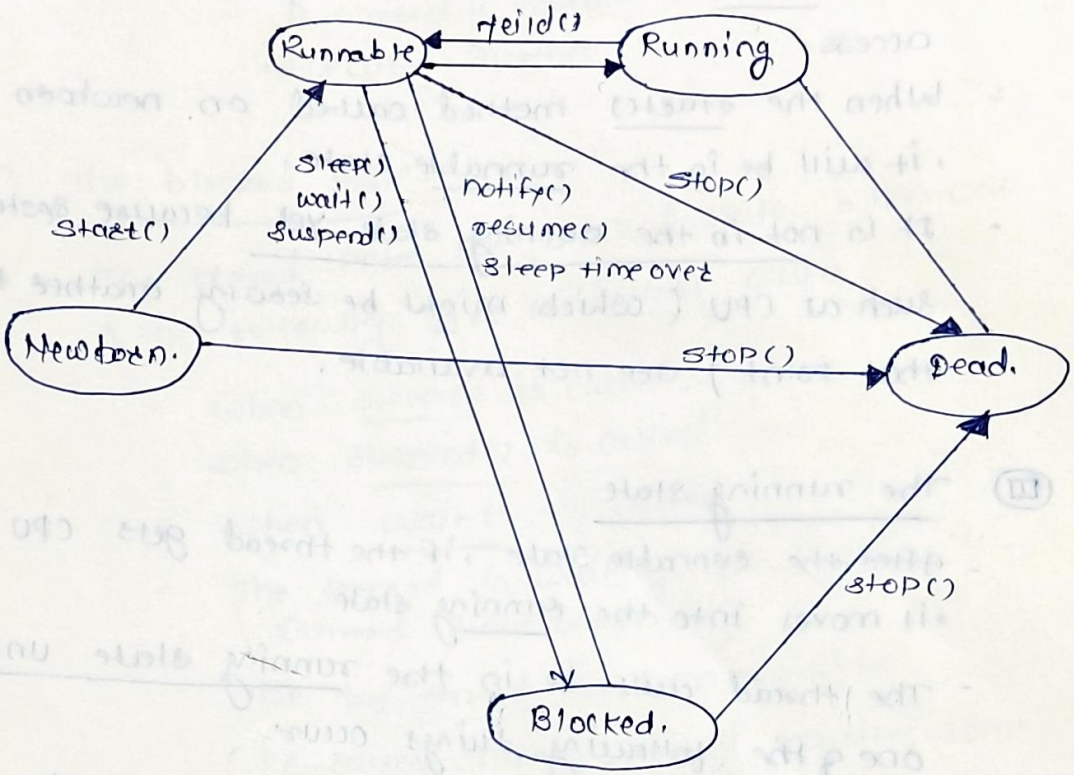
* Thread Life Cycle

A thread is always in one of five states :

- newborn
- runnable
- running
- dead
- Blocking / blocked.

fig shows the life-cycle of a thread.



(I) Newborn state

When a thread is called, it is in the newborn state, that is, when it is created and is not yet running

In other words, a start() method has not been invoked on this thread.

In this state, system resources not yet alloted / allocated to the thread.

when the thread is in the newborn state, calling any method other than sleep start() method causes an IllegalThreadStateException.

(II) The runnable state

- A thread in the runnable state is ready for execution but is not being executed currently.
- Once the a thread is in the runnable state, it gets all the resources of the system (e.g access to the CPU) and moves on to the running state
- All runnable threads are in a queue and wait for CPU access.
- When the start() method called on newborn thread, it will be in the runnable state.
- It is not in the running state yet because system resources such as CPU (which might be serving another thread at that point) are not available.

(III) The running state

- After the runnable state, if the thread gets CPU access, it moves into the running state
- The thread will be in the running state unless one of the following things occur.
  - It dies (this is, the run() method exits)
  - It calls sleep() method.
  - It calls wait()
  - It calls yield().
- A thread with highest priority than the running thread can preempt the running thread if the thread with the highest priority comes out of the sleeping state

(IV) The dead state

The thread goes into dead state in two ways.

- If its run() method exits
- A stop() method is invoked
- A stop() method · kills the thread.
- run() method ~~execute~~ exits, when its finished execution naturally
- A thread is dead state cannot be executed further.

