**Multi-Threading**

Multitasking is the simultaneous execution of multiple tasks or processes over a certain time interval.

Windows operating system is an example of multitasking because it is capable of running more than one process at a time like running Google Chrome, Notepad, VLC player, etc. at the same time.

The operating system uses a term known as a process to execute all these applications at the same time.

A process is a part of an operating system that is responsible for executing an application.

Every program that executes on your system is a process and to run the code inside the application a process uses a term known as a **thread**.

A thread is a lightweight process, or in other words, a thread is a unit which executes the code under the program. So every program has logic and a thread is responsible for executing this logic.

Every program by default carries one thread to executes the logic of the program and the thread is known as the **Main Thread**, so every program or application is by default **single-threaded model**.

This single-threaded model has a drawback:

* The single thread runs all the process present in the program in synchronizing manner, means one after another.
* So, the second process waits until the first process completes its execution, it **consumes more time** in processing.
* For example, we have a class named as Geek and this class contains two different methods, i.e. method1, method2. Now the main thread is responsible for executing all these methods, so the main thread executes all these methods one by one.

**Multi-threading** is a process that contains multiple threads within a single process.

Here each thread performs different activities.

For example, we have a class and this call contains two different methods, now using multithreading each method is executed by a separate thread.

So the major advantage of multithreading is it works simultaneously, which means multiple tasks execute at the same time. And also maximizing the utilization of the CPU because multithreading works on time-sharing concept mean each thread takes its own time for execution and does not affect the execution of another thread, this time interval is given by the operating system.

Whenever you create a process, a separate memory area is occupied. But threads share a common memory area.

**Advantages of Multithreading:**

* It executes multiple process simultaneously.
* Maximize the utilization of CPU resources.
* Time sharing between multiple process.

**System.Threading** Namespace

The System.Threading namespace contains classes and interfaces to provide the facility of multithreaded programming.

It also provides classes to synchronize the thread resource.

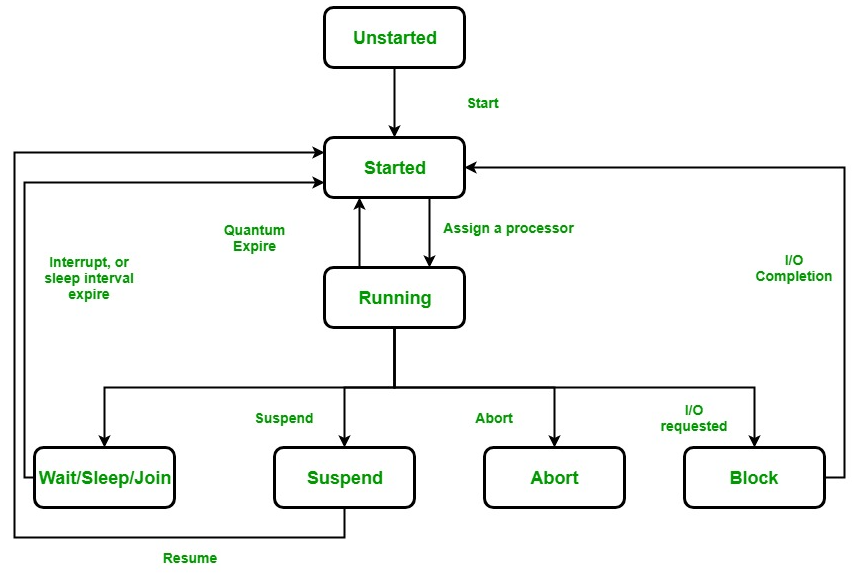
A list of commonly used classes are given below:

* Thread
* Mutex
* Timer
* Monitor
* Semaphore
* ThreadLocal
* ThreadPool
* Volatile etc.

**Thread Life Cycle**

In C#, each thread has a life cycle. The life cycle of a thread is started when instance of **System.Threading.Thread** class is created.

When the task execution of the thread is completed, its life cycle is ended.



There are following states in the life cycle of a Thread in C#.

* Unstarted :

When the instance of Thread class is created, it is in unstarted state by default.

* Runnable (Ready to run) :

When **start()** method on the thread is called, it is in runnable or ready to run state.

* Running:

Only one thread within a process can be executed at a time. At the time of execution, thread is in running state.

* Not Runnable:

The thread is in not runnable state, if sleep() or wait() method is called on the thread, or input/output operation is blocked.

* Dead (Terminated):

After completing the task, thread enters into dead or terminated state.

