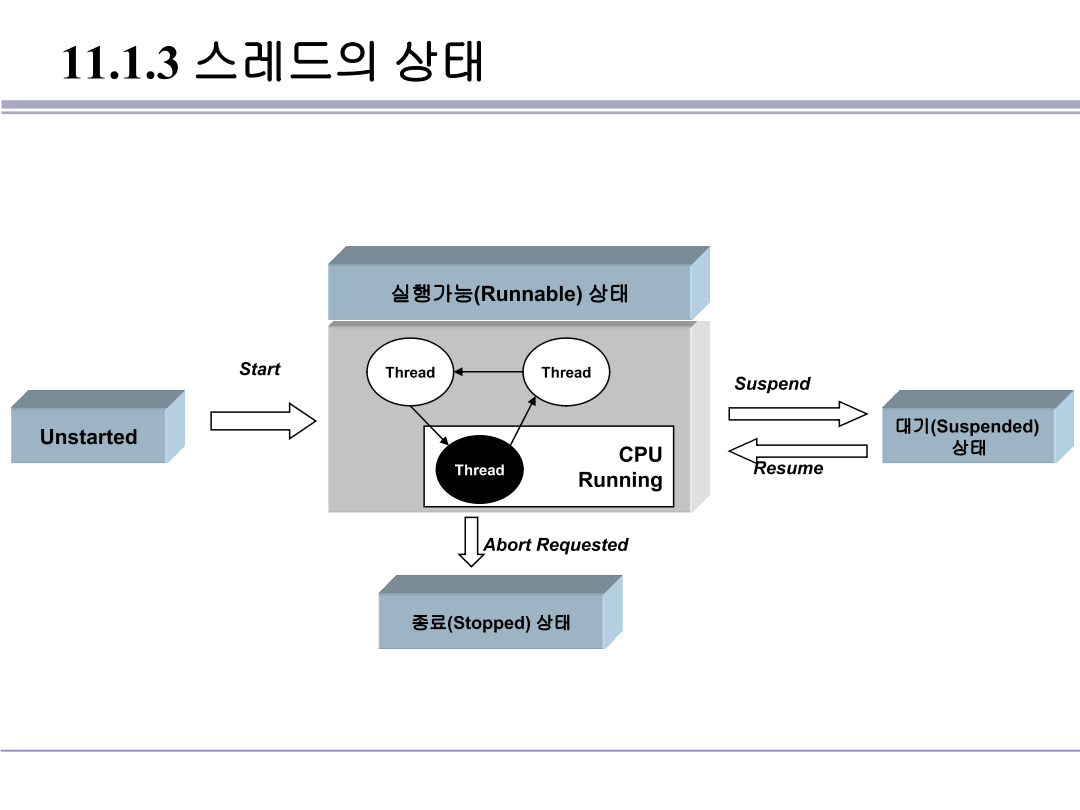
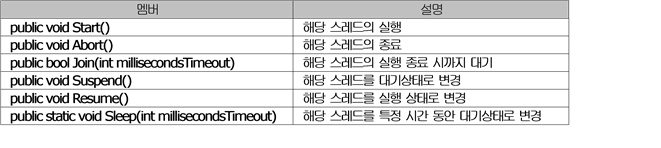
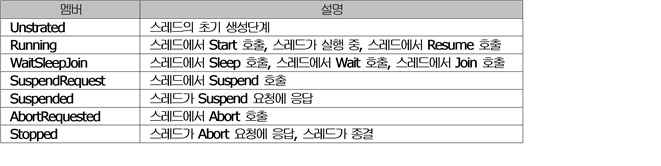
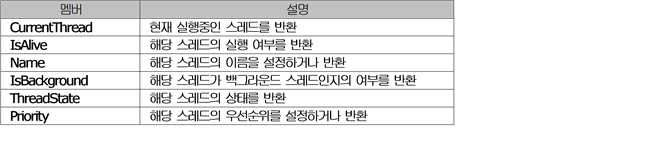
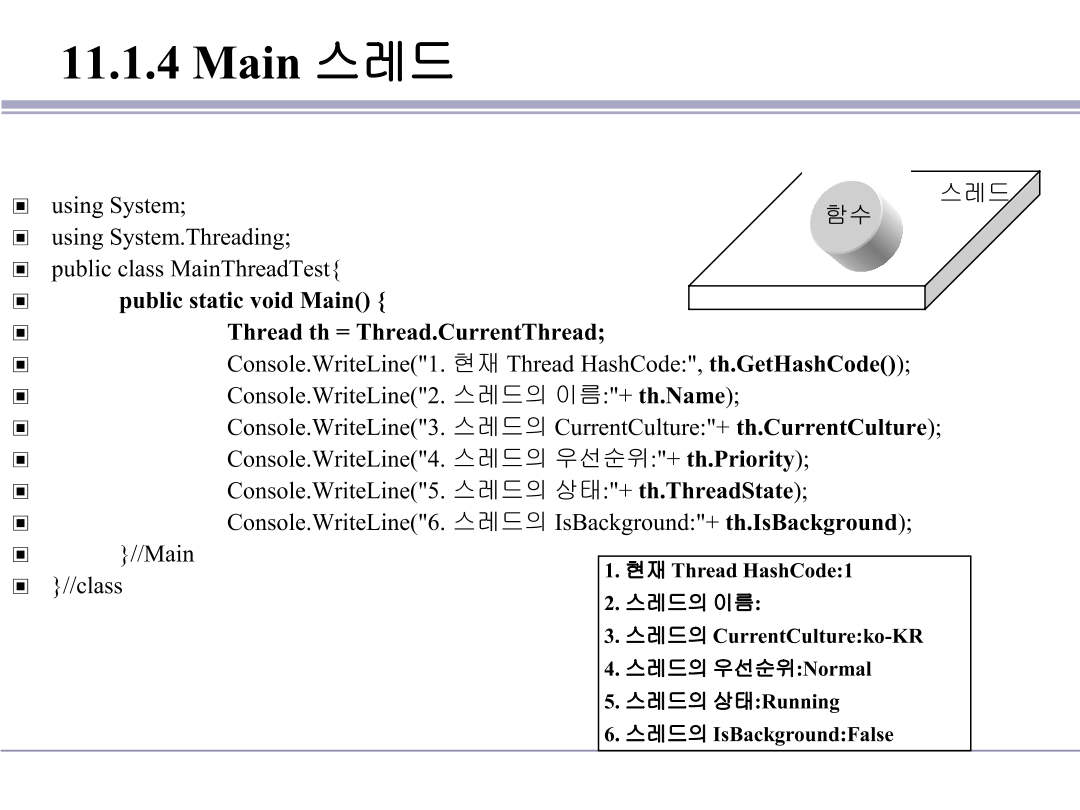
**11.1.1 개요**  
  
윈도우의 장점 중에서 가장 두드러진 것이 바로 멀티 태스킹(Multi-tasking)이다. 윈도우즈에서 작업을 할 때 워드창, 웹 브라우저, 음악 플레이어 정도는 동시에 켜둔 상태에서 작업을 한다. 동시에 이러한 작업이 가능한 것은 운영체제가 CPU의 시간을 분할해서 각각의 프로그램에게 제어권을 한번씩 나누어 주기 때문이다.   
  
이러한 작업의 대상인 프로그램은 하나의 프로세스 형태로 운영체제에서 관리된다. 운영체제 차원의 이러한 프로세스 관리법을 멀티 태스킹이라고 하며, 80386 계열에서는 CPU 차원에서 지원해주기 때문에 더욱 효율적으로 멀티 태스킹이 동작한다.  
  
**□ 프로세스(Process)**  
◇ 하나의 프로그램(Program)은 하나의 프로세스에 해당한다.   
  
프로세스의 경우에는 자동으로 운영체제가 관리하기 때문에 완벽하게 멀티 태스킹의 지원을 받을 수 있다. 프로그램을 만들고 실행만 시키면 운영체제가 알아서 관리해준다.  
  
