

Chapter – 9

Multi - Threading



ASHOK IT

Learn Here.. Lead Anywhere..!!

Single tasking:

Single tasking means executing 1 task at a time. Here much of the processor time will be wasted

because waiting time very high and processor gives response late.

Eg: DOS OS

Multi-tasking:

Multi-tasking means executing multiple tasks at the same time simultaneously. Here processor time utilized in a proper manner and performance of the application by reducing waiting time and processor gives response faster.

Eg: Windows OS, Student can listen and can write at a time

We can achieve multitasking in following 2 ways

1. Processor based multi-tasking

-> Processor based multi-tasking means executing multiple tasks at the same time simultaneously where each task is independent of other tasks having separate memory and resources

-> Processor based multi-tasking is an operating system approach

Eg:

java program,

listen to music,

download songs,

copy softwares,

chating,

....

2. Thread Based multi tasking

-> Thread based multi tasking means executing different parts of the same program at the same time simultaneously where each task is independent sometimes or dependent sometimes of other tasks which are having common memory and resources.

-> Thread based multi tasking is programming approach

-> Each different of the program is called Trheads

Eg: games, web based apps,....

Thread vs Process :

Process : a program execution is often referred as process.

Thread : a Thread is subset(part) or sub process of the process.

(or)

A thread is flow of execution and for every thread separate independent job(task).

NOTE: weather it is process based or thread based the main objective multitasking is reduce responsive time of the system and improve the performance.

Advantage of Multithreading

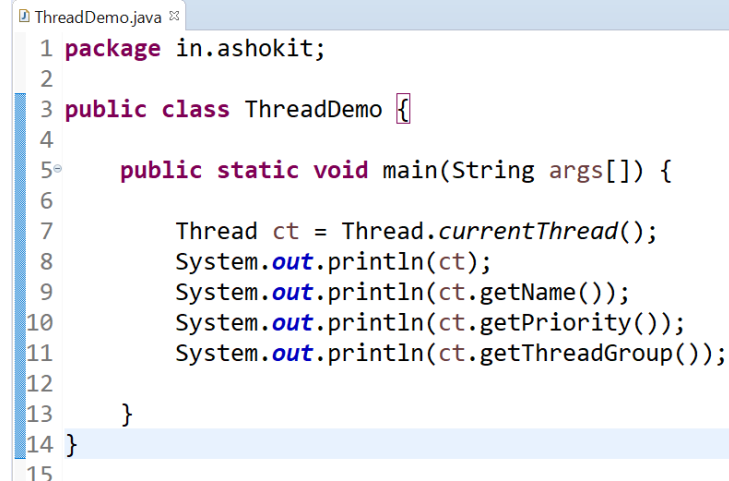
Multithreading reduces the CPU idle time that increase overall performance of the system. Since thread is lightweight process then it takes less memory and perform context switching as well that helps to share the memory and reduce time of switching between threads.

Main Thread

-> For every java program there will be a default thread by JVM which is called as Main Thread.

-> The entry point for Main Thread is main() method

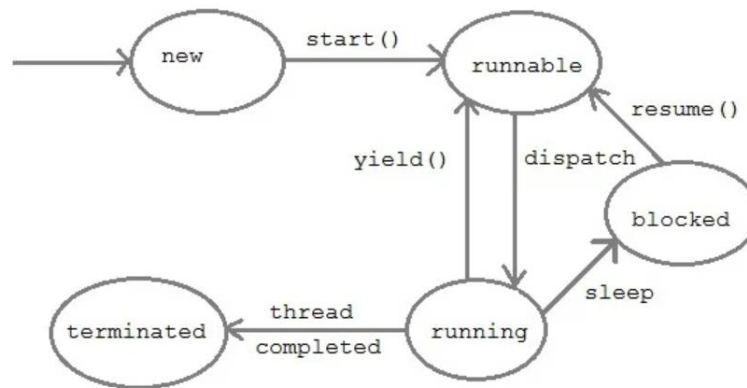
// wap to display the info of currently Running Thread



```
ThreadDemo.java
1 package in.ashokit;
2
3 public class ThreadDemo {
4
5     public static void main(String args[]) {
6
7         Thread ct = Thread.currentThread();
8         System.out.println(ct);
9         System.out.println(ct.getName());
10        System.out.println(ct.getPriority());
11        System.out.println(ct.getThreadGroup());
12
13    }
14 }
15
```

Life cycle of a Thread

Like process, thread have its life cycle that includes various phases like: new, running, terminated etc. we have described it using the below image.



New: A thread begins its life cycle in the new state. It remains in this state until the `start()` method is called on it.

