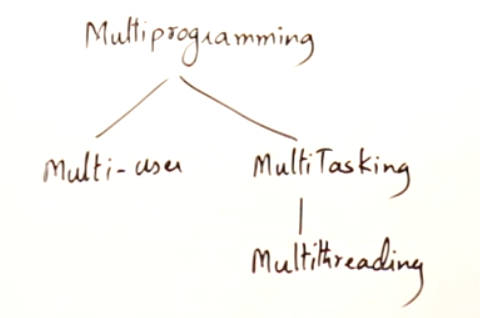
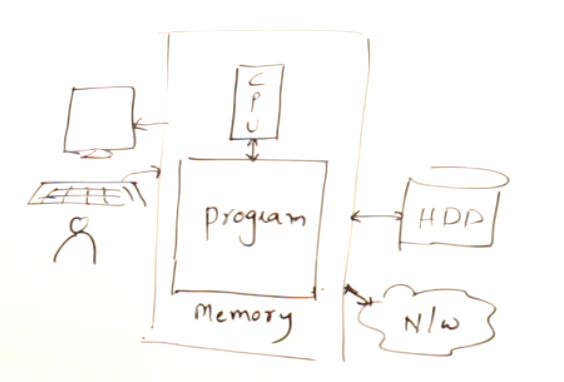
# **Multi-programming**



Need for multi-programming

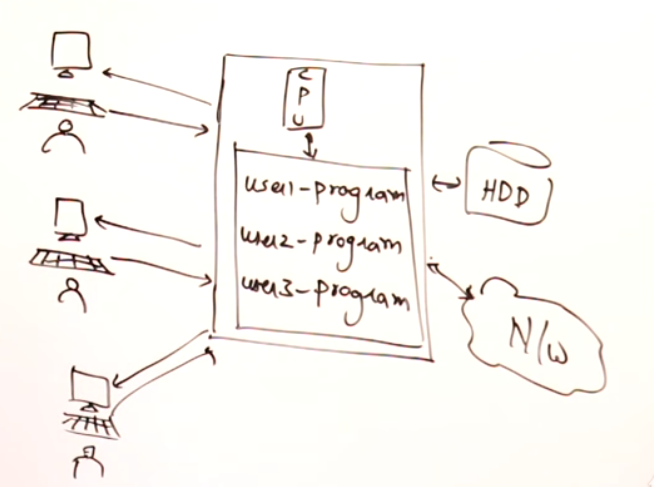


Suppose when you type 10-15 pages in MS-word which takes nearly an hour to complete, CPU will be utilized efficiently only for 5 minutes.   
CPU is utilized only for checking spelling mistakes, etc..

Rest of the time, CPU is idle. So to make use of the CPU efficienlyt, the concept of multi-programming comes into picture.

So that when you type 10-15 pages in MS-word, side by side you can also run some other programs at the background which will make use of the CPU so efficiently. Hence our aim is always to keep CPU busy all the time. This will taken care by the OS.

# **Multi-user**



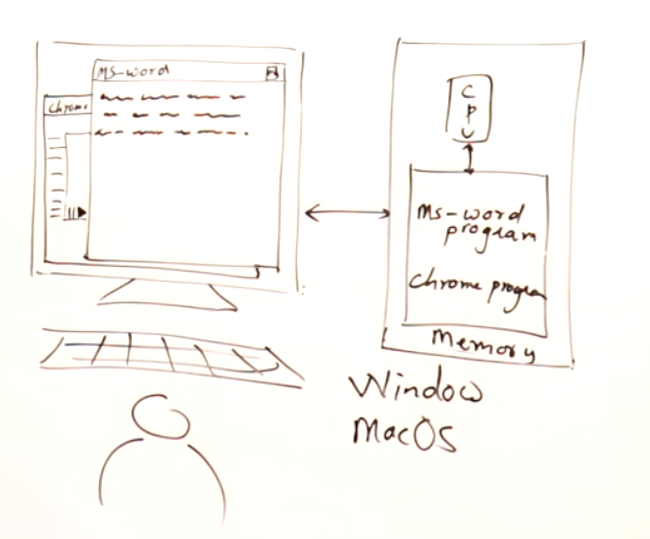


Multi-user 🡪 More than 1 user uses the same PC at the same time.

3 individual users are using the same PC, so that the CPU is always busy. These are taken care by famous OS such as Unix and Linux. These kind of multi-user practices are now outdated.

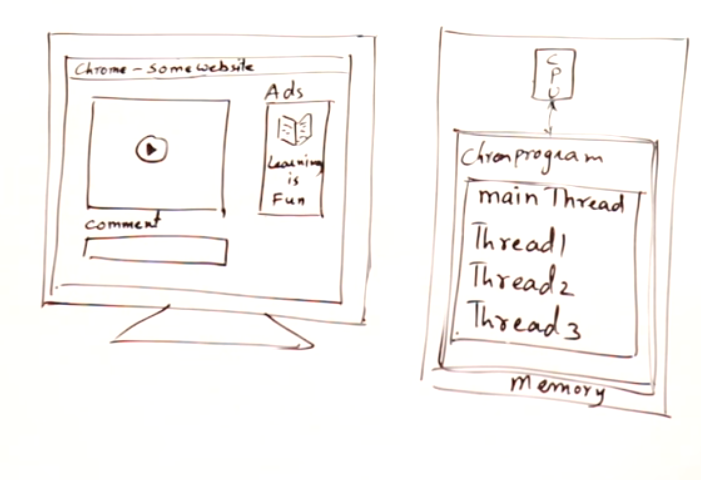
# **Multi-tasking**

Multi-tasking 🡪 A single user making multiple tasks on a PC simultaneously.

  
Actually CPU cannot proceed all programs simultaneously, it running alternatively (i.e changing programs) at a faster rate, so we don’t feel or see the shift. Programs are running on the CPU alternatively and not simultaneously.

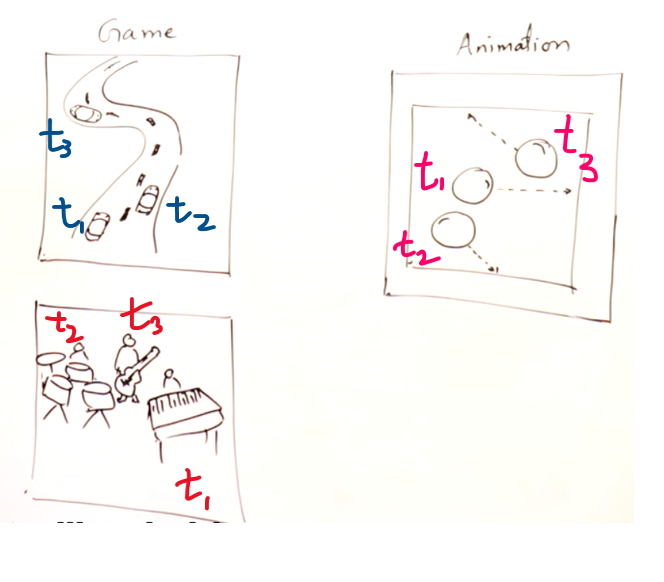


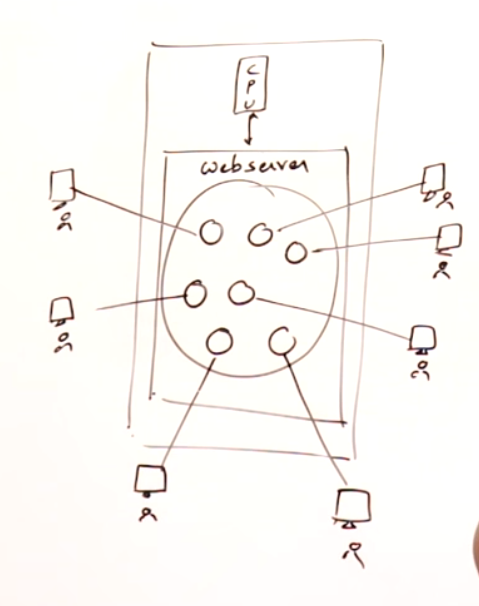
# **Multi-threading**



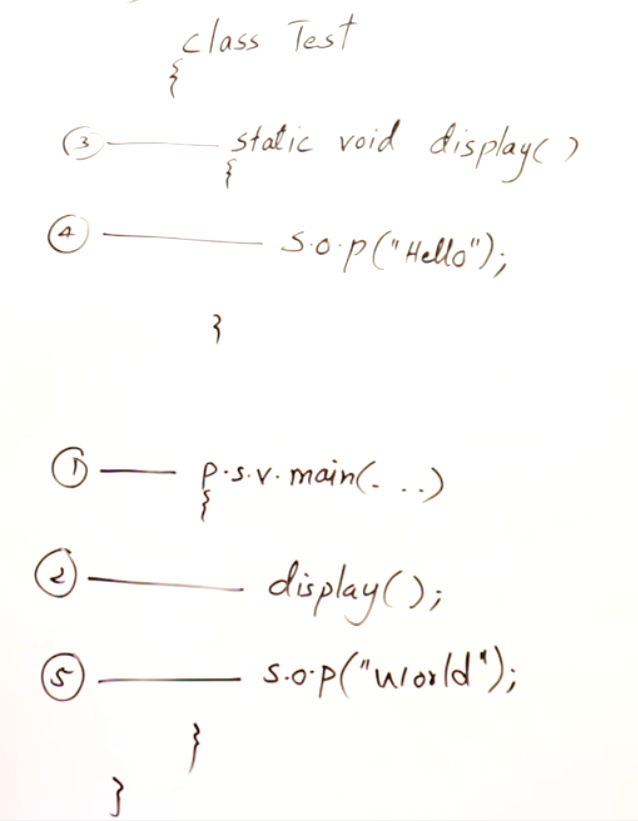
main thread () 🡪 Overall chrome application  
thread1 🡪 to play the video  
thread2 🡪 to showcase the adds  
therad3 🡪 to put the comments

So a single chrome application is divided into multiple threads and executed.  
Threads can also be executed one-by-one, by the speed of CPU we cannot able to see the shift and we are assuming that all threads are executing simultaneously.

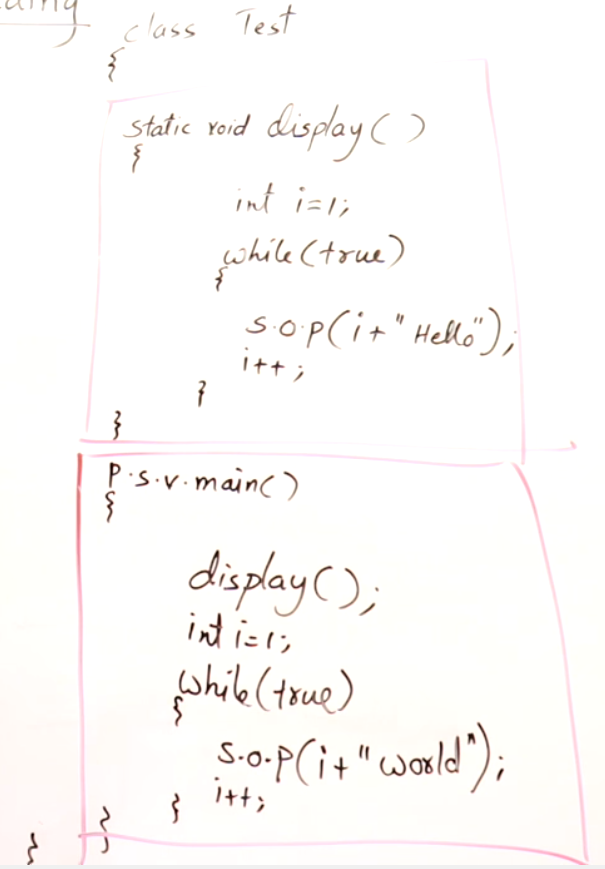


 Bubbles 🡪 threads

# **Control flow of a program**



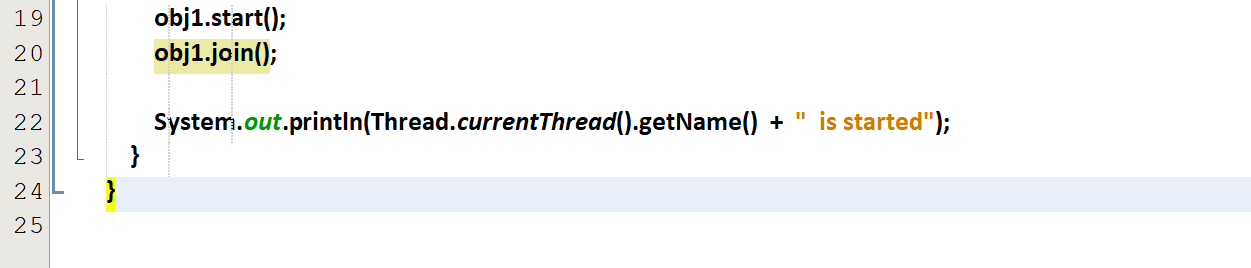
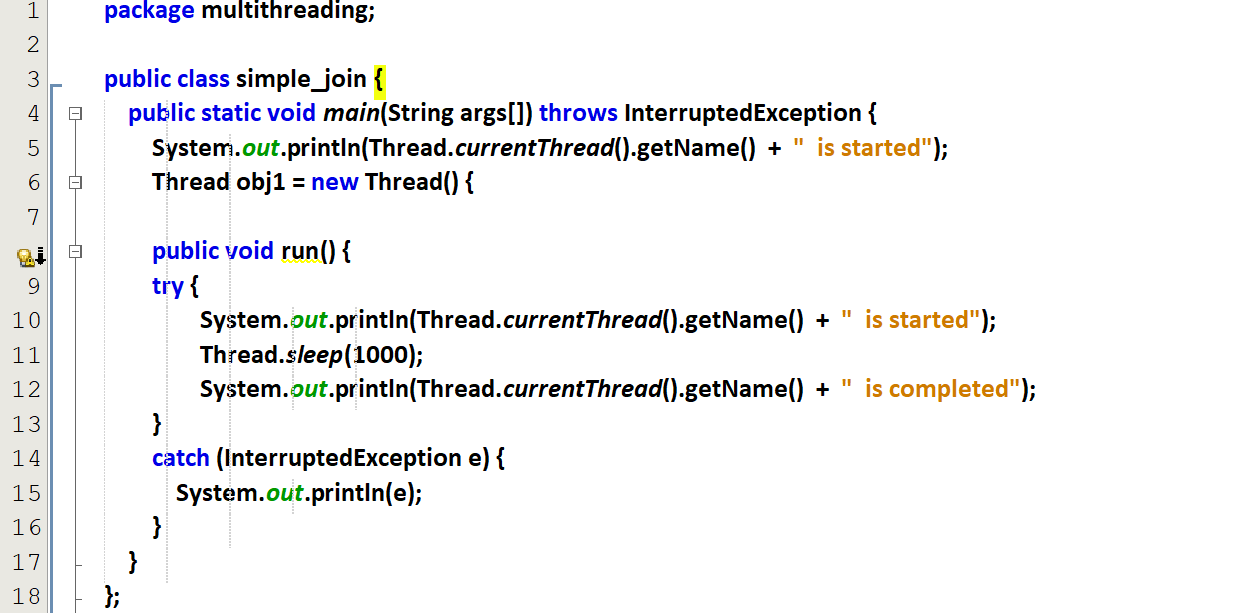
This is the normal control flow while handling with functions

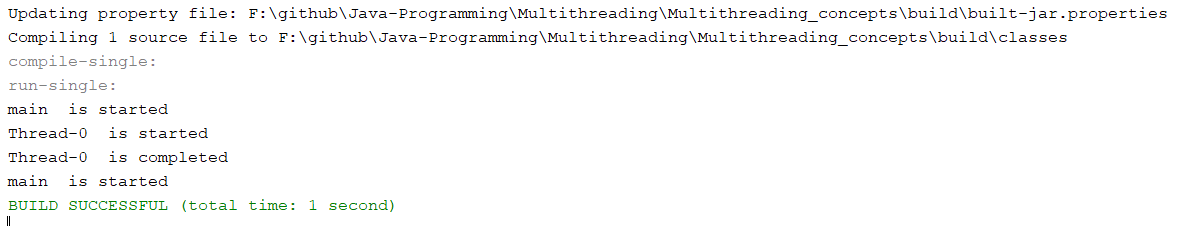




i+“hello” and i+“world” should run simultaneously. How to do???  
Assign thread1 to i+”hello” and thread2 to i+”world” , so that two threads can run simultaneously.  
Multi-threading concepts come into picture.

# **Multi-thread creation inside main()**





# **Multi-thread creation using Thread class**

How to achieve multi-threading ????

1. Thread class
2. Runnable Interface

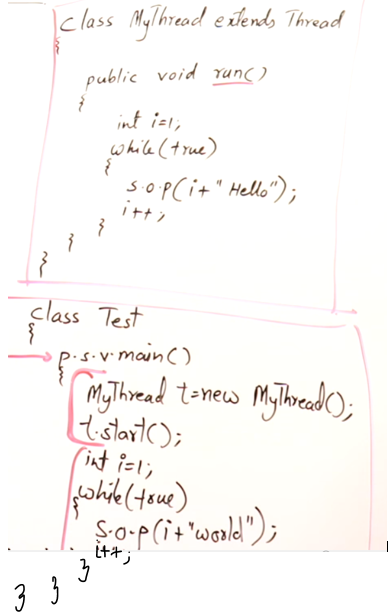
If your class is not inheriting any other class, then you can inherit thread class and can achieve multi-threading.