**□ 멀티 태스킹(Multi-Tasking)**  
◇ 프로세스의 경우 운영체제에서 자동으로 관리해준다.   
◇ 운영체제 차원의 프로세스 관리를 멀티 태스킹이라고 한다.   
  
우리가 목표로 하는 것은 프로그램 단위의 멀티 태스킹이 아니라 하나의 프로그램 내에서 실행되는 멀티 함수이다. 즉 하나의 프로세스 내에 여러 개의 함수를 동시에 실행하고자 하는 것이다.  
  
**□ 프로세스와 스레드**  
◇ 하나의 프로세스 내에는 여러 개의 스레드가 존재할 수 있다.   
  
스레드란 바로 하나의 프로세스 내에서 실행되는 함수를 의미한다. 일반적인 프로세스의 경우 한순간에 하나의 함수만 실행되지만, 스레드를 이용하면 여러 개의 함수를 동시에 실행시킬 수 있다.   
  
**□ 스레드(Thread)란?**  
◇ 하나의 프로그램 내에서 실행되는 함수   
◇ 한순간에 두 개의 함수가 동시에 실행되면 두 개의 스레드가 동작하는 것이다.   
  
이 장에서는 두 개의 함수를 동시에 실행시키는 방법에 대해서 알아볼 것이다. 사실 알고 보면 하나의 함수는 이미 실행된 상태이다. Main()도 하나의 함수이기 때문에 이미 프로그램에서는 Main 스레드가 동작 중이다. Main() 이외의 다른 함수를 실행하기 위해서 스레드를 이용하는 것이다.  
  
의미상으로 하나의 스레드는 하나의 함수에 해당한다고 했지만 실제 프로그램에서도 그러하다. 단! 조건이 있다. 하나의 스레드로서의 조건을 만족하기 위해서는 프로그램 내에서 하나의 함수가 다른 함수와 동시에 실행되어야 한다. 즉 하나의 독립된 함수로 실행되면 그것을 스레드라고 말할 수 있다.  
  
다음은 이러한 스레드 프로그램을 할 때 주의해야 할 사항이다.  
  
**□ 스레드 프로그램을 할 때 주의해야 할 사항**  
◇ 우선권(Priority)   
◇ 동기화(Synchronization)   
  
동시에 두 개 또는 여러 개의 함수가 실행되기 때문에 어느 함수에게 작업할 권한을 많이 줄 것인지에 대해서 생각해보아야 한다. 즉 우선권을 어떤 함수에게 많이 줄 것인지에 관한 것이다.  
  
우선권보다도 더 중요한 것이 바로 공유 자원의 문제이다. 두 개의 함수가 동시에 작업을 진행할 때 둘 다 하나의 자원을 상대로 작업을 하면 미세한 문제가 발생한다. 예를 들어 은행 계좌에 10000원이 있다고 가정할 때, 이 계좌를 A와 B라는 두 사람이 공동으로 이용한다고 하자. 그런데 공교롭게도 A와 B가 서로 다른 장소에서 A는 돈을 인출하려고 하고 B는 돈을 입금하려 한다고 가정하자.  
  
A와 B가 거의 동시에 작업을 진행한다면 은행 시스템은 A나 B 둘 중 한 사람을 먼저 처리한 후에 다른 사람을 처리해야 올바르게 작업이 이루어질 것이다. 만약 이것을 지키지 않는다면 아주 심각한 문제가 발생한다. A와 B가 둘 다 동시에 은행 계좌를 읽어가고 작업을 했다면 둘 다 10000원으로 작업을 할 수도 있다. 이렇게 되면 계좌의 돈 계산은 엉망이 될 것이다. A가 먼저하든지 B가 먼저하든지 순서를 지키면서 작업을 해야 한다. 이렇게 하나의 자원을 상대로 순서대로 작업이 처리되는 것을 보장하는 기법을 동기화(Synchronization)라고 한다.  
  
**□ 동기화의 보장**  
◇ 공유자원을 상대로 순서대로 작업이 이루어지는 것을 "동기화가 보장된다"라고 말한다.   
  
이러한 문제는 프로그램에서도 똑같이 발생한다. 위의 A와 B는 스레드에 해당하며, 은행 계좌는 공유자원에 해당한다. 공유자원을 사용할 때 동기화를 지키지 않으면 프로그램에서 예상치 못한 결과를 초래할 수도 있다. 이러한 측면에서 이 장에서는 스레드 프로그램을 하는 방법과 우선권 처리 그리고 동기화 기법에 관련된 사항들을 학습할 것이다.

**11.1.2 스레드란**  
  
일반적인 프로그램의 경우에는 Main()을 실행시키면 순서대로 작업이 진행된다. 한마디로 "시퀀셜(Sequential)하게 진행된다"라는 의미이다.  
  
**□ 일반적 함수의 특징**  
◇ 한순간에 하나의 함수가 동작하는 것을 "시퀀셜하게 동작한다"라고 한다.   
  
우리의 목표는 하나의 함수씩 시퀀셜하게 동작하는 것이 아니라 두 개의 함수를 동시에 실행하는 것이다. 이것이 바로 스레드의 기본 목표이다. 한순간에 두 개의 함수가 동시에 실행된다면 실행된 두 개의 함수를 스레드라고 부를 수 있다. 두 개의 작업을 동시에 실행한다는 의미는 다음과 같이 정의를 내릴 수 있다.   
  
**□ 스레드의 기본**  
◇ 한순간에 두 개의 함수가 동시에 실행되었을 때 실행된 함수를 스레드라고 한다.   
  
일반적인 프로그램들의 작업은 꼬리에 꼬리를 무는 형식으로 진행된다. Main()도 하나의 함수이기 때문에 하나의 Main()이 실행되고 내부에 또 다른 함수가 실행되더라도 대부분의 경우 동시에 작업이 진행되지는 않는다. 그렇다면 다음과 같이 생각해 볼 수 있다.   
  
**□ 생각**  
◇ 어떻게 동시에 두 개의 함수를 실행시킬 수 있는가?   
  
대부분의 프로그램 언어에서 동시에 두 개의 함수를 실행하기 위한 스레드 라이브러리를 제공하고 있다. 이것은 C# 또한 마찬가지다. 우리는 이 장에서 C# 스레드 라이브러리를 사용하는 방법에 대해서 배우게 될 것이다.  
  
이쯤에서 우리는 스레드의 정의를 내려보자.  
  
**□ 스레드란?**  
◇ 프로그램에서 독립적으로 실행되는 함수   
  
스레드란 하나의 프로그램에서 독립적으로 실행되는 함수를 의미한다. 스레드가 함수인 것은 사실이지만 조건이 있다. 하나의 함수가 실행되는 것도 하나의 스레드가 실행되는 것이지만, 진정한 스레드는 동시에 두 개 이상의 함수가 실행되는 것을 의미한다. 두 개의 함수가 존재하고 서로 독립된 함수로써 동시에 작업을 진행할 수 있으면, 이 때 이 함수들을 스레드라고 부를 수 있다. 스레드를 배우면서 다음의 사항을 생각하면서 학습하기 바란다.  
  
**□ 스레드의 학습목표**  
◇ 스레드가 어떻게 만들어져서 실행되는가?  
◇ 왜 스레드를 함수라고 하는가?  
  
스레드 프로그램을 해 나가면서 스레드를 함수라고 말할 수 있다면 스레드의 개념을 정확하게 잡은 것이다. 자! 그렇다면 다른 것은 전부 제쳐두고 두개의 함수를 만든 후 동시에 실행시키는 방법부터 알아보자. 그리고 나서 스레드의 상태(Status), 우선권(Priority) 그리고 동기화(Synchronization)에 대해서 알아보자.



**11.1.3 스레드의 상태**  
  
스레드의 상태를 아는 것은 스레드를 어떻게 제어하느냐와 관련이 있다. 스레드의 상태는 대표적으로 생성 직후의 상태(Unstarted), 실행 가능한 상태(Runnable), 대기상태(Suspended), 실행을 끝내고 죽는 상태(Stopped)로 나눌 수 있다.   
  