Runnable: After invocation of `start ()` method on new thread, the thread becomes runnable.

Running: A thread is in running state if the thread scheduler has selected it.

Waiting: A thread is in waiting state if it waits for another thread to perform a task. In this stage the thread is still alive.

Terminated: A thread enters the terminated state when it completes its task.

What is Thread Priority?

Learn Here.. Lead Anywhere..!!

-> When we create a thread, we can assign its priority. We can set different priorities to different threads but it doesn't guarantee that a higher priority thread will execute first than a lower priority thread.

-> The thread scheduler is the part of Operating System implementation and when a thread is started, its execution is controlled by Thread Scheduler and JVM doesn't have any control over its execution.

-> Thread priorities are integer numbers.

-> Using thread priority, we can decide when we should switch from one thread in running state to another. This is the context switching process wherein we switch contexts of the threads.

Anytime a thread can voluntarily release its control over CPU. Then the thread with the highest priority can take over.

-> Similarly, a higher priority thread can preempt any other thread with lower priority.

-> Thread class provides a `setPriority ()` method that is used to set the priority for the thread.

-> We can also use constants MIN_PRIORITY, MAX_PRIORITY, or NORM_PRIORITY in the place of integers.

User defined threads

-> In java we can also create user defined Threads by using any one of the following 2 approaches.

1. By extending the Thread class
2. By implementing Runnable interface

Creating the user defined Thread by extending the Thread class

if we want to create the user defined Threads using this approach we have to follow following 5 steps

Step-1: creating a class that extends a Thread class

```
class MyThread extends Thread{  
}
```

Step-2: overriding the run() method

```
public void run(){  
//statements;  
}
```

Note: Here run() method is called entry-point for the user defined thread and all the job done by the user defined thread will be written here itself.

Step-3: creating an object for user defined Thread class

```
MyThread mt = new MyThread();
```

Step-4: attaching user defined thread with main ThreadGroup

```
Thread t = new Thread(mt);
```

Step-5: starting the execution of the Thread by calling start()

```
t.start();
```

-> When we call start() method the thread will be registered with ThreadScheduler and next will invoke the run() method.

```
MyThread.java
1 package in.ashokit;
2
3 class MyThread extends Thread {
4     public void run() {
5         for (int i = 1; i <= 10; i++) {
6             System.out.println("User Thread Value:" + i);
7         }
8     }
9
10    public static void main(String args[]) {
11        MyThread mt = new MyThread();
12        Thread t = new Thread(mt);
13        t.start();
14    }
15 }
```

Creating user defined Thread by implementing Runnable interface

-> if we want to create the user defined Threads using this approach, we have to follow following 5 steps

Step1: creating a class that implements Runnable interface

```
class MyThread implements Runnable {  
}
```

Step2: overriding the run() method

```
public void run(){  
    //statements;  
}
```

Step3: creating an object for user defined Thread class

```
MyThread mt = new MyThread();
```

Step4: attaching user defined Thread with main ThreadGroup

```
Thread t = new Thread(mt);
```

Step5: starting user defined Thread by calling start() method

```
t.start();
```

```
MyThread.java
1 package in.ashokit;
2
3 class MyThread implements Runnable {
4
5     public void run() {
6         for (int i = 1; i <= 10; i++) {
7             System.out.println("User Thread Value:" + i);
8         }
9     }
10
11     public static void main(String args[]) {
12         MyThread mt = new MyThread();
13         Thread t = new Thread(mt);
14         t.start();
15     }
16 }
```

Q) What is the difference between extending Thread class and implementing Runnable interface

-> If we create any thread by extending Thread class then we have no chance for extending from any other class.

-> But if we create any thread by implementing Runnable interface then we have a chance for extending from any one class.

-> It is always recommended to create the user defined threads by implementing Runnable interface only.

What is the difference between calling t.run() and t.start() ?

-> We can call run() method directly but no thread will be created / registered with Thread scheduler but run() method executes like a normal method by Main Thread.

-> But if we call start() method thread will be registered with thread scheduler and it calls run() method.

```
MyThread.java
1 package in.ashokit;
2
3 class MyThread implements Runnable {
4
5     public void run() {
6         Thread t = Thread.currentThread();
7         for (int i = 1; i <= 5; i++) {
8             System.out.println(t.getName()+ " Thread Value:" + i);
9         }
10    }
11
12    public static void main(String args[]) {
13        MyThread mt = new MyThread();
14        Thread t = new Thread(mt);
15        t.start();
16        //t.run();
17    }
18 }
```

Note: When we call start () method it is creating thread and printing output like below