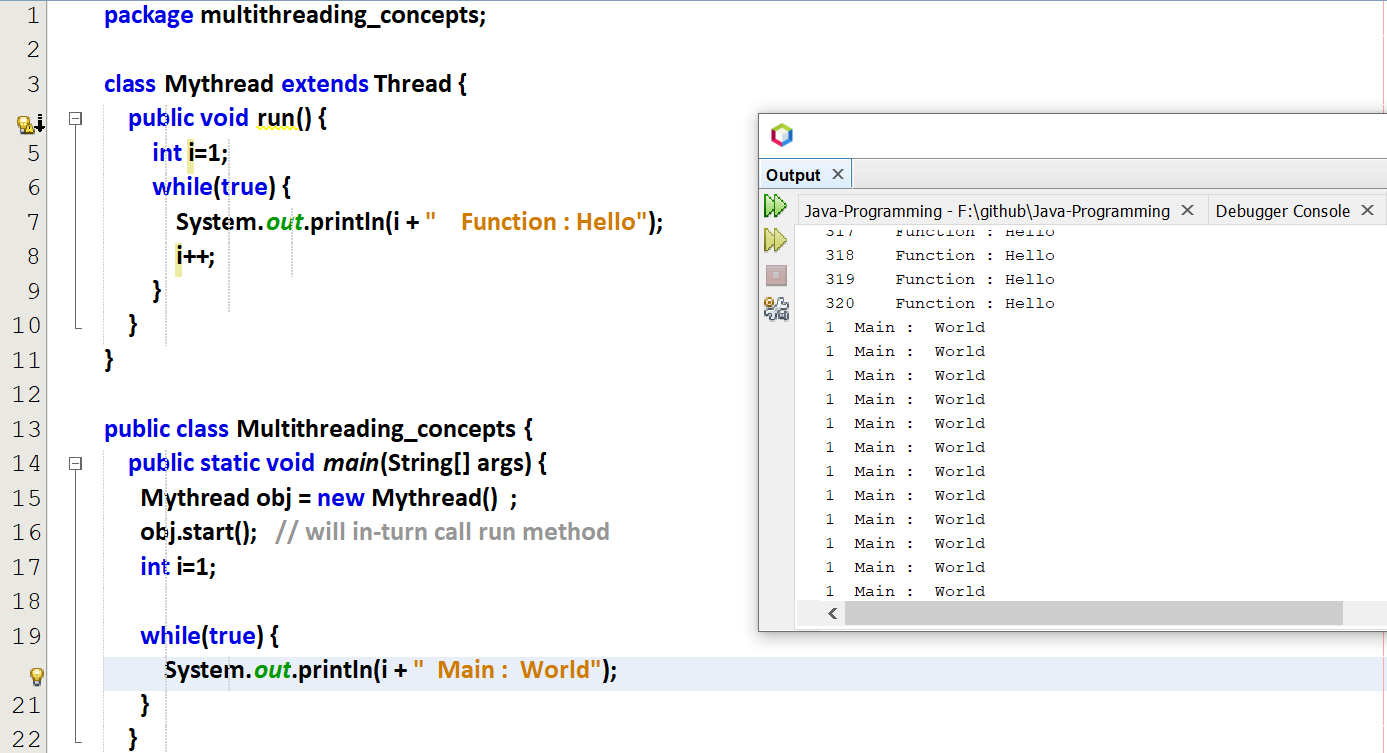
If your class is already inheriting some other class, then you cannot inherit Thread class. Since JAVA class can only inherit one class. In this case your class can implement Runnable interface and can achieve multi-threading. Since JAVA class can implement multiple interfaces but can extends(i.e inherit) only 1 class.

What-ever logic we want to execute in a thread, we should write that logic inside the run().  
We are over-riding the run() since run() is already defined in Thread class.  
main() 🡪 starting point of a program.  
run() 🡪 starting point of a thread.

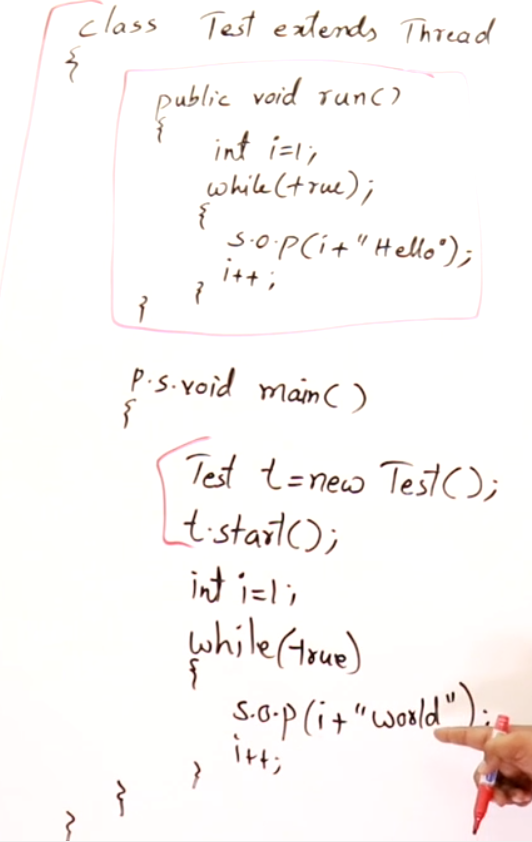
1)Create an object of the class which you want to execute as a thread.  
2) Then obj.start(), start() is already defined in Thread class and it knows how to start a thread and class the run()

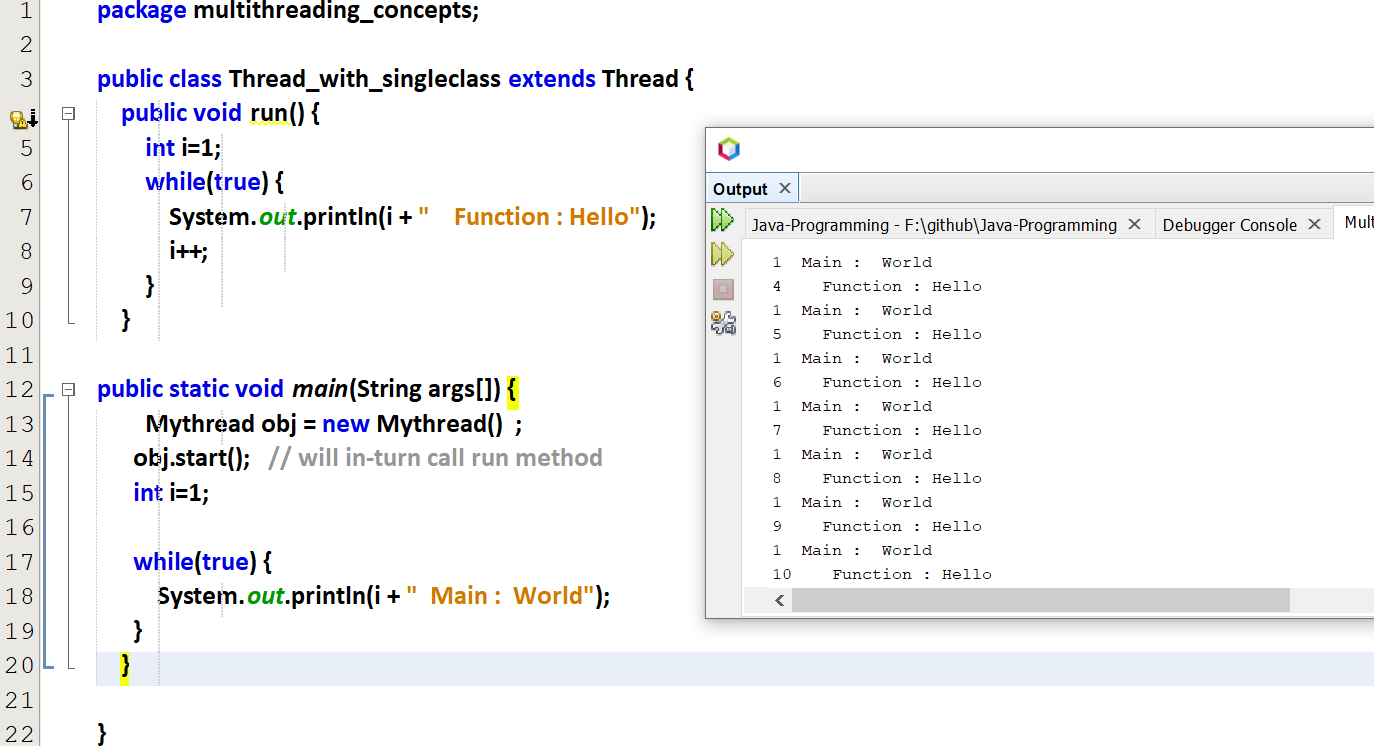
## **Using two classes**





## **Using a single class**





# **Multi-thread creation using Runnable Interface**

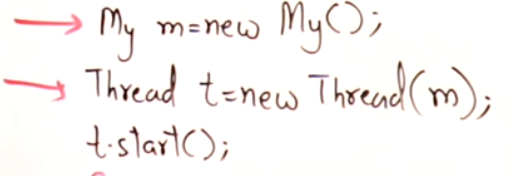


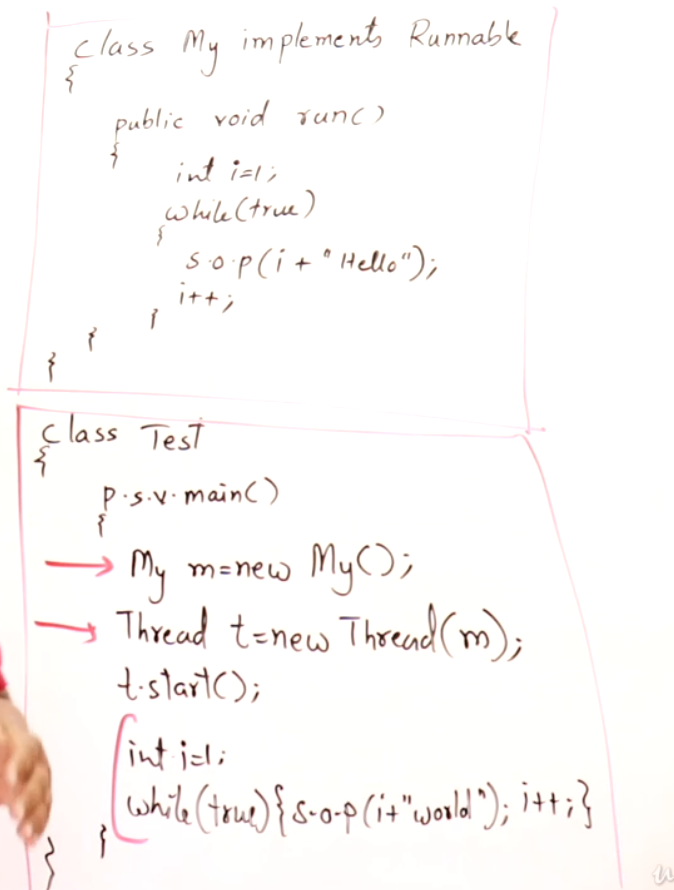
When a class implements an interface, It should over-ride all the methods in that interface.  
So it is very-very important to over-ride run(). And also only run() is available in Runnable interface.

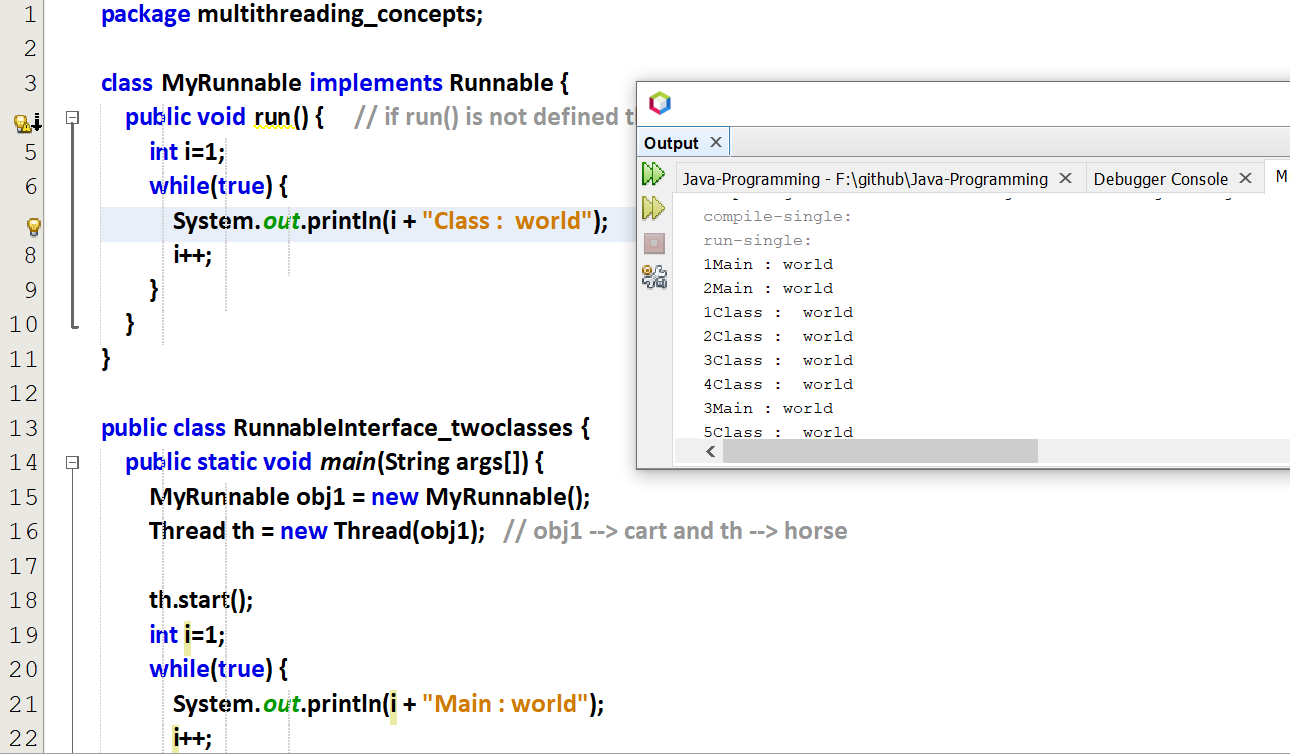
## **Using two classes**

Runnable interface is capable of running a thread, but it cannot run by-itself. So it seeks the help of Thread class.

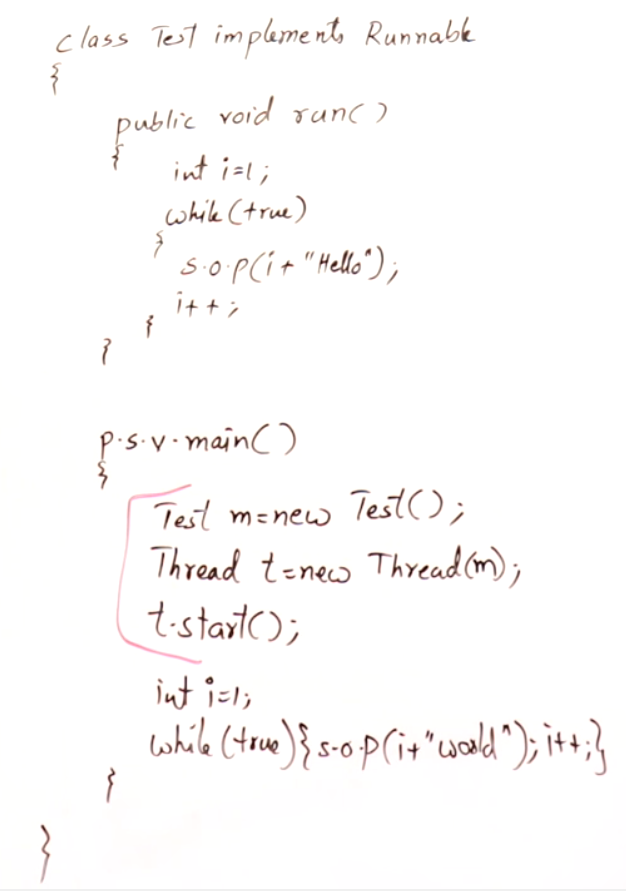
t 🡪 horse  
m 🡪 cart attached to that horse  
So that when the horse runs, it will also drag the cart along with it.

