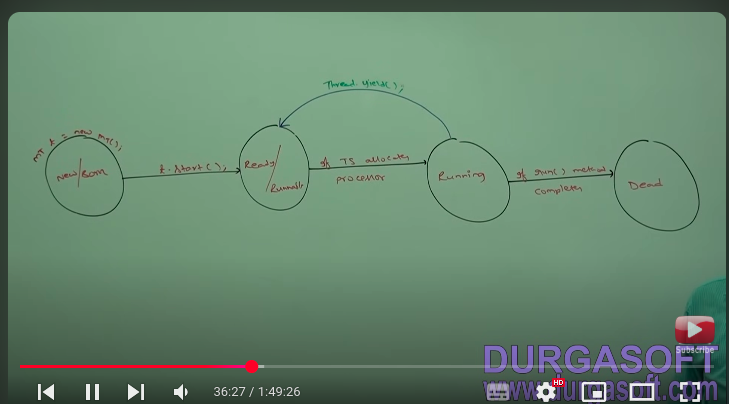
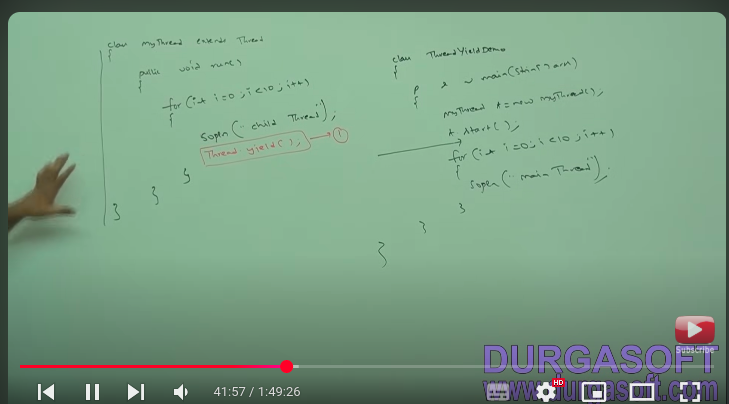
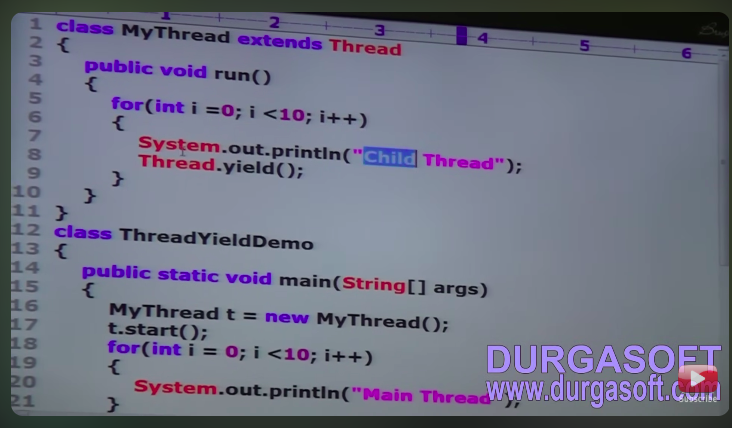
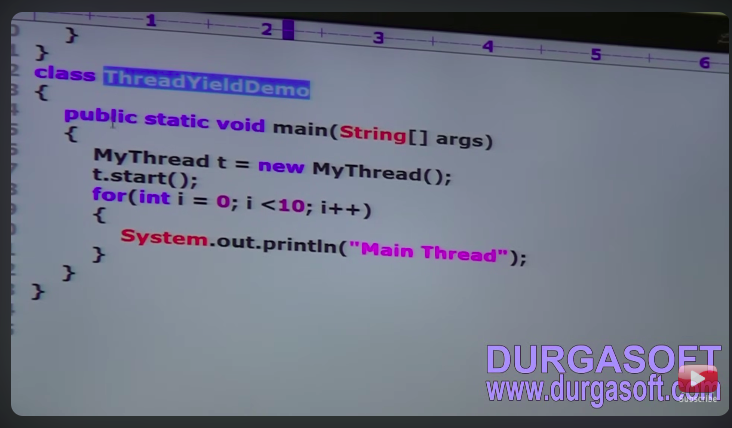
