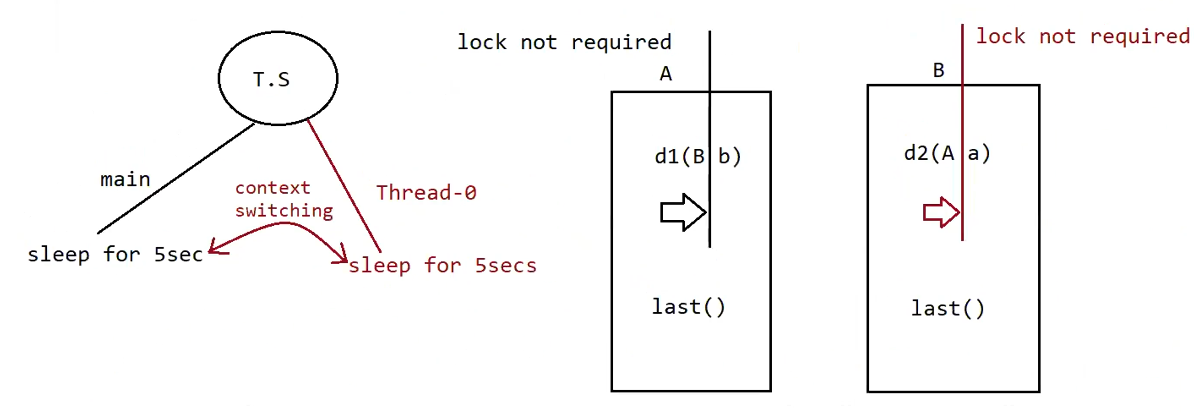


Deadlock :

If 2 threads are waiting for each-other forever( without end) such type of situation ( infinite waiting) is called dead lock

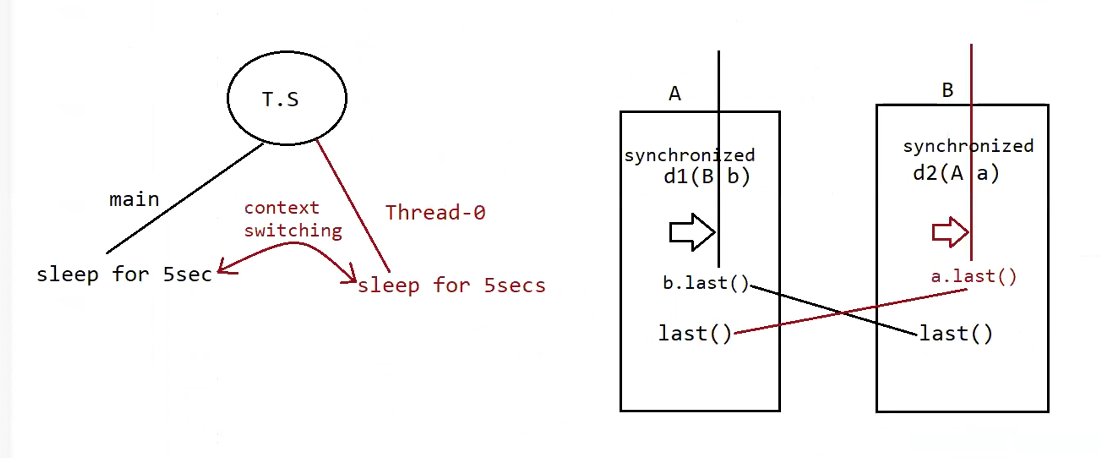
There are no resolution techniques for dead-locks ,but several prevention techniques are possible

Synchronized keyword is the cause for the deadlock, hence whenever we are using synchronized keyword we have to take special care