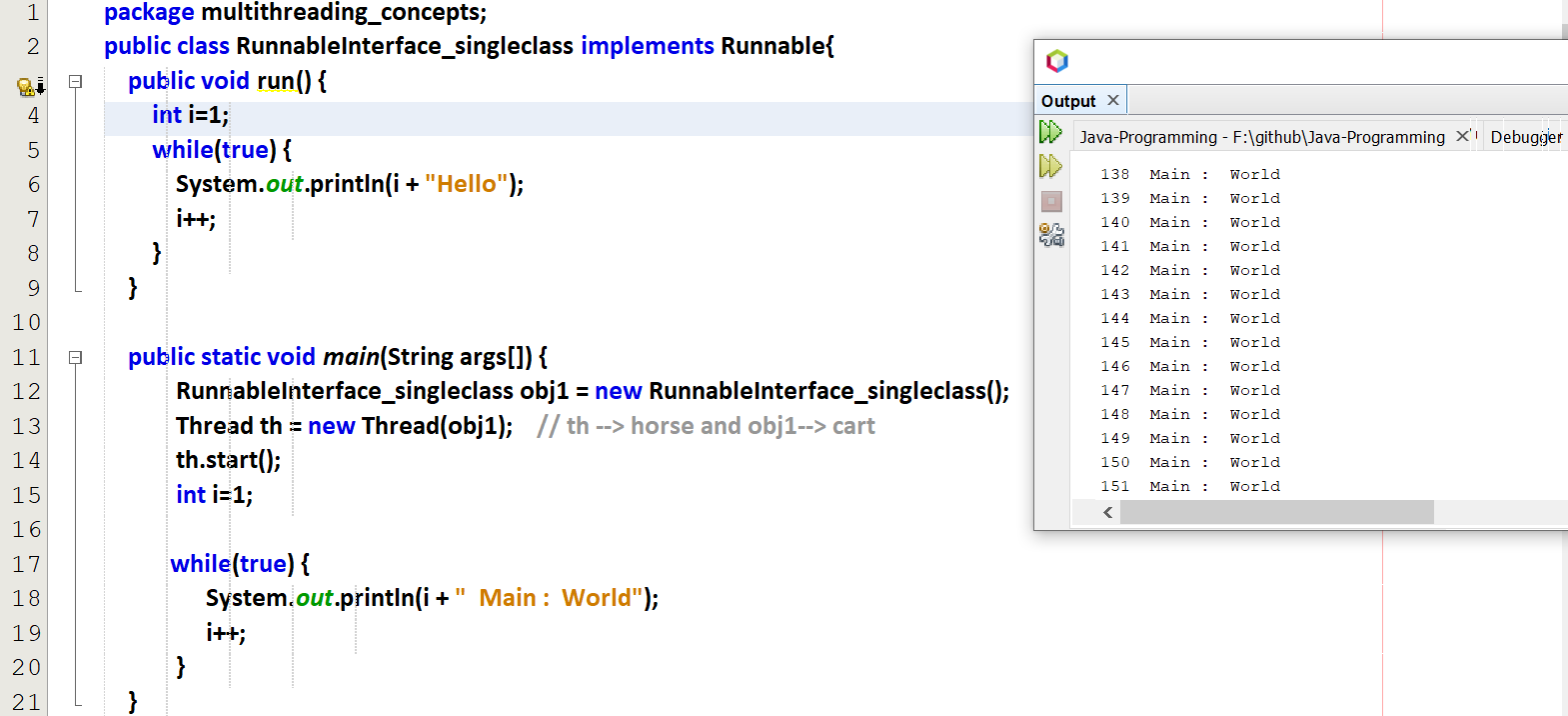




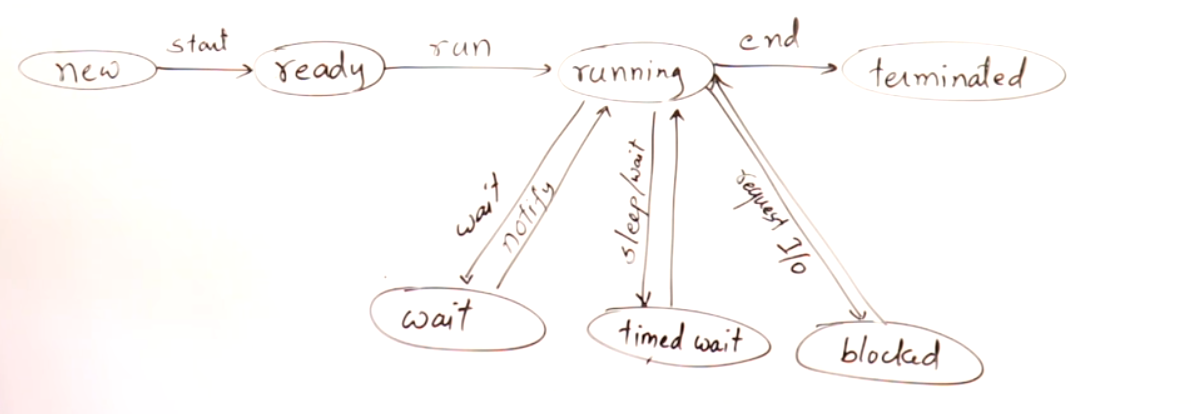


## **Using single class**



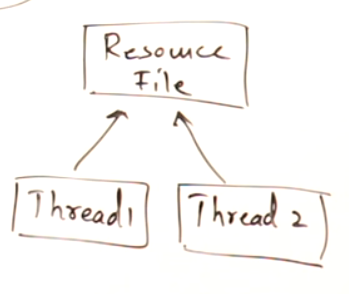


# **States of a thread**



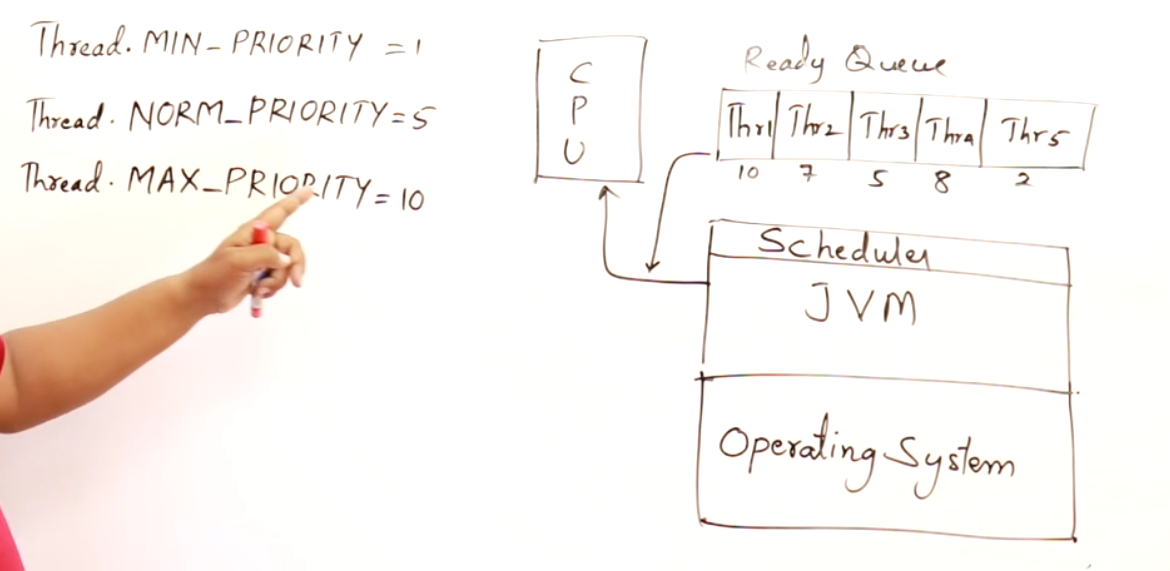
If we call start, in-turn it will call run()  
If a thread is terminated we cannot able to restart that thread, instead we have to create a new thread and start the process.

Resource should be accessed by one thread at a time.



Similar to a printer used in office, where there will be only 1 printer and all other users wants to access the printer, so they must wait in a queue and take the printout one-by-one at a time.

# **Thread Priorities**

  
Scheduler in JVM will maintain the ready queue.

A thread having higher priority will be executed first.

## **Eg: MS-Word**

t1 🡪 getting input from the user  
t2 🡪 spell check  
t3 🡪 autosaving the document

In the above operations how the priorities are provided ???  
If the 1st priority is given to t3, then what all you type will not apply on the screen time to time, there will be slight delay in displaying which will irritate the user while typing.  
So 1st priority 🡪 t1 , 2nd priority 🡪 t2 , 3rd priority 🡪 t3  
if this takes place in-order then there will be no problem in MS-Word.

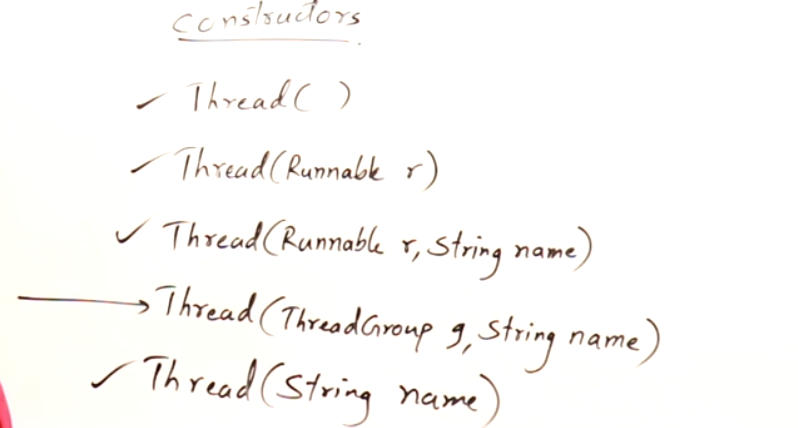
## **Eg: Chrome Browser**

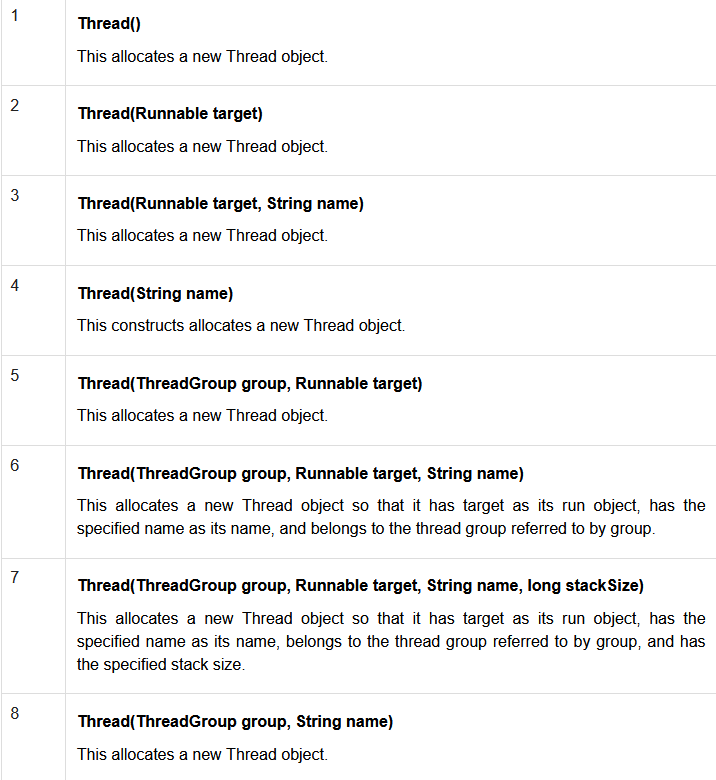
t1 🡪 pulling the data from the server  
t2 🡪 rendering the pulled data to the display

Some-times it takes for a page to reload to some extent. This is because without pulling the complete data from the web-server, it cannot able to render that pulled data into the monitor. So the web-pages loads and after that only it renders on the monitor display.  
So 1st priority 🡪 pulling the data from the server  
2nd priority 🡪 rendering the pulled data to the display

# **Thread Class**

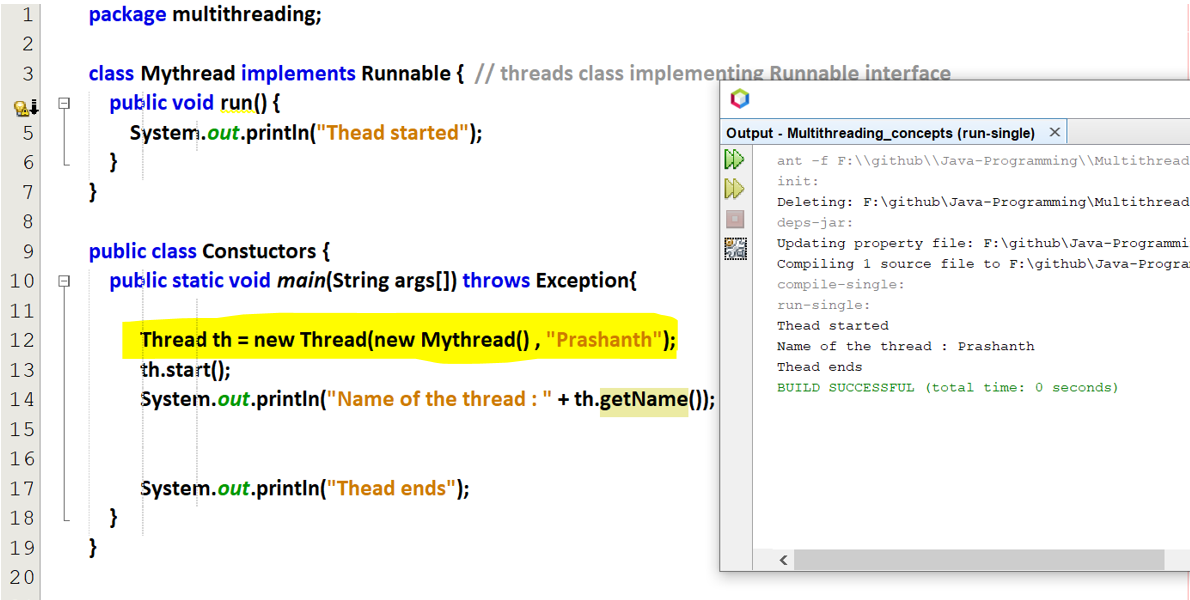
## **Constructors in thread class**



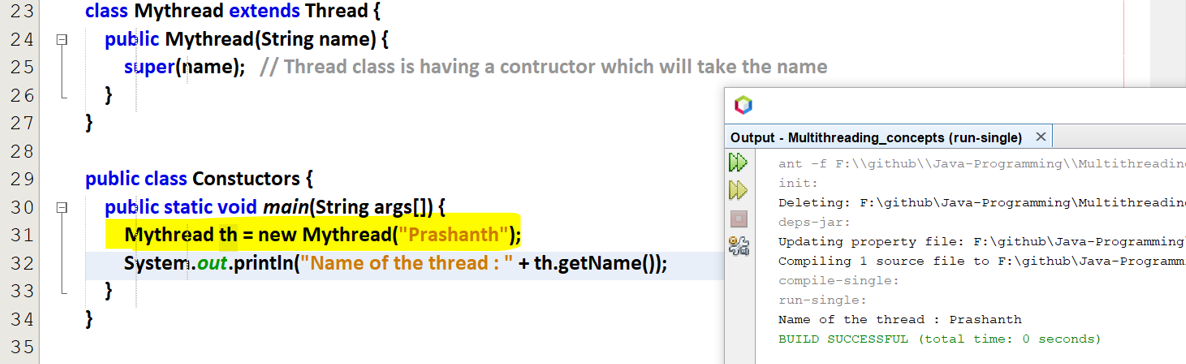


ThreadGroup 🡪 we can create a group of threads and make a sync among them.  
Eg: While designing a game, we assign all threads to all small-small operations through out the game and have a complete control over-it. So if we pause that game then all the threads will be stopped.

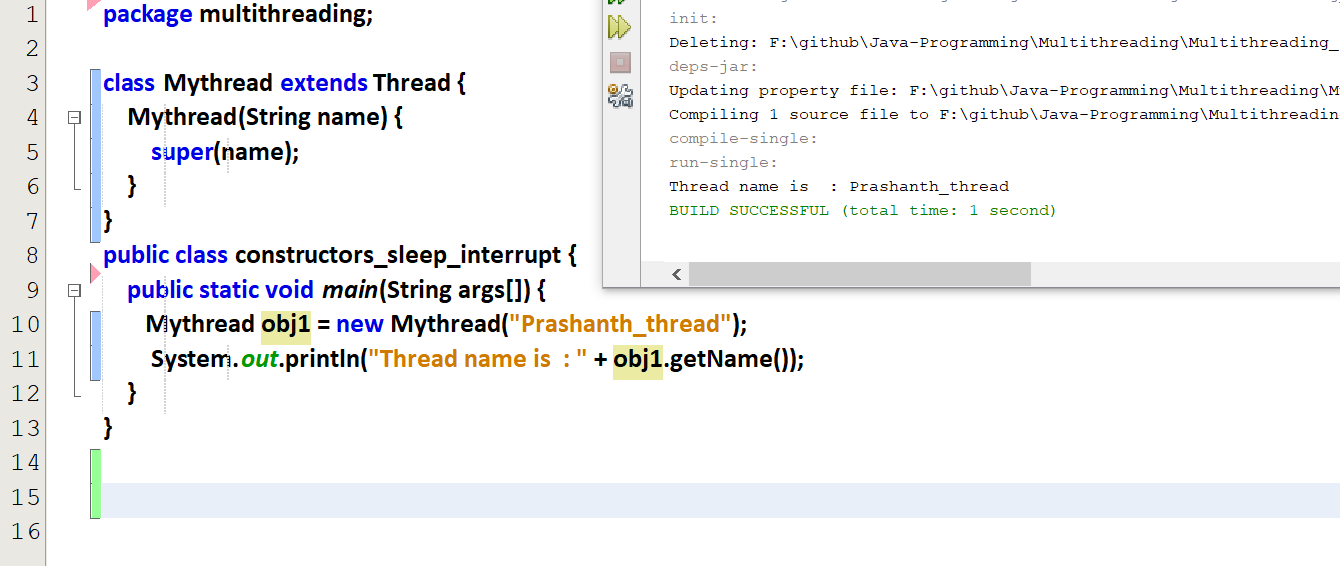
### **Thread(Runnable target, String name)**