일단 스레드가 생성된 후 Start를 하면, 스레드는 3가지 상태로 전이(轉移)가 가능하다. 위의 그림은 대표적인 스레드의 상태를 보여주고 있다.  
  
**□ Unstarted 상태**  
◇ 스레드에 해당하는 함수를 디자인하고, 이 함수의 대리자인 ThreadStart 델리게이트를 생성한다. 그리고 생성된 ThreadStart 델리게이트를 매개변수로 하여 Thread를 생성하게 된다. Unstarted 상태는 스레드가 생성된 직후를 의미한다. 이것은 Start하기 바로 직전의 상태를 나타낸다. 이 상태에서 스레드의 Start() 함수가 호출되면 실행가능 상태가 되고, 해당 스레드의 함수가 종료하면 바로 Stopped 상태가 된다.   
  
**□ 실행가능(Runnable) 상태**  
◇ 새로운 스레드가 생성되어 그 스레드의 Start() 함수를 호출하면, 스레드는 Runnable 상태가 된다. Runnable 상태는 CPU를 점유하느냐 하지 못하느냐에 따라 두가지로 나뉘게 된다. CPU를 차지한 스레드가 있을 것이고, 그 나머지는 후보(候補) 스레드가 될 것이다. Runnable 상태의 스레드들은 실행큐라고 하는 저장공간에 넣어두고 CLR에 의해서 관리된다. 실행큐란 실행가능 상태에 있는 스레드들을 넣어두는 자료구조이다. 알고 보면 CPU는 한순간에 단 하나의 스레드만을 실행할 수 있다. 실행큐에 있는 스레드들 중에서 하나만을 골라서 실행하는데 번갈아 가면서 실행하는 것이다. 물론 단 하나의 스레드만이 실행 스레드이며 나머지는 후보 스레드가 된다. 이 두가지 경우를 모두 Runnable 상태라고 말한다.  
  
**□ 대기상태(Suspended)**  
◇ 현재 실행 중인 스레드에서 Suspend()와 Sleep() 함수를 호출하게 되면 스레드는 Runnable 상태에서 Suspended 상태로 전환하게 된다. Runnable 상태를 벗어난 것이다. 조금 전에 말한 실행큐에서 제거되고 대기큐라는 곳으로 자리를 옮기게 된다. 그리고는 실행큐에 들어갈 조건이 될 때까지 기다리게 된다. 대기큐에 존재하는 스레드들은 CPU를 사용할 권한이 없기 때문에 대기상태가 된다.   
◇ 이렇게 Suspend()나 Sleep() 함수가 호출될 경우 스레드는 실행가능 상태에서 대기상태로 전환하게 되는데, 반대로 대기상태에서 실행가능 상태로 전환해주기 위해서는 Suspend() 함수가 호출된 경우에는 Resume() 함수를 호출하고, Sleep() 함수가 호출된 경우에는 주어진 시간이 경과하면 자동으로 실행가능(Runnable) 상태가 된다.  
  
**□ 종료상태(Stopped)**  
◇ 스레드는 하나의 실행 단위이다. 따라서 스레드가 할 일을 모두 마치면 스레드는 Stopped 상태가 된다. 또한 작업 도중에 Abort() 함수를 호출해도 스레드는 종료하게 된다. 이렇게 해당 스레드의 함수가 끝나거나 리턴된 경우 또는 강제로 종료될 경우 스레드는 Stopped 상태가 된다.  
  
이러한 스레드의 상태를 변경하려면 Thread 클래스에 정의된 함수를 사용하면 된다.  
  
  
  
[표 11\_1] Thread 클래스에 정의된 함수  
  
스레드의 상태가 변경되었는지 알아볼 방법이 있을까? 물론 있다. Thread 클래스의 ThreadState 속성은 스레드의 상태 집합을 정의하고 있다.   
  
  
[표 11\_2] ThreadState 열거형의 멤버  
  
이 이외에 스레드의 정보를 알아내기 위한 다양한 속성들을 제공하고 있다. 스레드 정보에 관련된 속성들은 다음과 같다.  
  
  
[표 11\_3] Thread 클래스의 속성  
  
Thread의 상태를 정확히 알아야만이 스레드를 제어할 수 있다. Thread의 속성을 이용하면 현재 스레드의 상태와 우선권 등 다양한 정보들을 얻을 수 있다.



**11.1.4 Main 스레드**  
  
진입점이 있는 C# 애플리케이션은 기본적으로 하나의 스레드이다. C#에서는 프로그램의 시작을 나타내는 Main() 함수가 가장 기본적인 스레드이다. 그래서 스레드와 관련해서 Main() 함수를 Primary Thread라고 부르기도 한다.   
  
다음의 예는 현재 Main() 함수가 동작하는 스레드를 얻어낸 후 스레드의 정보를 출력하는 예이다.

**§ chap11\MainThreadTest.cs**

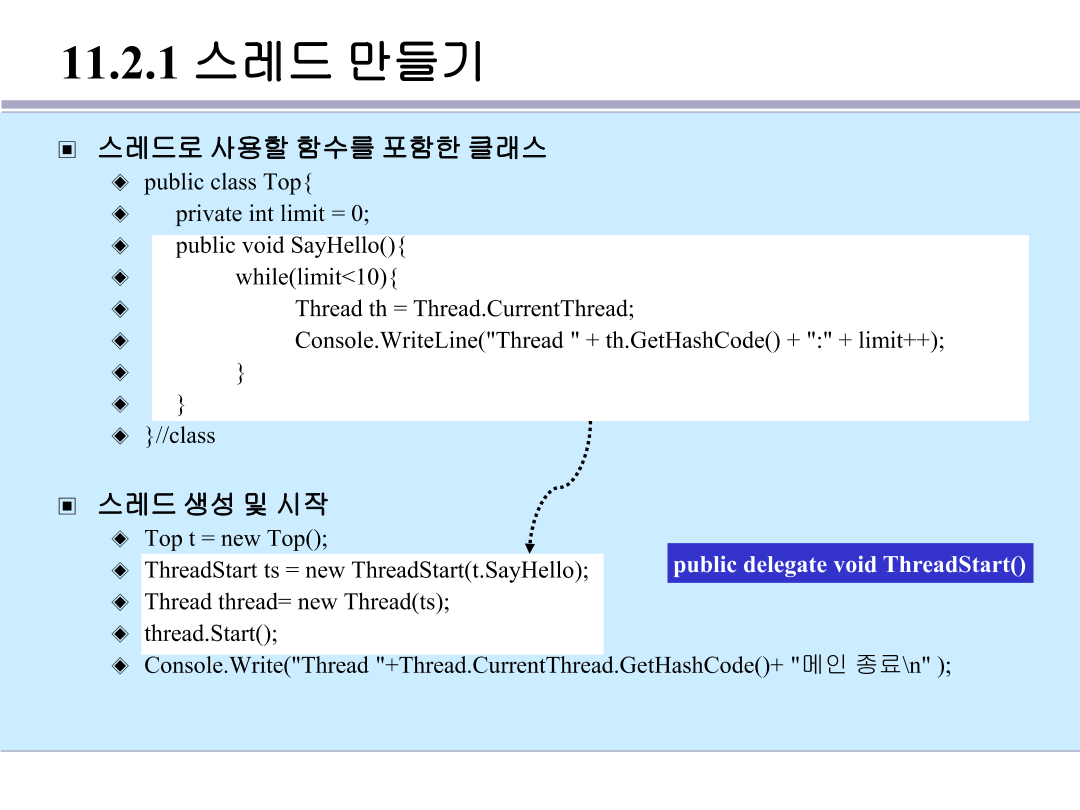
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | /\*\*  Man() 함수도 스레드의 한 종류  \*\*/  using System;  using System.Threading;  public class MainThreadTest{      public static void Main (){          Thread th = Thread.CurrentThread;          Console.WriteLine("1. 현재 Thread HashCode:" + th.GetHashCode());          Console.WriteLine("2. 스레드의 이름:" + th.Name);          Console.WriteLine("3. 스레드의 CurrentCulture:"+ th.CurrentCulture);          Console.WriteLine("4. 스레드의 우선순위:"+ th.Priority);          Console.WriteLine("5. 스레드의 상태:"+ th.ThreadState);          Console.WriteLine("6. 스레드의 IsBackground:"+ th.IsBackground);      }//main  }//class  /\*\*\*  1. 현재 Thread HashCode:1  2. 스레드의 이름:  3. 스레드의 CurrentCulture:ko-KR  4. 스레드의 우선순위:Normal  5. 스레드의 상태:Running  6. 스레드의 IsBackground:False  \*\*\*/ |

Main()이 동작하고 있는 곳의 스레드를 얻어내기 위해서 Thread의 CurrentThread 속성을 이용하고 있다.  
  
