# Daemon Threads:

**These threads are running internally to provided services to some other thread and it will be terminated along with the threads which are taking services.**

**Note: Garbage collector is treated as DAEMON THREAD as internally it is providing service to the JVM by destroying the objects whenever heap memory is full.**

**And also daemon threads are going to be stopped automatically when the threads which are taking services are going to be stopped.**

**To make a thread as daemon thread we have to use the following method.**

**public void setDaemon(boolean b)**

--> If 'b' value is true then thread will be daemon thread.

--> If 'b' value is false then thread will not be daemon thread.

**EX:** mt.setDaemon(true);

**NOTE: this method must be called before calling start() method,If we call after calling start() method then JVM will rise an exception called java.lang.IllegalThreadStateException.**

**To check whether a thread is daemon thread or not we have to use the following method.**

**public boolean isDaemon()**

**Coding Example:**

**class MyThread extends Thread**

**{**

**public void run()**

**{**

**while(true)**

**{**

**System.out.println(" USER THREAD ");**

**}**

**}**

**}**

**public class Test**

**{**

**public static void main(String args[])throws Exception**

**{**

**MyThread mt=new MyThread();**

**mt.setDaemon(true);**

**mt.start();**

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

**{**

**System.out.println("MAIN THREAD ");**

**}**

**System.out.println(mt.isDaemon());**

**}**

**}**

# Synchronization:

In java applications, if we execute more than one thread on a single data item then there may be a chance to get data inconsistency, it may generate wrong results in java applications.

In java applications, to provide data consistency in the above situation we have to use "Synchronization".

**"Synchronization" is a mechanism , it able to allow only one thread at a time , it will not allow more than one at a time, it able to allow other threads after completion of the present thread only.**

**In java applicatinos, synchronization is going on the basis of Locking mechanisms, If we send multiple threads at a time to synchronized area then Lock Manager will assign lock to a thread which is having highest priority, once a thread gets lock from Lock manager then that thread is eligible to enter in synchronized area, once a thread is available in synchronized area then Lock Manager will not assign lock to other threads, when Thread completes its execution in synchronized area then that thread has to submit lock back to Lock Manager, once Lock is given back to Lock manager then Lock Manager will assign that lock to another thread which is having next priority.**

**In java applications, to provide synchronization JAVA has provided a keyword in the form of "synchronized".**

**In java applications, we are able to achieve "synchronization" in the following two ways.**

1.synchronized method

2.synchronized block

1. **synchronized method:**

**It is a normal java method, it will allow only one thread at a time to execute instructions, it will not allow more than one thread at a time, it will allow other threads after completion of the present thread execution only.**

**EX:**

class A

{

synchronized void m1()

{

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

{

String thread\_Name=Thread.currentThread().getName();

System.out.println(thread\_Name);

}

}

}

class MyThread1 extends Thread

{

A a;

MyThread1(A a)

{

this.a=a;

}

public void run()

{

a.m1();

}

}

class MyThread2 extends Thread

{

A a;

MyThread2(A a)

{

this.a=a;

}

public void run()

{

a.m1();

}

}

class MyThread3 extends Thread

{

A a;

MyThread3(A a)

{

this.a=a;

}

public void run()

{

a.m1();

}

}

class Test

{

public static void main(String[] args) throws Exception

{

A a=new A();

MyThread1 mt1=new MyThread1(a);

MyThread2 mt2=new MyThread2(a);

MyThread3 mt3=new MyThread3(a);

mt1.setName("AAA");

mt2.setName("BBB");

mt3.setName("CCC");

mt1.start();

mt2.start();

mt3.start();

}

}

**Note: Thread scheduler running internally only knows which thread will enter into synchronized area first,second and last…..**

1. **In java applications, we have already synchronized methods to achieve synchronization then what is the requirement to use synchronized block?**

**Ans:**

**In java applications, if we use synchronized method to achieve synchronization then it will provide synchronization through out the method irrespective of the actual requirment. If we need synchronization upto a block inside the synchronized method then it will provide unneccessary sychronization for the remianing part of the method, it will increase execution time and it will reduce application performance.**

**In the above context, to provide synchronization upto the required part then we have to use synchronized block.**

**Synchronized block:**

**It is a set of instructions, it able to allow only one thread at a time to execute instructions, it will not allow more than one thread at a time, it will allow other threads after completion of the present thread execution only.**

**Syntax:**

synchronized(Object o)

{

-----

-----

}

**EX:**

class A

{

void m1()

{

String thread\_Name=Thread.currentThread().getName();

System.out.println("Before Synchronized Block :"+thread\_Name);

**synchronized(this)**

{

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

{

String thread\_Name1=Thread.currentThread().getName();

System.out.println("Inside Synchronized Block :"+thread\_Name1);

}

}

}

}

class MyThread1 extends Thread

{

A a;

MyThread1(A a)

{

this.a=a;

}

public void run()

{

a.m1();

}

}

class MyThread2 extends Thread

{

A a;

MyThread2(A a)

{

this.a=a;

}

public void run()

{

a.m1();

}

}

class MyThread3 extends Thread

{

A a;

MyThread3(A a)

{

this.a=a;

}

public void run()

{

a.m1();

}

}

class Test

{

public static void main(String[] args) throws Exception

{

A a=new A();

MyThread1 mt1=new MyThread1(a);

MyThread2 mt2=new MyThread2(a);

MyThread3 mt3=new MyThread3(a);

mt1.setName("AAA");

mt2.setName("BBB");

mt3.setName("CCC");

mt1.start();

mt2.start();

mt3.start();

}

}