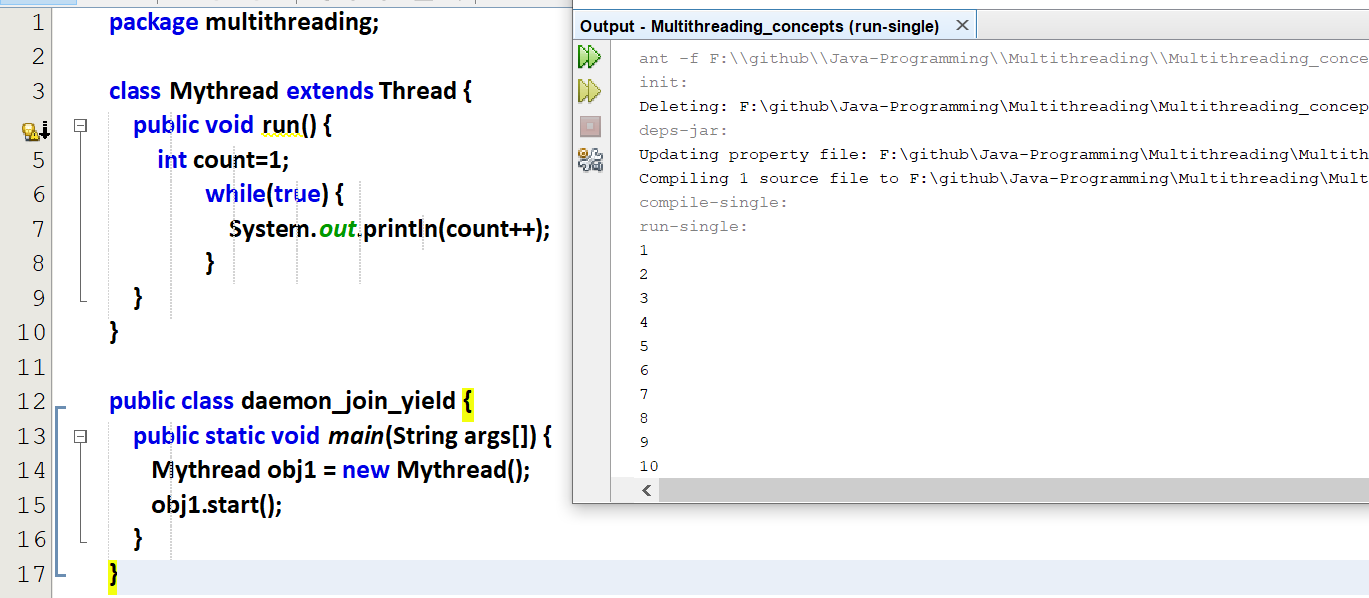
### **Thread(String name)**



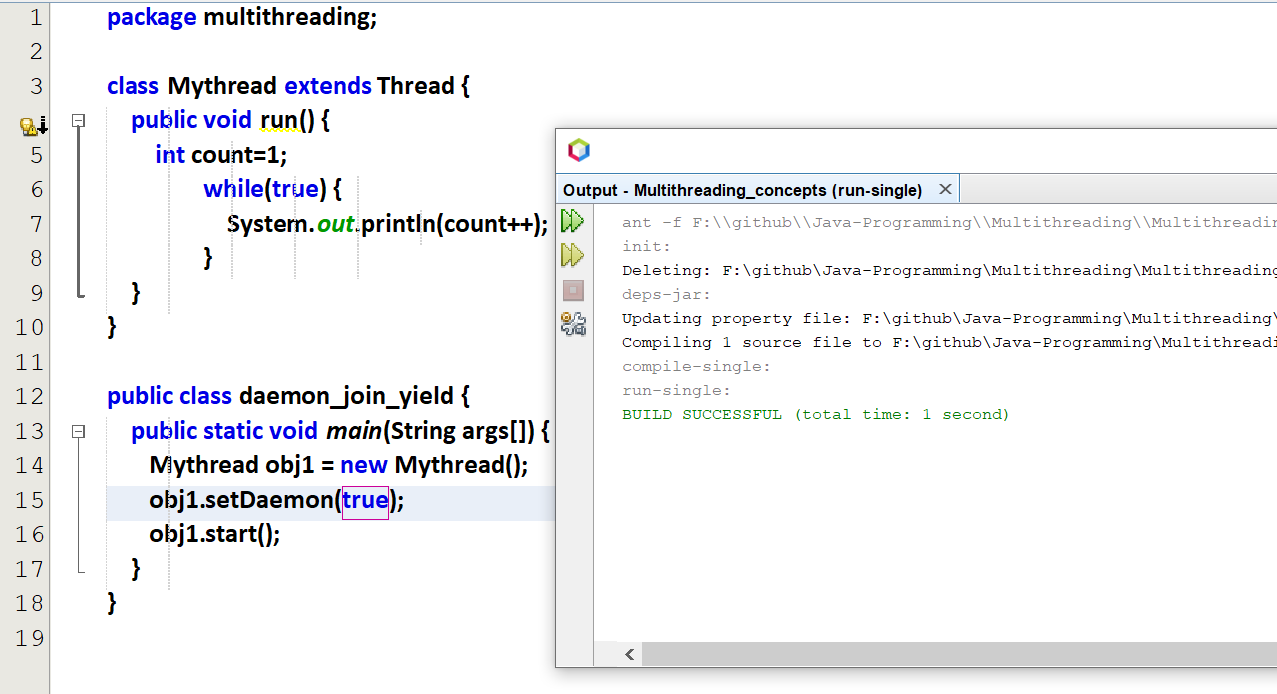
We can get id of a thread, but we cannot set id of a thread.  
We can get and set a name to a thread.



Daemon thread 🡪 A thread which is having very least priority.  
Eg: In MS-Word, autosave option is done only at the last, so the autosave option is set to as demon thread. So that it happens at the last.  
If we give the boolean d 🡪 true [ Demon ]   
If we give the boolean d 🡪 false [ Not Demon ]

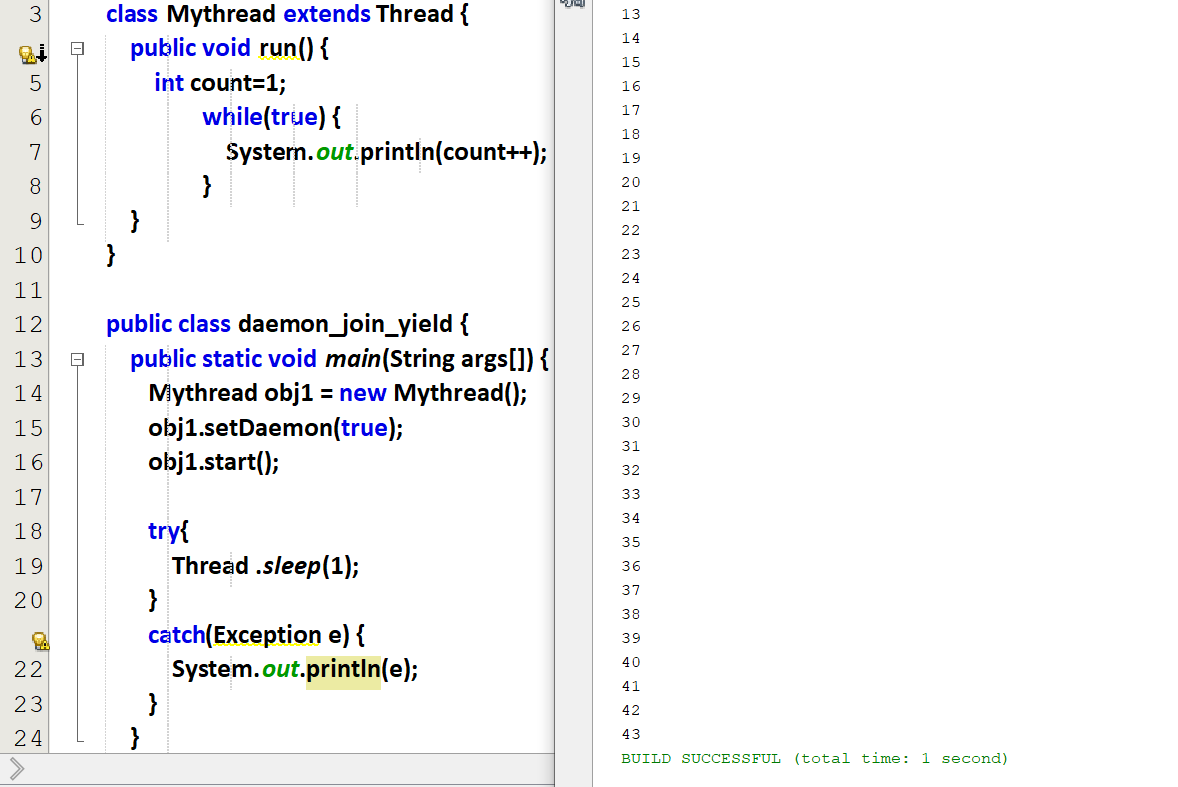


When the thread (i.e obj1) stops, main() thread also stops and the program ends. Here main() is waiting for the thread (i.e obj1) to finish. So we are forcefully terminating the process.



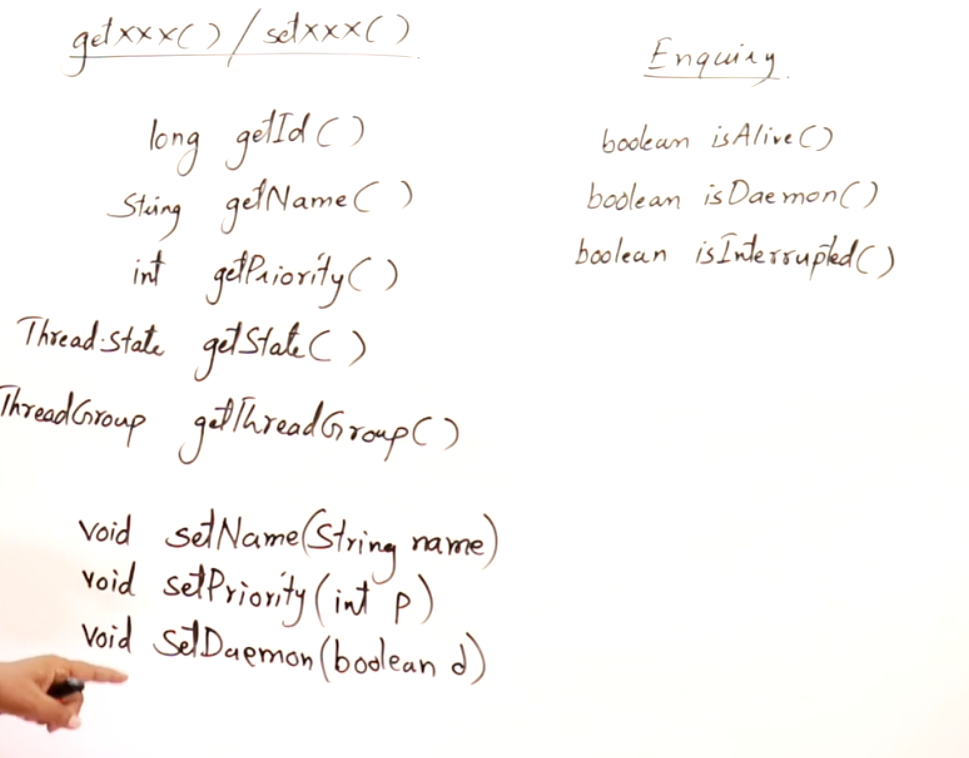
When the main() thread is terminating, thread (i.e obj1) is also terminating. Main() is not waiting for obj1 thread.

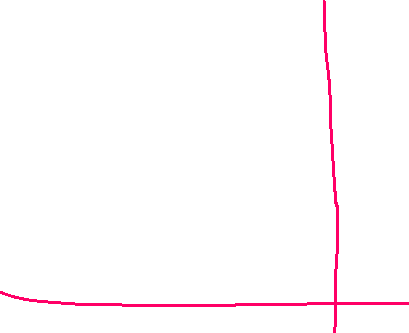
In-order to observe the output.



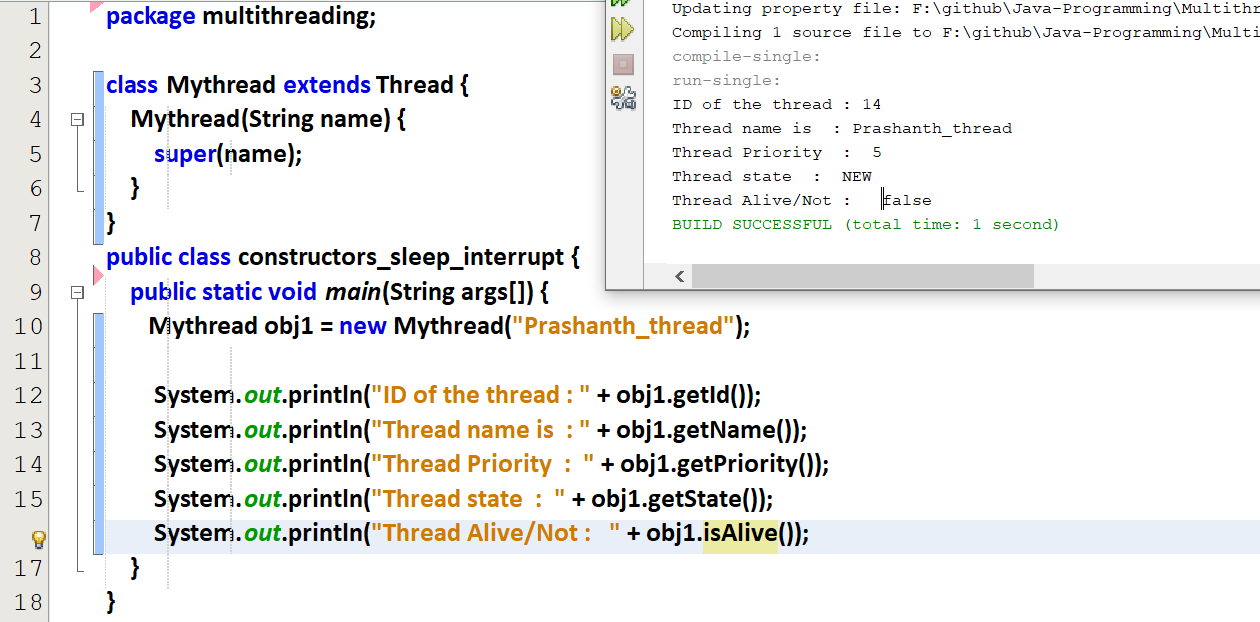
Now the main() thread is sleeping for 1 milli second and exits . In that 1 milli second obj1 thread is executing the infinite loop and then exiting before main().

## **Get and Set method in thread class**



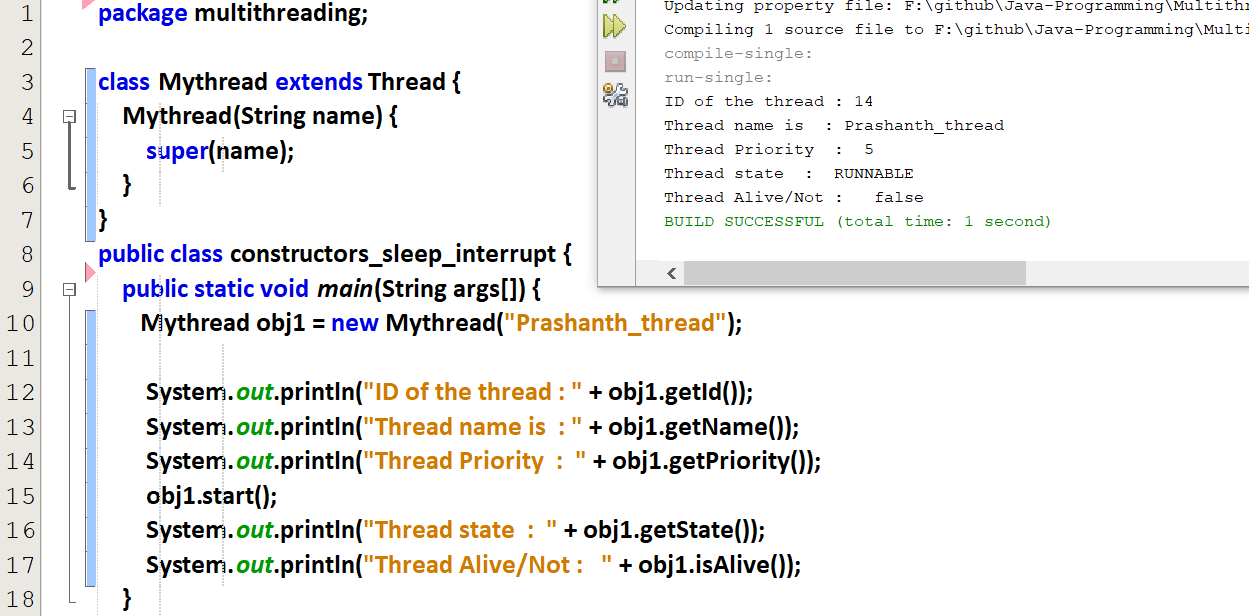


We can interrupt a thread by stopping the thread for a while or we can terminate a thread.  
If a thread is sleeping, we can interrupt and stope them sleeping. A thread can interrupt it-self or a thread can interrupt some other thread.