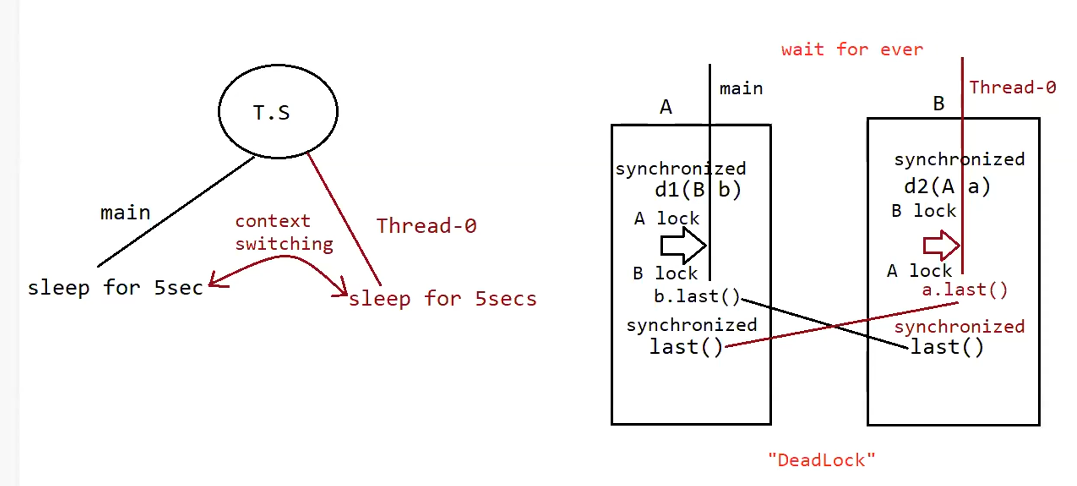
Eg: Without\_Synchronization\_DeadLock

// go through the code



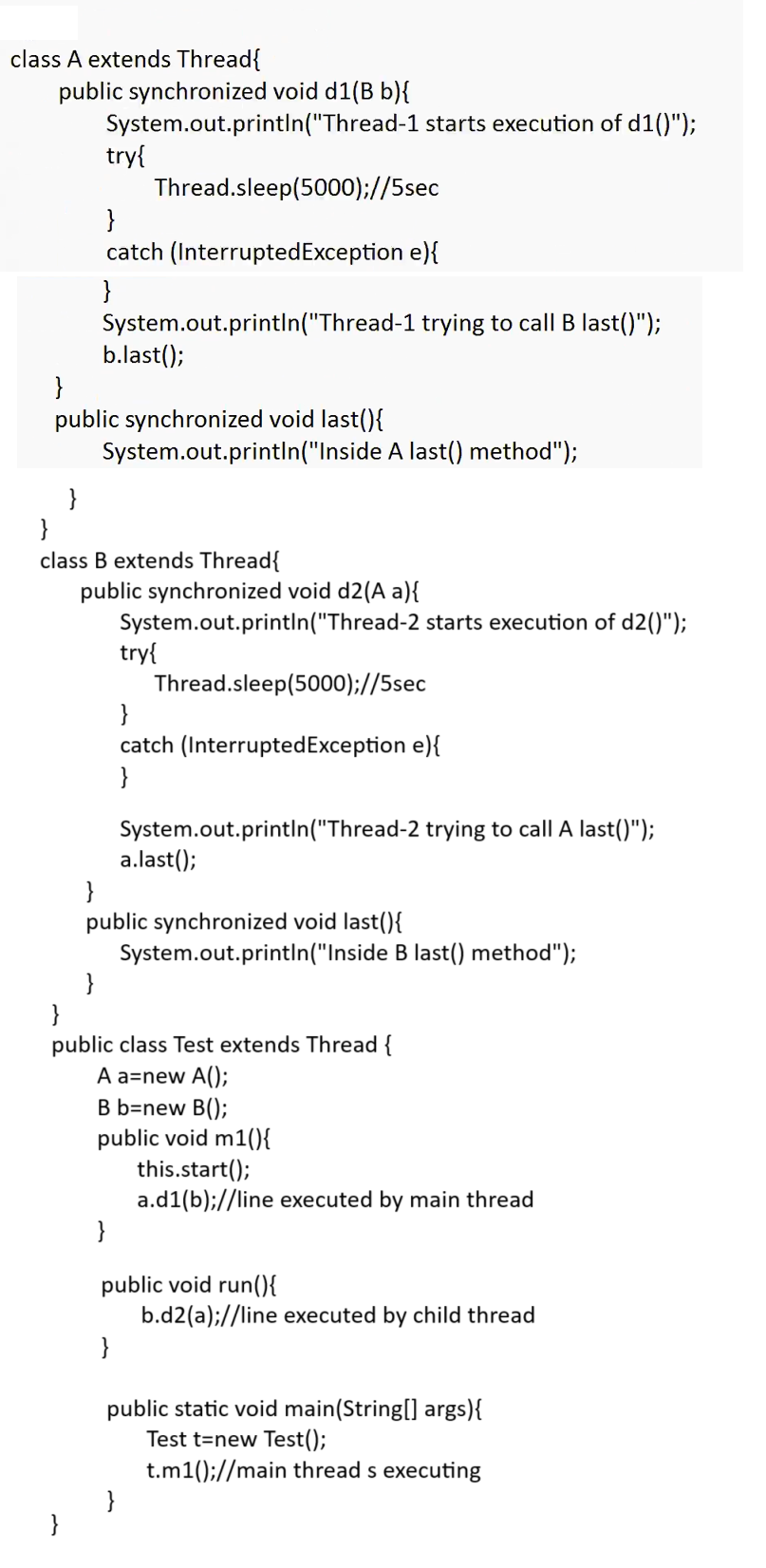
Eg: With\_Synchronization\_Deadlock\_

// go through the code



Eg: With\_Synchronization\_Deadlock\_Eg2

// go through the code



In the above program, there is a possiblity of "deadlock".

Output

Thread-1 starts execution of d1()

Thread-2 starts execution of d2()

Thread-1 trying to call B last()

Thread-2 trying to call A last()

//here cursor will be waiting

t1 => starts d1(),since d1() is synchronized and a part of 'A' class so t1 applies lockof(A) and

starts the execution, while executing it encounters Thread.sleep().so T.S gives chance

for t2 thread.

After getting a chance again by TS, it tries to execute b.last.

but lock of b is with t2 thread, so t1 enters into waiting state.

t2=> starts d2(),since d2() is synchronized and a part of 'B' class so t2 applies lockof(B) and

starts the execution, while executing it encounter Thread.sleep(),so TS gives chance again for t1 thread.

