" >
</Param>
```

write a Java Programm to create Simple applet:

```
import java.applet.*;
import java.awt.*;
/**
<applet. code ="Simple.class" width="500"
                              height="500" >
</applet>
**/
Public class Simple Extends Applet
{
 public void paint (Graphics g)
 {
 g.drawString("welcome to EEE", 10,20);
 }
}
```

Output:-

→ Save → Simple.java

Compile → javac Simple.java

run → appletviewer Simple.java

```
applet
    welcome to EEE

applet started
```

Write a Java Programm to creat an applet.

```
import java.applet.*;
import java.awt.*;
/**
<applet code="week11a.class" width="500" height="5oc
</applet>
**/
```

```
Public class Week11a Extends Applet
 {
  String msg;
  Public void init()
   {
  msg = "I am in init() method ----";
  }
 Public void start()
  {
 msg + = "I am in start() method-----";
    SetBackground (color. red);
   Set Foreground (color. Yellow);
 }
 Public void paint (Graphics g)
  {
 g. drawString (msg, 10, 20);
 g. draw String ("welcome to EEE", 50, 100);
 Showstatus ("This is my first applet");
  }
 }
```

Ex: 2

Passing Parameters to the Applet:

we can Pass the Parameters to the applet using Param tag. The Param tag has Parameter name & Value. we can get Parameters from the Param tag using getParameter (). This method has only one argument i.e, Parameter name.

Syntax:

```
String getParameter(String str);
```

The syntax of param-tag as follows

<Param name = "Parame Name" Value="Param-value">
    <\Param>

write a Java Programm to pass Parameters to the Programm:

```
          import Java.applet.*;
          import Java.awt.*;
        /**
    <applet code="weeklb.class width="500" height="500">
   <Param name="message" value="Hello EEE" >
    </Param >
    </applet >
   **/
   Public class weekllb extends Applet
     {
     String msg;
     Public void start( )
     {
    Set Background(color.red);
   Setforeground (color.pink);
     msg = get Parameters("message");
     }
     Public void paint (Graphics g)
     {
     g.drawString (msg, 50, 100);
      }
   ShowStatus ("This is my second Applet")
     }
     }
```

```
import java.applet.*;
import java.awt.*;
/**
<applet code="week11c.class" width="1300" Height="100">
</applet>
**/
Public Class week11c Extends Applet
{
Image img;
Public Void start()
{
img =getImage(getDocumentBase(),"boy.gif");
}
Public void Paint (Graphics g)
{
for (int ~~i~~ i=0; i<1300; i++)
{
g.drawImage(img, i, 30, this);
try
{
Thread.sleep(20);
}
catch (InterruptedException ie)
{
System.out.Println (ie);
}
}
}
}
```

Note:-
getImage() is used to get a Particular specified image. It has two Parameters one is the path of a specified image & second Parameter is specify the image name

using html file:- 23-03-19

we can Execute the applet using appletviewer or using html file. we can Execute the applet Programm using applet viewer as follows.

appletviewer classname. java,

we can also Execute applet Program by using html file. we can Embedde the applet tag inside html file rather than java file.

~~for~~ Example:

```
import . java . applet .*;
import java. awt .*;
Public class Simple extends Applet
{
Public void paint (Graphics g)
{
g. drawString ("welcome to EEE", 40, 50);
}
}
```

Save it as Simple. java