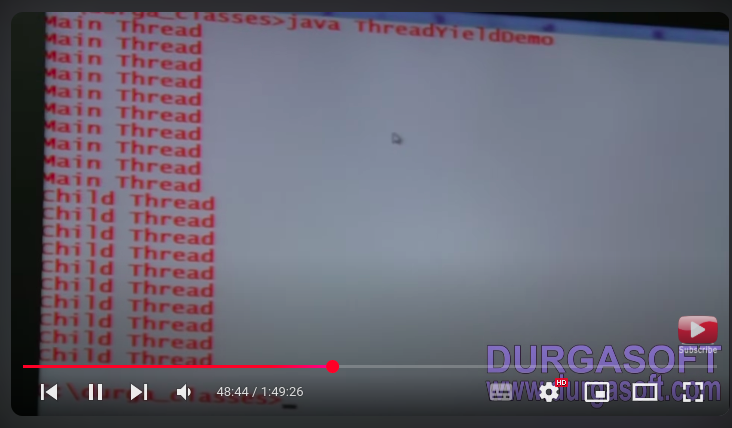
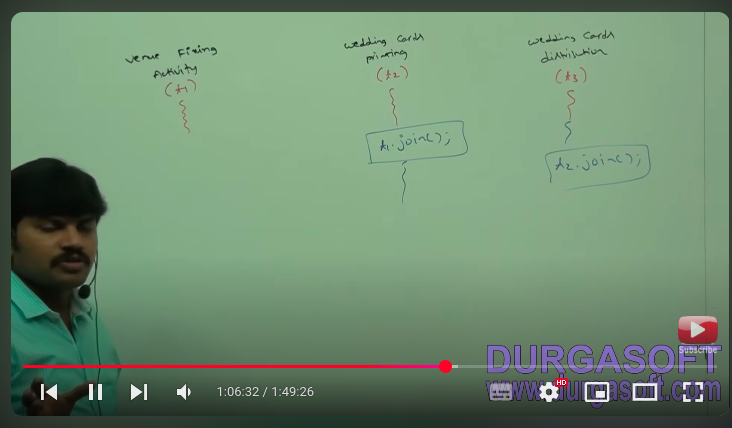
**Video 5 : preventing thread form execution 1.yield() and join()**