(V) The blocked state

The thread enter the blocked state. when one of the following five conditions occurs.

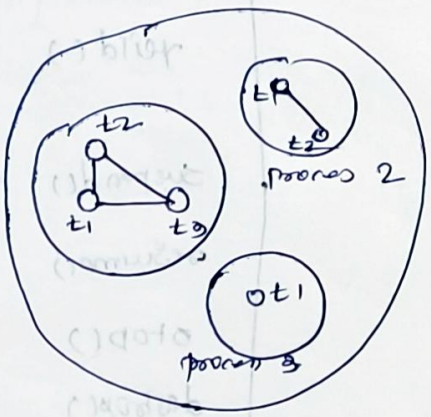
- when sleep() is called.
- when suspend() is called
- when wait() is called
- The thread is waiting for monitor (thread synchronization)
- The ~~the~~ thread calls an operation.

(i.e current running thread requires some I/O operation, it will enters into blocked state by giving the access to the CPU to another thread · After completion of the I/O operation it will enter into the runnable state).

- A thread must move out of a blocked state into the runnable (or running) state using the opposite of whatever phenomenon put into the blocked state.

Multithreading

- Multithreading in a Java is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing.
- Multiprocessing and Multithreading both are used to achieve multitasking.
- We use multithreading than multiprocessing because thread are used to share memory area. They don't allocate separate memory area so using multithreading save memory.
- Threads are independent. If exception occurs in one thread it doesn't affect on other threads.



Java Thread methods (80+ plus methods are there)

modifier & return type	method	
	start()	
	run()	
	sleep()	
	currentThread()	
	join()	It wait for a thread to die.
	getPriority()	
	setPriority()	
	getName()	
	setName()	
	isAlive()	
	yeild()	It causes the currently executing thread to call other thread with equal priority.
	suspend()	
	resume()	
	stop()	
	destroy()	
	isDemon()	It test the thread is daemon (low priority) thread
	setDemon()	It makes the thread as daemon or user thread.
	activeCount()	It returns the no. of active threads in a current thread group.
	wait()	
	notify()	It is used to send notification for only one thread which is waiting for particular object
	notifyAll()	It is used to send notification for all waiting threads of a perticular object.

Creation of a thread

There are two ways to create thread.

1) By extending Thread class
2) By implementing Runnable interface

- Thread class provide constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.
- Thread class constructors are

Thread()
Thread(String name)
*. Thread(Runnable r)
Thread(Runnable r, String name)

Creating thread using Thread class

```
class MyThread multi extends Thread
{
p.s.v.main(String args[])
{
Thread t=new Thread("my first thread");
t.start();
Multi m1 = new Multi
}
class
```

P(4)

Creating a thread by using Thread Class

```
class Multi extends Thread
{
    Public void run()          // overriding run() method
    {                          //  in a Thread class
      S.O.pl("Thread is running")
    }
  P.S.V.main(String args[])
    {
      Multi t1 = new Multi();
          t1.start();
    }
}
```

Creating a thread by using Runnable Interface

```
class Multi implements Runnable
{
   Public void run()
   {
    S.O.pl("thread is running");
   }
   P.S.V.main(String args[])
     {
      Multi m1 = new Multi()
      Thread t1 = new Thread(m1)  //using Constructor
                                  Thread(runnable
                                  obj).
          t1.start();
     }
}
```

∴ => Basic diff. bet?. Thread and Runnable is that each thread defined by extending Thread class creates unique object and get associated with that object whereas each Thread defined by ~~usi~~ implementing Runnable interface share the same object

	Extends Thread class	Implements Runnable Interface
Basic	- Each thread creates unique object and gets associated with it	- Multiple thread share the same object
Memory	As each thread creates a unique object. more memory required	As multiple thread share same resources req'd. less memory
Extending	∵ Multiple inheritance not possible ∴ - once create A Thread by using extends Thread class it can not extend any other class	If a class define thread Implementing the Runnable interface it has a chance of extending one class
Use	If use wants override other methods in Thread class ~~user extends must~~ ~~exte~~ need to create a thread by extends Thread	If user want to specialize run() method then implementing is Runnable is better option
coupling	Tight coupling	loose coupling.

