**Multithreading – Part-06 – Preventing Thread From Execution**

* Case\_02: Waiting of child thread until completing main thread:

class MyThread extends Thread{

static Thread mt;

public void run(){

try{

mt.join();

} catch(InterruptedException ie){

}

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

System.out.println(“Child thread”);

}

}

}

class ThreadJoinDemo{

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

MyThread.mt = Thread.currentThread();

MyThread t = new MyThread();

t.start();

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

System.out.println(“Main Thread”);

}

}

}

Note:

In the above example child thread calls join method on main thread object, hence child thread has to wait until completing main thread. In this case output is

Main Thread …

Child Thread…

* Case\_03:

If main thread calls join method on child thread object and child thread calls join method on main thread object, then both threads will wait for forever and the program will be stucked. (This is something like deadlock).

* Case\_04:

If a thread calls join method on the same thread itself then the program will be stucked (This is something like deadlock).

In this case thread has to wait infinite amount of time.

Example:

class Test{

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

Thread.currentThread().join();

}

}

* **Sleep Method**

If a thread doesn’t want to perform any operation for a particular amount of time, then we should go for sleep method.

Signature:

public static native void sleep(long ms) throws InterruptedException

public static void sleep(long ms, int ns) throws InterruptedException

Note:

Every sleep method throws InterruptedException, which is checked exception. Hence whenever we are using sleep method compulsory we should handle InterruptedException either by try catch or throws keyword otherwise we will get compile time error.

* **Impact of thread life cycle with sleep method:**

**MyThread t = new MyThread();**

**(New / Born state)**

**t. start();**

**(Ready / Runnable)**

**If Thread Scheduler allocates time**

**(Running) if Thread.sleep(1000);**

**Thread.sleep(1000, 100); (Note)**

**If run() method completes**

**(Dead)**

**Note: Threads from sleep state will go to runnable state if the following things occurs.**

1. **If the time expires.**
2. **If sleeping thread got interrupted.**

* **Example:**

class SlideRotator{

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

for(int i = 1; I <=10; i++){

System.out.println(“Slide -”+i);

Thread.sleep(5000);

}

}

}

Output:

Slide-1 to 10 with 5seconds pause in-between.

* **How a thread can interrupt another thread?**

A thread can interrupt a sleeping thread or waiting thread by using interrupt method of thread class.

public void interrupt();

Example:

class MyThread extends Thread{

public void run(){

try{

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

System.out.println(“I’m lazy thread”);

Thread.sleep(2000);

}

}

catch(InterruptedException ie){

System.out.println(“I got interrupted”);

}

}

}

class ThreadInterruptDemo{

public static void main(String[] args){

MyThread mt = new MyThread();

mt.start();

mt.interrupt(); // Line-1

System.out.println(“End of main”);

}

}

If we comment Line-1 then main thread won’t interrupt child thread. In this case child thread will execute for-loop 10 times.

If we are not commenting Line-1 then main thread interrupts child thread. In this case output is

End of main thread

I’m lazy thread

I got interrupted.

Note:

\* Whenever we are calling interrupt method, if the target thread not in sleeping state are waiting state then there is no impact of interrupt call immediately.

Interrupt call will be waited until target thread entered into sleeping or waiting state.

If the target thread entered into sleeping or waiting state then immediately interrupt call will interrupt the target thread.

\* Recall the snake example.

If the target thread never entered into sleeping or waiting state in its lifetime then there is no impact of interrupt call. This is the only case where interrupt call will be wasted.

Example:

class MyThread extends Thread{

public void run(){

for(int i =0; i< 10000; i++){

System.out.println(“I’m lazy thread-”+i);

}

System.out.println(“I’m entering into sleeping state”);

try{

Thread.sleep(10000);

}

catch(InterruptedException ie){

System.out.println(“I got interrupted”);

}

}

}

class ThreadSleepDemo{

public static void main(String[] args){

MyThread t = new MyThread();

t.start();

t.interrupt();

System.out.println(“End of main”);

}

}

**Note:**

In the above example interrupted call waited until child thread completes for-loop 10000 times.

* **Comparison table of yield(), join() and sleep()**

|  |  |  |  |
| --- | --- | --- | --- |
| **Purpose** | **yield()** | **join()** | **sleep()** |
| Purpose | If a thread wants to pause its execution to give the chance for remaining threads of same priority. Then we should go for yield() | If a thread wants to wait until completing some other thread, then we should go for join method. | If a thread don’t want to perform operation for a particular amount of time, then we should go for sleep method. |
| Is it overloaded? | NO | YES | YES |
| Is it final? | NO | YES | NO |
| Does it throw IE? | NO | YES | YES |
| Is it native | YES | NO | sleep(long ms) – native  sleep(long ms, int ns) – non-native |
| Is it static? | YES | NO | YES |