**▒ Main()이 동작하고 있는 곳의 스레드 얻어내기**

|  |  |
| --- | --- |
| 1 | Thread th = Thread.CurrentThread; |

생성된 스레드의 참조 변수를 이용해서 해시코드와 여러 가지 속성값들을 출력하고 있다.  
  
여기서 함수와 스레드의 관계를 알면 위의 개념이 쉬워진다. 간단히 스레드라는 판 위에서 함수가 동작한다고 생각하라! 하나의 스레드 위에서 Main()이 동작하는 것이며 Main() 안에서 또 다른 함수들이 동작하는 것이다. 하지만 모두 하나의 스레드 위에서 동작하는 것이다.   
  
그렇기 때문에 현재 함수가 속한 스레드를 얻어낼 수 있는 것이다. 이 때 사용하는 속성이 바로 Thread의 CurrentThread이다.  
  
**☞ 스레드와 함수의 관계**

|  |
| --- |
| 스레드라는 판 위에 함수가 동작한다고 생각하라. Main() 함수도 하나의 스레드 위에서 실행되는 방식으로 동작한다. 2개의 스레드를 만들고 각각의 스레드에 함수를 하나씩 올려서 동시에 실행하면 다중 스레드가 된다. |



**11.2.1 스레드 만들기**  
  
C#에서 스레드를 생성하고 관리하는 모든 작업은 System.Threading 네임스페이스를 사용해서 이루어진다. Threading 네임스페이스 내의 클래스 중에서 스레드를 대표하는 클래스가 바로 Thread이다.  
  
**□ Thread 클래스**  
◇ System.Threading.Thread  
◇ public sealed class Thread  
◇ 스레드를 만들고 제어하고, 스레드의 우선 순위를 설정하고 스레드의 상태를 가져오는 클래스  
  
Thread는 sealed 즉 상속금지 클래스로 선언되어 있다. 이 말은 직접 상속해서 사용할 수 없다는 뜻이다. 따라서 스레드를 만들고 제어하기 위해서는 오직 Thread 클래스의 객체를 생성한 후 이용하는 방법뿐이다. 그리고 스레드로 사용할 함수를 참조하기 위해 ThreadStart 델리게이트를 이용해야 한다.  
  
**□ ThreadStart 델리게이트**  
◇ 스레드로 사용할 함수를 참조할 수 있게 해주는 델리게이트(Delegate)  
  
델리게이트는 함수의 역할을 위임받는 객체를 생성한 후 그 함수의 역할을 대신하는 것이다. 결국 하나의 함수가 하나의 스레드 역할을 하는 것이다. 이 때 함수는 ThreadStart 델리게이트라는 표준형식을 사용하게 되는 것이다. 다음은 스레드를 생성하고 시작하는 절차를 보여주고 있다.  
  
**▒ 스레드 만드는 절차**

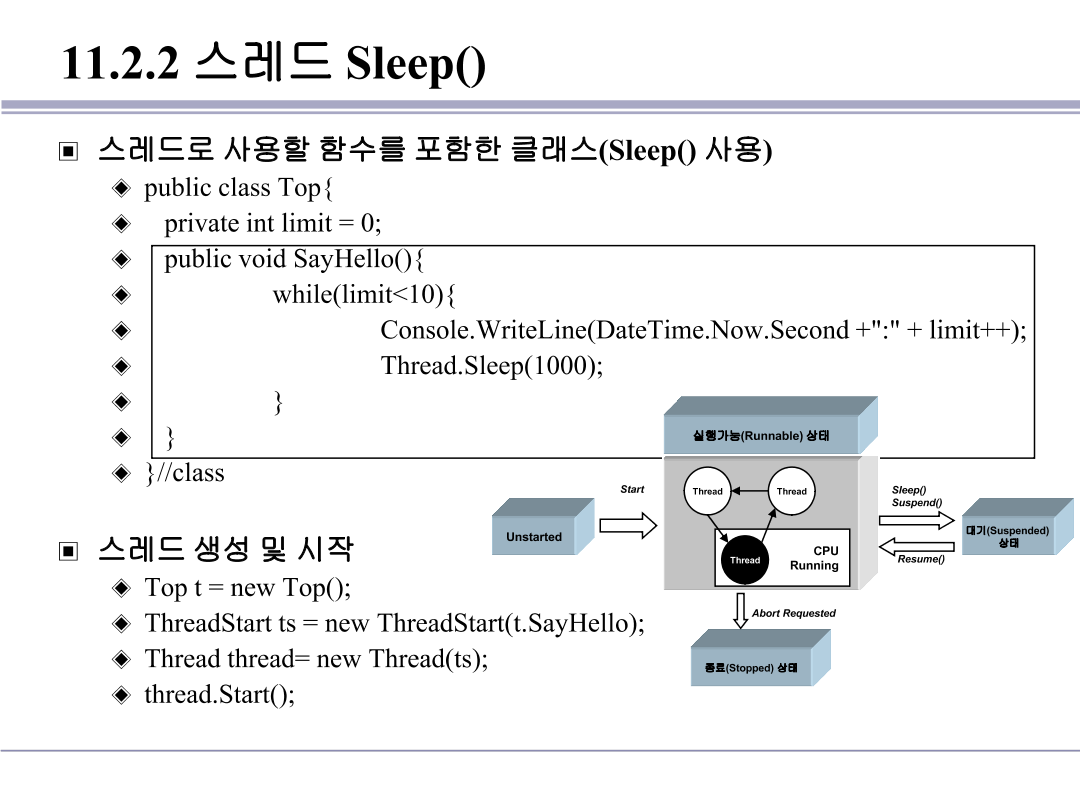
|  |  |
| --- | --- |
| 1  2  3  4  5 | Top t = new Top(); //객체 생성  //함수의 이름은 public void SayHello()라고 가정  ThreadStart ts = new ThreadStart(t.SayHello); //ThreadStart 델리게이트를 생성  Thread thread = new Thread(ts); //Thread 생성  thread.Start(); //Thread의 시작 |

위의 절차에 의해서 예제를 만들어보면 다음과 같다.

**§ chap11\SimpleThreadTest.cs**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | /\*\*  스레드 만들고 시작하기  \*\*/  using System;  using System.Threading;  public class SimpleThreadTest{      public static void Main(){          Top t = new Top();          ThreadStart ts = new ThreadStart(t.SayHello);          Thread thread= new Thread(ts);          thread.Start();          Console.Write("Thread " + Thread.CurrentThread.GetHashCode() + "메인 종료\n" );      }//main  }//class    public class Top{      private int limit = 0;      public void SayHello(){          while(limit<10){              Thread th = Thread.CurrentThread;              Console.WriteLine("Thread " + th.GetHashCode() + ":" + limit++);          }      }  }//class  /\*\*\*  Thread 1메인 종료  Thread 3:0  Thread 3:1  Thread 3:2  Thread 3:3  Thread 3:4  Thread 3:5  Thread 3:6  Thread 3:7  Thread 3:8  Thread 3:9    [참고] 스레드를 띄운 뒤 Main() 함수는 자신의 작업을 계속한다. 그렇기 때문에 "메인 종료"라는 메시지가 출력된 뒤 스레드가 동작하는 것을 볼 수 있다.  \*\*\*/ |

위의 예에서는 Thread를 생성한 후 실행하는 표준적인 방법을 보여주고 있다. 스레드에 해당하는 함수를 만들고, 그 함수를 위임하는 ThreadStart 델리게이트로 스레드를 생성한 후, Start() 함수를 호출함으로써 스레드는 동작한다.