Here the thread is running

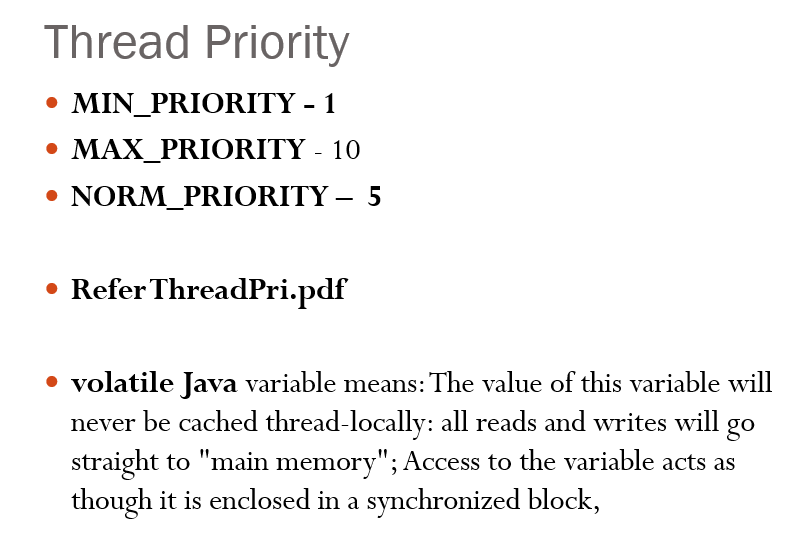




Just now the thread has started to run, so only the state = RUNNABLE

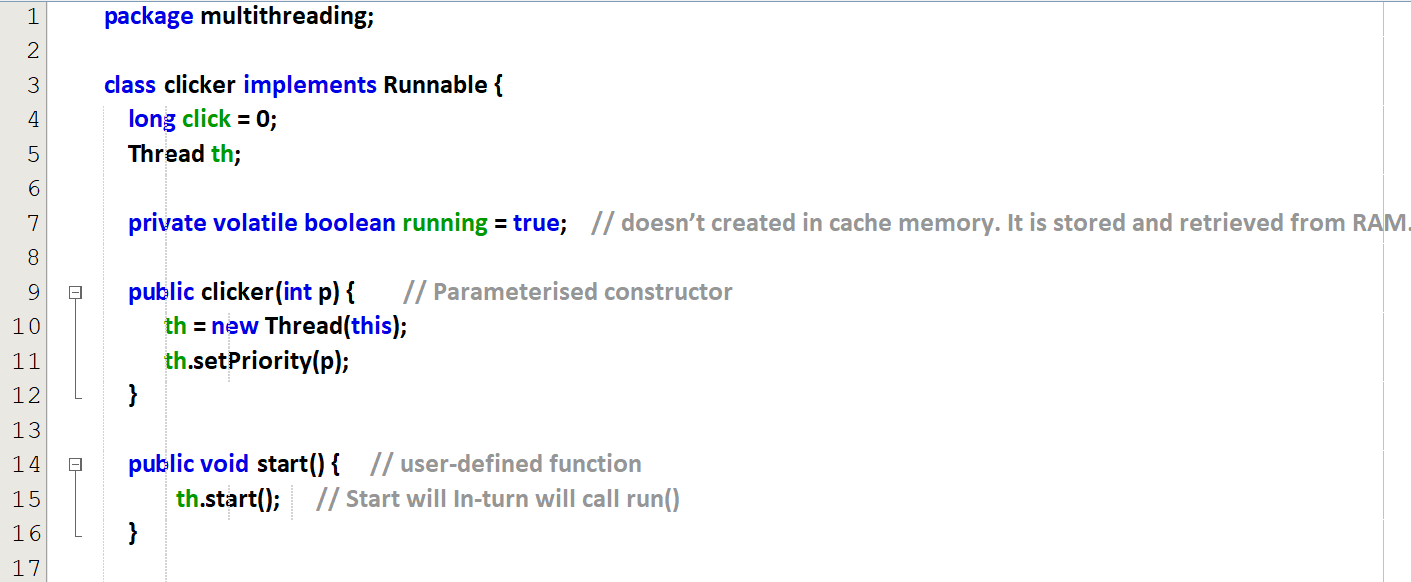
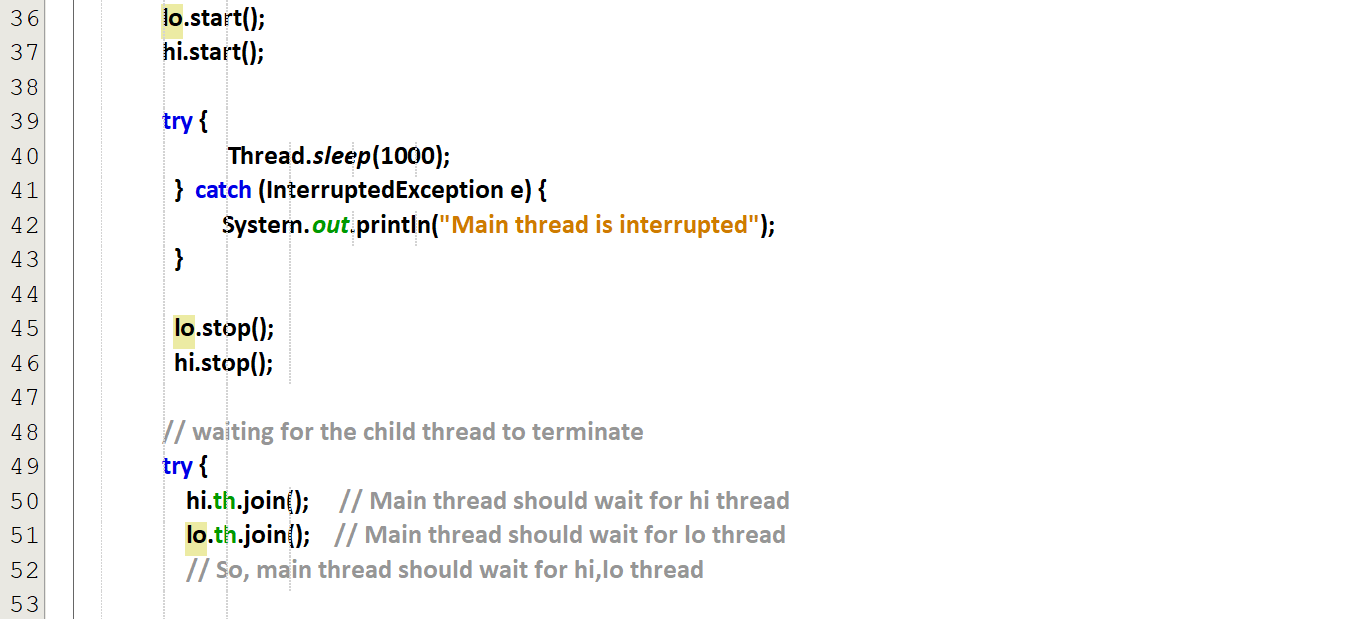
### **Thread Priority**

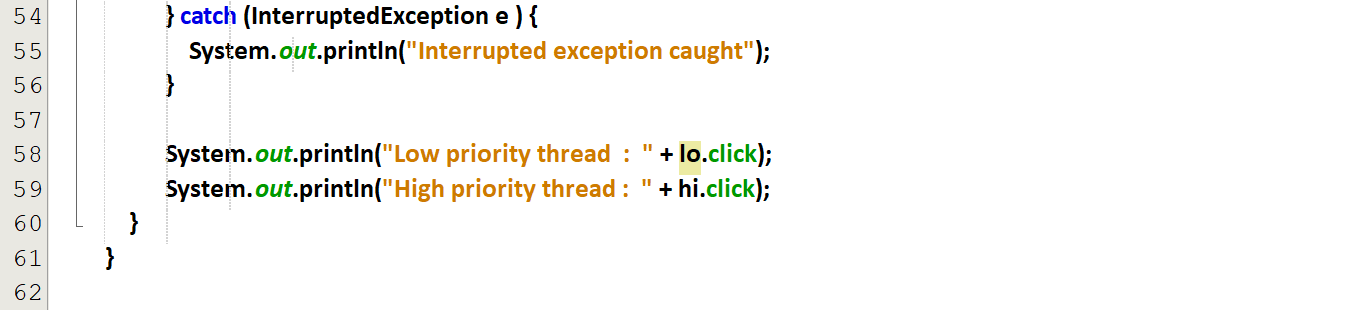
Set the priory of a thread



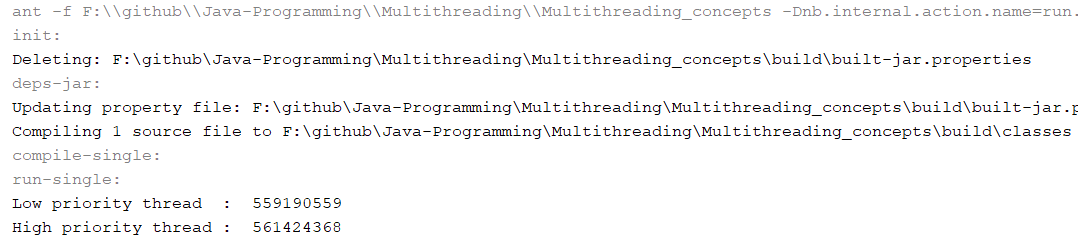
volatile 🡪 doesn’t created in cache memory. It is stored and retrieved from RAM.

Refer thread\_priority.JAVA

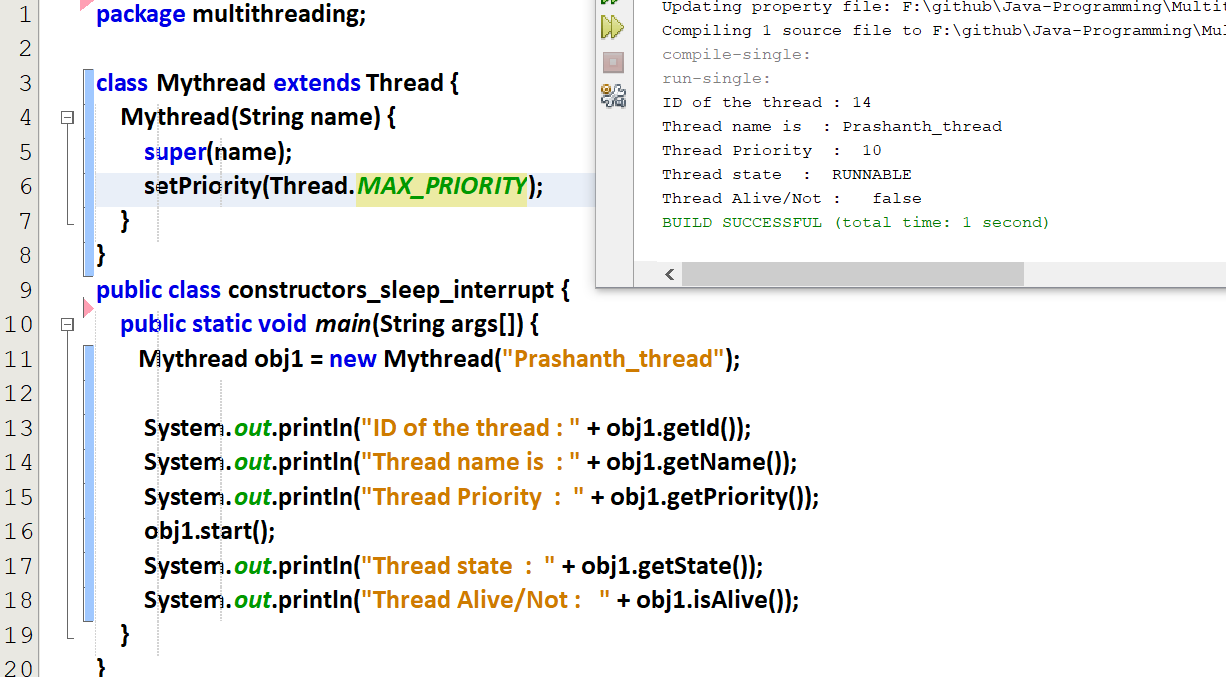
  
  




Output

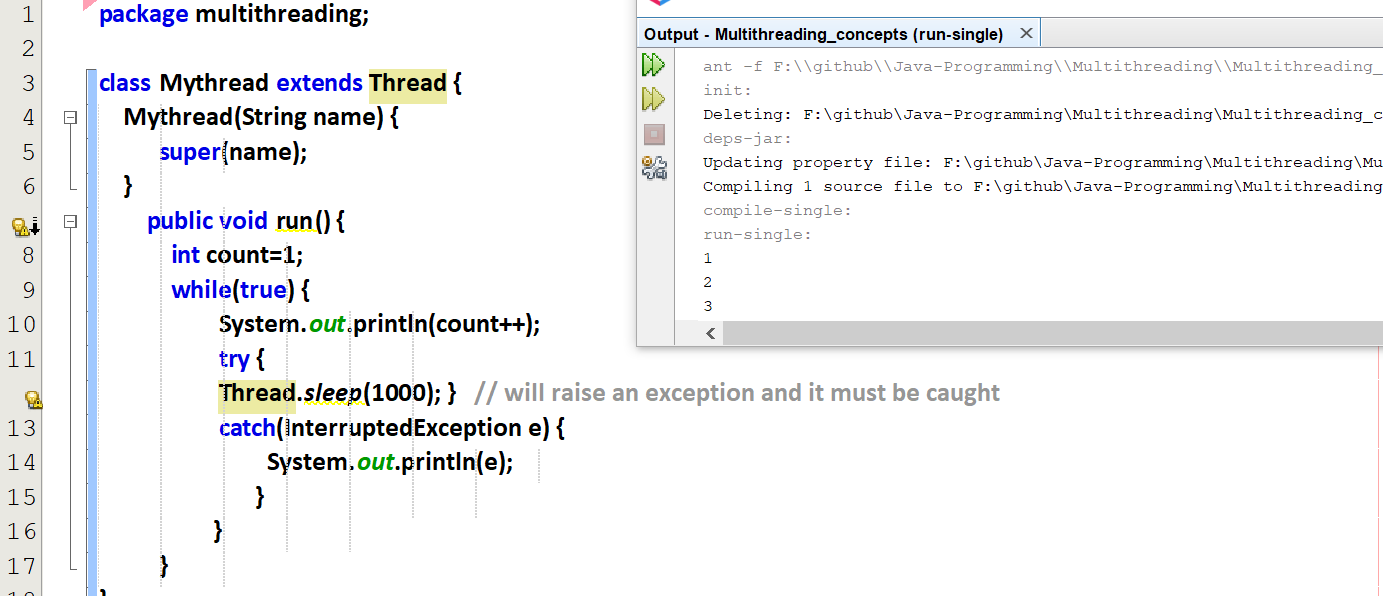


li 🡪 having low priority since it gets very less CPU time.  
hi 🡪 having high priority since it gets very more CPU time.