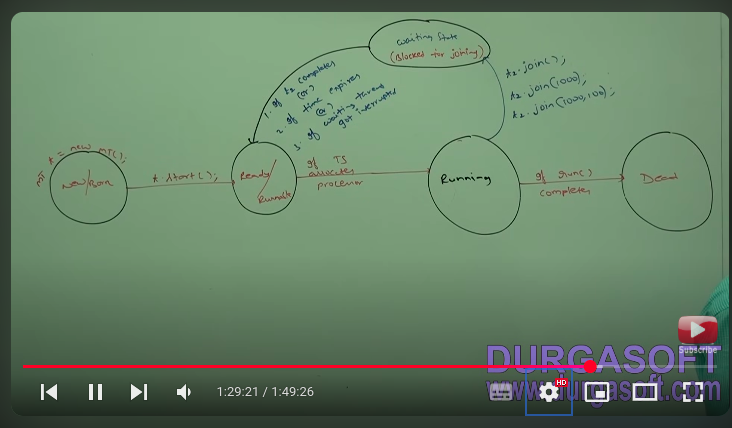
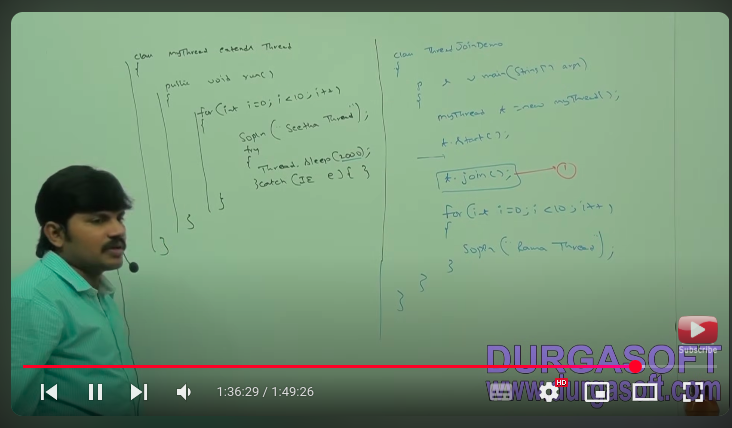
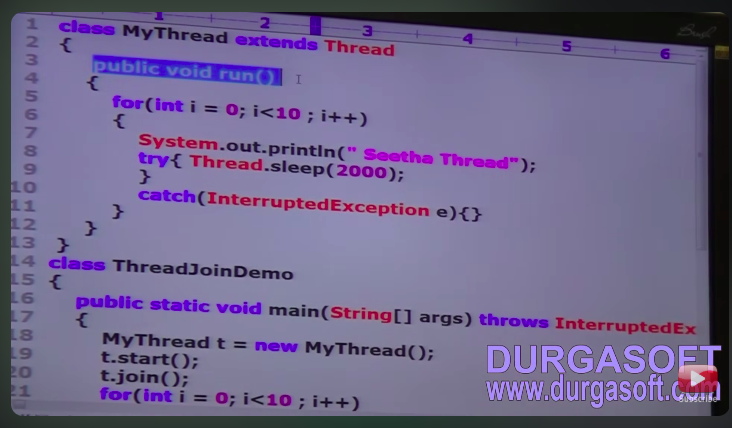
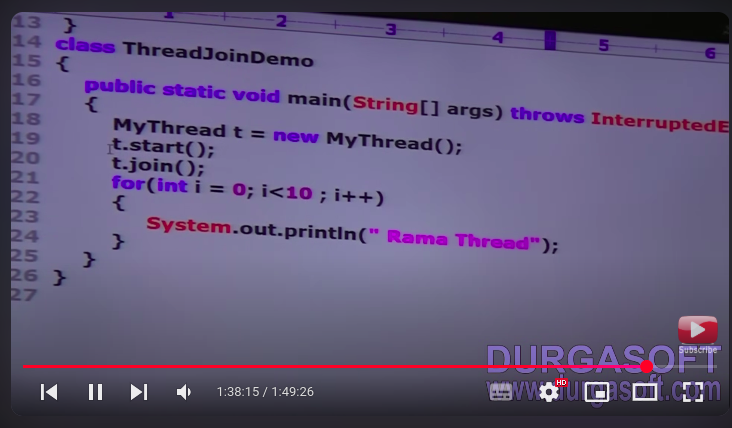
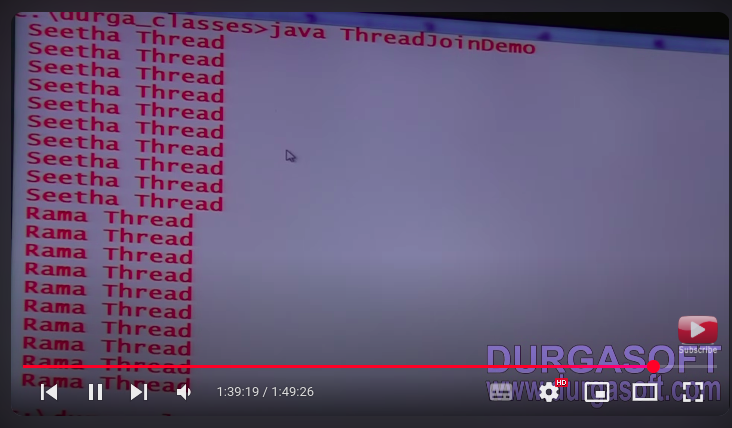
1. we can **prevent thread execution** by using following method
   1. **yield()** -
      1. **yield cause** to **pause current execution** thread to give **chance** for **waiting thread of same priority.**
      2. if there is **no waiting thread** or **all waiting** thread have **low priority** then **same thread** can **continue it’s execution**.
      3. if **multiple thread** are **waiting** with **same priority** then **which** waiting thread will **get the chance** we can’t expect it’s **depend on thread scheduler**.
      4. the which is **yielded**, when it will **get chance** ones again it is **depends** on **thread scheduler** we **can’t** expect **exactly**.
      5. **syntax** -- public static native void yield();
      6. 
      7. 
      8. 