**11.2.2 스레드 Sleep()**  
  
스레드를 제어하기 위한 방법들 중에서 Runnable 상태에 있는 스레드를 일정시간 동안 대기시키는 방법이 있다.   
  
스레드를 일정시간 동안 대기시키면 당연히 다른 스레드가 실행상태에서 작업을 더 많이 할 수 있게 된다. 즉 실행되는 스레드를 잠시 멈추어 대기상태로 만드는 것이다. 이렇게 스레드를 대기상태로 보내는 방법은 두가지로 나누어 볼 수 있다. Sleep() 함수를 호출하는 것과 Suspend() 함수를 호출하는 것이다.   
  
**□ 스레드를 대기상태로 보내는 방법1 - Sleep()**  
◇ 지정한 시간(Millisecond)동안 스레드를 대기상태로 만든다.  
◇ 지정된 시간이 지나면 자동으로 Runnable 상태가 된다.  
◇ .NET Framework 2.0에서 obsolete되었다.  
  
**□ 스레드를 대기상태로 보내는 방법2 - Suspend()**  
◇ 스레드를 대기상태로 보낸다.  
◇ 다시 실행상태가 되기 위해서는 Resume() 함수를 호출해야 한다.  
◇ Resume()을 호출하지 않으면 스레드는 Runnable 상태로 되돌아 갈 수 없다.  
◇ .NET Framework 2.0에서 obsolete되었다.  
  
먼저 Sleep() 함수를 이용한 예를 알아본 후 다음 절에서 Suspend() 함수를 살펴보자. Sleep() 함수를 추가했을 때 어떤 결과가 나오는지 테스트하는 예는 다음과 같다.

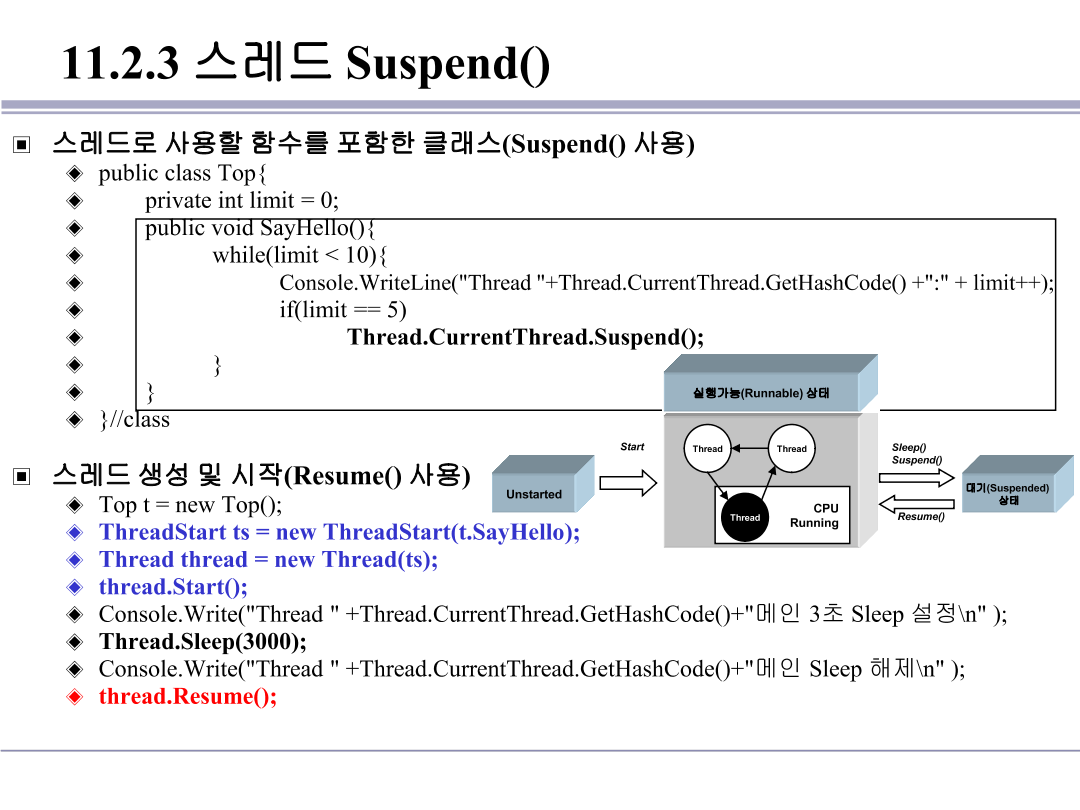
**§ chap11\SimpleThreadSleepTest.cs**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | /\*\*  스레드의 Sleep()을 테스트하는 예제  \*\*/  using System;  using System.Threading;  public class SimpleThreadSleepTest{      public static void Main(){          Top t = new Top();          ThreadStart ts = new ThreadStart(t.SayHello);          Thread thread= new Thread(ts);          thread.Start();          Console.Write("\nThread " + Thread.CurrentThread.GetHashCode() + "메인 종료\n" );      }//main  }//class    public class Top{      private int limit = 0;      public void SayHello(){          while(limit<10){              Console.WriteLine(DateTime.Now.Second + ":" + limit++);              Thread.Sleep(1000);          }      }  }//class  /\*\*\*  Thread 1메인 종료  42:0  43:1  44:2  45:3  46:4  47:5  48:6  49:7  50:8  51:9  \*\*\*/ |

코드를 보면 앞 절에서 사용한 스레드 생성 예제에서 Thread.Sleep()만 추가된 것을 확인할 수 있다.   
  
앞 절의 예제와 달리 1초마다 한번씩 출력되는 것을 확인할 수 있다. Sleep() 함수는 Thread의 스태틱 함수이기 때문에 어디서나 필요한 곳에서 사용할 수 있다. 다음은 1초 동안 해당 스레드를 Sleep 상태로 만드는 구문이다.  
  
**▒ Sleep() 호출하기**

|  |  |
| --- | --- |
| 1 | Thread.Sleep(1000); |

단순하지만 Sleep() 함수는 아주 중요한 역할을 한다. 특정 스레드가 잠시 Sleep 상태에 있는 동안 다른 스레드가 CPU를 차지할 확률이 높아지는 것이다. 여러 개의 스레드들이 동작하고 있는 상태라면 Sleep을 얼마나 주는가는 아주 중요한 문제이다. 너무 많이 주어도 안 되고 너무 적게 주어도 안 된다. 적절하게 주는 것이 필요하다. 물론 멀티 스레드에서 Sleep()은 반드시 사용된다고 봐도 과언이 아니다.   
  
**□ Sleep() 함수의 핵심**  
◇ 지정된 시간동안 대기상태가 되었다가 시간이 지나면 자동으로 Runnable 상태가 된다.  
  
Sleep()에서 중요한 것은 주어진 시간이 지나면 자동으로 Runnable 상태로 되돌아와서 CPU를 차지할 수 있는 상태가 된다는 점이다. 반면에 Suspend() 함수는 누군가가 깨워주지 않으면 영원히 잠들어 버린다. 영원히 잠들지도 모를 Suspend()에 대해서 알아보자.