Thread-0 Thread Value:1

Thread-0 Thread Value:2

Thread-0 Thread Value:3

Thread-0 Thread Value:4

Thread-0 Thread Value:5


```
MyThread.java
1 package in.ashokit;
2
3 class MyThread implements Runnable {
4
5     public void run() {
6         Thread t = Thread.currentThread();
7         for (int i = 1; i <= 5; i++) {
8             System.out.println(t.getName() + " Thread Value:" + i);
9         }
10    }
11
12    public static void main(String args[]) {
13        MyThread mt = new MyThread();
14        Thread t = new Thread(mt);
15        // t.start();
16        t.run();
17    }
18 }
```

Note: When we call run () method it is not creating thread and printing output like below with main thread

main Thread Value:1

main Thread Value:2

main Thread Value:3

main Thread Value:4

main Thread Value:5

 **ASHOK IT**
Learn Here.. Lead Anywhere..!!

Creating Multiple Threads

When we execute multiple threads at the same time simultaneously then we never get same output for every execution because Thread Scheduler will decide which thread should execute in next step.

```
MyThread.java
1 package in.ashokit;
2
3 class MyThread implements Runnable {
4
5     public void run() {
6         Thread t = Thread.currentThread();
7         for (int i = 1; i <= 5; i++) {
8             System.out.println(t.getName() + " Thread Value:" + i);
9         }
10    }
11
12    public static void main(String args[]) {
13        MyThread mt1 = new MyThread();
14        MyThread mt2 = new MyThread();
15        MyThread mt3 = new MyThread();
16
17        Thread t1 = new Thread(mt1);
18        Thread t2 = new Thread(mt2);
19        Thread t3 = new Thread(mt3);
20
21        t1.start();
22        t2.start();
23        t3.start();
24    }
25 }
```

Data in consistency problem

-> When we execute multiple Threads which are acting on same object at the same time simultaneously then there is chance of occurring data inconsistency problem in the application

-> Data inconsistency problem will occur because one thread is updating the value at the same time other thread is using old value.

-> To resolve this data inconsistency problem, we have to **synchronize** the object on which multiple Threads are acting.

Thread synchronization

-> Thread synchronization is a process of allowing only one thread to use the object when multiple

Threads are trying to use the particular object at the same time.

-> To achieve this Thread synchronization, we have to use a java keyword or modifier called "synchronized".

-> We can achieve Thread synchronization in following 2 ways

1. synchronized blocks

If we want to synchronize a group of statements then we have to go for synchronized blocks

syntax:

```
synchronized(object) {  
    //statements  
}
```

2. synchronized methods

If we want to synchronize all the statements of the particular method then we have to go for synchronized methods.

syntax:

```
synchronized returntype methodname(parameters){  
    //statements;  
}
```

Note: Thread synchronization concept is recommended to use only when we run multiple threads which are acting on same object otherwise this concept is not required.

Q) Which is more preferred - Synchronized method or Synchronized block

-> In Java, synchronized keyword causes a performance cost.

-> A synchronized method in Java is very slow and can degrade performance. So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

Deadlock

-> When we execute multiple Threads which are acting on same object that is synchronized at the same time simultaneously then there is another problem may occur called deadlock

-> Dead lock may occur if one thread holding resource1 and waiting for resource2 release by the Thread2, at the same time Thread2 is holding on resource2 and waiting for the resource1 released by the Thread1 in this case 2 Threads are continuously waiting and no thread will execute this situation is called as deadlock.

-> To resolve this deadlock situation there is no any concept in java, programmer only responsible for writing the proper logic to resolve the problem of deadlock.

// below is the java program which gives dead lock

package in.ashokit;

class MyThread {

 public static void main(String[] args) {

 final String resource1 = "Ashok IT";

 final String resource2 = "Java Training";

 // t1 tries to lock resource1 then resource2

 Thread t1 = new Thread() {

 public void run() {

 synchronized (resource1) {

 System.out.println("Thread 1: locked resource 1");

 try {

 Thread.sleep(100);

 } catch (Exception e) {

 }

 synchronized (resource2) {

 System.out.println("Thread 1: locked resource

2");

 }

 }

 }

 };

 // t2 tries to lock resource2 then resource1

 Thread t2 = new Thread() {

 public void run() {

 synchronized (resource2) {

 System.out.println("Thread 2: locked resource 2");

 try {

 Thread.sleep(100);

 } catch (Exception e) {

```

    }

    synchronized (resource1) {

        System.out.println("Thread 2: locked resource
1");

    }

}

}

};

t1.start();

t2.start();

}

}

```

Daemon Thread

-> Daemon threads is a low priority thread that provide supports to user threads. These threads can be user defined and system defined as well.

-> Garbage collection thread is one of the systems generated daemon thread that runs in background.