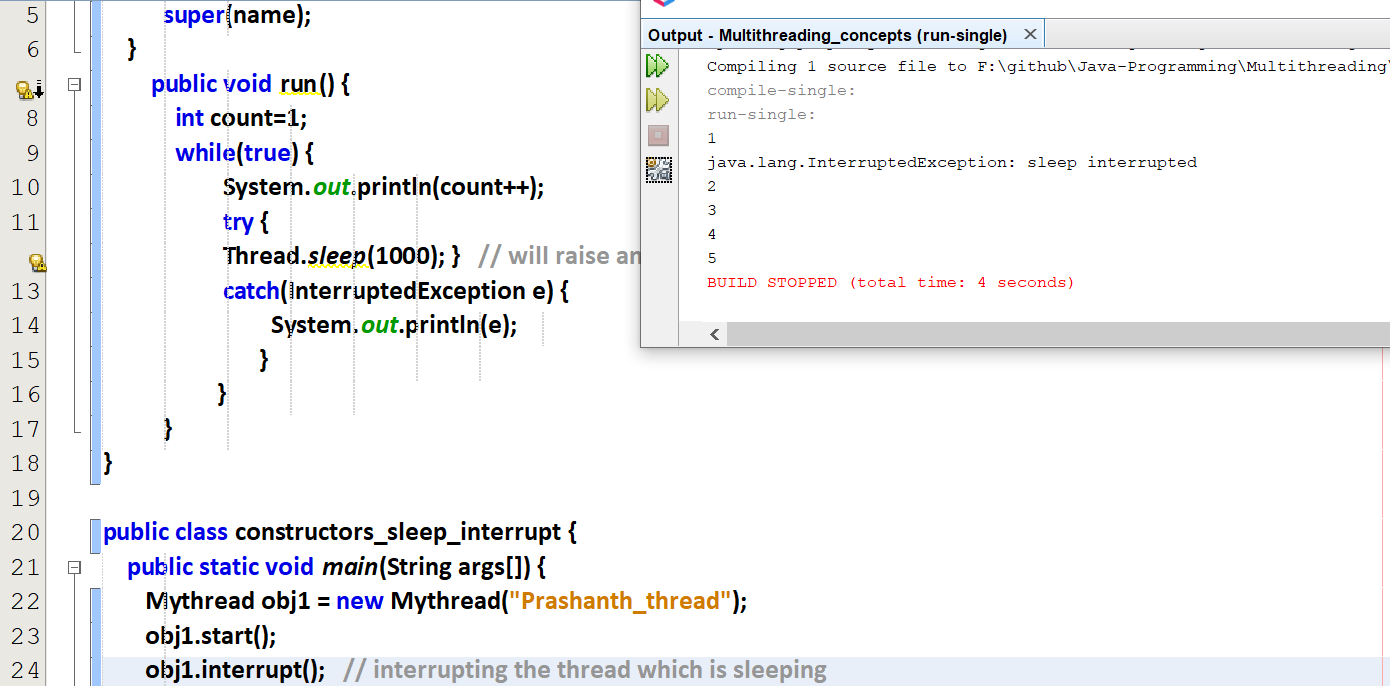




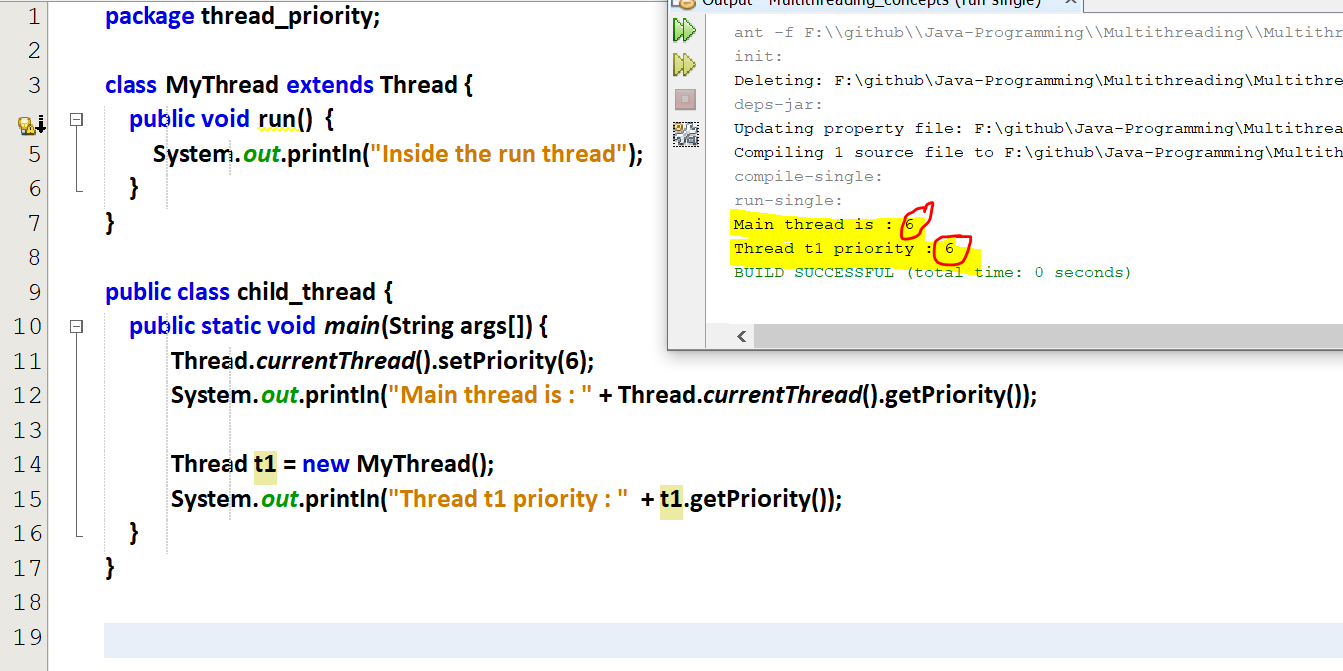
Using sleep



Output will be slow since the thread will sleep for 1000 milli seconds.  
Thread.sleep() 🡪 will raise an Exception and it must be caught. So only using try and exception.

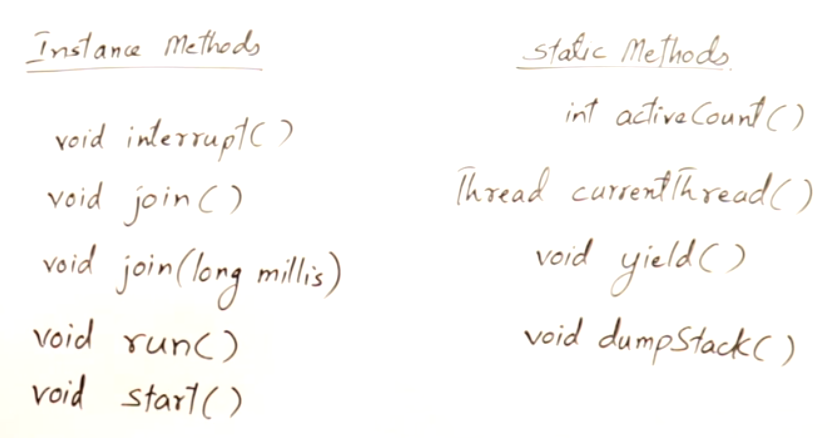
Interrupting the thread purposefully from main()

The thread is interrupted and it is caught using try and catch.



Thread t1 is the child of main thread, so t1 will also have the priority of the main thread

## **Instance method in Thread class**



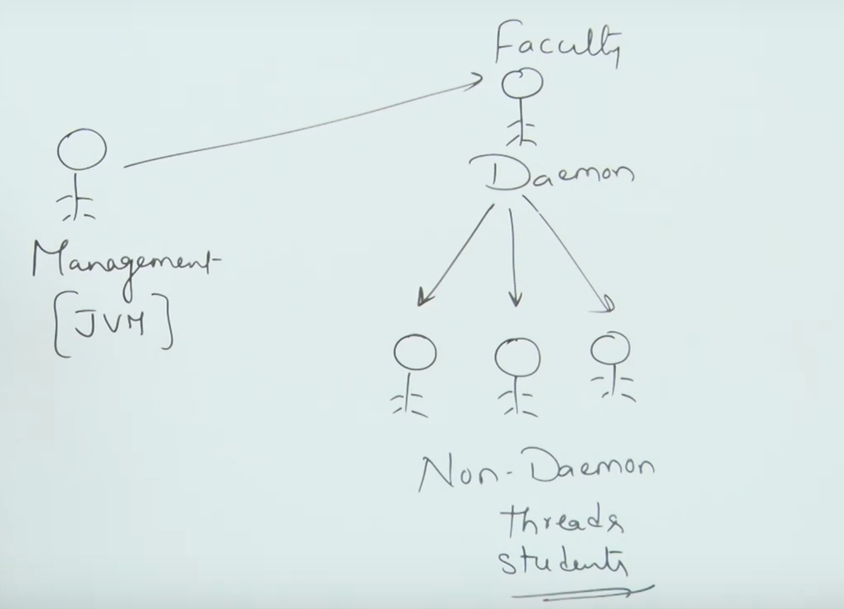
### **void interrupt()**

Can interrupt itself/can interrupt a thread.

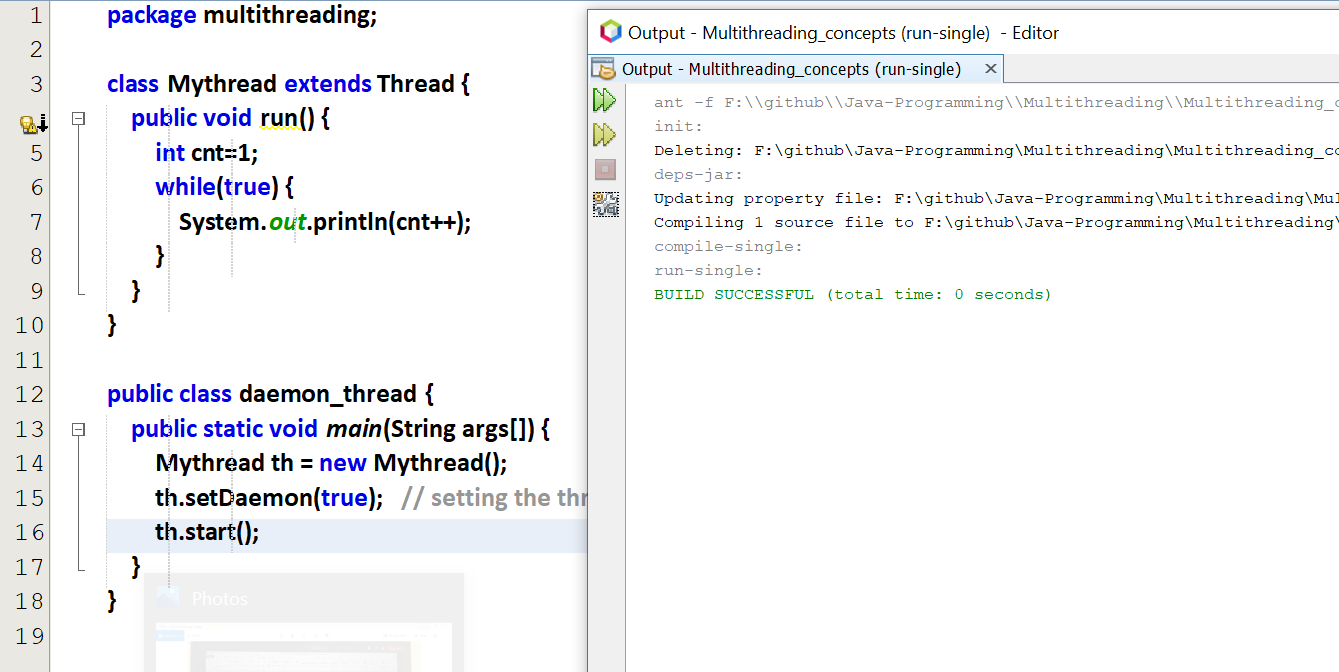
### **Daemon thread (daemon 🡪 invisible)**

Daemon threads (back-end) [i.e background logic]  
Non-daemon threads (front-end) [i.e getting input from the user, displaying the output]

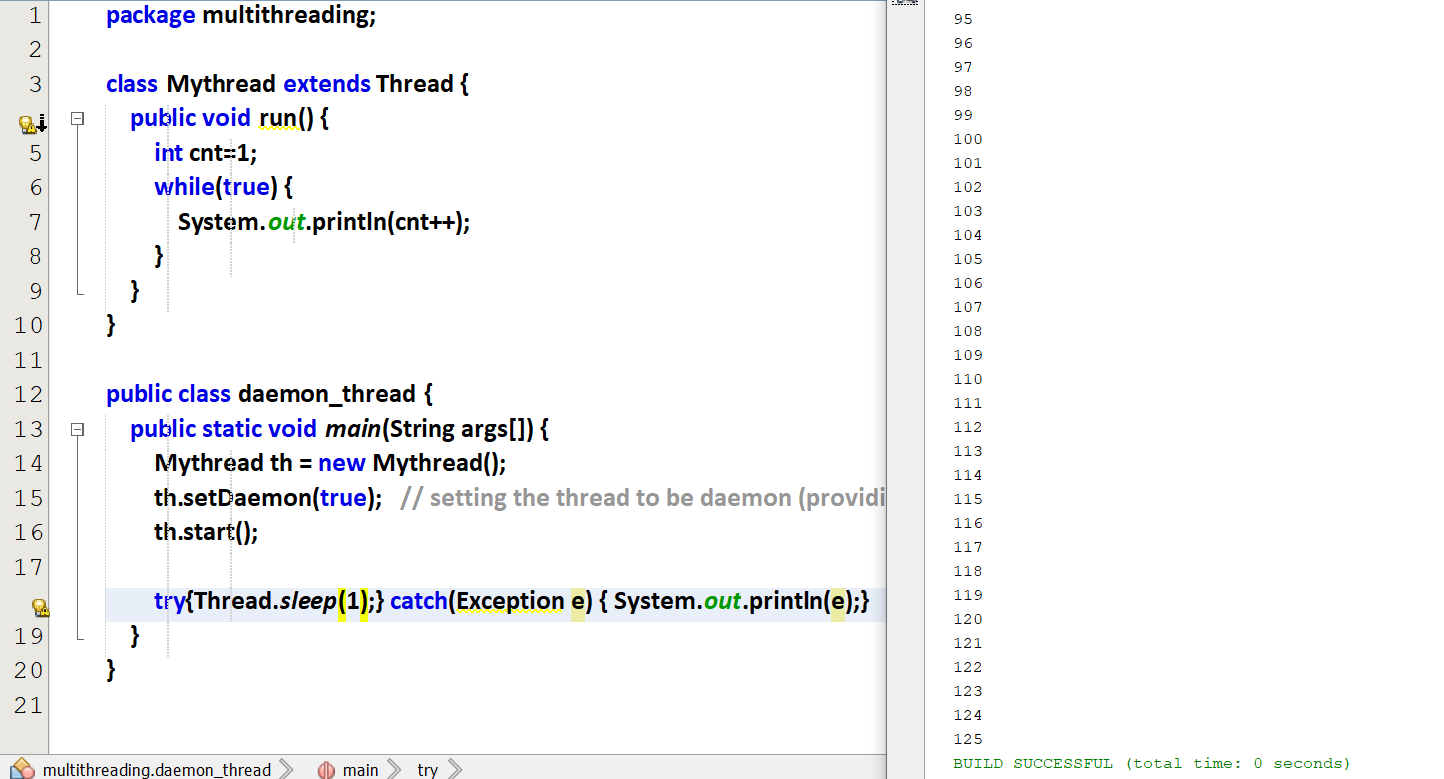
Daemon threads provides service to Non-daemon thread. When all non-daemon threads finishes its execution, JVM forcefully stops all the daemon threads.



When there are no students in the class(No Non-Daemon threads), then there is no job for the faculty (Daemon threads). So Management(JVM) will automatically stops the Daemon thread.



run() is executed but If the main application is terminated, then the daemon thread is also terminated, So the execution of run() is not shown.

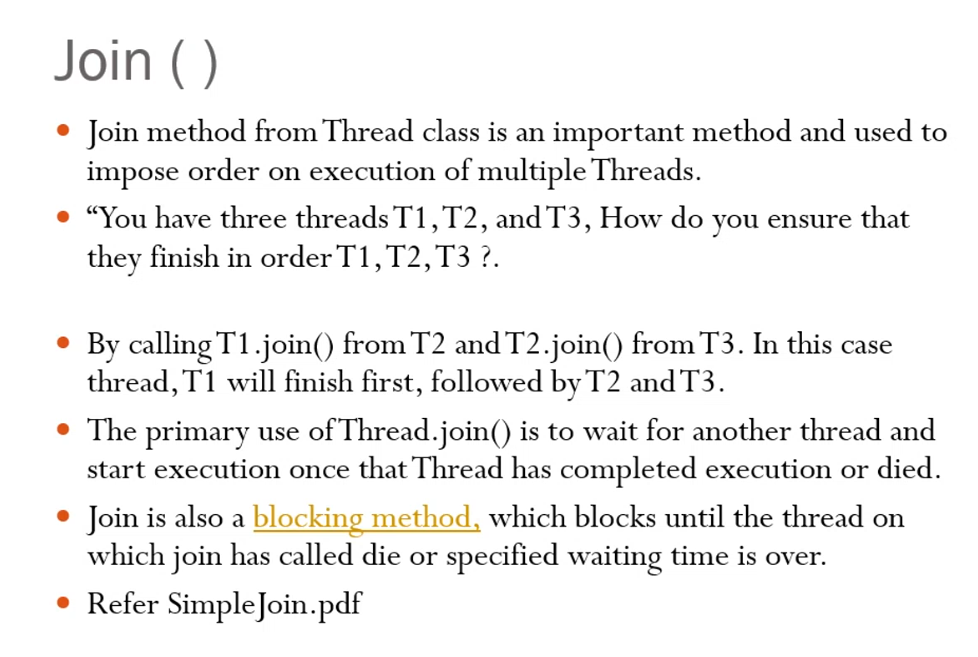


So before terminating main() will sleep for a while, so the execution of run() can be seen very easily. After 125, main() will wake up from the sleep and ends, so the daemon also ends.

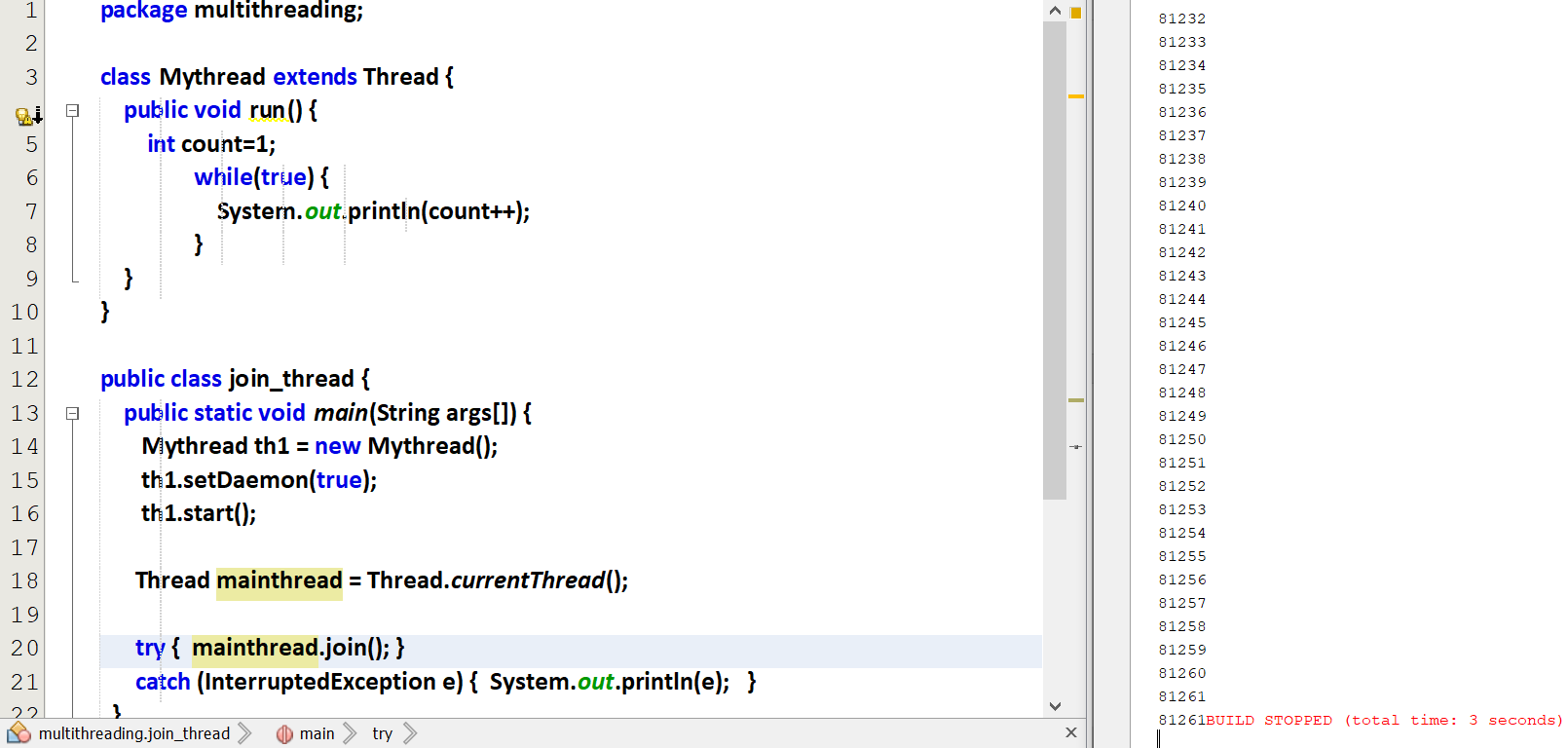
So how to do make main() to wait for the other threads to complete its execution. So that main() will end after all other main()’s child threads are executed. This can be done by using join.