**11.2.3 스레드 Suspend()**  
  
Suspend() 함수는 스레드를 대기상태로 보낸다. 다시 Runnable 상태가 되기 위해서는 Resume() 함수를 호출해주어야 한다. 만약 Resume()을 호출해주지 않으면 대기상태에서 빠져나올 수 없다. 일정 시간이 지나면 자동으로 대기상태를 빠져나오는 Sleep()과는 달리 외부에서 반드시 Resume() 함수를 호출해야만 대기상태를 벗어날 수 있다.  
  
**□ Suspend()의 핵심**  
◇ Suspend()를 호출하면 대기 상태가 된다. 하지만 Resume()을 호출해주지 않으면 영원히 대기 상태가 된다.  
  
다음의 예제는 Suspend() 함수를 이용해 스레드를 정지시키고, Resume() 함수를 호출해서 스레드를 Runnable 상태로 만드는 예제이다. 

**§ chap11\SimpleThreadSuspendTest.cs**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | /\*\*  스레드의 Suspend()를 테스트하는 예제  \*\*/  using System;  using System.Threading;  public class SimpleThreadSuspendTest{      public static void Main(){          Top t = new Top();          ThreadStart ts = new ThreadStart(t.SayHello);          Thread thread = new Thread(ts);          thread.Start();          Console.Write("Thread " + Thread.CurrentThread.GetHashCode()                                              + "메인 3초 Sleep 설정\n" );          Thread.Sleep(3000);          Console.Write("Thread " + Thread.CurrentThread.GetHashCode()                                                      + "메인 Sleep 해제\n" );          thread.Resume();      }  }//class    public class Top{      private int limit = 0;      public void SayHello(){          while(limit<10){              Console.WriteLine("Thread " + Thread.CurrentThread.GetHashCode()                                                              + ":" + limit++);              if(limit == 5)                  Thread.CurrentThread.Suspend();          }      }  }//class  /\*\*\*  1. C# 컴파일  ---------------------  SimpleThreadSuspendTest.cs(17,3): warning CS0618:  'System.Threading.Thread.Resume()'은(는) 사용되지 않습니다.  'Thread.Resume has been deprecated.  Please use other classes in  System.Threading, such as Monitor, Mutex, Event, and Semaphore,  to synchronize Threads or protect resources.  <http://go.microsoft.com/fwlink/?linkid=14202>'  SimpleThreadSuspendTest.cs(28,5): warning CS0618:  'System.Threading.Thread.Suspend()'은(는) 사용되지 않습니다.  'Thread.Suspend has been deprecated.  Please use other classes in System.  Threading, such as Monitor, Mutex, Event, and Semaphore, to synchronize Threads  or protect resources.  <http://go.microsoft.com/fwlink/?linkid=14202>'    2. 실행결과  ---------------------  Thread 3:0  Thread 3:1  Thread 3:2  Thread 3:3  Thread 3:4  Thread 1메인 3초 Sleep 설정  Thread 1메인 Sleep 해제  Thread 3:5  Thread 3:6  Thread 3:7  Thread 3:8  Thread 3:9    [참고]C# 버전이 올라가면서 Resume()과 Suspend()가 obsolete되었다. 더이상  사용하지 않기를 권고하는 것이다. 그렇기 때문에 위의 컴파일 경고대로 Monitor,  Mutex, Event, Semaphore 등을 사용하기 바란다. 이들에 대해서는 이 장의 후반부에서  배우게 될 것이다.  \*\*\*/ |

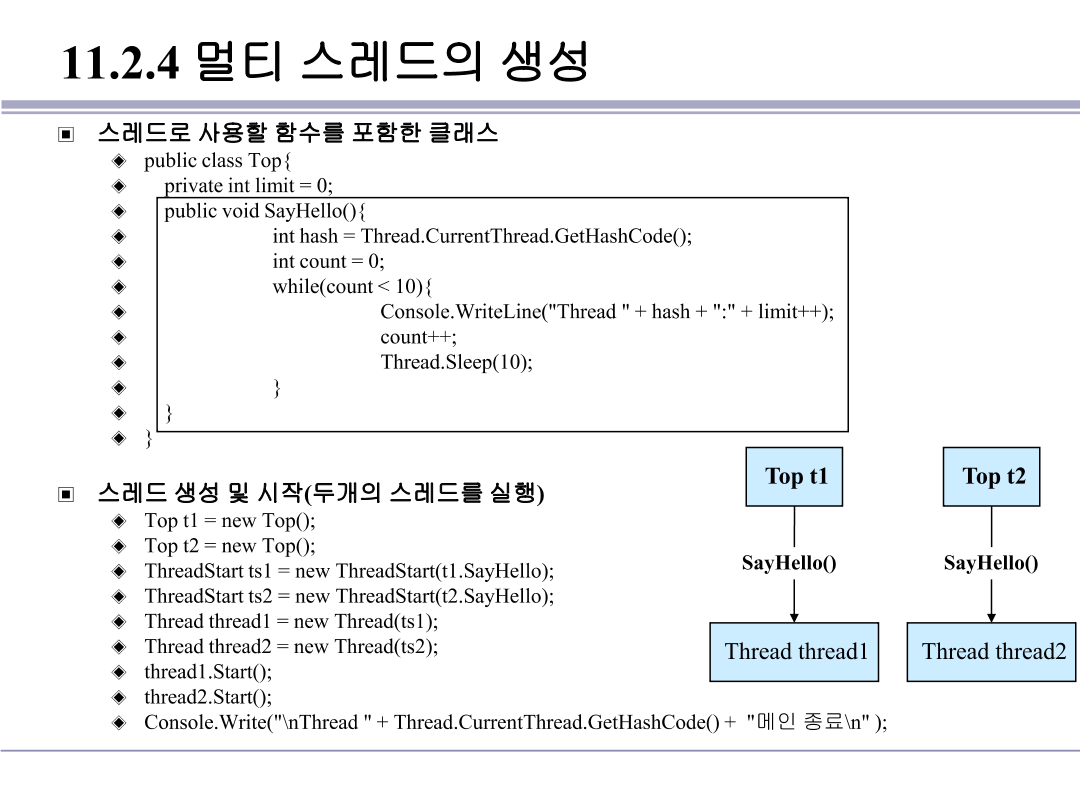
스레드를 자동으로 Suspend 시키기 위해 스레드로 사용할 함수 내에서 자신이 속해 있는 스레드를 얻어내어서 Suspend 시킨다.   
  
**▒ Suspend() 호출하기**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void SayHello(){     while(limit < 10){        //내용출력        if(limit == 5)           Thread.CurrentThread.Suspend();     }  } |

limit이 5의 값을 가지는 순간 SayHello()를 포함한 스레드는 Suspend 상태가 될 것이다. Suspend 상태가 된 스레드를 외부에서 Resume() 함수를 호출해서 다시 깨워주고 있다.   
  
**▒ Resume() 호출**

|  |  |
| --- | --- |
| 1 | thread.Resume(); |

만약 thread.Resume() 함수 부분을 주석 처리한다면 해당 스레드는 영원히 Suspend 상태에 빠지게 된다. 실행결과를 보면, 해당 스레드는 함수 내부에서 지정한대로 Suspend() 함수가 호출되어 동작을 멈추게 된다. 그리고 외부에서 Resume() 함수를 호출하면 작업이 재개된다.  
  
이와 같이 Suspend() 함수가 호출되면 스레드가 대기상태가 된다. Runnable 상태에 있는 다른 스레드가 CPU의 제어권을 가지게 되며, 대기상태의 스레드는 Resume()이 호출되었을 때 Runnable 상태가 되어 작업을 다시 시작할 수 있다.



**11.2.4 멀티 스레드의 생성**  
  
멀티 스레드란 스레드를 여러 개 생성하는 것을 말한다. 일반적인 스레드를 만드는 방식으로 스레드를 여러 개 만들면 된다.   
  
**□ 멀티 스레드**  
◇ 스레드를 여러 개 생성하는 것을 말한다.  
  
다음과 같이 두개의 스레드를 독립적으로 실행시키면 멀티 스레드가 동작한다.