Thread Synchronization and Interthread Communication

1) Thread Synchronization

by 2 methods (i) Using Synchronized statements

Syntax:-

```
Synchronized <object>
{
  statements;
}
```

ii) Using Synchronized methods.

```
Synchronized <return type> methodName()
{
  statements;
}
```

2) Interthread Communication

It is achieved by using wait(), notify() and notifyAll() inside the Synchronized statements.

eg MoreeBookApp.java.

```
class Total_Earning extends Thread
{
  int total = 0
  Public void run()
  {
    for(int i=0 ; i<=10 ; i++)
    {
      total = total + 100;
      S.O.Pl("Total Book in Amt is" + (total));
    }
  }
}
class MoreeBookApp
{ p.s.v.main(String args[])
  { Total_Earning Te = new Total_Earning()
    Te.start();
    S.O.Pl("Total Earning is" + te.total + "Rupees");
  }}
```

(P-8)

o/p => Total Earning 0 Rupees

Expected is 1000 Rupees

We achieved this by use of wait() and notify() inside Synchronized statement.

```
Class Total_Earning
{
    int Total = 0
    public void run()
    {
        Synchronized (this)
        {
            for(int i=0 ; i<=10; i++)
            {
                Total = Total + 100
            }
            this.notify();
        }
    }
}
```

```
Class Movee Book APP
{
    p.s.vm main (String args[])
    {
        Total_Earning te = new Total_Earning()
        te.start();
        Synchronized (te)
        {
            te.wait();
            S.O.Pl(" Total Earning "
                + te.Total
                + " Rs");
        }
    }
```

* Multithreading *.

- 1) We implement a thread ~~by by using~~ by using a object of a class which contains run() method.

2) by using run() method we implements behaviour of the thread.

3) We do call this run() method by using object of class contains run() method. start() method.

i.e t_1.start().

(*) ⇒ t_1 is the object of class contains run() method.

TO call the run() method use start() method.

(*) ⇒ This class extends Thread class / implements Runnable. contains run() method

Let us take e.g

Class A	Class B	Class C	class ThreadTest.
print i = 0 to 5	print j = 0 to 5	print k = 0 to 5	

```
Class A extends Thread
{
    Public void run()
    {
        for(int i=0 ; i<=5 ; i++)
        {
            S.O.Pl(" \t from thread A : i = " +i);
        }
    }
}
class B extends Thread
{
    Public void run()
    {
        for(int j=0 ; j<=5 ; j++)
        {
            SOPl(" \t from thread B : j " = +j);
        }
    }
}
class C extends Thread
{
    public void run()
    {
        for(int k=0 ; k<=5 ; k++)
        {
            SOPl(" \t from thread C : k " =+k);
        }
    }
}
class ThreadTest.
{ P.S.V. main (String args[])
    {
        A threadA = new A();   threadA.start();
        B threadB = new B();   threadB.start();
        C threadC = new C();   threadC.start();
    }
```

getting current thread details / setName() / sleep()

(3)

```
class CurrentThread
{
    P.S.V.main (String args[])
    {
Thread. t1 = Thread.currentThread();
S.O.Pl(" current thread details are : " + t1)
```

O/P

current thread details are :

Thread[main, 5, main]

- main → Name of the thread
- 5 → default priority
- main → group name

```
    t1.setName("XYZ");
SOPl (" After changing Name" + t1);
```

O/P => Thread[XYZ, 5, main]

```
try
{
for (int i=0 ; i<=5 i<=10 ; i++)
    {
        S.O.Pl(i);
        t1.sleep(1000);   (controlling main() Thread)
    }
}
Catch ( InterruptedException e)
    {
    S.O.Pl( "main thread Interrupted" +e);
    }
}
}
```

Create a Thread by Implementing Runnable Interface (4)

```
class NewThread Implements Runnable
{
    Nt Thread t.
    NewThread()
    {
        t = new Thread(this, "demo Runnable Thread");
        t. sopl("child Thread details" + t);
        t.start();
    }
    Public void run()
    {
        try
        { for(int i=0; i<=5; i++);
          sopl("child thread: i=" + i);
          sopl. t.sleep(500);
        }
        catch (Interrupted exception e)
        {
          sopl("child thread Interrupted" + e);
        }
        sopl("exit from child thread");
    }
}
class RunInterfaceDemo
{ p.s.v.main(String args[])
  {
    new NewThread();
    try
    { for(int j=0; j<=5; j++)
      {
        s.o.pl("Main Thread" + j)
        Thread.sleep(1wo);
      }
    }
    Catch (Interrupted exception e)
    {
      sopl("Main thread Interrupted" + e);
    }
    sopl("exit from main thread");
}
```

