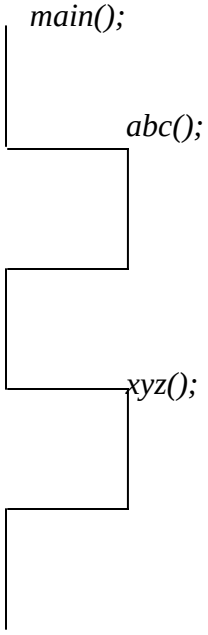
THREADS & THREADING

```
/*C_PROGRAM*/
 main()
 {
 int i,j,k;
 printf("------");
 i=j+k;
 abc();
 xyz();
}
 abc()
 {
   -----
   -----
}
 xyz()
 {
 ------
 ------
}
```



At any point of time, there is a single point of control i.e. program is going to take only one path at any point of time.

In c language we cannot execute two parts of program at a point of time.

This is also in java language, but after the AWT & JFC this is not going to take place. Using AWT & JFC, we can execute multiple tasks at the same time.
main() method is the starting of the program and the last line in your main() function is the ending of your program

In AWT & JFC even though the main program is executed the last line, the program is still executing. This is because of something is happening at the back (O/S level) apart from our main(), i.e. something similar to the main() is executing at the back.

WEB SERVER: is a program that serves the web pages to the web browsers when requested. Ex.:-Telephone operator

While he/she is answering to one call (client), the other people those who are trying the same number will hear the engage sound, and they can't able to connect to that number until unless he/she terminates the call i.e. at point of time telephone operator can't handle multiple calls. (He/she can't provide concurrent service to multiple clients unless he/she has multiple phones)

SOLUTION FOR THIS PROBLEM:

```
main()
{
 while(time) {
   Listen();           // Listens the incoming call.
   provideService();     // returns content to the Server.
 }
}
provideService(){
      ------
      ------
 }
```

* Have multiple lines.
* Set the time for each and every call i.e. 30sec.

In this case, until the time is complete for the call other person cannot able to call the same number for the same number. And if the content to be delivered is more, then also can't work well.

**When we interact more about the services provided by an O/S, then only we can say that we know better c-language

Process:-
A program under execution is known as process. Almost all the O/S can able to run multiple processes at a point of time simultaneously/concurrently.
Ex: If we opened 3 Notepad applications, then at the O/S level, it is running 3 processes of notepad concurrently.

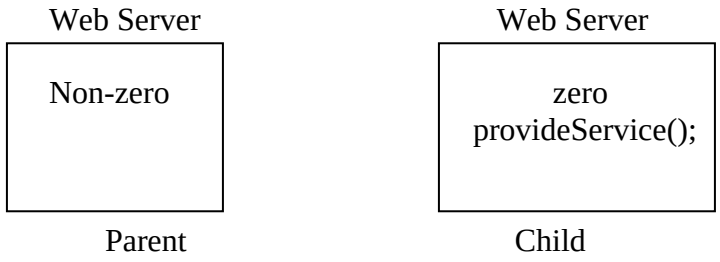
Better approach for the problem:-
Pseudo code:-

```
main() {
      while(11) // Other than zero (non-zero) in a while loop causes infinite loop in C-language.
        {
          Listen();
          pid = fork();  // pid =process Id
          if(pid != PARENT)
            provideService();
      }
   }
```

fork() --> If the program name is a web Server, it creates a same copy of the Web Server.

When a call arrives, the *Listen()* method will be executed and Listens the call. After this *pid = fork();* method will be executed, and it will creates the same copy of a process Web

Server. Now there are two processes available here, and continue the execution at the same path (line) in both the processes.



Task Manager/Process Manager are used to know the processes details like pid and so on.

Process Table/Task Table consists the indexes of process-id's (each and every program has a separate identifier).

No two processes running currently have same process-id. The O/S is responsible to concurrently run the two processes.

In our code, one process *if(pid!=PARENT)* is executed and becomes 0 (zero) which is a child, and in other case it cannot be zero, it is non-zero that is a parent. In child process case, it executes the *provideService()* method and in the parent process it listens another call from the other client. Because *provideSevice()* method will be skipped in the parent process, so it receives the call.

In this code, both the *Listen()* and *provideService()* methods are concurrently executing. i.e., the same program services to the multiple clients at the same time.

In each and every process, there will be one point of control. Even though there are multiple processes, only one client is able to receive the services at a point of time.

Problems with this approach:
1. Writing this kind of programs is not that much easy because the separate memory is required for each and every process.
2. Creating a process at the O/S level is very expensive on time consuming using *fork()* method. If this is going on, at some point of time, there is very less memory to execute the program causes lower performance. This is a heavy weight component Technique.

Footprint: The total memory that is occupied by the program is known as "foot-print".

So, we can search for the light-weight component technique for our problem.

Light-weight Technique offered by the Windows NT O/S:

We can able to see multiple copies of Oracle while running on UNIX O/S with the same name.