**§ chap11\MultiThreadTest.cs**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | /\*\*  멀티스레드를 테스트하는 예제  \*\*/  using System;  using System.Threading;  public class MultiThreadTest{      public static void Main(){          Top t1 = new Top();          Top t2 = new Top();          ThreadStart ts1 = new ThreadStart(t1.SayHello);          ThreadStart ts2 = new ThreadStart(t2.SayHello);          Thread thread1 = new Thread(ts1);          Thread thread2 = new Thread(ts2);          thread1.Start();          thread2.Start();          Console.Write("\nThread " + Thread.CurrentThread.GetHashCode() + "메인 종료\n" );      }  }  public class Top{      private int limit = 0;      public void SayHello(){          int hash = Thread.CurrentThread.GetHashCode();          int count = 0;          while(count < 10){              Console.WriteLine("Thread " + hash + ":" + limit++);              count++;              Thread.Sleep(10);          }      }  }  /\*\*\*  Thread 1:0  Thread 2:0  Thread 3메인 종료  Thread 1:1  Thread 2:1  Thread 2:2  Thread 1:2  Thread 2:3  Thread 1:3  Thread 2:4  Thread 1:4  Thread 2:5  Thread 1:5  Thread 1:6  Thread 2:6  Thread 1:7  Thread 2:7  Thread 1:8  Thread 2:8  Thread 2:9  Thread 1:9  \*\*\*/ |

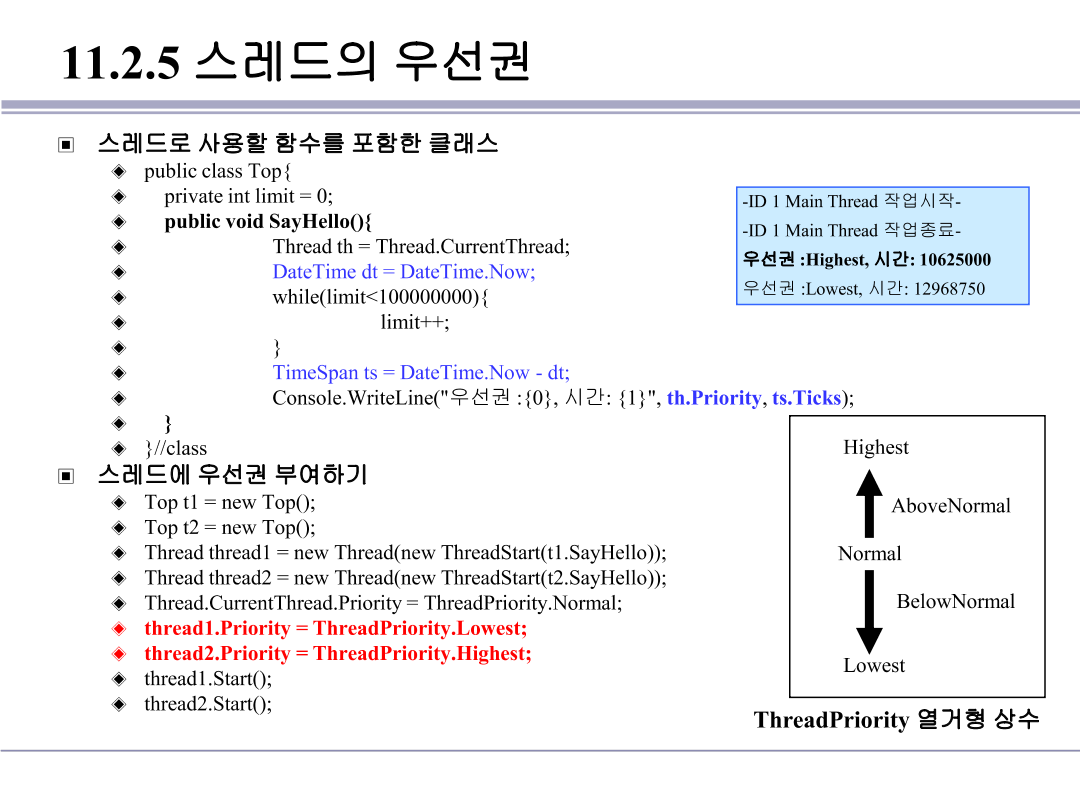
스레드를 여러 개 생성하기 위해서 각각의 객체를 생성하고 ThreadStart와 Thread를 따로 만들어서 각각의 스레드에 해당하는 Start() 함수를 호출하고 있다. 멀티 스레드를 만든다는 것은 단순히 두개의 스레드를 만든 후 동시에 실행시키면 된다.  
  
**▒ 두개의 스레드를 만들고 실행시키기**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Top t1 = new Top();  Top t2 = new Top();  ThreadStart ts1 = new ThreadStart(t1.SayHello);  ThreadStart ts2 = new ThreadStart(t2.SayHello);  Thread thread1 = new Thread(ts1);  Thread thread2 = new Thread(ts2);  thread1.Start();  thread2.Start(); |

두개의 스레드를 사용하는 것은 어려운 일이 아니다. 실제 멀티 스레드를 사용할 때 주의해야 하는 것은 바로 공유자원(Shared Data)이 있을 때이다. 멀티 스레드를 사용할 때 공유자원에 대한 동기화를 해결하는 것은 그렇게 만만한 작업이 아니다. 이것에 대해서는 동기화 부분에서 위의 예제를 약간 변형해서 자세히 다루게 될 것이다. 이 절에서는 단순히 여러 개의 스레드를 동작시키는 방법만 알아두기 바란다.   
  
**☞ 참고**

|  |
| --- |
| 위의 경우 Top t1과 t2를 이용해서 별도의 메모리로 스레드를 실행시키고 있기 때문에 공유자원이 없는 상태이다. |

다음으로 여러 개의 스레드가 동작할 때의 우선권(Priority) 문제에 대해서 알아보자.



**11.2.5 스레드의 우선권**  
  
여러 개의 멀티 스레드가 동작할 때 스레드들은 동시에 동작하는 것처럼 보이지만 한순간에 하나의 스레드만이 CPU를 점유하게 된다. 그리고 시간이 지나면 다른 스레드가 CPU를 점유할 것이다.   
  
그렇다면 과연 어떤 스레드가 CPU의 자원을 확보할 가능성이 높을까? 이러한 스레드의 실행 기준을 스레드의 우선권(Priority)이라고 한다.   
  
당연히 스레드의 작업 우선권을 어떻게 주느냐에 따라서 스레드의 작업 순서가 달라진다. C#에서는 이런 스레드의 우선권을 설정하기 위해서 Thread.Priority 속성을 제공한다. 그리고 이 속성은 ThreadPriority 열거형에 정의된 다음의 값으로 표현되어진다.   
  
  
[표 11\_4] ThreadPriority 열거형의 멤버  
  
ThreadPriority를 이용해서 스레드의 우선권 즉 Thread.Priority의 값을 Highest에서 Lowest 사이의 값으로 세팅할 수 있다. 또한 반대로 현재 실행 중인 스레드의 우선권을 Thread.Priority 속성을 이용해 얻어낼 수도 있다. 우선권을 세팅하지 않고 스레드를 만들었다면 디폴트 우선권인 Normal로 세팅된다.  
  
Thread.Priority 속성을 이용해서 스레드의 우선권을 제어해보자. 아래의 예제는 스레드의 우선권을 설정해준 후 우선권이 높은 스레드가 먼저 작업을 끝내는 예를 보여주고 있다.