### **void join()**

Refer join\_thread.java file



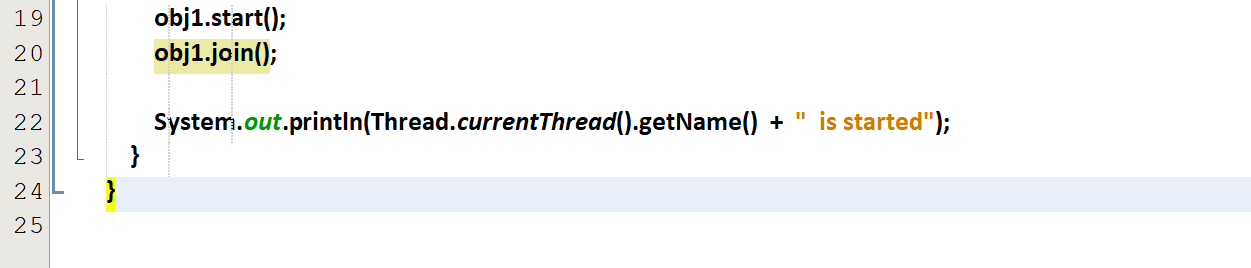
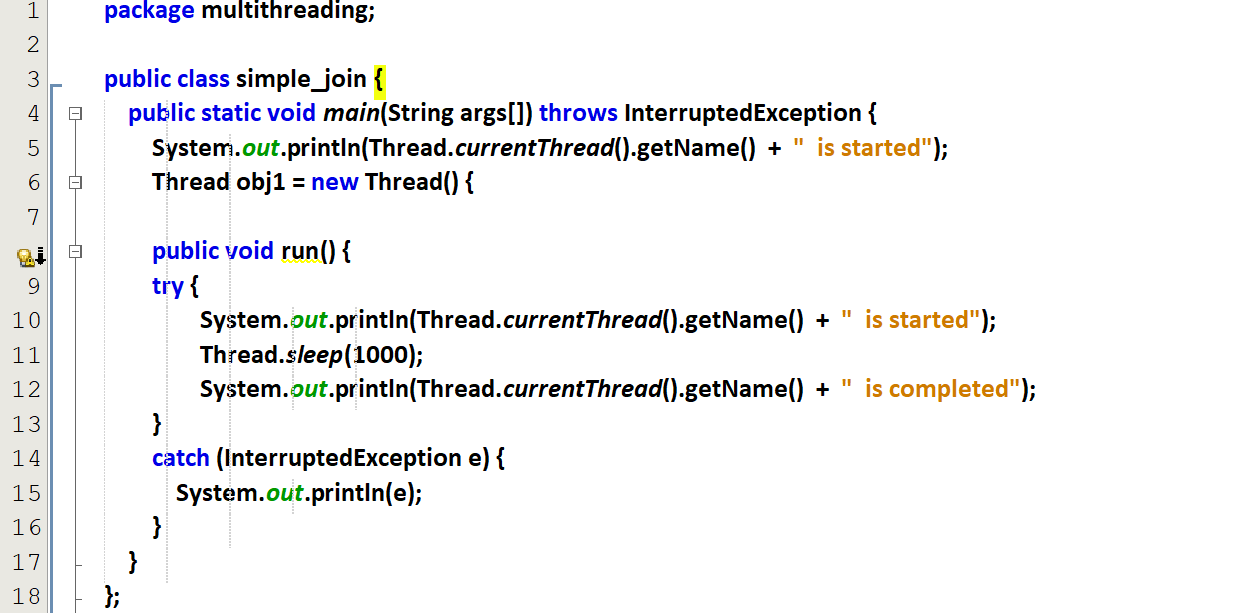
***Generally main thread will not wait for their child thread to finish.  
If the child thread wants the main thread to wait.  
Child thread wants to specify thread.join() in main thread area.***

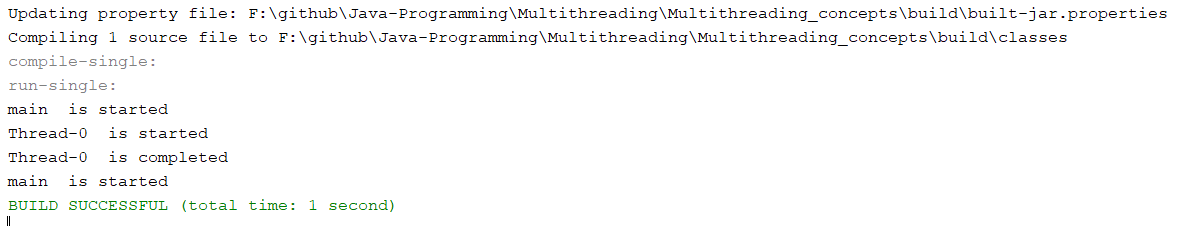


The program goes on an infinite loop, since without exiting th1 main() will not exit.  
main() will keep on waiting for th1.

Join is a function which helps us to wait till other thread finishes its execution.

main will wait till the child thread finishes because of obj1.join()



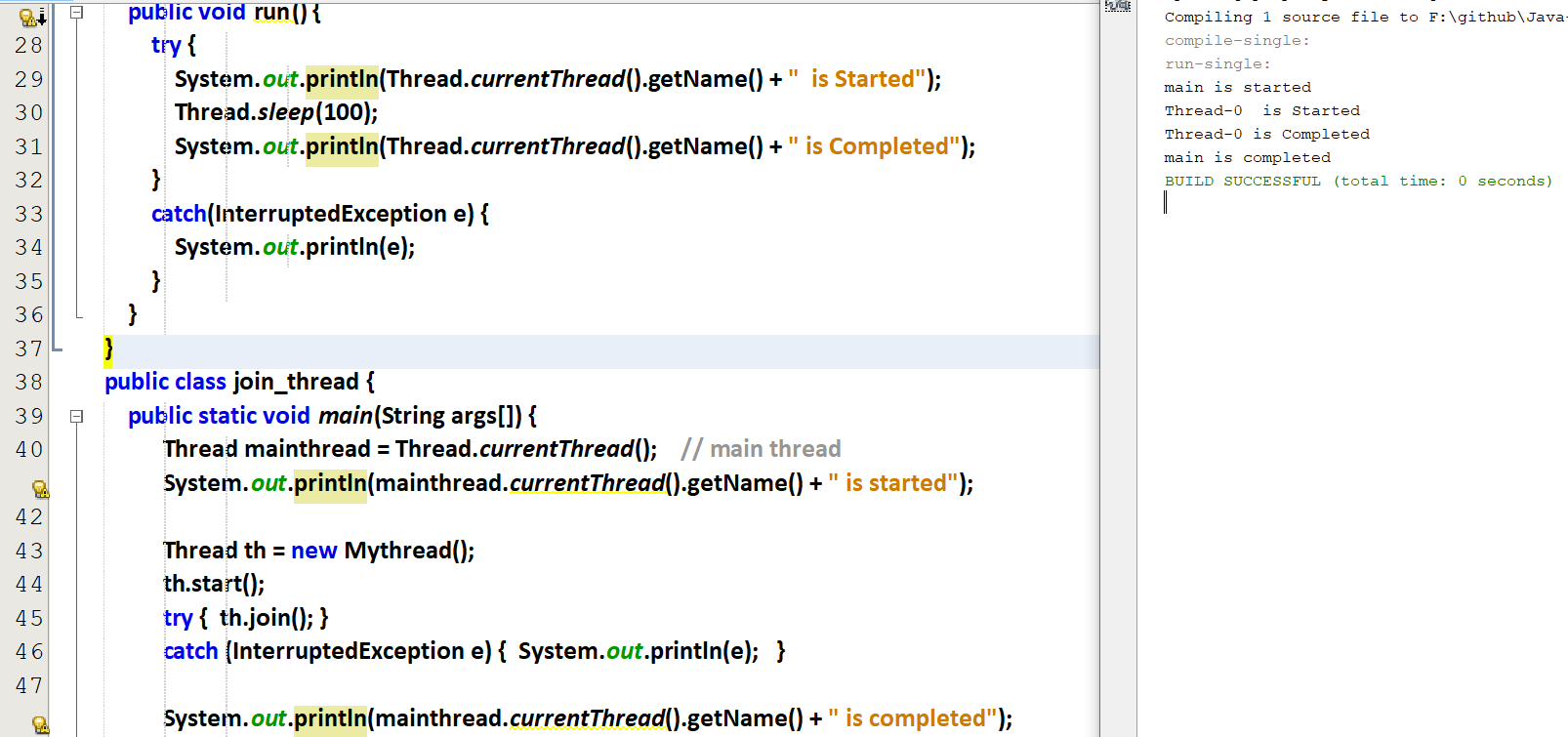


Here main() waits till thread-0 completes its execution.

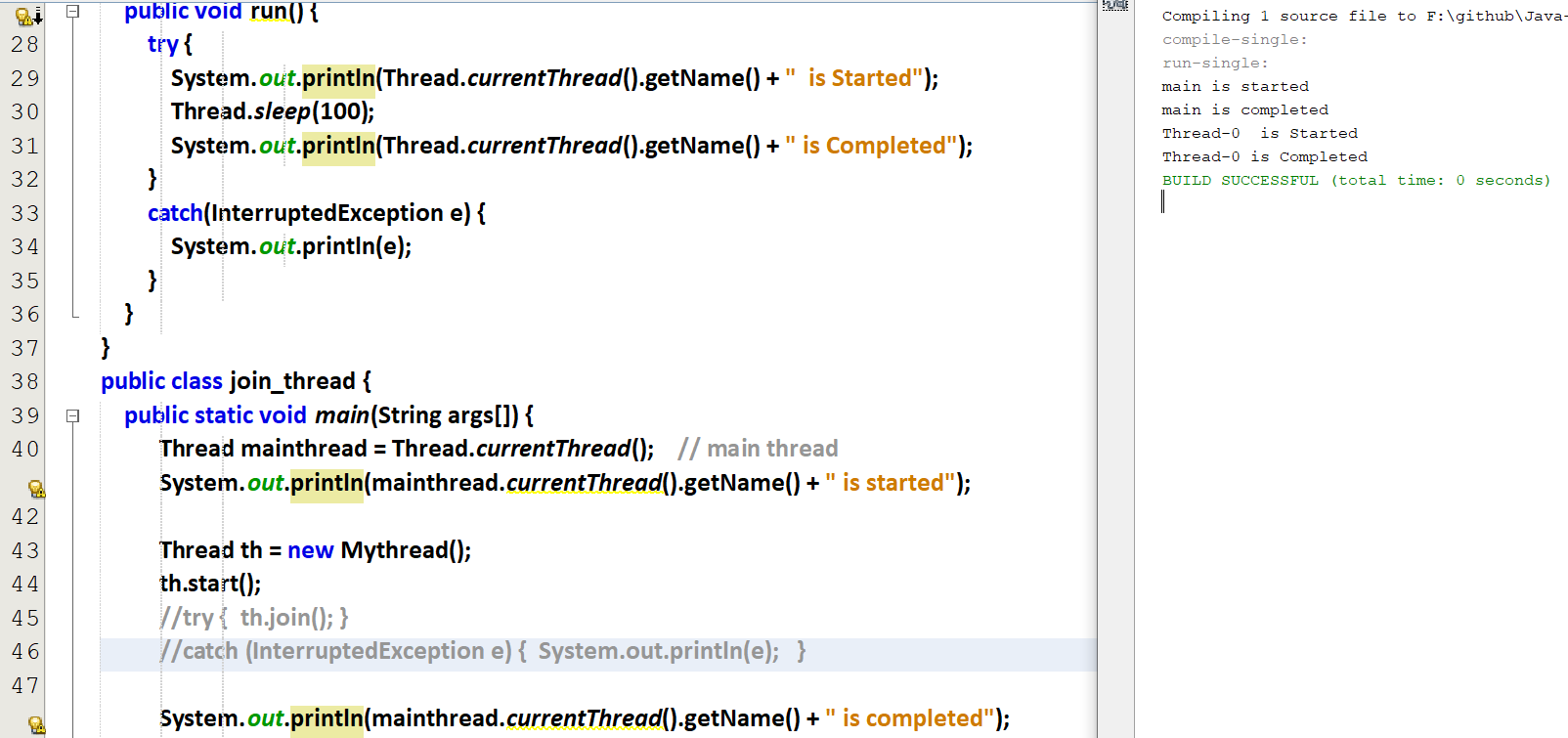
Without join()

Now main() will not-at-all wait for other threads.

The same above code and concept in different aspect



main() is waiting for th1.

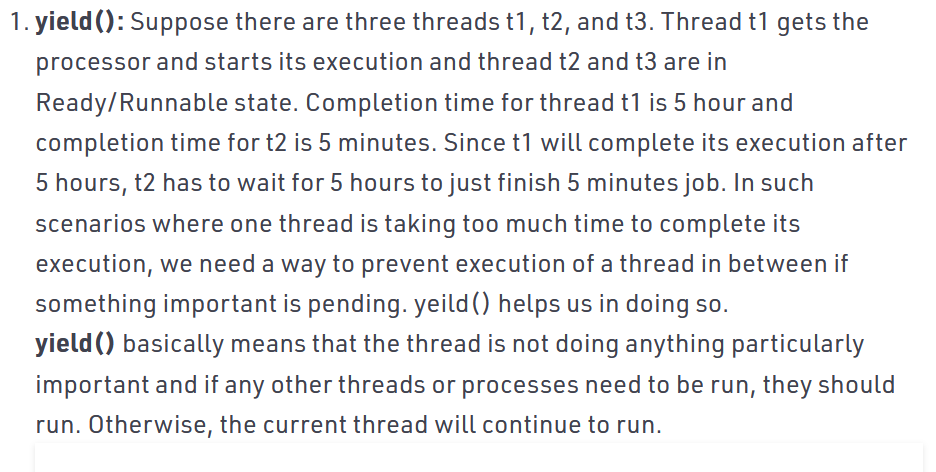


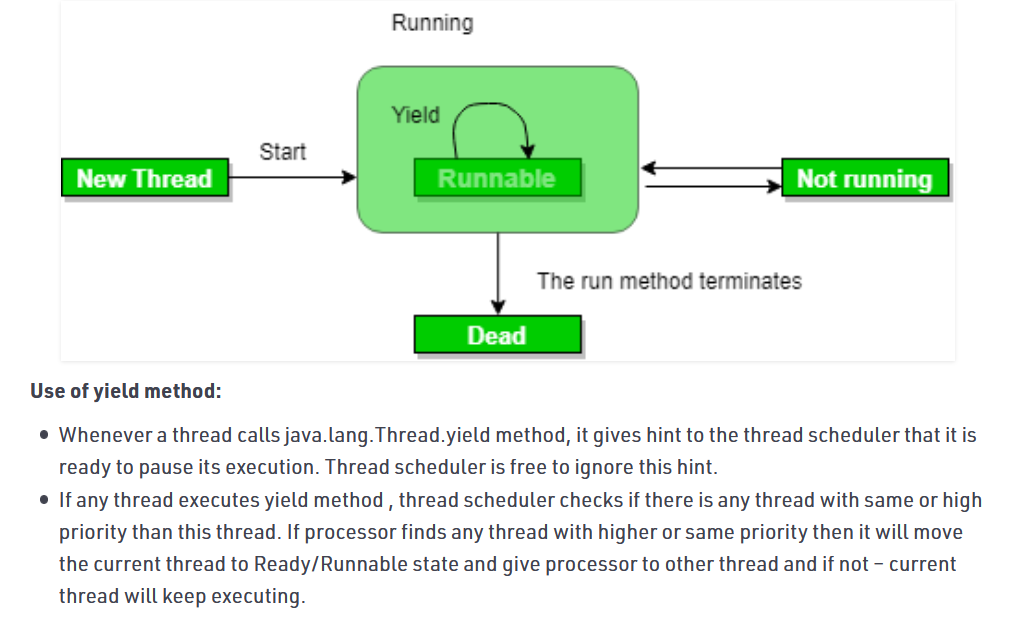
main() is not at-all waiting for th1.