```
<html>
<body>
<applet code = "simple. class" width="500" height="500">
</applet>
</body>
</html>
```

html → hyper text markup language

Save it as: Simple. html

Execution of applet:-

① first we have to compile javafile

javac Simple. java

It will generates class file.

② Now we have to run ~~simple~~ Either java file or html file using appletviewer tool.

3) we can also Execute applet with out using appletviewer tool as follows

a) first we have to compile Java file.

b) Now open html file with the browser. The browser simply gets an applet and Execute it but here ~~to~~ execute applet using browser we need Java plug-in (JVM)

Streams in java:- 26-03-19

In Java stream is a Sequence of data. A stream can be represented Either in bytes or characters. so that we have the following two categories of streams.

1. Byte streams
2. Character streams

Byte stream:-

The byte stream has the data interms of Bytes. we can read the data or write the data interms of bytes [1 byte = 8 bits]. The bytes can be represented in multiples of 0's & 1's. The Byte streams are divided into following Categories.

1) I/P stream - Input stream
2) O/P stream - output stream

Input stream:-

The input stream is a class which is used to read data from the file or console, It has the following subclasses

a) fileInputStream
b) BufferedInputstream
c) DataInputStream
d) ByteArrayInputStream
e) SequenceInputStream

The i/p stream class has the following methods to read the data from the Either file or console

read():- read the data from eith file or console.

close():- close the opened stream.

Output Stream:-

→ The o/p stream is a class which is used to write the data to the file or console it has the following subclasses.

a) fileoutputStream
b) Bufferedoutput stream
c) Data output stream
d) ByteArray output stream.
e) Print stream.

It has the following methods.

write ():- write data to the file or console

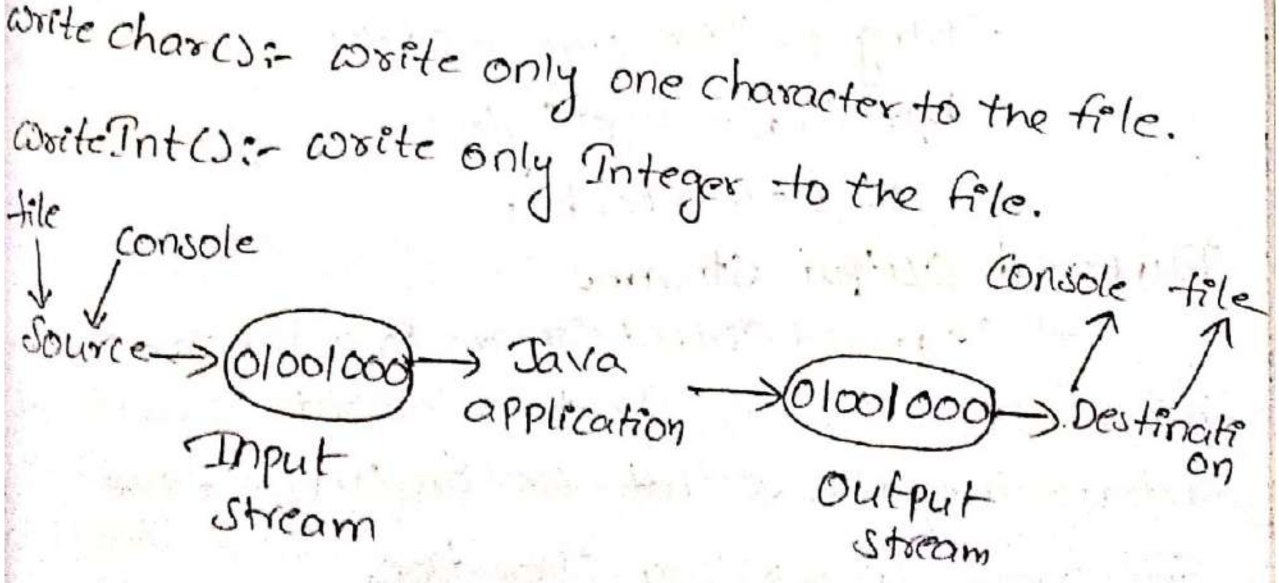
close ():- close the opened stream

writeTo ():- The ByteArray output stream uses writeTo(). to write same data to the two different files.

write chars():- The DataOutputStream uses write chars(). To write all characters to the file.

write char():- write only one character to the file.

writeInt():- write only Integer to the file.



FileOutputStream:-

It is a Sub class of output stream class. It will be used to write the data to the file. It has only One Parameter i.e., filename. It will use write method to write the data to the file.

Eg:-
```
import java.io.*;
Class FOS
{
Public static void main(String args[])
{
try
{
Fileoutputstream fout =new fileoutputstream
                                ("EEE.txt");
fout.write(65);
fout.close();
System.out.Println("success");
}
Catch(Exception e)
{
System.out.Println(e);
}
}
}
```

→ To write multiple characters to the file we have to add the following lines.

After fileoutputstream
```
String s = "welcome to EEE";
byte[] b = s.getBytes();
fout.write(b);
```

Buffered output stream:-

A Buffered Output Stream is a Subclass of output Stream. It is used to Create a seperate Buffer to add data to the file, so that we can improve the Efficiency of writing operation.

Eg:-
```
import Java.io.*;
class BOS
{
Public static Void main (String args[])
{
try
{
fileoutputStream fout = new fileoutput stream
                                    ("ECE.txt");
Bufferedoutputstream bout = new Bufferedoutpt
                                    Stream(fout);
String s = "welcome to ECE";
byte[] b = s.getBytes();
bout.write(b);
bout.close();
System.out.Println("succes");
}
catch (Exception e)
{
system.out.Printn(e);
}
}
}
```

Data OutPutstream:-

The Dataoutputstream is a subclass of output stream It is also similar to other o/p stream class but it is machene independent. It will use writechars() to write all characters to the file.

Eg:-
```
import Java.io.*;
  Class Dos
  {
  Public static Void main(string args[])
   {
   try
   {
fileoutputstream fout = new fileoutputStream("bbc.txt");
Dataoutput Stream dout = new Data output stream
                                    (fout);
```

