**Multithreading – Part-05 – yield() and join()**

* **Preventing Thread Execution:**

We can prevent the thread execution by using following methods.

yield()

join()

sleep()

* **yield()**

yield method causes to pause current executing thread to give the chance for waiting threads of same priority.

If there is no waiting thread or all waiting threads have low priority then same thread can continue its execution.

If multiple threads are waiting with same priority, then which waiting thread will get the chance we can’t expect it depends on thread scheduler.

The thread which is yielded, when it will get the chance once again? It depends on thread scheduler and we can’t expect exactly.

**public static native void yield();**

Note: Recall about the telephone problem and solution.

Change of life cycle because of yield method.

MT t = new MT();

**t.start();**

**If TS allocates processor Thread.yield()**

**If run method completes**

* **Example:**

class MyThread extends Thread{

public void run(){

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

System.out.println(“Child Thread”);

Thread.yield(); // Line-1

}

}

}

class ThreadYieldDemo{

public static void main(String[] args){

MyThread t = new MyThread();

t.start();

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

System.out.println(“Main Thread”);

}

}

}

In the above program if we are commenting Line-1, then both threads will be simultaneously and we can’t expect which thread will complete first.

If we are not commenting Line-1, then child thread always calls yield method, because of that main thread will get chance more number of times and the chance of completing main thread first is high.

Note: The thread has more processing time, that has to call the yield method in between the execution to give a chance to the other waiting thread.

Note:

Some platforms won’t provide proper support for yield().

Because of preemptive scheduling.

* **join():**

If a thread wants to wait until completing some other thread, then we should go for join method.

For example if a thread t1 wants to wait until completing t2. Then t1 has to call [t2.join()].

If t1 executes t2.join, then immediately t1 will be entered into waiting state until t2 completes.

Once t2 completes then t1 can continue its execution.

Note: recall the friend waiting for another friend example.

Example:

Venue Fixing Wedding Cards Printing Card distribution

t1 t2 t3

. . .

t1.join(); t2.join();

Wedding cards printing thread (t2) has to wait until venue fixing thread (t1) completion. Hence t2 has to call t1.join();

Wedding cards distribution thread(t3) has to wait until wedding cards printing thread (t2) completion. Hence t3 has to call t2.join();

public final void join() throws InterruptedException

public final void join(long ms); throws InterruptedException

public final void join(long ms, int ns); throws InterruptedException

ns is int, because in number once we cross the last digit it will automatically add 1 to the previous number.

After 99 🡪 100, 999 🡪 1000 etc.

So once the nana seconds crosses some limit it will become a millisecond, so instead of defining the ns as long, they defined it as int.

Note:

When our friend is waiting, someone may come and call him out for some time, so this is interruption, hence all the join method will throws InterruptedException.

Note:

Every join method throws InterruptedException which is checked exception, hence compulsory we should handle this exception either by using try/catch or by throws keyword., otherwise we will get compile time error.

Change of thread life cycle because of join method:

MT t = new MT();

**t.start();**

**possibility**

**If TS allocates processor Thread.yield()**

**t2.join()**

**t2.join(1000)**

**t2.join(1000, 100)**

**If run method completes**

Possibility:

1. If t2 completes (or)
2. If time expires (or)
3. If waiting thread got interrupted.

* **Example Program:**

**Case\_01: Waiting of main thread until completing child thread.**

class MyThread extends Thread{

public void run(){

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

System.out.println(“Seetha Thread”);

try{

Thread.slee(2000);

} catch(InterruptedException e){

}

}

}

}

class ThreadJoinDemo{

public static void main(String[] args) throws InterruptedException{

MyThread t = new MyThread();

t1.start();

t1.join(); // Line-01

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

System.out.println(“Raama Thread”);

}

}

}

If we comment Line-1, then both main and child threads will be executed simultaneously and we can exact output.

If we are not commenting line-1, then main thread calls join method on child thread object. Hence main will wait until completing child thread. In this case output is.

Seetha Thread, Seetha Tread 10 times

Rama Thread, Rama Thread 10 times