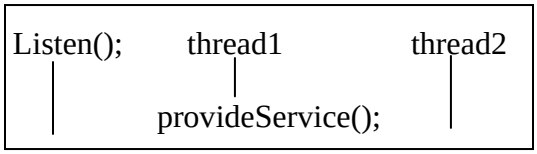
On Windows-NT, we have a single process and we have so many threads to perform multiple tasks concurrently, i.e. here we are using the feature of multi-threading. This technique is different that of the previous technique is known as "Light-weight component technique".

On Linux O/S give the command PS (Process Status), we can see the multiple copies of Java Programs that are executed concurrently.
$ PS Java

Code for Windows-NT O/S:

```
main() {
  while(11) {
    Listen();
    Startthread(fnptrofprovideService);       // fnptr --> function pointer
  }
 }
```

Startthread method will takes the address of a function pointer.



Web Server

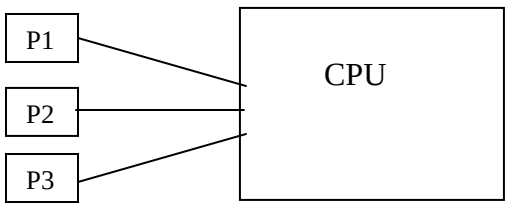
When a call is received, it Starts executing the *Listen()* method, and at some point *Startthread* creates a thread, there it Starts executing the p*rovideService()* method. This is executed in concurrent with the *main()* method. Here the separate copy is not created, that's why this is called as a "Light-Weight component technique".

Thread: Thread is a short form of thread of control.

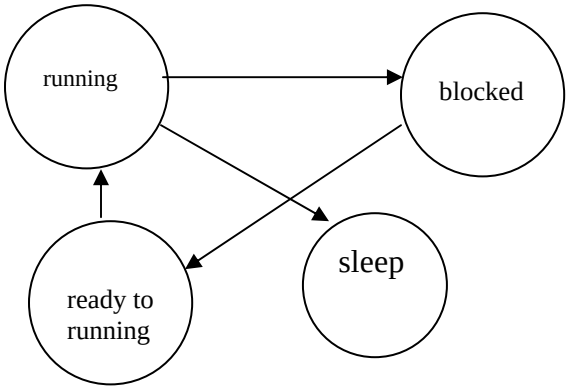
* In each and every thread, the *provideSerice()* method is executed and the thread will be automatically terminates after the method is completed.

In this method also, there are some disadvantages, but the total amount of memory that is used to create a separate (copy of) process is much more than creating the threads.

How the O/S gives an illusion to the developer while running multiple processes:
The most important information of the process is State.
At a point of time only one process can be executed because the CPU is capable of running only one process at a point of time.



If the process P1 is continuously utilizing the CPU, then P2 and P3 are not able get the chance for execution. If the CPU has given 10ms for each and every process, in this case after 10 ms, P2 gets the chance to run on the CPU. In that case P1 forcefully moved from running to the block state, and P3 will be in the ready to running State, this is will be continued P2 is in running State. When a particular process is blocked in that case we can't give or execute input/output operations on that particular process.
Ex: Input from the Keyboard, display to the console etc.
Task is a Process here.



All the processes are shared the time consumed on CPU and providing the illusion to the developer that they are running concurrently. Every process gets its own execution time on CPU.

Scheduler: Piece of S/W that is responsible for allocating the CPU time to each and every process is known as "Scheduler".

Preemptive (cutting shot) multiple task technique is forcibly used on the CPU, to perform this type of operations. The change of process from running state to ready to running state is known as preemptive multiple task technique

The way in which the technique used in windows-3.1 is different that of the technique used in windows-NT O/S.

In windows-3.1, it allows multiple processes, known as "co-operative multi tasking". i.e., in the process itself, the developer should write the code such that I want to yield for some time, to give a chance to the other process.

But in windows-NT O/S takes the decision of sharing the CPU time for every process. In windows-NT O/S it provides multi-threading. Windows-NT O/S is responsible to schedule the threads and processes.

**A thread at Java level is a thread at O/S level in Windows-NT O/S, but every process in UNIX O/S.

Some O/S's does not directly support multiple threads in the same process. On such O/S's we can simulate the threads by using multiple processes.

How to write the Threads:

1. First identify what has to be done in the thread. Create a sub-class of a Thread class and over-ride the method *run().*

```
class Thr1 extends Thread{
  public void run(){
    for(int i=0;i<50;i++)
      System.out.println("=======" + i);
  }
 }
```

2. start() method is responsible to create a Thread and to run the run() method.
When a Java program is compiled JVM is responsible to create a main thread, and the thread will be created when *Thr1.start()* method is executed, and this will runs at the main method of a Java program. JVM is responsible for using the O/S facility and giving the time to each and every thread.

The way in which the JVM implemented by SunSoft on windows-NT O/S (creates one thread at O/S level, to represent the thread that we created at the Java level) have one to one map to the java and the O/S level.

But in JRocket, even though we have created multiple threads at Java level, but it is mapped to a single thread at the O/S Level, of the JVM implemented in the JRocket. The big advantage in this is, the Switching time is very less than that of the JVM implemented by the JavaSoft. But, this design is very difficult. And it is very costly (around $1000 per CPU).

