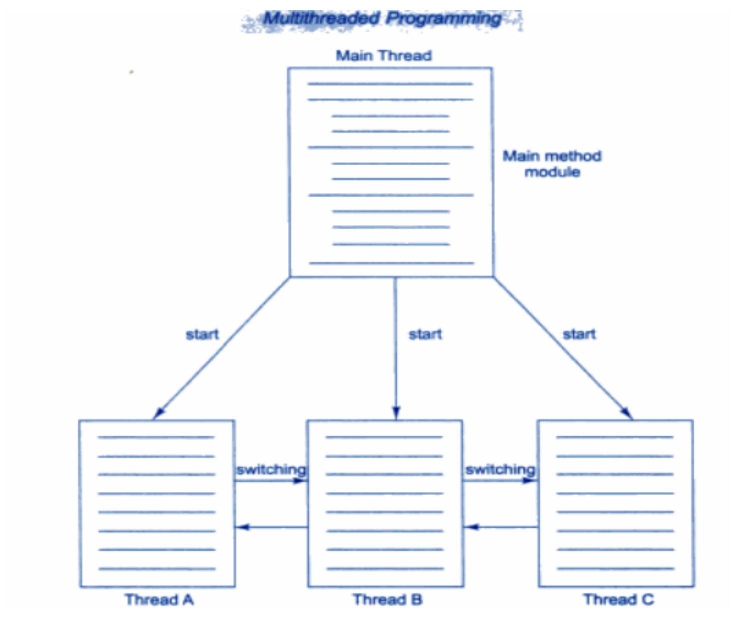
Syllabus:

Multithreading: The Java Thread Model, The Main Method, Creating a Thread, Creating Multiple Thread, Using isAlive() and join(),Thread priorities,Synchronization,Interthread Communication,Suspending,Resuming and Stopping Threads, Using Multithreading

Module 3: Multithreading and Event handling

1. Multithreading programming :

- Multithreading is a conceptual programming paradigm where a program is divided into two or more subprogram, which can be implemented at the same time in parallel.
- This is something similar to dividing a task into subtask and assigning them to processor for execution independently and simultaneously.
- Multithreading is a specialized form of multitasking. In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.
- In a *thread-based* multitasking environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously.



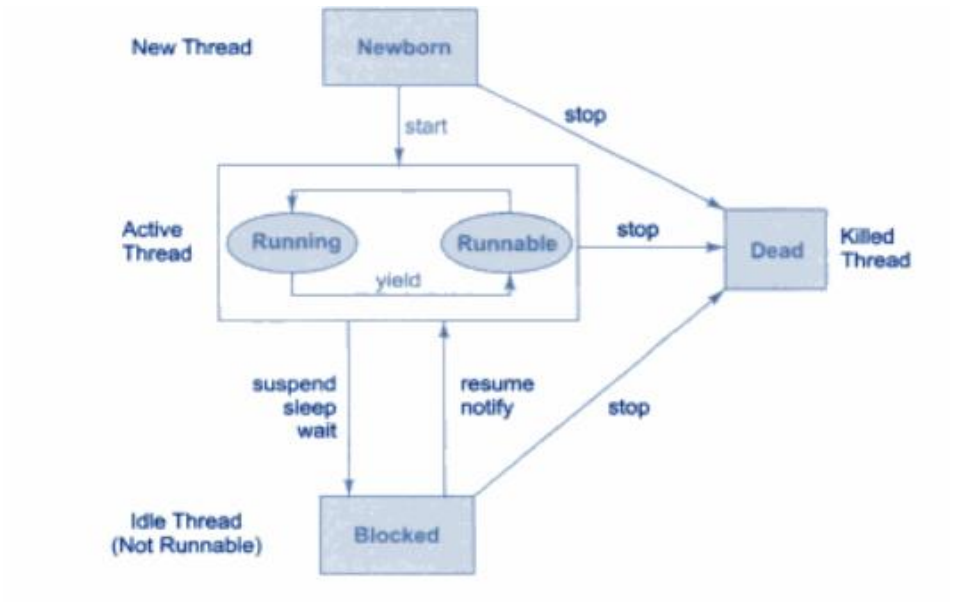
2 .**Thread:**

- The small unit of program or sub module is called as thread.
- Each thread defines a separate path of execution
- Threads that do things like memory management and signal handling but from the application programmer's point of view, you start with just one thread, called the main thread.
- Main thread has the ability to create additional threads.

3. Life cycle of thread:

Thread can enter into different state during life of thread, different stages in thread are as follows:

- New born
- Runnable
- Running
- Dead
- Blocked



New Born state:

- When we create the thread class object, then thread is said to be born and move to new born state. But still thread is not under running state.

- Thread can be moved to either one of two state as follows:
- Thread can move from born state to dead state when we invoke stop() method.
- Thread can move from born state to runnable state when we invoke start() method.