* + 1. 
    2. in the above program if we are commenting line 1 then both thread will be executed simultaneously and we can’t expect which thread will complete first.
    3. if we are not commenting line 1 then child thread always call yield method because of that main thread will get chance more numbers of times and chance of completing main thread first is high.
    4. **Note :** some platform(operating system) won’t provide proper support for yield method.
  1. **join()** -
     1. if a thread **want** to **wait** until **completing** some **other thread** then we **should go** for **join** method.
     2. e.g . if thread t1 wants to wait until completing t2 then t1 has to call t2.join();
     3. if t1 executed t2.join() then t1 will be enter into waiting state until t2 completes.
     4. ones t2 complete then t1 can complete it’s execution.
     5. 
     6. wedding card printing thread (t2) has to wait until venue fixing thread (t1) compilation hence t2 to has to call t1.join().
     7. wedding card distribution thread (t3) has to wait until wedding card printing thread (t2) compilation hence t3 has to call t2.join().
     8. **syntax** -
        1. public final void join() **-- may throws interruptedexception**
        2. public final void join(long milliseconds) **-- may throws interruptedexception**
        3. public final void join(long milliseconds, int nanoseconds) **-- may throws interruptedexception.**

· nanos represents additional time in **nanoseconds** (1 nanosecond = 10−910^{-9}10−9 seconds).

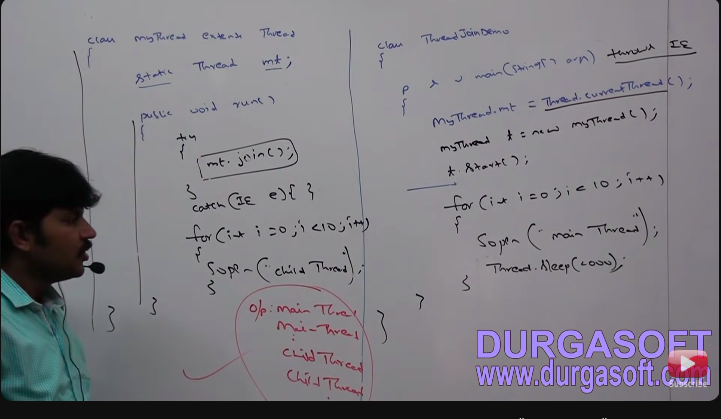
· The **valid range** is **0 to 999999** nanoseconds (since 1,000,000 nanoseconds = 1 millisecond).

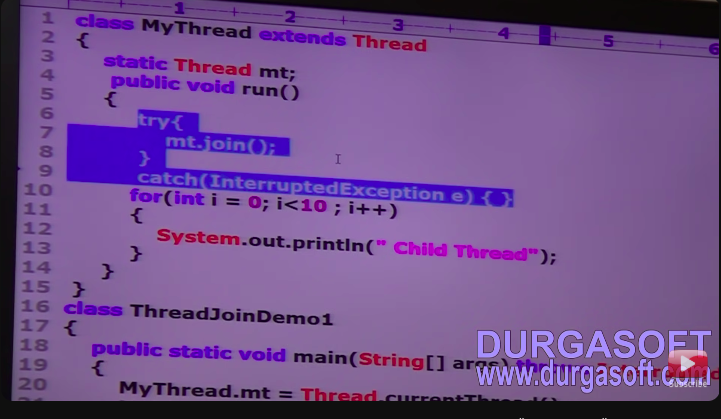
* + 1. every join method throws interruptedexception which is checked exception hence compulsory we should handle this exception either by using try catch or throws keyword other wise we will get compile time error.
    2. 
    3. 
    4. 
    5. 
    6. 
    7. if we comment line 1 then both main and child threads will execute simultaneously and we can’t expect output.
    8. if we are not commenting line 1 then main thread calls join method on child thread object hence main thread will wait until completing child thread in this case output is seeta thread seeta thread 10 times Ram thread Ram Thread 10 times