```
 String s = "welcome to AITS";
  dout. writechars (s);
  dout. close( );
  fout. close();
 System. out. Println ("Success");
  }
 Catch (Exception e)
  {
  System. out. Println (e);
  }
 }
}
```

ByteArrayoutputStream :-

It is a subclass of output stream. It is used to write Bytes to the different files write method is used to write the data to the ByteArrayoutputstream class then the same data is written to using two different files using writeTo ()

Eg:-

```
   class BAOS
   {
   Public static void main(String args[])
   {
   try
   {
  fileoutputstream fout1 = new fileoutputStream ("f1.txt");
 fileoutpustream fout2 = new fileoutputstream("f2.txt");
 ByteArrayoutput stream bout = new ByteArrayOutput
                                          stream();
  bout. write(65);
  bout. writeTo (fout1);
  bout. writeTo (fout 2);
  bout. close();
  fout1 . close ();
 S.o.p ("success");
 }
 catch (Exception e)
 {
    S.O.P(e);
  }
 }
}
```

PrintStream:-

It is used to Print something in a file or console Here we will use Either Print or Println method to write the data to the file or console.

Eg:-
```
class Ps
{
public static void main(String args[])
{
try
{
fileoutputStream fout=new fileoutput stream("1.txt");
PrintStream Ps = new PrintStream(fout);
  Ps.Println(2016);
  Ps.Print("Hello world");
   Ps.close();
   fout.close();
   S.O.P("success");
}
catch (Exception e)
{
S.O.P(e);
}
}
}
```

InputStream:-

The Input stream Class is used to read data from the file or Console. Here all subclass of Inputstream gives read() to read the data from Either file or Console.

The following Classes are Subclasses of Input Stream.

1. file InputStream
2. Buffered Input stream
3. Data Input stream

4. ByteArray InputStream.

5. Sequence InputStream.

- File InputStream & Buffered InputStream (FIS & BI

The FIS is used to read the data from file. The BIS is used to store the data in a buffer so that it will increase the efficiency of read operation.

Eg:- Class FISBIS
```
      {
      Public Static Void main (String args[])
      {
      try
       {
  fileInputStream fis = new fileInputStream ("EEE.txt");
  BufferedInputStream bis = new BufferedInputStream
                                              (fis);
    int i=0;
    while((i=bis.read())!=-1)
     S.O.P ((char) i);
     bis.close();
     fis.close();
     S.O.P ("success");
     }
     Catch (Exception e)
      {
      S.O.P(e);
      }
    }
   }
```

DataInputStream:-

The DataInputStream is used to read the data from a file or console in machine independent way.

```
Eg:-  Class DIs
      {
  Public static void main (String args[])
    {
   try
    {
 fileInputStream fis = new fileInput stream("EEE.TXT");
 DataInputStream dis = new DataInput stream(fis);
   int i=0;
  while ((i= dis.read())! =-1)
   dis. close();
   fis . close();
   System.out.Println ("succes");
     }
   Catch (Exception e)
     {
   System.out.Println (e);
     }
   }
  }
```

ByteArrayInputStream:

. It is used to read data from the Byte Array

```
Eg:- Class BAIs
      {
     Public static void main (String args[])
    {
    try
    {
    byte[]b={65, 66, 67}.
  ByteArrayInputStream bais = new ByteArrayInput
                                        Stream(b).
    int i=0;
   while( (i=bias.read()! = -1)
    S.O.P ((char)i),
    bias. close():
   S.O.P("succes");
}
catch(Exception e)
{
S.O.P(e);
}
}
}
```

Sequence InputStream :- SIS

It is used to read the data from two different files & combined those files data & write to another file.

Eg:- Class SIS