Blackdown JVM -- 1:1 process on LINUX.
Blackdown has provided two different implementations of JVM on LINUX O/S. Where we run the JVM with default setup, the JVM creates one process per thread. This implementation by blackdown is known as "Native Threading".

The other implementation is called as "Green Threading". (Clean implementation of threading). In this Schema, JVM uses its own code to schedule the threads without using the O/S Service.

```
$Java -DTHREADS_FLAG=green filename
```

and run the java program in a new window, then you can able to see the "green threading" effect.

How to analyze the multi-threaded program:

When we run a Java program, JVM creates one thread to run main method. While creating the thread, try to write the code as part of the main.

Every thread has its own section of code for execution. Thread is represented by a box.

```
Thread T1 = New MyThr();
Thread T2 = New MyThr();
    T1.start();
    T2.start();
```

Thread.setName("Thr-1")-- Used to set the names for a thread or box.
If we don't set the names, by default JVM gives the box names like Thr-0, Thr-1...etc.

Execute the run method as a part of the current thread.

Thr-0	Thr-1
for(i=0;i<50;i++)	*for(i=0;i<50;i++)*
System.out.println(Thread.currentThread);	*System.out.println("Student");*
T1.start();	*T2.start();*

Every thread has its own method stack area and whenever the method is executed of that particular method; all the local variables are stored in method stack of the particular thread.

When *T2.start()* is executed, T1 box will be removed.

Even though our threads are using I/O operations like *System.out.print* and Inputs from the keyboard etc, it may or may not be blocked the current thread, because of the features provided at the O/S levels. According to the time given to each thread it executes and after the time completed. The chance will be given to the second thread and so on.

According to the JavaSoft people, do not use *Thread.suspend(), Thread.resume() and Thread.stop()* methods according to them, because these methods are developed for Java-1.0.

Whenever we analyze a multi-threaded program, we need to keep the following points in mind:

1. It is not always possible to predict when the thread Switching happens.
2. In case of methods *Thread.Yield()* and *Thread.Sleep()*, we can assume that the thread is blocked.

Examples:-

Thr-0	*Thr-1*
for(i=0;i<50;i++)	*for(i=0;i<50;i++)*
i = 1+k+j+f+d	*a = b+c;*
.....	.....

JVM cannot execute the operations statement by statement in multi-threading programs. It simply performs the operations on the byte code developed after compiling the program.

In the above example, the threaded switching can be occurred only when the operation is completely executed i.e. 1+k (or) k+j (or) j+f (or) f+d (or) i= , in integer arithmetic operations. It cannot switch while these operations are going on.
Floating point arithmetic operations are thread unsafe (f1*f2), because some of the micro processors does not support the floating point operations.

According to the JVM Specification of a JVM implementation need not ensure thread safety of floating point operations. i.e., a JVM may switch from one thread to another thread when it is performing floating point operations.
*Thread.enumerate(array as parameter): G*ives the number of threads currently as part of your JVM/currently executing.
Thread.currentThread(): Runs the code that is as part of the current thread.

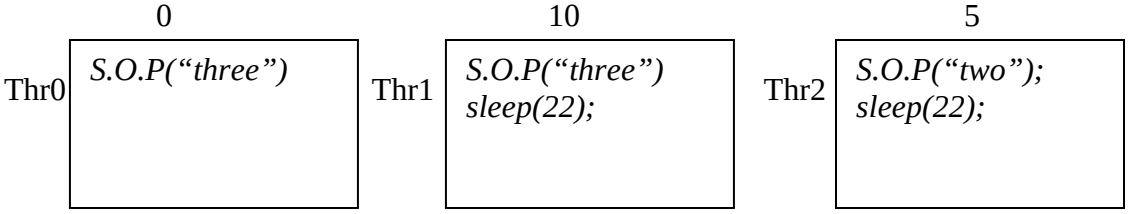
Multiple threads are part of the other main thread group. Whenever a thread is created a default priority of 5 will be assigned to the thread. The priority of the threads can be set by the *Thread.setPriority()* method.

Name of the thread, priority, Thread group

th2.setPriority(Thread.MAX_PRIORITY): 10
MIN_PRIORITY : 0
NORM_PRIORITY : 5
th2.start();
Priority issue is used to give a chance for execution.

Whenever the JVM schedule the threads, it's going to check the priorities and gives the CPU time to a thread which is having highest priority.

sleep(22): It may be greater than 22ms the thread will sleep .i.e., the thread is not going to sleep not exactly 22ms.



Whenever multiple threads with the same priority are there in the JVM, then JVM is responsible for giving the time to the thread which has entered in to ready to run state first?