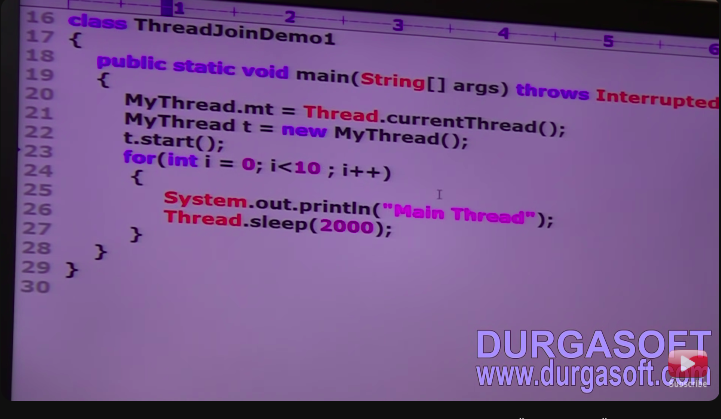
**Video 6 : Multi Threading | join(), sleep(), Thread Interruption**

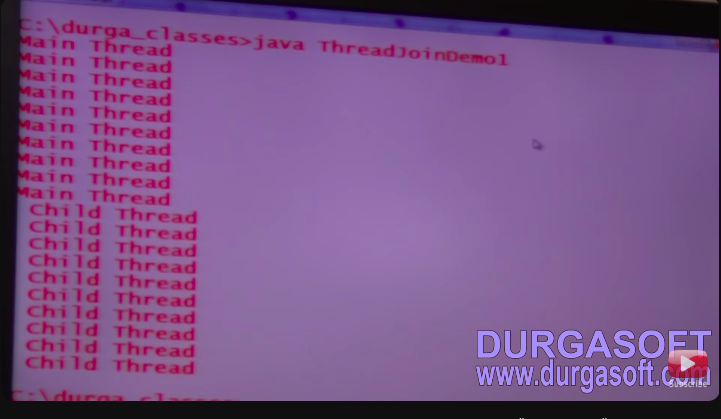
1. **join()**

**case 2 : waiting of child thread until completing main thread**





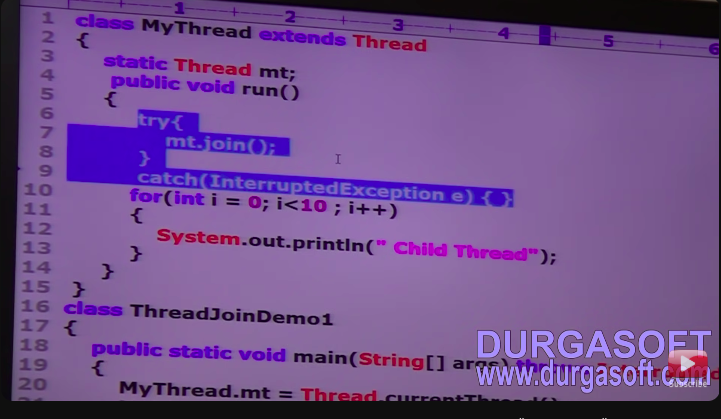


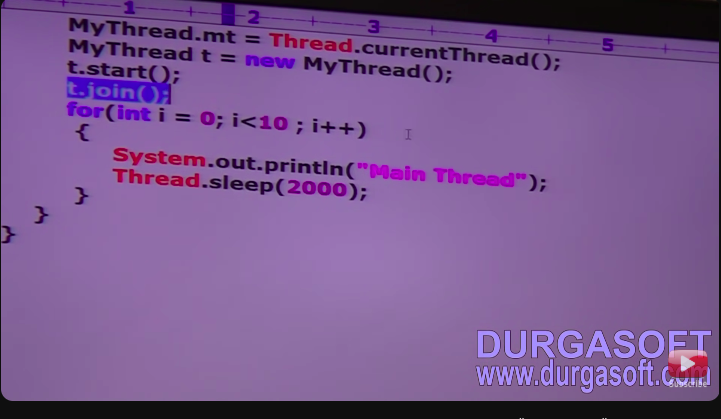


the above example child thread called join method on main thread object hence child thread has to wait until completing main thread in the case output is main thread main thread 10 times and child thread child thread 10 times.