```
{
Public static void main(String args[]).
{
try
{
FIS fis1 = new fis("ccc.txt");
FIS fis2 = new fis("bbc.txt");
fos fos = new fos("merge.txt");
SIS sis = new SIS (fis1, fis2);
int i=0;
while((i=sis.read()).!=-1)
fos.write(i);
fis1. close();
fis2. close();
Sis. Close();
fos. close();
System.out.Println("success");
}
catch(Exception e)
{
System.out.Println(e);
}
}
}
```

22-03-19

Character stream:

Character streams are used to read or write data in the form of characters. we have the following two classes. 1) Writer
2) Reader

writer:

writer class is an abstract class it is used to perform write operation and character set. It has the following sub class,

1) fw — file writer   5. Pw - Print writer
2) BW — Buffered writer
3) CAW - Character Array writer
4) OSW - output stream writer

The above sub classes are use write( ) to write either strings or character to the specified file are console

file writer & Buffered writer:

file writer class is used to write data to the file before writting the buffered writter class will create the buffer to store the data so that it will give more fast performance in write operation both classes are extends writer class.

Ex:
```
Import java.io.*;
    Class FWBW
    {
    Public static void main(String args[])
    {try
    {filewriter fw=new filewriter("one.txt");
    Bufferedwriter bw = new Buffered writer(fw);
      bw.write("welcome to java class");
     bw.close();
     fw.close();
     S.O.P (" succes")
    }
```

```
      catch(Exception e)
      {
      S.O.P(e);
  } } }
```

Character Array Writer:

It is used to write some data to the differen files. first it will write data to the Character Array using write(), then it will write some data to the files using writeTo().

Ex:- import java.io.*;

```
    class CAW
    {
    Public static void main(String args[])
    {
    try
    {
    filewriter fw1 = new file writer("Two.txt");
    file writer fw2 = new filewriter ("Three.txt");
    charArraywriter caw = new charArraywriter();
     caw.write ("I don't like java");
     caw.writeTo(fw1);
     caw.writeTo(fw2);
     caw.close();
     fw1.close();
     fw2.close();
     S.O.P ("success");
     }
     catch (Exception e)
     {
      S.O.P(e);
  } } }
```

Output Stream writer (OSW)

This class is used to convert the Character stream into byte stream. the write method simply calls Encoding converter to ~~converter~~ the characters into bytes. To write data to the file we have to use file output stream class insteade d of file writer.

Ex:
```
import Java.io.*;
·class OSW
{
public static void main (String args[])
{
try
{
fileoutputStream fout= new fileoutputStream
                                ("four.txt");
Outputstream writer OSW = new Outputstreamwriter
 osw.writer ("I like C programming");   (fout);
  osw.close ();
 fout.close ();
 s.o.p("success");
 }
 catch (Exception e)
 {
 s.o.p (e);
}
}
}
```

Print writer:

It is used to Print the data to the file or console using write method

Ex:
```
import Java.io.*;
   class PW
{
 public static void main(String args[])
 {
 try
{
file writer fw= new file writer ("file.text");
Print writer Pw = new Print writer (fw);
Pw. write ("I like php Programming");
  Pw. close();
  fw. close();
 S.o.p("success");
 }
catch(Exception e)
  {
  s.o.p(e);
  }
 }
}
```

Reader:-

It is an abstract class which is used to read the data from either file or console it will use Read or Read line method to read the data. It has the following subclasses

1) FR - file Reader
2) BR - BufferedReader
3) CAR - character Array Reader
4) ISR - Input Stream Reader.

file Reader & Buffered Reader:-

file Reader Class is used to read the data from the file. Buffered Reader will create Buffer to store the data. Both the classes are extends Reader class

Ex:-
```
Import java.io.*;
Class FRBR
{
```

Scanned by CamScanner

```
Public static void main(String args[])
{
try
{
file Reader fr= new fileReader ("one.text");
Buffered Reader br=new BufferedReader(fr);
  int i=0;
  while ((i=br.read())!=-1)
   S.O.P ((char)i);
   br.close();
  fr.close();
   S.O.P ("success");
   }
  catch (Exception e)
   {
  S.O.P(e);
   }
 }
}
```

Character Array Reader:
It is used to read the data from character array.

Ex:
```
 import java.io.*;
  Class CAR
 {
 Public static void main(String args[])
  {
  try
  {
  char [] ary = {'J', 'A', 'V', 'A'};
  CharArrayReader car= new charArrayReader
                                   (ary);
   int i=0;
  while ((i= car.read()) !=-1)
   S.O.P((char)i+":"+i);
   Car.close();
```