Creating Multiple threads Using Runnable

```
class Newthread implements Runnable
{
    String name
    Newthread (String threadname)
    {
      name = threadname
      Thread t = new Thread (this, "name");
      S.O.Pl ("New thread" +t);
      t.start
    }
    Public void run()
    {
      try
        {
          for (int i=5 ; i>0 ; i--)
          {S.O.Pl ("name +" "+i);
           Thread.sleep (1000);
          }
        }
      catch (Interruptedexception e)
        {
          S.O.Pl (name +"interrupted");
        }
    }
}
class RunMultiThreadDemo
{ P.S.V.main()
  {
    new Newthread ("First thread");
    new Newthread ("second thread");
    new Newthread ("Third thread");
    try
    { // Wait for other thread to end
      Thread.sleep (10000);   // sleeping main
    }
    catch (Interruptedexception e)
    { SOPl ("Main thread interrupted");
    }
    SOPl ("Exit main thread");
  }
}
```

How can one thread know when another thread has ended ?

in the previous program in main() method we use sleep(10000) time in nanoseconds to pause main thread, by the time op^n/calling for other thread finish and main() thread will end at last

instead of sleep(10000) we do as

1) isAlive() to know whether the thread is running or not.

Syntax

final boolean isAlive()

⇒ isAlive() occasionally used by us.
we do use join() method regularly

2) join() ⇒ This method wait until the thread upon which, it is called are terminates.

Syntax

final void join() throws InterruptedException

e.g joinDemo.java

Thread Priorities

Thread Priority are used to decide when to switch from one running thread to the next thread called Context switching

```
Public class Fivetable extends Thread
{
    public void run()
    {
        for(int i=1 ; i<=10 ; i++)
            S.o.pl(i + " Fives are " + (i*5));
    }
}
public class Seventable extends Thread
{
    public void run()
    {
        for(int i=1 ; i<=10 ; i++)
            So pl(i + " Sevens are " + (i*7));
    }
}
public class Thirteentable extends Thread
{
    public void run()
    {
        for(int i=1 ; i<=10 ; i++)
            So pl(i + " Thirteen are " + (i*13));
    }
}
Class PriorityDemo
{
    P.S.V. main()
    {
        Fivetable five = new Fivetable();
        Seventable seven = new Seventable();
        Thirteentable Thirteen = new Thirteentable();
        five.SetPriority(1);
        seven.setPriority(6);
        Thirteen.SetPriority(9);
        five.start();
        seven.start();
        Thirteen.start();
    }
}
```

(1) (*) Synchronized by statement

Syntax

Synchronized (object) < Statement >

e.g

```
Synchronized (obj1)
{
   obj1.printValue();
}
```

e.g Printing the Value 0 to 5 by calling three diff. threads

```
class class_A
{
synchronized void printValue()
  {
    try
    {
      for (int i=0 ; i<=5 ; i++)
        {
          S.o.pe(i+" ")
          Thread.sleep(1000);
        }
    }
    catch (InterruptedException e)
    {
      S.o.pe(e);
    }
  }
}
```

getValue() & 2
setValue() & 1

```
import java.io.*;
class SynchThreadExample implements Runnable
{
  Public static void main()
  {
    class_A ca = new class_A();
    SynchThreadExample one = SynchThreadExample(ca)
                one.t.start();
    SynchThreadExample two = SynchThreadExample(ca)
                two.t.start();
    SynchThreadExample three = SynchThreadExample(ca)
                three.t.start();

  }
  class_A obj; Thread t;
  SynchThreadExample(class_A c)
  {
    obj = c;
    t = new Thread(this);
  }
public void run()
  {
    Synchronized(obj)
      {
        obj.printValue();
      }  obj.setValue();
    }      obj.getValue();

}
```

Synchronized by statement

(II) Synchronization by Method ()