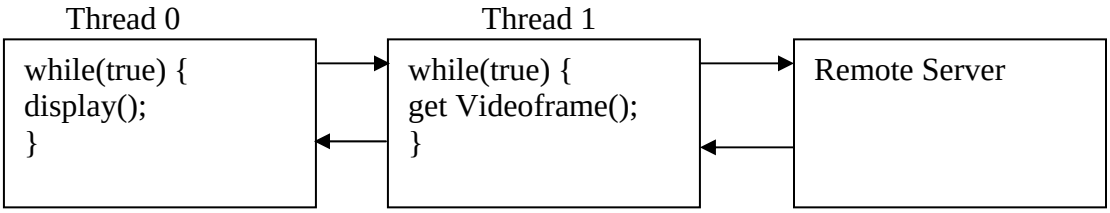
The output of multi-threaded programs should be depending upon the m/c configurations. We can't analyze the output of this type of programs.

```
/* Program to create thread by sub classing Thread class
   class Thread1
    {
      public static void main(String args[])
       {
            Thread th=new MyThread();
            Thread tharr[]=new MyThread1[1000];
            System.Out.println(Thread.currentThread());
            th.setName("FirstThread");
            th.setPriority(2);
            th.start();
            int not=Thread.activateCount();
            System.out.println("Active thread count: "+not);
            Thread tha[]=new Thread[not];
            Thread.enumerate(tha);
            for(int i=0; i<not; i++)
              System.out.println(tha[i]);
         }
     }

   class  MyThread1 extends Thread
    {
      public void run()
       {
          for(int i=0; i<10;i++)
            System.out.println(Thread.currentThread());
        }
    }
```

output:

```
Thread[main,5,main]
 active thread count2
Thread[main,5,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
Thread[Thread one,2,main]
```

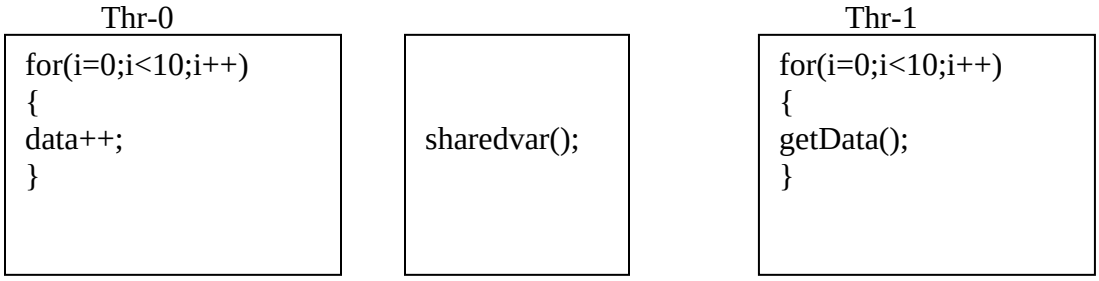
Imp points while writing the multi-threaded programming:
1. What the thread to do
2. The input of one thread is an output of the other.
3. Whether there are any shared resources in the multiple threads.



If we are writing a program to display video frames on the monitor by retrieving them from a remote server, here thread-0 is to display the video frame, thread-1 is to display the video frames from a remote server and to give them to the thread-0.

GENERAL PROBELMS THAT OCCURS WITH THIS APPROACH:
1. If the consumer is faster than the producer. Unnecessarily we are displaying the same frame again and again when the consumer is faster than producer.
2. If the producer is faster than consumer, it goes on retrieving the frames, even though the consumer is not consuming them.

SOLUTION FOR THIS:
Producer should wait till the consumer consumes the produced data item or consumer should wait until the producer produces the next data item.



It may work properly in some machines, but not in all machines. It will depend upon the load on that particular machine.

The sequence in which you write the code will affect the output.

Efficiency will come down because forcefully we are keeping the thread idle in both the consumer and the producer.

If we are using the high configuration machine, it performs the task very fast. Suppose, we kept the sleep time 10ms (*sleep(10);*) but it will perform the operation in 1ms. Then the machine kept idle for 9+9 ms time. Here we are under-utilizing the CPU time.

In a low configuration machine, we are testing the condition so many times. By checking

the condition again and again we are over-utilizing the CPU time.

We have some automated mechanism, directly build into the system that the producer and consumers should intimate one another to balance the under and over utilization of CPU (i.e. until the task completed by one the other one should wait).