Learn Here.. Lead Anywhere..!!

-> When a JVM finds daemon threads it terminates the thread and then shutdown itself, it does not care Daemon thread whether it is running or not.

```

MyThread.java
1 package in.ashokit;
2
3 class MyThread extends Thread {
4
5     public void run() {
6         if (Thread.currentThread().isDaemon()) { // checking for daemon thread
7             System.out.println("daemon thread work");
8         } else {
9             System.out.println("user thread work");
10        }
11    }
12
13    public static void main(String[] args) {
14        MyThread t1 = new MyThread(); // creating thread
15        MyThread t2 = new MyThread();
16        MyThread t3 = new MyThread();
17
18        t1.setDaemon(true); // now t1 is daemon thread
19
20        t1.start();
21        t2.start();
22        t3.start();
23    }
24 }

```

Methods of Object class which are related to threads

1. wait(): : This method used to make the particular Thread wait until it gets a notification.

2. notify(): This method used to send the notification to one of the waiting thread so that thread enter into running state and execute the remaining task.

3. notifyAll(): This method used to send the notification to all the waiting threads so that all thread enter into running state and execute simultaneously.

-> All these 3 methods are available in Object class which is super most class so that we can access all these 3 methods in any class directly without any reference.

-> These methods are mainly used to perform Inner thread communication

-> By using these methods we can resolve the problems like Producer-Consumer problem, Reader-Writer problem, ...

```
MyThread.java
2 class MyThread extends Thread {
3     int total;
4     public void run() {
5         for (int i = 1; i <= 1000; i++) {
6             total = total + i;
7             if (total > 10000) {
8                 synchronized (this) {
9                     notify();
10                }
11            }
12            try {
13                Thread.sleep(5);
14            } catch (InterruptedException ie) { }
15        }
16        System.out.println("User Thread total:" + total);
17    }
18
19    public static void main(String args[]) {
20        MyThread mt = new MyThread();
21        Thread t = new Thread(mt);
22        t.start();
23        try {
24            synchronized (mt) {
25                mt.wait();
26            }
27        } catch (InterruptedException ie) { }
28        System.out.println("Main Thread total:" + mt.total);
29    }
30 }
```

IT
...!!

Note: We must call wait(), notify(), notifyAll() methods inside the synchronized blocks or synchronized methods otherwise it will throw a runtime exception called `IllegalMonitorStateException`.

// Write a java program on Produce – Consumer

package in.ashokit;

public class Store {

int value;

boolean pro_thread = **true**;

synchronized void produce(**int** i) {

if (pro_thread == **true**) {

 value = i;

 System.out.println("Procuder Has Produced Product " + value);

 pro_thread = **false**;

 notify();

 }

try {

 wait();

 } **catch** (InterruptedException ie) {

 }

 }

synchronized int consume() {

if (pro_thread == **true**) {

try {

 wait();

 } **catch** (InterruptedException ie) {

 }

 }

 pro_thread = **true**;

 notify();

return value;

 }

}

class Producer **implements** Runnable {

 Store sr;

 Producer(Store sr) {

this.sr = sr;

 }

public void run() {

int i = 1;

while (**true**) {

 sr.produce(i);

```

        try {
            Thread.sleep(1000);
        } catch (InterruptedException ie) {
        }
        i++;
    }
}

```

```

class Consumer implements Runnable {

```

```

    Store sr;

```

```

    Consumer(Store sr) {
        this.sr = sr;
    }

```

```

    public void run() {
        while (true) {
            int res = sr.consume();
            System.out.println("Consumer Has Taken Product " + res);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ie) {
            }
        }
    }
}

```

```

class ProducerConsumer {
    public static void main(String args[]) {
        Store sr = new Store();
        Producer p = new Producer(sr);
        Consumer c = new Consumer(sr);
        Thread t1 = new Thread(p);
        Thread t2 = new Thread(c);
        t1.start();
        t2.start();
    }
}

```


Knowledge – Check

- 1) What is Multi - Threading?
- 2) Why we need multi-Threading?
- 3) What is Thread Scheduler?
- 4) What is Thread Priority?
- 5) What is Thread Life Cycle?
- 6) How to create Thread?
- 7) Difference between start () method and run () method ?
- 8) Thread vs Runnable
- 9) Runnable Vs Callable
- 10) What is Synchronization
- 11) Synchronized Method Vs Synchronized Block
- 12) What is Dead Lock
- 13) Write a java program which gives Dead Lock
- 14) What is Inter - Thread Communication
- 15) What is Producer – Consumer problem
- 16) What is Executor Framework

**ASHOK IT***Learn Here.. Lead Anywhere..!!*