**Thread class**

C# Thread class provides **properties** and **methods** to create and control threads. It is found in System.Threading namespace.

**C# Thread Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| CurrentThread | returns the instance of currently running thread. |
| IsAlive | checks whether the current thread is alive or not. It is used to find the execution status of the thread. |
| IsBackground | is used to get or set value whether current thread is in background or not. |
| ManagedThreadId | is used to get unique id for the current managed thread. |
| Name | is used to get or set the name of the current thread. |
| Priority | is used to get or set the priority of the current thread. |
| ThreadState | is used to return a value representing the thread state. |

A list of important properties of Thread class are given below:

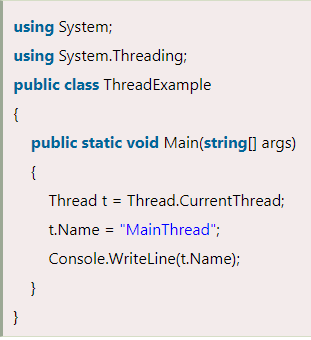
**C# Thread Methods**

A list of important methods of Thread class are given below:

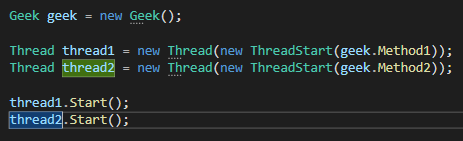
|  |  |
| --- | --- |
| **Method** | **Description** |
| Abort() | is used to terminate the thread. It raises ThreadAbortException. |
| Interrupt() | is used to interrupt a thread which is in *WaitSleepJoin* state. |
| Join() | is used to block all the calling threads until this thread terminates. |
| ResetAbort() | is used to cancel the Abort request for the current thread. |
| Resume() | is used to resume the suspended thread. It is obselete. |
| Sleep(Int32) | is used to suspend the current thread for the specified milliseconds. |
| Start() | changes the current state of the thread to Runnable. |
| Suspend() | suspends the current thread if it is not suspended. It is obselete. |
| Yield() | is used to yield the execution of current thread to another thread. |

**Main Thread**

The first thread which is created inside a process is called Main thread. It starts first and ends at last.

Multi-Threading Example :



It’s a delegate.

Represents a method that executes on thread.

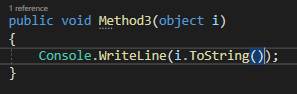
class System.Threading.Thread

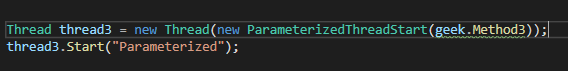
Creates & controls thread, sets its priority and gets its status.

Method which we are passing to ThreadStart should have

**No Parameters** and should **return void**.

We have an option to pass method to thread which takes parameter. But this method should only take **one** parameter of **object** type and return type should be **void**.





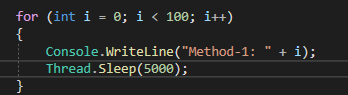
Option is of **ParameterizedThreadStart** delegate supports only a single parameter.

You can pass multiple data items to the [ParameterizedThreadStart](https://docs.microsoft.com/en-us/dotnet/api/system.threading.parameterizedthreadstart?view=net-6.0) by making that parameter one of the following:

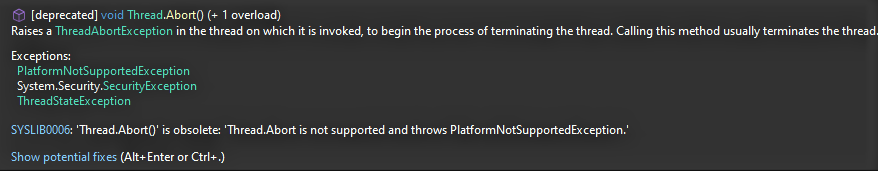
* An array.
* A collection type, if all of the data items are of the same type.
* A tuple type, such as [Tuple<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.tuple-2?view=net-6.0) or [Tuple<T1,T2,T3,T4>](https://docs.microsoft.com/en-us/dotnet/api/system.tuple-4?view=net-6.0).

Sleep ():

The Sleep() method **suspends** the current thread for the specified **milliseconds**. So, other threads get the chance to start execution.



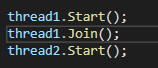
Abort ():



It is not used in .net Core as Abort does not support for different platforms.

Join ():

It causes all the calling threads to wait until the current thread (joined thread) is terminated or completes its task.



It will stop the execution of thread2 till the execution of thread1 is completed.