Shows how to synchronize and problem presented in WhySync.java program

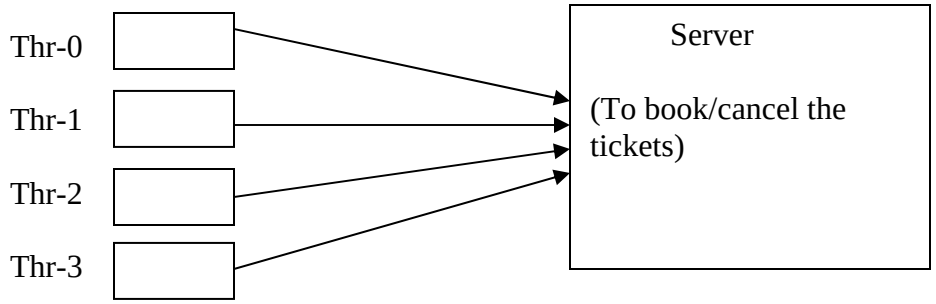
```
class Sync{
  static public void main(String[] args)throws Exception{
    SReservation reservation = new SReservation(10); // ten tickets are available for us
    Thread banglore = new SThread(reservation);
    Thread banglore1 = new SThread(reservation);
    Thread hyderabad = new SThread(reservation);
    Thread hyderabad1 = new SThread(reservation);
    banglore.setName("blr");
    hyderabad.setName("hyd");

          // let us reserve five tickets from banglore
          banglore.start();
          banglore1.start();
          // let us reserve five tickets from hyderabad
          hyderabad.start();
          // let us reserve another five tickets from hyderabad
          // try using hyderabad.start();
          hyderabad1.start();
  }
}

class SThread  extends Thread{
 SReservation r;
 public SThread(SReservation r){
          this.r = r;
 }
 public void run(){
          System.out.println(Thread.currentThread()+" Tickets are booked " +
r.reserveTicket(5));
 }
}

class SReservation{
          int not;
          public SReservation (int not){
                    this.not = not;
          }
```

```
// we will lock this so that no one can perform this operation till this is completely
finished by a thread

synchronized public boolean reserveTicket(int n){
        // get the not available from database
                if (not < n)
                        return (false);
                try{
                Thread.currentThread().sleep(20); // we are doing some activity here to
simulate most common
                }catch (Exception e){}
                //problem with threads
                not = not - n;
                System.out.println("no of tickets currently available = "  + not);
                return true;
        }
// we are not going to call this at all !!!
        public void cancelTicket(int n){
                not = not + n;
        }
}
```

Output

```
no of tickets currently available = 5
Thread[blr,5,main] Tickets are booked true
no of tickets currently available = 0
Thread[Thread-1,5,main] Tickets are booked true
Thread[hyd,5,main] Tickets are booked false
Thread[Thread-3,5,main] Tickets are booked false
```

RAILWAY RESERVATION SYSTEM: **not =10**




```
 class reservation
  {
    int not;                                    // not – number of tickets
```

```
    booktickets()                    // nor – number of requested tickets
    {
      not=not-nor;
    }
    canceltickets()
    { }
}
```

First create an object, based on the reservation class. If four concurrent requests send to sever to book five tickets each here all the 4 requests are pointing to a single object (reservation). The main point here is to access the resources from multiple threads.

Thread.currentThread().sleep(20);

This statement used to simulate the situation i.e. to give a chance to the other threads for execution. In real time applications, this is not required.

NOTE: While accessing the resources from multiple threads, you have to take some precautions to get the required output.

```
if(not<nor)
    return false;
else
     not=not-nor;
     return true;
```

In our problem, apart from the four requests, in the first thread the condition (not<nor) is checked and after coming to switching occurs to thr-1, and there is also it has not performed any task and switched to the thr-2. In the thr-2 and thr-3, some operations are performed. After this again switched to thr-0 and performed the remaining task. Then it gives wrong information there are tickets are available and returns true. But actually tickets are reserved for the requests of thr-2, thr-3 and there are no tickets for thr-0, thr-1.

This problem can be solved by holding the lock from a thread. If one thread is holding the lock on an object, no one else cannot lock the particular object until the first thread performs the operation and released the lock.

In java language only one lock is available for an object. At point of time only one thread can hold the lock. But multiple threads can request to obtain the lock.
In our example, none of the threads locks the object that is why our system failed to give the accurate results/correct output.

How the lock can be held by a thread:-

By using the '*Synchronized*' method. Whenever a thread executes a synchronized method

on an object, the JVM has to allocate the lock on that object on behalf of the thread. If it is unable to acquire a lock on the object, then the JVM will block the thread.
In this case even the switching occurs; the other thread cannot proceed because it requires a lock. Until unless the first thread completes the operation and release the lock, to get the second thread for execution.

In java language,
Entering to the monitor -- acquires a lock.
Leaving the monitor -- Leaving the lock.

Even though you have written all the methods as synchronized methods, nothing can be carried out concurrently i.e. if the access to be given for multiple threads, then avoid using synchronized methods.

Thread Unsafe: In some cases, a vendor of a class may say that their class is not thread safe or thread unsafe. So, when a class is thread unsafe, we should not access the objects based on this class from multiple threads.

JFC & AWT classes are not thread safe. Since these classes are very complex, JavaSoft has not designed them as thread safe classes. Because of the complexity in design of these classes, writing the thread safe classes is very difficult. We may lose control over the code if the complexity of the thread safe class design increases.

