A **thread** is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

Threads are **lightweight processes.** One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.

So far we wrote the programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute more than one task at a time, it could be divided into smaller threads.

Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread –

- **The Unstarted State** – It is the situation when the instance of the thread is created but the Start method is not called.
- **The Ready State** – It is the situation when the thread is ready to run and waiting CPU cycle.
- **The Not Runnable State** – A thread is not executable, when
  - Sleep method has been called
  - Wait method has been called
  - Blocked by I/O operations
- **The Dead State** – It is the situation when the thread completes execution or is aborted.

* Main Thread :- The first thread to be executed in a process is called Main thred.

e.g :-

```
using System;

using System.Threading;
```

```
namespace MultithreadingApplication
{
   class MainThreadProgram {
      static void Main(string[] args) {
         Thread th = Thread.CurrentThread;
         th.Name = "MainThread";

         Console.WriteLine("This is {0}", th.Name);
         Console.ReadKey();
      }
   }
}
```

Destroying Threads

The **Abort()** method is used for destroying threads.

The runtime aborts the thread by throwing a **ThreadAbortException.** This exception cannot be caught, the control is sent to the *finally* block, if any.

The following program illustrates this –

Live Demo

```
using System;
using System.Threading;

namespace MultithreadingApplication {
   class ThreadCreationProgram {
      public static void CallToChildThread() {
         try {
            Console.WriteLine("Child thread starts");
```

```
            // do some work, like counting to 10
            for (int counter = 0; counter <= 10;
counter++) {
                Thread.Sleep(500);
                Console.WriteLine(counter);
            }

            Console.WriteLine("Child Thread Completed");
        } catch (ThreadAbortException e) {
            Console.WriteLine("Thread Abort Exception");
        } finally {
            Console.WriteLine("Couldn't catch the Thread
Exception");
        }
    }
    static void Main(string[] args) {
        ThreadStart childref = new
ThreadStart(CallToChildThread);
        Console.WriteLine("In Main: Creating the Child
thread");

        Thread childThread = new Thread(childref);
        childThread.Start();

        //stop the main thread for some time
        Thread.Sleep(2000);

        //now abort the child
        Console.WriteLine("In Main: Aborting the Child
thread");

        childThread.Abort();
        Console.ReadKey();
    }
  }
}
```