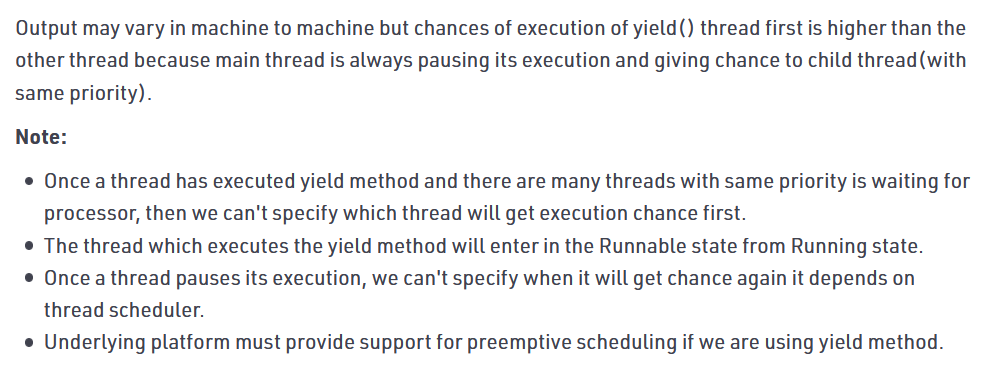
### **void join(long milliseconds)**

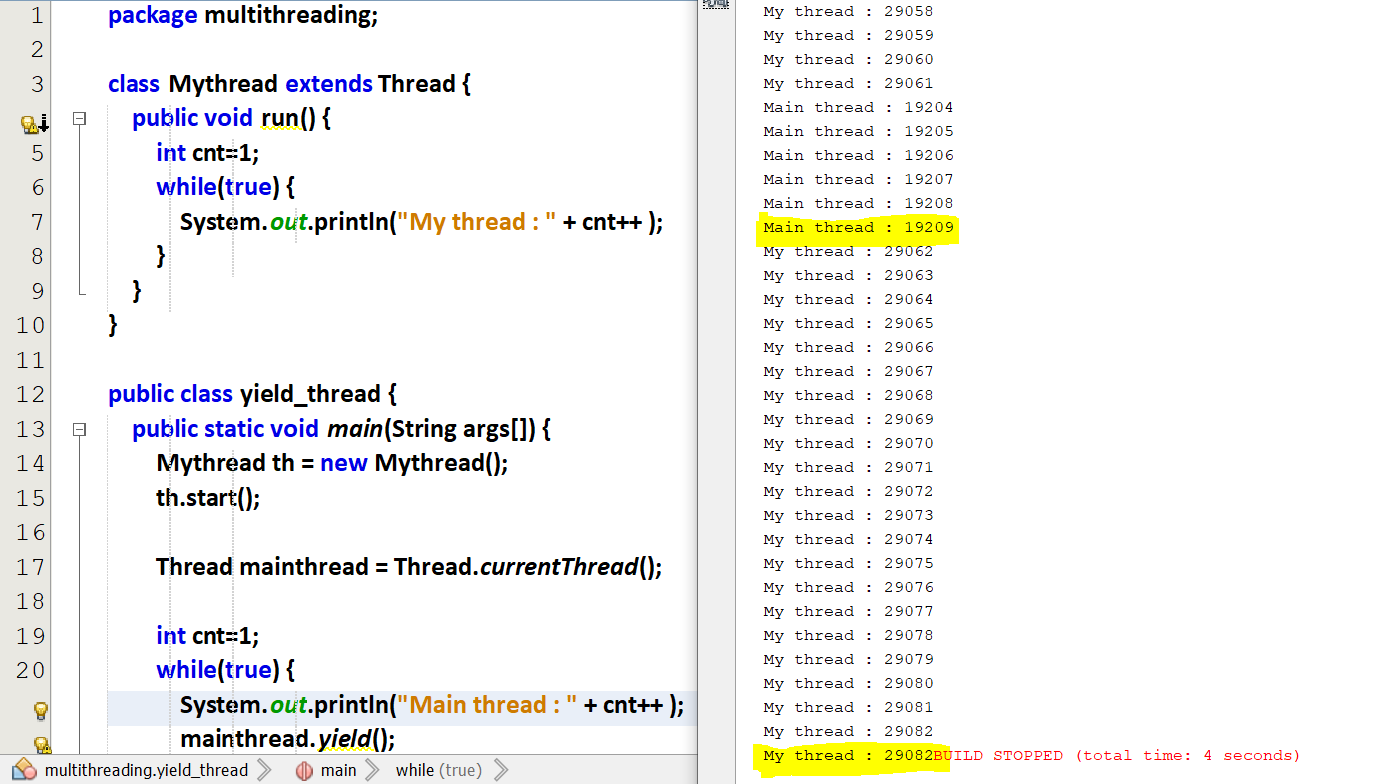
We can also make a thread(t1) to wait for specific time-period for the thread (t2) rather waiting for the complete execution of the thread t2.

### **void yield()**

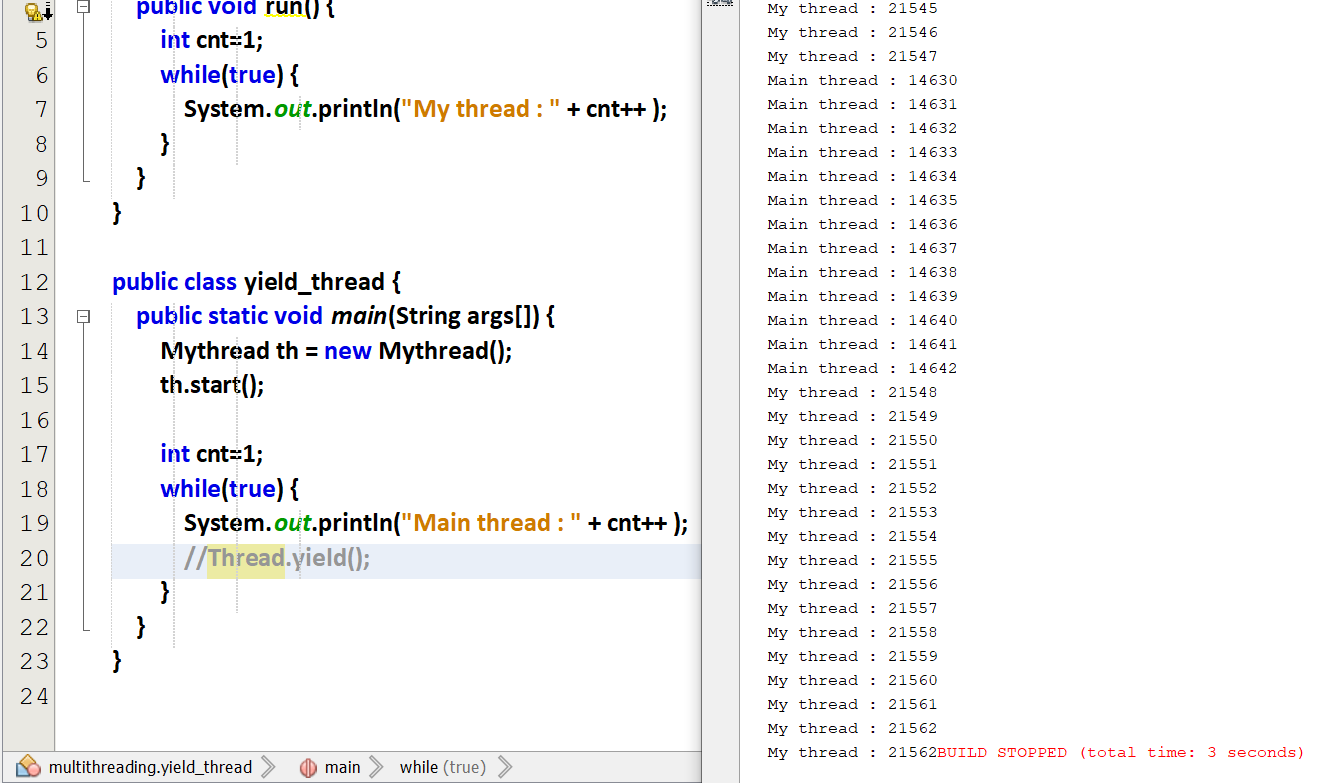








With yield, Mythread(29082) has got more chance to execute than Mainthread(19209 times).



DOUBT Without yield, Mythread has got more chance to execute than Mainthread.

### **void run() and void start()**

## **Static method in Thread class**

### **int activeCount()**

How many threads are active in a group ??

### **Thread currentThread()**

We can get a refence to the current thread.

### **void yield()**

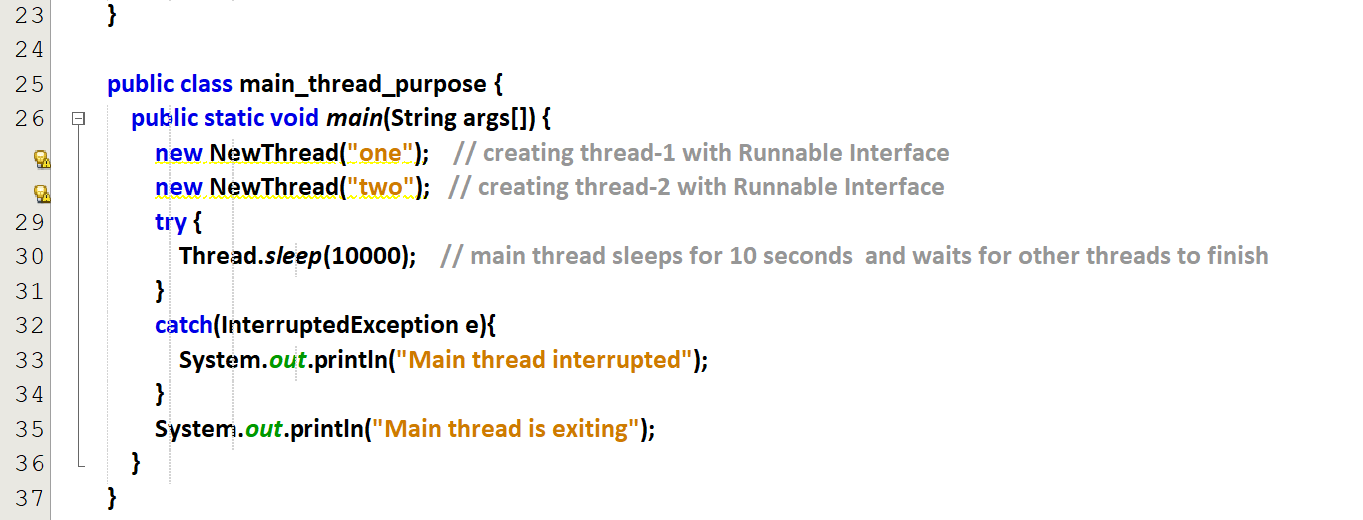
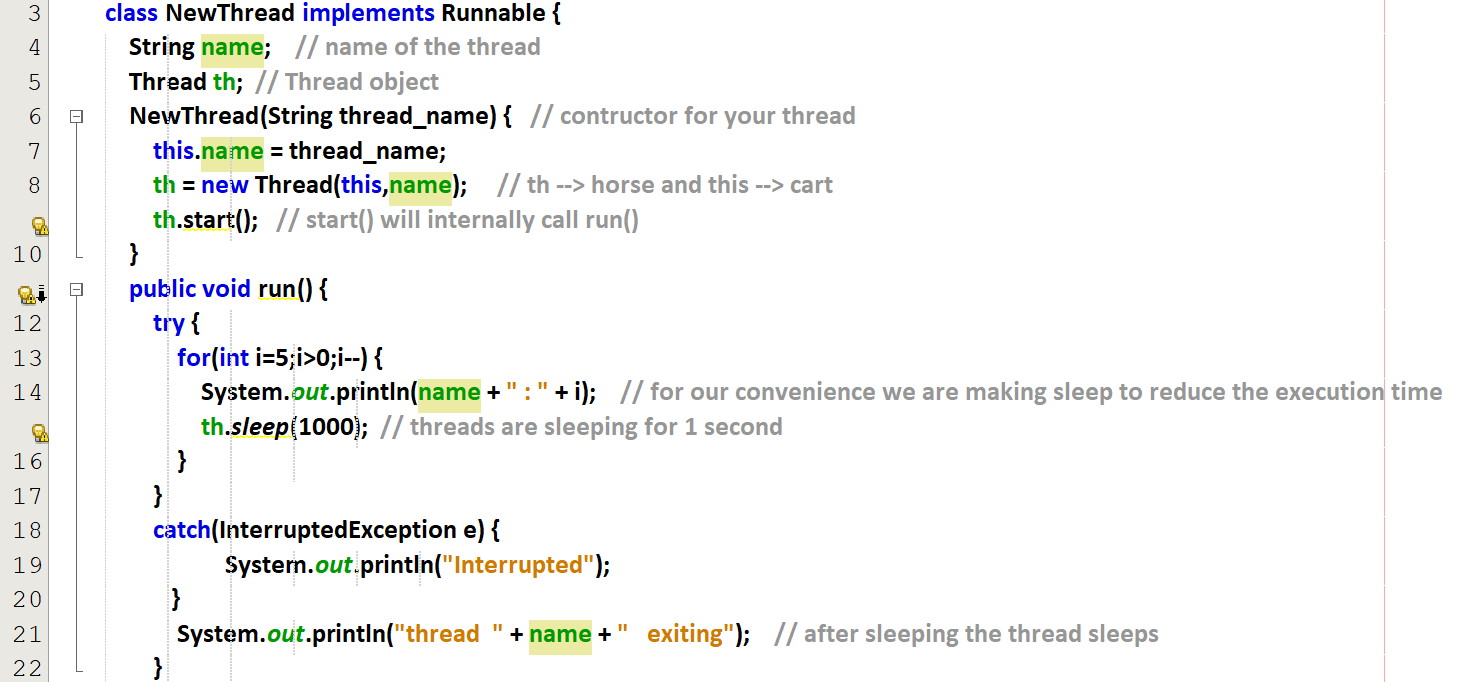
In a restaurant   
Person1 🡪 Ordering 10 different items.  
Person2 🡪 Ordering just 1 small item.  
Server guy taking order from Person1. Suddenly person1 asks the server guy to finish serving Person2 first and then come to me. So person1 can yield person2.

Like that a higher priority thread can give chance for the low priority threads to execute and then gets the CPU back to execute the remaining high priority threads. So high priority thread can yield all the low priority threads.

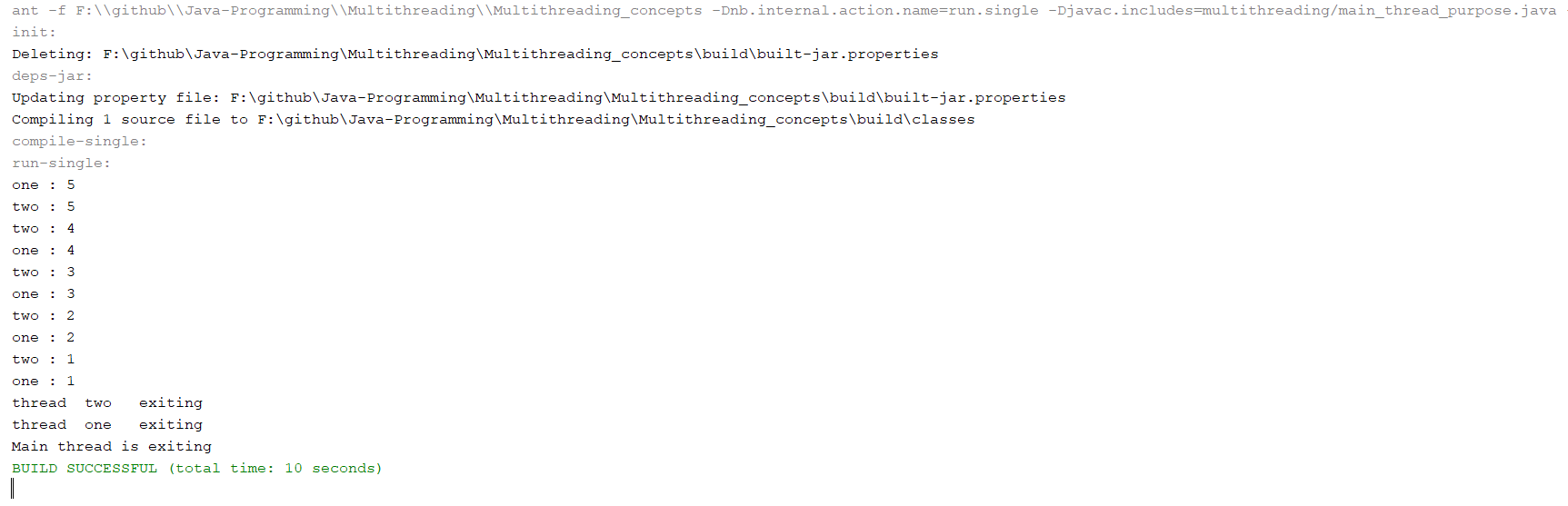
### **void dumpStack()**

We can the sequence of thread calls.

# **Controlling the Main thread via sleep**



main thread will exit only after the child thread exits since   
thread-1(1s)  
thread-2(1s)   
main\_thread(1000s)



main thread will exit before the child thread exits since   
thread-1(1000 millisecond)  
thread-2(1000 millisecond)   
main\_thread(100 millisecond)

Hence by changing the sleep\_time() we can make any thread of our project to exit.

### **isAlive() and join()**