/* Program to show how to solve producer consumer

```
class PCPSolv1{
  static public void main(String[] args)throws Exception{
        sharedvar v = new sharedvar();
    Thread producer = new SProducer(v);
    Thread consumer = new SConsumer(v);
        //producer.setPriority(Thread.MIN_PRIORITY);
        producer.start();
        consumer.start();
  }
}
class SProducer  extends Thread{
sharedvar v;
 public SProducer(sharedvar s){
        v = s;
 }
 public void run(){
        for(int i= 0;i <10; i++){
                // check to see if the consumer has consumed prev data item or not
        try{
        while ( v.consumed == false)
                        sleep(10);
```

```
        }catch(Exception e){}
                v.setData((v.getData())+1);
                System.out.println("Producer ");
                v.consumed = false;    }   }
  public int getData(){
        return v.getData();
  }
}

class SConsumer extends Thread{
sharedvar v;
 public SConsumer(sharedvar s){
        v = s;
 }
 public void run(){
        for(int i = 0;i <10;i++){
                try{
                //        sleep(3);
                }catch(Exception e){}
        try{
        while ( v.consumed == true)
                        sleep(10);
        }catch(Exception e){}
                System.out.println("Consumed " + v.getData());
                v.consumed = true;
        }  } }
// this class object is used as a shared data object
class sharedvar{
        public boolean consumed; // let us have a lock here,to reduce code we declared
this as public var
        int data;
        public void setData(int data){
                this.consumed = true; //set this to true so that the producer
                // can start producing the items
                this.data = data;
        }
        public int getData(){
                return (data);
        }
}
```

Output:-

```
Consumed 0
Producer
Consumed 1
Producer
```

…….
Consumed 9
Producer

DAEMON PROCESS: Which runs at the back ground, to help the other programs/processes, this is in UNIX OS. In windows-NT OS, this process is known as windows-NT service.

They want to differentiate with the other process, that's why the Java people called this as "Daemon process".

Daemon thread: is a thread which helps the other threads. This is very rarely used.

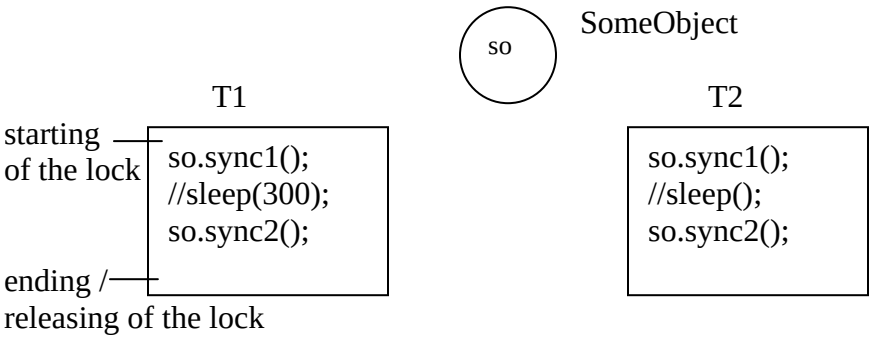
t1.setDaemon(True);
t1.start();

JVM treats these threads differently from the other threads.

When all the non-daemon threads exits, then the JVM automatically terminates the program even through the daemon thread is exists/running.
Ex: Garbage collector executes as a part of daemon thread.

In real time when you are developing a very big project with server-side programs and you are creating too much number of objects. The server will find out which objects are not used by the client (The objects that are in-active) and return to the disk, to clean up the memory space. This can be done by writing the daemon threads.

In order to create a thread as a daemon thread, we have to use setDaemon() method before starting the thread.
Main thread should be a non-thread; otherwise JVM simply terminates your program.



/* shows what happens when more than one sync method is there in a class

```
class Sync1{
  static public void main(String[] args)throws Exception{
    SomeObject so = new SomeObject();
    SomeThread st1 = new SomeThread(so);
```

```
   st1.setName("Thread One");
   st1.start();
   SomeThread st2 = new SomeThread(so);
   st2.setName("Thread Two");
   st2.start();
  }
}

class SomeThread  extends Thread{
 SomeObject  so;
 public SomeThread(SomeObject so){
        this.so = so;
 }
 public void run(){
        so.sync1();
        try{
                sleep(300);// comment out this line and see the result
        }catch(Exception e){}
        so.sync2();
 }
 public String toString(){
        return(getName());
 }
}

class SomeObject{
        synchronized void sync1(){
                System.out.println(Thread.currentThread()+ "  in sync 1 method");
        }
        synchronized void sync2(){
                System.out.println(Thread.currentThread()+"  in sync 2 method");
        }
}
```

Output

```
Thread One  in sync 1 method
Thread Two  in sync 1 method
Thread One  in sync 2 method
Thread Two  in sync 2 method
```

If sleep(300); statement is commented, then you can't predict the output, because at any point of time switching from one thread to another thread can occur.

Java provides two different mechanisms of getting lock on the object.
1. By executing the synchronized method.