**Case 3 : deadlock situation**

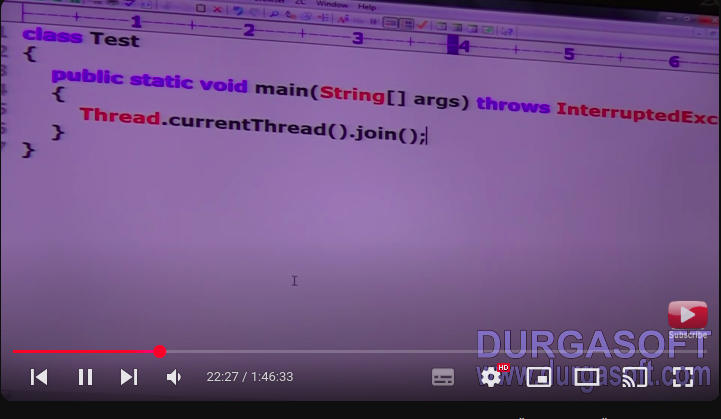
**A deadlock in Java occurs when two or more threads are blocked forever, waiting for each other to release resources. This typically happens in multithreaded applications when threads acquire locks in an inconsistent order.**





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 forever and program will be stuck (this is some thing like deadlock).

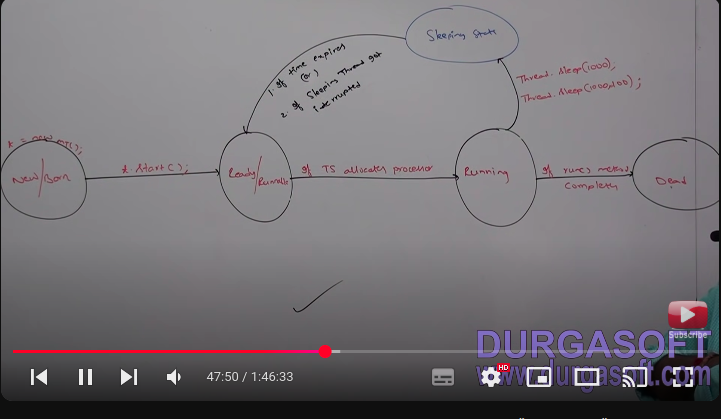
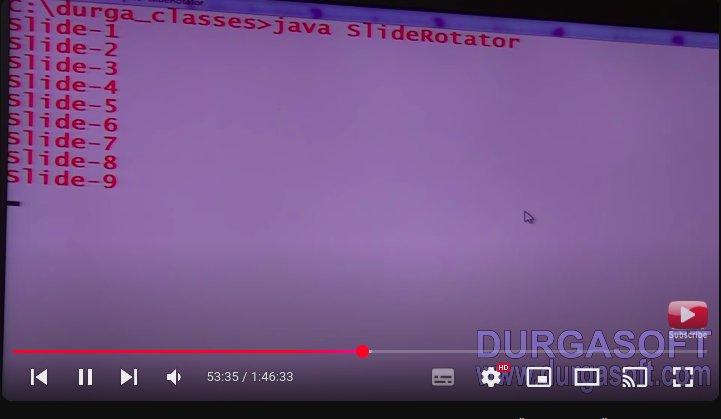
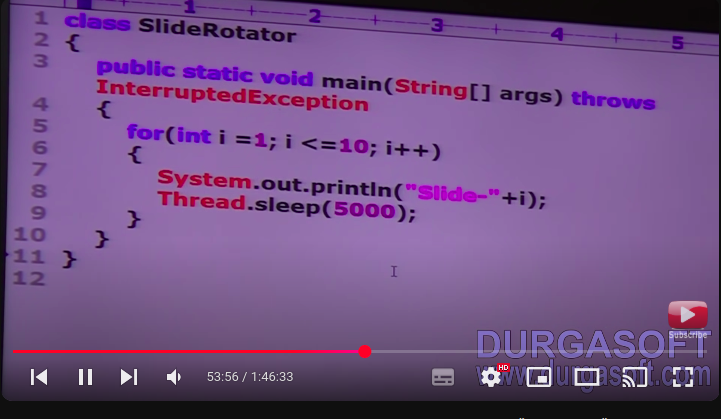
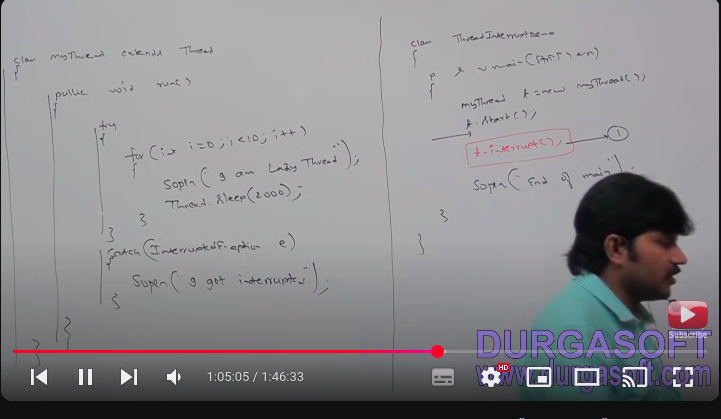
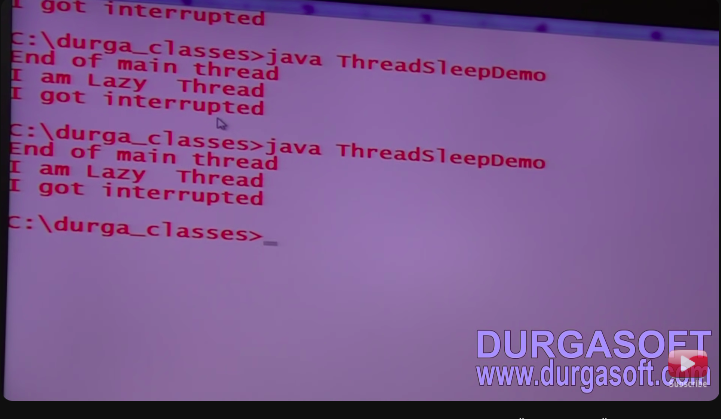
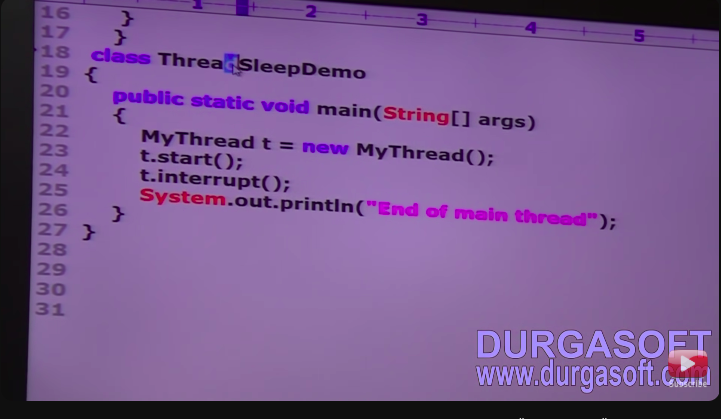
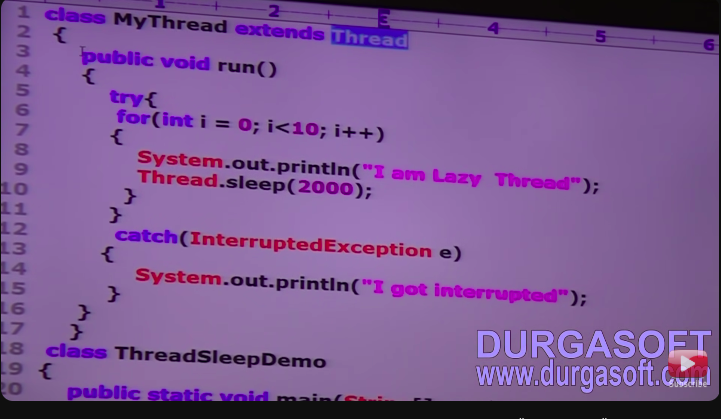
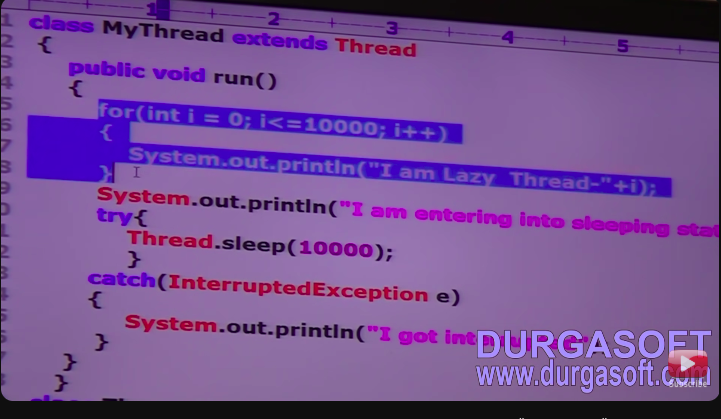
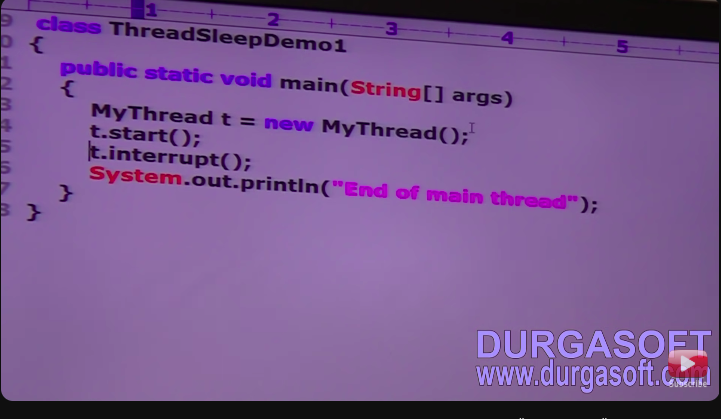
**Case 4 : another example of deadlock condition:**



