**Multithreading**

* **Multi-tasking :** An art of handling different (more than one) tasks simultaneously at a given point of time also called as multi-processing.



* **Multi-threading:** Multithreading in C# is a process in which multiple threads work simultaneously. It is a process to achieve multitasking. It saves time because multiple tasks are being executed at a time. To create multithreaded application in C#, we need to use **System.Threding** namespace.

**Real World Example**

1.

Single Thread



* Account
* Billing
* Customer Handling
* Manage
* Clean



Multiple Thread





2. Browser

Opening multiple tabs in a single browser, each tab is handled by a single thread.

**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
* Runnable (Ready to run)
* Running
* Not Runnable
* Dead (Terminated)

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

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

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

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

**Dead State:** 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**

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

|  |  |
| --- | --- |
| **Property** | **Description** |
| CurrentThread | returns the instance of currently running thread. |
| Is Alive | checks whether the current thread is alive or not. It is used to find the execution status of the thread. |
| Is Background | 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. |
| Thread State | is used to return a value representing the thread state. |

## C# Thread Methods

|  |  |
| --- | --- |
| **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. |

# **C# Thread Synchronization**

Synchronization is a technique that allows only one thread to access the resource for the particular time. No other thread can interrupt until the assigned thread finishes its task.

## Advantage of Thread Synchronization

* Consistency Maintain
* No Thread Interference

## C# Lock

We can use C# **lock keyword** to execute program synchronously. It is used to get lock for the current thread, execute the task and then release the lock. It ensures that other thread does not interrupt the execution until the execution finish.

# **C# Threading Example**



|  |
| --- |
|  |

**output**