2. By using the synchronized blocks.

Synchronized block:-

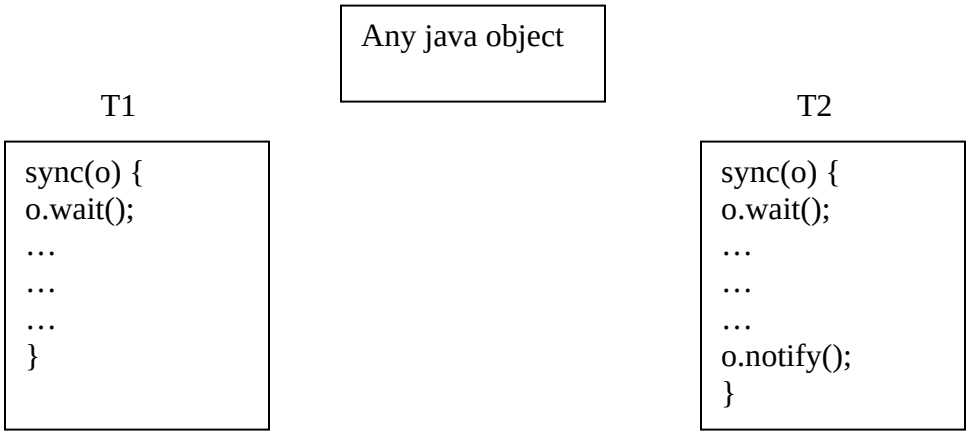
```
    public void run(){

                Synchronized(so){
                   so.sync1();
try{
   sleep(300);
 }catch(Exception e){}
 so.sync2();
 }//Block ends.
}
```

In order to carryout multiple actions on a single object without releasing the lock; we can use a synchronized block.

When a synchronized block is executed by the JVM, the JVM has to ensure that the thread gets, the lock on that object, and the lock on that object will be released after the execution of the synchronized block.

Wait & Notify Methods:-

When we write multi-threaded applications, in some cases we need to implement the technique Thread1 has to wait till Thead2 finishes some action.



o.wait()-- It releases the lock, but the JVM marks down that, it is waiting on the object o;

o.notify()-- JVM marks down to know that the other thread is ready for execution.

Thread1 woke up after the T2 completed its task and when *o.Notify* is executed, to

perform the task that is pending in T1, i.e., it comes to the ready to run state from the block state.
notifyAll()-- sequence of the switching cannot be predicted when this method is used.

Waiting in java language is the waiting for an execution of that particular thread.

/* This program shows how to use wait and notify

```
class WaitNotify{
  static public void main(String[] args)throws Exception{
    Thread t1 = new DataProducer();
    Thread t2 = new DataConsumer((DataProducer)t1);
    t1.start();
    t2.start();
  }
}

class DataProducer  extends Thread{

StringBuffer sb;
boolean dataprodover = false;

 public DataProducer(){
                System.out.println("In data Prod");
                sb = new StringBuffer("");
 }
 public void run(){
        for(int i= 0;i <10; i++){
        try{
                sb.append("."+i);
                sleep(100);
                System.out.println("appending");
        }catch(Exception e){System.out.println(e);}
        }
        dataprodover = true;
  }
  public StringBuffer getData(){
        return sb;
  }
  public boolean isDPOver(){
        return dataprodover;
  }
}

class DataConsumer extends Thread{
DataProducer dp;
```

```
public DataConsumer(DataProducer dp){
        this.dp = dp;}
 public void run(){
        try{// if data production is not over sleep for some time and check again
//System.out.println(dp.isAlive());
         //while(! dp.isAlive())
        //        wait(1); // let us wait till data producer starts
//System.out.println(dp.isAlive());
                while( ! dp.isDPOver())
                        sleep(10);
        }catch(Exception e){}
        System.out.println(dp.getData());
  }
}
```

output:-

```
In data Prod
appending
appending
appending
……..……
appending
.0.1.2.3.4.5.6.7.8.9
```

Threadgroup: Threads adding to an array/group is known as "Threadgroup". If you don't add any thread group, by default 'main' threadgroup will be created.

Java doesn't support multiple-implementation inheritance. In this case we can't extend this class to write the thread.

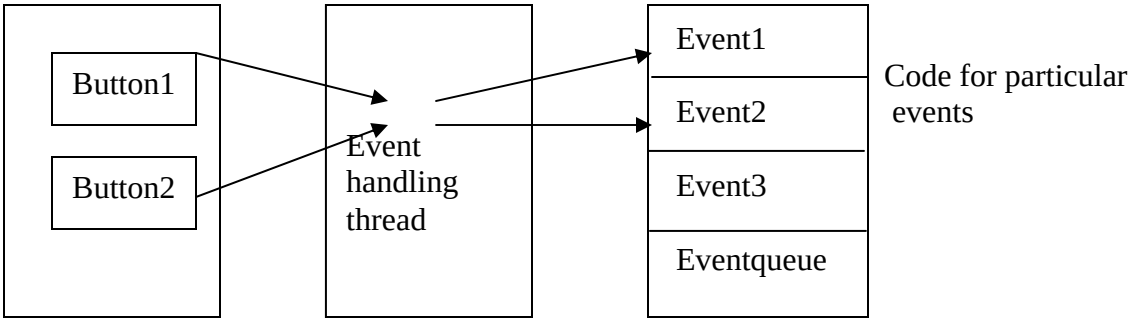
To do this JavaSoft has provided a very simple mechanism/technique by extending the *Runnable* class.

The thread class simply implements the *Runnable* class directly here, in the previous examples we are indirectly implementing the *Runnable* class.

One object constructor:-

Even though you have not written any threads in AWT/JFC programs, JVM internally creates some threads. This will depends upon the implementation of the JVM. In this one thread is responsible for handling the events. This thread is known as "Event handling Thread". But the JavaSoft people had given the name for this as: Event queue".