Syntax

```
Synchronized <return type> methodname ( parameter)
```

Ex

```
Synchronized Void printValue()
 {
  Statements;
 }
```

~~Impl~~

```
class Class_A
{
 Synchronized Void printValue()        } Synchronized & method
  {
    try
     {
       for (int i=0 ; i<=5 ; i++)
        {
          sopl( i+" ");
          Thread.sleep(1000);
        }
    }
    catch (InterruptedException e)
     {
       sopl(e);
     }
}
```

Inter Thread Communication

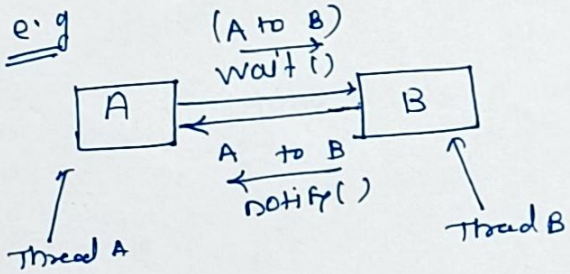
1) wait()
2) notify()
3) notifyAll()
and 4) yeild()
methods

1) wait() :- " The wait() method causes the current thread to wait till executing until another thread invokes the notify() or notifyAll() methods for that object.

2) notify() :- notify() method wakes up a single thread that is waiting on that object's monitor

3) notifyAll() :- notifyAll() method wakes up all threads that are waiting on that object's monitor.

e.g



Syntax :

if you want use wait() method then we have to use Sechronized method in program & lock object (i.e thread object)

e.g

```
Synchronized (lock object)
{
while (! condition)
  {
  lockobject.wait();
  }
}
```

```
import java.io.*;
class SynchMethodThreadExample
{
    Public static void main(String args[])
    {
declare globally // → Class_A ca = new class_A();
    SynchMethodThreadExample one = new SynchMethodThreadExample(ca)
    SynchMethodThreadExample two = new SynchMethodThreadExample(ca)
    SynchMethodThreadExample three = new SynchMethodThreadExample(ca)
        one.start();   one.t.start();
                       two.t.start();
                       three.t.start();
declare globally // Class_A obj; Thread t;
    SynchMethodThreadExample(Class_A c)
    {
        obj1 = c;
        t = new Thread(this);
    }
    Public void run()
    {
        obj1.printValue()
    }
}
```

Syntax . notify ()

```
synchronized (lockobject)
  {
    // code ;
    lockobject . notify () ;        // it wake up Single thread.
    // code
  }
```

Syntax notifyAll()

```
synchronized ( lockobject
  { //code
    lockobject . notifyAll () ;
  }
```

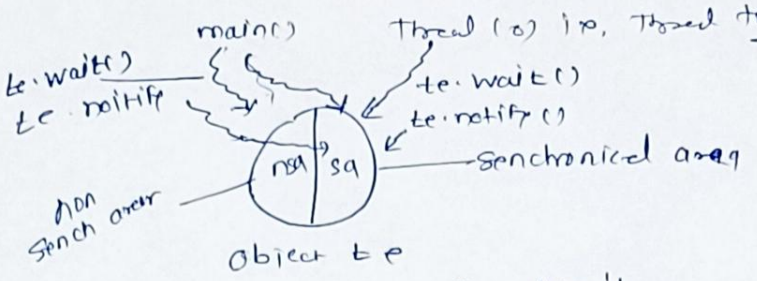
Program ⇒ MoreBookApp . java

```
class TotalEarning extends Thread
{
  int total = 0 ;
  public void run ()          // overriding run() method
  {

    for ( int i=1 ; i<=10 ; i++)
      {
        total = total + 100 ;
      }

  }
}
class MoreBookApp
{ p.s.v. main ( ) throws InterruptedException
  {
    TotalEarning te = new TotalEarning ( ) ;
                te . start ( ) ;
    SOPl ( "Total Earning" + te . total + "Rupees" ) ;

} }
```



time limit

in te.wait( )

yeild() method:- It is static method of a Thread class and it can stop the currently executing thread and will give a chance to other waiting threads of the same priority in case there are no waiting thread or if all the waiting threads have low priority then the same thread continue its execution.

WAP To create a file and content into that file.