Runnable state:

- Runnable state means thread is ready for execution and waiting for the processor to free.
- All thread are joined in queue and waiting for execution.
- If all the threads have equal priority, Then they are given a time slot for execution in round robin fashion ie. FCFS fashion.

Running state:

Running state means, Thread is under execution. Thread will run until its relinquish control on its own or its is preempted by the higher priority thread.

Blocked state:

- Thread is said to be blocked when it is pre-empted from entering into runnable state and subsequently the running state.
- This can be happen when thread is suspend, wait, sleep in order to satisfy certain requirements.

Dead State:

- Thread can be killing as soon as its born state by calling stop() method.
- Thread will automatically kill, as soon as its completed the operation

3.Creating thread:

Thread can be created in two ways:

- By creating a thread class(**Extending Thread class**)
- By converting a class to a Thread Class(**Implementing Runnable interface**)

1.By Creating a Thread class(Extending Thread class):

- Declare a class by extending Thread Class
- implement the run() method,Where run() method is the override method in order to implement the actual code to be executed by Thread.
- Create a thread object and call a start method to initiate the thread execution

Syntax:

```
class class_name extends Thread
{
public void run()
{
/*Implement actual code*/
}
public static void main(String ar[])
{
class_name object=new class_name();
object.start();
}
}
```

Program:

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

```
for(int i=0;i<10;i++)
System.out.println("class B="+i);
}
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
a1.start();
b1.start();
}
}
```

2.By converting a class to a Threadable Class(Implementing Runnable interface)

- Runnable interface declare the run() method that is required for implementing thread in our program. The following are the steps are taken to implement the runnable interface.
- Declare the class by implementing Runnable interface
- Implement the run() method.
- Crteate a thred by defining an object that is instantiated from this Runnabel class as the target of that thread
- Call the start() method to run the thread.

Syntax:

```
class Class_Name implements Runnable
{
public void run()
{
/* Implements operation */
}
public static void main(String ar[])
{
Class_Name object=new Class_Name();
Thread object1=new Thread(object);
object1.start();
}
}
```

Program:

```
class A implements Runnable
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println("Class A="+i);
}
}
class B implements Runnable
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println("class B="+i);
}
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
Thread t1=new Thread(a1);
Thread t2=new Thread(b1);
t1.start();
t2.start();
}
}
```

5. Threads methods

The different threads methods are as follows:

Yield(),stop(),suspend(),resume(),wait(),notify(),notifyall().

1.yield()

Calling **yield()** will move the current thread from running to runnable, to give other threads a chance to execute. However the scheduler may still bring the same thread back to running when processor is free.

Program:

```
class A extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) yield();
}
}
}
class B
{
public static void main(String ar[])
{
A a1=new A();
a1.start();
}
}
```

Stop():

When stop() is called then processor will kill thread permanently.It means thread move to dead state.

```
class A extends Thread
{

public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) stop();
}
}
}

class B
{
public static void main(String ar[])
{
A a1=new A();
a1.start();
}
}
```

Sleep():

- When sleep() is called then processor will stop the execution of thread for the specified amount of time from the execution.
- This static **sleep()** method causes the thread to suspend execution for a given time. The sleep method has two overloaded versions:
  - static void sleep (long milliseconds) throws Interrupted Exception
  - static void sleep (long milliseconds, int nanoseconds) throws Interrupted Exception

Suspend():

- When suspend() is called then processor Sends the calling thread into block state.
- Using resume() method its bring the thread back from block state to running state.

Program for sleep and suspend resume methods:

```
class A extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) try { sleep(100);} catch(Exception e){ s.o.p(e) resume();}
}
}
}
class B extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) suspend();
}
}
}
class Mainclass
{
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
a1.start();
b1.start();
}
}
```

Thread Priority:

- Each thread assigned a priority,which effects the order in which it is scheduled for running.
- Thread of same priority are given equal treatment by the java scheduler and there for they share the processor on FCFS basis
- Java permits us to set the priority of the thread using setPriority()methods.

  Final void setPriority(int level)
- Where level specify the new priority setting for the calling thread. Level is the integer constant as follows:

  MAX_PRIORITY

  MIN_PRIORITY

  NORM_PRIORITY
- The MAX_priority value is 10,MIN_PRIORITY values is 1 And NORM_PRIORITY is the default priority whose value is 5.
- We can also obtain the current priority setting value by calling getPriority() method of thread.

  Final int getPriority()

Program:

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

```
class B extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
s.o.p("Class b thread="+i);
}
}
class MainThread
{
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
a1.setPriority(Thread.MAX_PRIORITY);
b1.setPriority(Thread.MIN_PRIORITY);
a1.start();
b1.start();
b1.setPriority(a1.getPriority()+10);
}
}
```

Synchronization:

- When two or more thread needs access to the shared resource, they need some way to ensure that the resource will be used by only one thread at a time.
- The process by which this is achieved is called synchronization.
- Key to synchronized is the concepts of monitor or semaphores. A monitor is an object that is used as mutually exclusive lock or mutex. Only one thread can own a monitor at a given time. When one thread acquires a lock it is said to have entered the monitor.