EventQueue consists of all the details of the events of that particular component| object.



When we click the Button1, it will calls the *processxxx* Event and this internally responsible to intimate the action Listeners of that particular component. This operation takes lot of time and the Button will be disabled until the task completed.

Since JVM/AWT objects are not thread safe, it is not advisable to manipulate the JFC objects from multiple threads concurrently. In some situations an event handler may take a lot of time, and this gives an impression to the user that the program is not responding. So we can use a very simple technique to solve this problem.
From the event handler, we create a thread which should takes care about a long-running operation (From this thread we can't access JFC component). At the end of this thread, we have to use *swingutilities.invokeLater();* which ensures that the remaining task will be carried out from EventQueue thread.

In *actionperformed()*,we have to write the thread and you can write a program to retrieve the list from a remote server in swingprobsolv example. But this is not recommended by the JavaSoft because you can't update the data (list) from this thread. The updation of the data can be done by the Event-handling (EventQueue) thread. The new thread is only responsible for picking up to data from a remote server.

* sleep(10000); used here to simulate the data that is taking lot of time.

swingutilities.invokeLater(runnable object);

This provides an implementation of run method.
Make sure that the run method will be executed from the EventQueue.
Listupdate(); This class provided by the developer, to provide the execution of run method from AWTEventQueue, and to update the list.
dtm.addElement(v.elementAt(i));

The data is stored in a vector, and we are adding the data to the list with the updated data.

/* Shows problem how to solve the problem of fetching bulk data or doing expensive operation in event handlers
*/

```
// import java.awt package to use awt classes
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

public class SwingProbSolv{
        public static void main(String args[]) throws Exception{
                SwingProb jprob = new SwingProb("this shows problem with swing");
                jprob.show();
    }
}

class SwingProb extends JFrame implements ActionListener{
 JButton jb;
 JList jl;
 Container cp;
 DefaultListModel  dlm;
 public SwingProb(String s){
        super(s);
        dlm = new DefaultListModel();
        cp = getContentPane();
        jb = new JButton("Fecth Data");
        cp.add(jb);
        dlm.addElement("one");
        dlm.addElement("two");

        jl = new JList(dlm);
        jl.setPreferredSize(new Dimension(200,200));
        JScrollPane scrollPane = new JScrollPane(jl);
        cp.add(scrollPane);
        cp.setLayout(new FlowLayout());
        jb.addActionListener(this);
        pack();
 }
public void actionPerformed(ActionEvent ae){
        // now spin off data thread and return immediately
DataThread dt = new DataThread(dlm);
        dt.start();
}
}
```

```
class DataThread extends Thread{
 DefaultListModel  dlm;
 Vector v;
        public DataThread( DefaultListModel  dlm){
                this.dlm = dlm;
        v = new Vector();
        }
        public void run(){
       try{
                System.out.println("fetching data from data base");
                sleep(10000);
        }catch(Exception e){}
                for(int i = 0; i <20;i++)
                        v.addElement(i +" element added");
    ListUpdater lu = new  ListUpdater(v,dlm);
                System.out.println("Placing runnable object on event dispatch thread");
        SwingUtilities.invokeLater(lu);
 }
}

class ListUpdater extends Thread{
 DefaultListModel  dlm;
 Vector v;
  public ListUpdater(Vector v,DefaultListModel  dlm){
        this.v = v;
        this.dlm = dlm;
  }
 public void run(){
// update list from fetched data
   for(int i= 0;i <20;i++) { // we have to use methods on vectors
// here than using i <20 here
                dlm.addElement(v.elementAt(i));
   }
// this will run on event dispatch thread.
 }
}
```

/* show why we have to use thread groups*/

```
class WhyTGroups{
  static public void main(String[] args)throws Exception{
        ThreadGroup tg = new ThreadGroup("Reservation Group");
        Thread res = new ResGroup(tg);       // add this thread to new group so that we
                        // can control all the threads in this group using a single method
     res.start();
  } }
```

```
class ResGroup extends Thread{

public ResGroup(ThreadGroup group){
        super(group,"reservsation group");
    }
Reservation r[] = new Reservation[30];
    // this class creates a thread when ever a request for reserving a ticket is there
    // may be more no of threads at a single point of time.
    public void run(){
    for(int i = 0;i < 30;i++){
        try {
            System.out.println("Reservation group started");
            sleep(100);
            r[i] = new Reservation();
// u can use this method to suspend the execution of all the threads, we can have control
over all the threads of the group like this dangerous

getThreadGroup().suspend(); // stop the execution of threads for a while
r[i].start();
getThreadGroup().resume(); // resume thread execution
    }catch(Exception e){}
    }
    }
}

class Reservation extends Thread{
    public void run(){
        try {
        System.out.println(" I am reservation thread and my priority is set to "+
getPriority());
            sleep(300);
        }catch(Exception e){}
    }
}
```

Output

```
Reservation group started
Reservation group started
 I am reservation thread and my priority is set to 5
Reservation group started
 I am reservation thread and my priority is set to 5
Reservation group started
 I am reservation thread and my priority is set to 5
……
…….
……..
```