```
S.O.P("succes");
}
Catch (Exception e)
{
S.O.P(e);
}}}
}
```

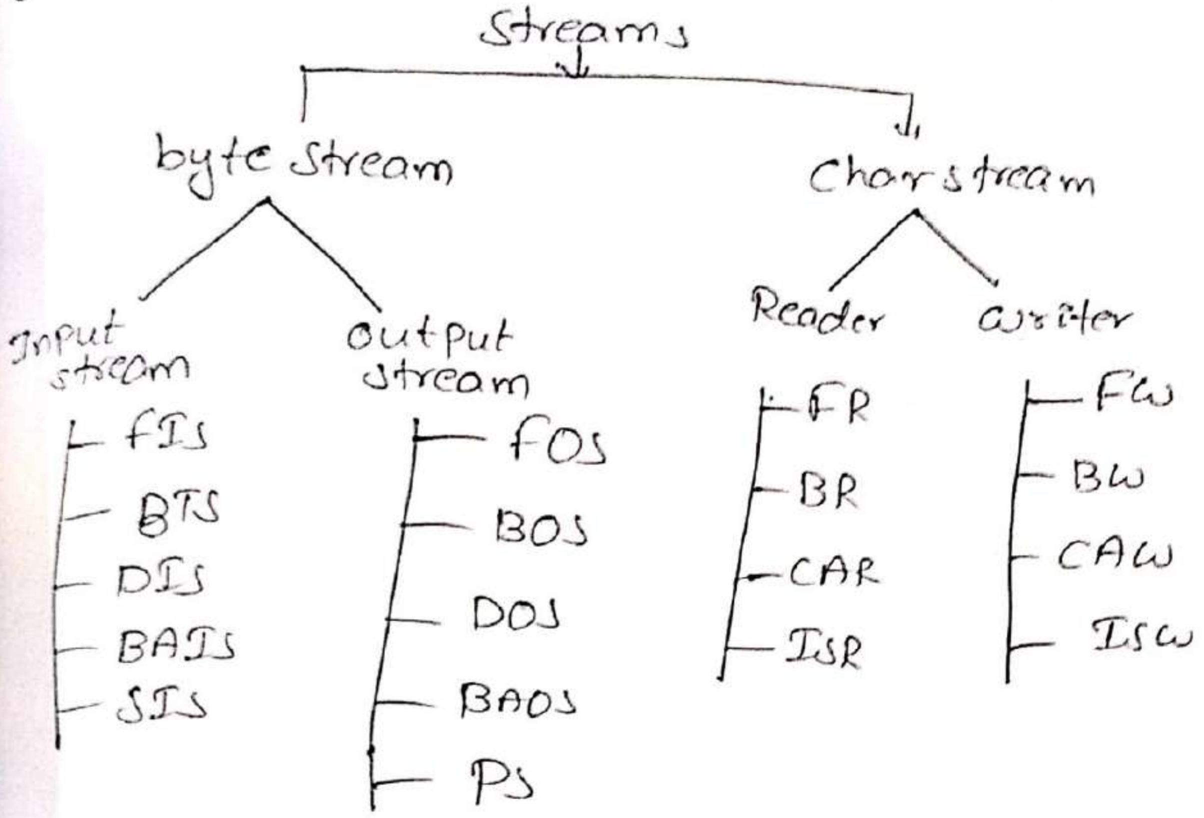
Input StreamReader:-
It is used to convert the byte stream into character stream. The Read() will call the Encoding converter to convert byte stream to charac-ter stream. Here we have to use fileInputStream ~~class~~ class. To read bytes from a file.

Ex:-

```
import java.io.*;
Class ISR
{
Public static void main(String args[])
{
try
{
file Input stream fis = new file Input stream
                                    ("Two.text");
Input stream Reader isr = new Input Stream
                                    Reader(fis);
int i=0
while(ci=isr.read())!=-1)
  S.O.P((Char)i);
  isr.close();
  fis.close();
  S.O.P("succes");
  }
  catch (Exception e)
   {
   S.O.P(e);
   }
  }
}
```

Scanned by CamScanner

Classification of streams:-



Along with above streams we have the following three streams. These stream are automatically created. Those are.

1) System.in - It is used to read the data from Console.

2) System.Out - It is used to write the data to the Console (I/P or O/P devices)

3) System.Err - It is used to write Error msg on console.

The above 3 streams are used to perform console I/P & O/P operations. (input & output device operations)