(1)

```
import java.io.*;
Class CreateFile
{
  p.s.v.main (String args[]) throws
                    FileNotFoundException,
                    IOException
  {
     String s = "Welcome";
     byte b[] = s.getBytes();
  FileOutputStream fout = new FileOutputStream ("path");
     fout.write(b);
     sopl("file created")
     fout.close();
  }
}
```

---

(2) To display the content in file

```
import java.io.*;
class DisplyFileContent
{
  ps.s.v.m (String) throws FNFE, IOException
  {
     FileInputStream fin = new FileInputStream ("path");
       int n;
     ~~while((n=fin.read())!=-1)~~
     while ((n = fin.read()) != -1)
     {
        sopl ((char) n)
     }
     fi.close();
  }
}
```

Note:

1) While we reading the file ~~from~~ by using InputStream / FileReader need to convert into characters.

2) While writing character into file by using OutputStream / ~~File Writer~~ FileWriter

character conversion is not req^ed, mentioned / specified characters will write directly into file.

(*) Write a Java Program to accept character from the user once user enters 'q' quit from the file, store the user entered character in file and display the characters in the file.

⇒

```
import java.io.*;
class demodis2
{
  p.s.v. main (String args[]) throws IOException,
                                     FileNotFoundException
  {
    try
    {
      int n;
      FileOutputStream fos = new FileOutputStream( "G:\Ch-9\
                                File_Ex / abc.txt");
                                // stream created to write the character
      DataInputStream dis = new DataInputStream
                            (System.in);   // used to access the i/p. character

      S.o.pl ( "Enter the character's to store into file
                : For quit enter 'q' " );

      while ( (char) (n = dis.read()) != 'q' )   // ∵ comparision with 'q' char
      {                                          // ∴ converting 'in' char of is req
        fos.write (n)
      }
      fos.close();
      dis.close();
```

```
// To display the content present in the file
s.o.pt(" file contains the following Data: ");
FileInputStream fis = new FileInputStream("G:/javaexample
                                /ch-9/File_Ex/
                                abc.txt");
~~while(n~~
while((n = fis.read()) != -1)
  {
     System.out.println((char) n);
     fis.close();
  }
} //end try
catch (IOException e)
{
  s.o.pt("Error");
}
}
```

* Append the content of file f2 to file f1

// Note : Need to use Constructor

FileOutputStream (String filepath , boolean append)

ex)

```
import java.io.*;
class FileAppend
{
  p.s.v. main (String args[]) throws IOException,
                                     FileNotFoundExcepting
  {
    File f1 = new file ("D:/.../abc.txt");
    File f2 = new file ("D:/..../xyz.txt");
    if (f1.exists() && f2.exists())
    {
      FileInputStream fin = new FileInputStream (f2);
FileOutputStream fout = new FileOutputStream (f1, true);
      int n;
      while ( (n=fin.read()) != -1 )
      {
          fout.write(n);
      }
      fin.close();
      fout.close();
  }
  else
  {
    S.o.pl("File doesnot exist");
  }
}
```

WAP to write A to Z into file

```
class FileWriteDemo
{
  p.s.v. main (--11--)
  {
    try
    {
      FileWriter FW = new FileWriter("D:\\----\\abc.txt");
      for(char i = 65; i<90; i++)
      {
        FW.write(i);
      }
      FW.close();
    }
    catch (FileNotFoundException e)
    {
      SOpl(e);
    }
  }
```

Reading A to Z from file

```
class FileReaderDemo
{
  p.s.v.main(--11-->)
  {
    try
    {
      FileReader fr = new FileReader("--path--");
      int i
      while( (i = fr.read()) != -1)
      {
        S.O.pl((char) i);
      }
      fr.close()
    }
    catch (exception e)
    { SOpl(e);
```

Use of reset() method to read the same input multiple times

demo on ByteArrayInputStream

```
import java.io;
class ByteArrayInputStreamDemo
{
  p.s.v.main (---------)
  {
    try
    {
      string str = "welcome";
      byte b[] = str.getBytes()
    ByteArrayInputStream bin = new ByteArrayInputStream
                                    (b);

    for(int i=0 ; i<2 ; i++)
    {
      int c
      while ( (c = bin.read()) !=-1)
      {
        if (i = 0)
          s.o.pl ((char) c);
        else
          so pl (Character.toUpperCase
                    ((char) c));
      }
      s.o.pl()  // be next line
      bin.reset  bin.reset() ; // reset stream for
                                  next i/p
    }
  }
}
```