After getting a chance again by TS, it tries to execute a.last

but lock of a is with t1 thread, so t2 enters into waiting state.

Since both the threads are in waiting state and it would be waiting for ever,so we say the above

pgm would result in "DeadLock".

Note:

synchronized is the only reason why there is a deadlock,so we should be careful when we use

synchronized keyword,if we remove atleast one synchronized word then the program wont enter intodead lock.

DeadLock vs starvation

===================

Long waiting of a thread, where waiting never ends is termed "deadlock".

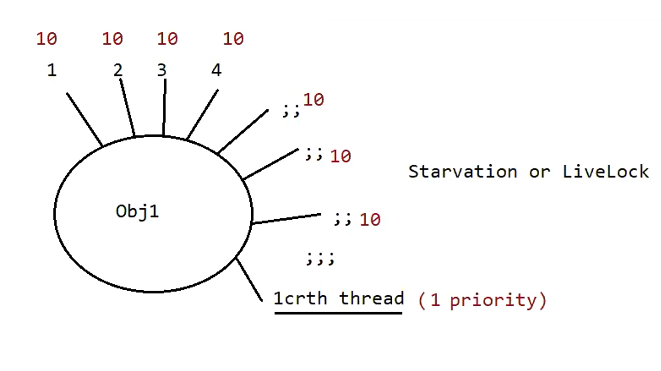
Long waiting of a thread, where waiting ends at certain point is called "starvation".

eg :: Assume we have 1cr threads, where all 1cr threads have priority is 10,but one thread is there

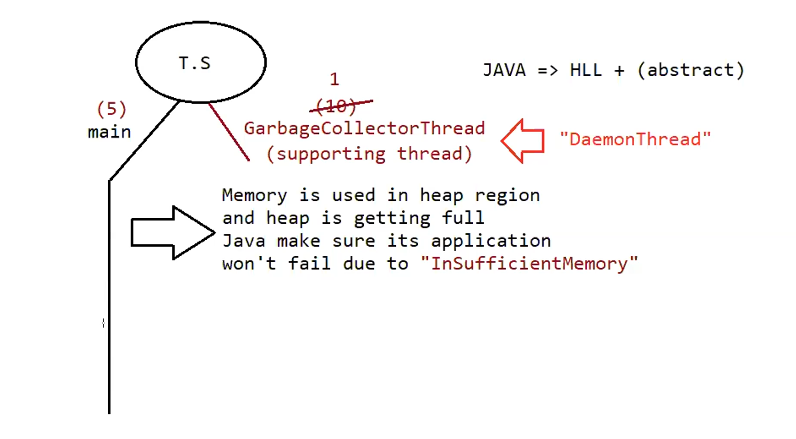
which has priority 0,now the thread with a priority-0 has to wait for long time but still it gets a chance, but it has to wait for long time, this scenario is called "Starvation".

Note :: Low priority thread has to wait untill completing all priority threads but ends at certain

point which is nothing but starvation.



Daemon thread :



A thread which is executing in the background is called “daemon thread”

eg: AttachListener , SignalDispatcher , GarbageCollector

// not used in the industry much as it there are lot of predefined daemon threads which comes by default .

Main Objective of the Daemon thread :

The main objective of the daemon thread is to provide support for the non-daemon threads (main thread)

eg: if main thread runs with low memory then jvm will call GarbageCollector thread , to destroy the useless objects , so that no of bytes of free memory will be improved . with this free memory main thread can continue its execution.

Usually daemon threads have low priority , but based on our requirement daemon threads can run with high priority also

Jvm creates 2 threads

1. Daemon Thread ( priority =1 , priority = 10)
2. main ( priority = 5)

while executing the main code , if there a shortage of memory then immediately jvm will change the priority of daemon thread to 10 , so garbage collector activates daemon thread and it frees the memory , after doing it immediately the priority changes to 1, so main thread will continue

How to check whether the Thread is Daemon or not?

public boolean isDaemon() => To check whether the thread is "Daemon"

public void setDaemon(boolean b) throws IllegalThreadStateException

b=> true, means the thread will become Daemon, before starting the Thread we need to make the thread as "Daemon" otherwise it would result in "IllegalThreadStateException".

What is the default nature of the Thread?

Ans. By default the main thread is "Non-Daemon".

for all remaining thread Daemon nature is inherited from Parent to child, that is if the parent thread is "Daemon" then child thread will become "Daemon" and if the parent thread is "Non-Daemon" then automatically child thread is also "Non-Daemon".

Is it possible to change the Non-Dameon nature of Main Thread?

Ans. Not possible, becoz the main thread starting is not in our hands, it will be started by "JVM".

Eg: Daemon\_Thread

// go through the code

Eg: Daemon\_Thread\_Dead\_State

// go through the code

Eg: Making\_Child\_Thread\_As\_Daemon\_After\_Start\_Method

// go through the code

Eg: Main\_Thread\_As\_Daemon\_Thread

// go through the code