- All other thread attempting to enter the locked monitor will be suspended until the first thread exits the monitor. These other thread are said to be waiting for monitor.
- This can be achieved by using keyword synchronized to method.

Syntax:

```
synchronized void method_name()
{  /* implementation or operation
}
```

Program:

```
class A
{
Synchronized void display()
{
for(int i=0;i<10;i++)
System.out.println("i="+i);
}
}
class B extends Thread
{
public void run()
{
A a1=new A();
System.out.println("class A thread");
For(int i=0;i<10;i++)
a1.display();
}
}
class C extends Thread
{
public void run()
{
A a1=new A();
Sysem.out.println("class B thread");
For(int i=0;i<10;i++)
a1.display();
}
}
class D
{
public static void main(String ar[])
{
B b1=new B();
C c1=new C();
b1.start();
c1.start();
}
}
```

Inter-thread communication:

- Inter-thread communication can be defined as exchange of message between two or more threads. The transfer of message takes place before or after changes of state of thread.
- The inter-thread communication can be achieved with the help of three methods as follows:

  Wait(),notify() notifyall()

  Wait()- tells the calling thread to give up the monitor and go to sleep mode until some of other thread enters the same monitor and call the notify() methods

  Notify()-wakes up a thread that called wait() method on the same object.

  Notifyall()-wakes up all thread that called wait() methods on the same object.
- Inter-thread communication can be implemented by using the key word as synchronized to methods.
- Different types of inter-thread communication example are as follows:

- Producer-consumer problem
- Reader –writer problem
- Bounded-Buffer problem(Also called as Producer-consumer problem)

Producer-consumer problem/bounded buffer problem:

- Producer thread goes on producing an item unless an until buffer is full.
- Producer thread check before producing an item weather buffer is full or not, if buffer is full producer wait producing an item unless and until consumer thread consume a item.
- Consumer thread goes on consuming an item which is produced by the producer. The consumer thread check weather buffer is empty before consuming. If buffer is empty consumer thread as to wait until producer has to produce an item.

Program:

```
class A
{
int stack[]=new int[10];
int top=-1;
Synchronized void produce(int item)
{
if(top==10)
try
{
 wait();
}
catch(Exception e)
{
System.out.println(e);
}
Satck[++top]=item;
notify();
}

Synchronized void consume()
{
if(top==-1)
try
{
 wait();
}
catch(Exception e)
{
System.out.println(e);
}
item=Satck[top++];
System.out.println("consumed item is"+item);
notify();
}
}
class Producer extends Thread
{
public void run()
{
A a1=new A();
for(int i=0;i<10;i++)
a1.produce(i);
}
}
class Consumer extends Thread
{
public void run()
{
A a1=new A();
for(int i=0 ;i<12;i++)
a1.consume();
}
```

```
}
class MainThread
{
public static void main(String ar[])
{
Produce p=new Produce();
Consume c=new Consume();
p.start();
c.start();
}
}
```

Reader-writer problem:

- Reader thread reading an item from the buffer, Where as writer thread writing an item to buffer.
- If reader is reading then writer has to wait unless and until reading is finish.
- While writing thread writing an content then no other thread read the content unless and until writing is over.
- This problem can be achieved using wait, notify and nitifyall method and using synchronized keyword to method.

isAlive() and join()

- The final **isAlive()** method returns true if the thread is still running or the Thread has not terminated.
- **final join()**
- The final **join()** method waits until thread on which it is called is terminated. For example, thread1.join() suspends the current thread until thread1 dies.
- The join() method can throw an Interrupted Exception if the current thread is interrupted by another thread.

Program:

```
 class A extends Thread
{

public void run()
{
for(int i=0;i<10;i++)
System.out.println("class A="+i);
}
}
Class B extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println("class B="+i);
}
}

class C
{
public static void main(String args[])
{
A a1=new A();
B b1=new B();
a1.start();
b1.start();
System.out.println(a1.isAlive());
System.out.println(b1.isAlive());
try
{
A1.join();
B1.join();
}
catch(Exception e)
{
System.out.println(e);
}
System.out.println("main thread dead");
}
}
```

Creating multiple thread:

- More than one thread can be created using single object of thread class. Where all the thread can execute parallel.

Program:

```
class A implements Runnable
{
A()
{
Thread t=new Thread(this);
t.start();
}
public void run()
{
```

```
for(int i=0;i<10;i++)

System.out.println(i);

}

}

class MainThread

{

public static void main(String args[])

{

A a1=new A();

A a2=new A();

}

}
```