if thread calls join method on same thread itself then program only stuck this is something like deadlock. in this case thread has to wait infinite amount of time.

1. **sleep()**
   1. if a thread **don’t want to perform** any **operation** for **particular** amount of **time** then we should go for **sleep method**.
   2. **syntax:**
      1. public static native void sleep(long millisecond); ---- **throws interruptionexception**
      2. public static native void sleep(long millisecond,int nanosecond); ---- **throws interruptionexception**

**note:-**Great observation! The reason you don’t see the native keyword when you Ctrl + Click in an IDE (like IntelliJ IDEA or Eclipse) is because your IDE typically shows **the decompiled or documented version** of the JDK source files, rather than the actual OpenJDK implementation.

* 1. every sleep method interruption exception, which is checked exception hence whenever we are using sleep method compulsary we should handle interruption exception either by try catch or throws keyword other wise we will get compiler time error.
  2. 
  3. example of sleep method
  4. 
  5. **how thread can interrupt another thread** 
     1. a thread can interrupt sleep thread or waiting thread by using interrupt method of thread class - public void interrupt();
     2. 
     3. if we comment line 1 then main thread won’t interrupt child thread in this case child thread will executed for loop 10 times.
     4. if we not commenting line 1 then main thtead interrupt child thread in this case output is **End of main thread ..I am lazy thread... I got interrupted.**
     5. **example 1:**
     6. 
     7. example 2:
     8. 
     9. 
     10. in the above example interrupt called waited until child thread complete for loop 10000 time.
     11. **Note :** whenever we are calling interrupt method if the target thread not in sleeping or waiting state then there impact of interrupt call immediately interrupt called wait until target thread enter into sleeping or waiting status.
     12. if the target thread enter into sleeping or waiting state then immediately interrupt called will interrupt target thread.
     13. if the target never enter into sleeping or waiting state in his life time then there is no impact of interrupt call this is the only case where interrupt called wasted.
     14. **comparison table of yield , join and sleep methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **property** | **yield** | **join** | **sleep** |
| purpose | if a thread wants to pause it’s execution to give the chance remaining thread of same priority then we should go for yield method. | if a thread want’s to wait until completing some other thread then we should go for join method . | if a thread don’t want to perform any operation for particular amount of time then we should go sleep method. |
| is it overloaded | no | yes | yes |
| is it final | no | yes | no |
| is it throws interrupted exception | no | yes | yes |
| is it native method | yes | no | sleep(long ml) ----> native  sleep(long ml,int ns)--->none-native |
| is it static | yes | no | yes |