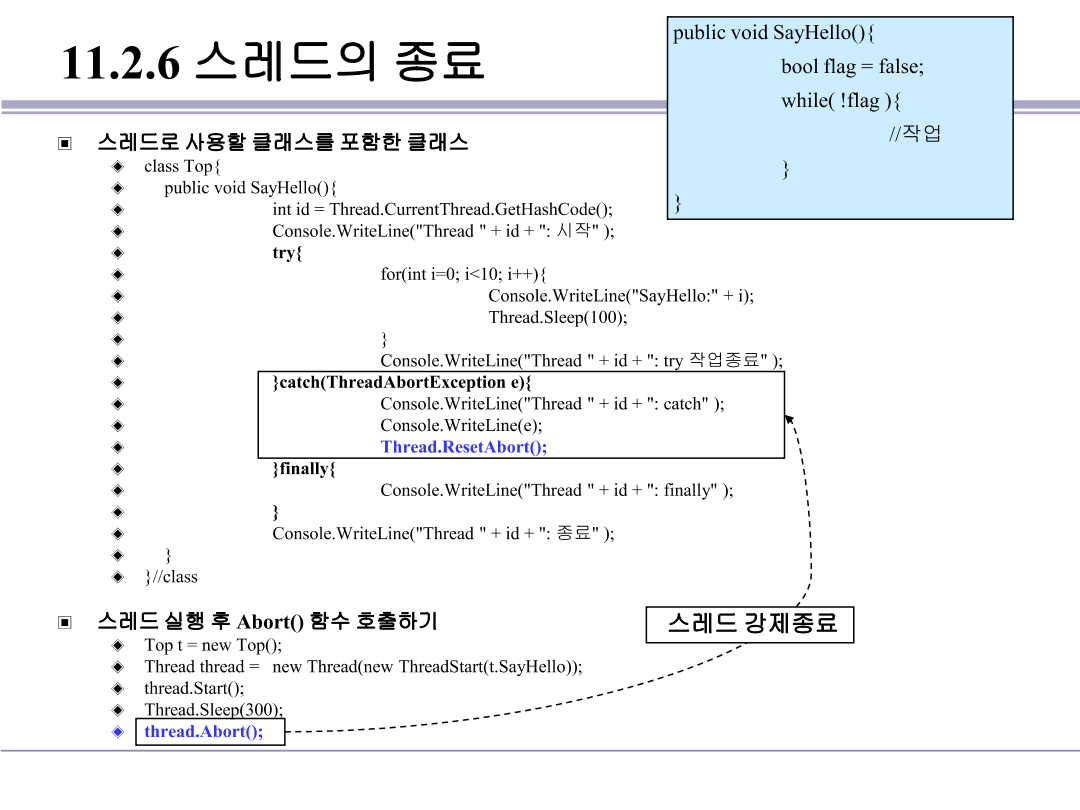
**§ chap11\SimpleThreadPriorityTest.cs**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | /\*\*  스레드의 우선권을 테스트하는 예제  \*\*/  using System;  using System.Threading;  class SimpleThreadPriorityTest{      public static void Main (){          int nowId = Thread.CurrentThread.GetHashCode();          Console.WriteLine("-ID {0} Main Thread 작업시작-", nowId );          Top t1 = new Top();          Top t2 = new Top();          Thread thread1 = new Thread(new ThreadStart(t1.SayHello));          Thread thread2 = new Thread(new ThreadStart(t2.SayHello));          Thread.CurrentThread.Priority = ThreadPriority.Normal;          thread1.Priority = ThreadPriority.Lowest;          thread2.Priority = ThreadPriority.Highest;          thread1.Start();          thread2.Start();          Console.WriteLine("-ID {0} Main Thread 작업종료-", nowId);      }  }//class    public class Top{      private int limit = 0;      public void SayHello(){          Thread th = Thread.CurrentThread;  DateTime dt = DateTime.Now;          while(limit<100000000){              limit++;          }  TimeSpan ts = DateTime.Now - dt;          Console.WriteLine("우선권 :{0}, 시간: {1}", th.Priority, ts.Ticks);      }  }//class  /\*\*\*  -ID 1 Main Thread 작업시작-  -ID 1 Main Thread 작업종료-  우선권 :Highest, 시간: 10625000  우선권 :Lowest, 시간: 12968750  \*\*\*/ |

스레드의 우선권을 설정해주지 않으면 Normal의 값을 가진다. 여기서는 스레드의 우선권을 다음과 같이 설정해 줌으로써 스레드의 우선권을 제어할 수 있다.   
  
**▒ Thread의 우선권 설정**

|  |  |
| --- | --- |
| 1  2  3 | Thread.CurrentThread.Priority = ThreadPriority.Normal;  thread1.Priority = ThreadPriority.Lowest;  thread2.Priority = ThreadPriority.Highest; |

Main 스레드의 우선권을 Normal로 주고, thread1 스레드의 우선권을 Lowest, thread2의 우선권을 Highest로 주었다. 결과를 보면 thread2가 늦게 시작했지만 가장 높은 작업 우선권을 가지고 있기 때문에 thread1보다 작업을 더 빨리 끝낸 것을 확인할 수 있다. 다음으로 스레드의 종료에 대해서 알아보자.



**11.2.6 스레드의 종료**  
  
스레드는 ThreadStart 델리게이트에 등록되어 있는 함수가 종료하면 자동으로 종료된다. 그래서 스레드를 계속적으로 동작시키기 위해 함수 내에 for문이나 while문 같은 반복문을 사용한다. 조건이 만족하면 계속적으로 반복한다는 특징 때문에 스레드 내에 단골 손님처럼 사용되고 있다.  
  
스레드의 종료는 함수의 종료를 의미하기 때문에 보통의 경우 while문을 빠져나오면서 스레드를 종료시키는 방법을 이용한다.  
  
**▒ while문을 이용한 함수의 종료**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public void SayHello(){     bool flag = false;     while( !flag ){        //작업     }  } |

함수가 종료되면 스레드가 자동으로 끝나지만 외부에서 강제로 종료시킬 수도 있다. Abort() 함수가 호출되면 ThreadAbortException 예외를 발생시켜서 스레드를 종료하게 만든다. ThreadAbortException 예외는 catch에서 처리할 수 있지만 catch 구문에서 ThreadAbortException 예외가 다시 발생한다.   
  
**□ 스레드의 자동 종료**  
◇ 해당 함수가 종료하면 스레드는 자동으로 종료한다.  
  
**□ 강제 종료**  
◇ Abort() 함수를 이용해서 강제로 종료할 수 있다. 이 때  ThreadAbortException 예외를 발생시켜서 스레드를 종료하게 만든다.  
  
**□ ThreadAbortException 예외**  
◇ Abort() 함수를 호출하면 발생하는 예외이며, catch에서 처리할 수 있지만 catch에서 ThreadAbortException 예외가 다시 발생한다.  
  
catch에서 다시 발생한 예외를 그냥 놓아두면 프로그램이 죽어버린다. 보통의 경우 catch에서 예외를 처리하면 되지만 ThreadAbortException 예외의 경우 catch에서 예외가 다시 발생하기 때문에 중단 요청을 취소해주어야 한다. 이를 위해 catch에서 ResetAbort()를 호출해주면 된다. ResetAbort()가 호출되면 ThreadAbortException 예외는 다시 발생하지 않는다.  
  
**□ ResetAbort() 함수**  
◇ 스레드의 중단 요청을 취소하는 함수  
  
결국 Abort() 함수를 호출해서 스레드를 먼저 중단시키고 catch에서 ResetAbort()를 호출해서 중단요청을 취소해주어야만 프로그램이 죽지 않고 정상적으로 동작할 것이다.  
  
그럼 Abort() 함수를 사용해서 스레드를 강제 종료하고, ResetAbort() 함수를 사용해서 중단요청을 취소하고 실행을 계속하는 예를 알아보자.

**§ chap11\SimpleThreadExitTest.cs**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | /\*\*  스레드 강제종료 기법  \*\*/  using System;  using System.Threading;  class SimpleThreadExitTest{      public static void Main (){          Top t = new Top();          Thread thread = new Thread(new ThreadStart(t.SayHello));          thread.Start();          Thread.Sleep(300);          thread.Abort();          thread.Join();      }  }//class    class Top{      public void SayHello(){          int id = Thread.CurrentThread.GetHashCode();          Console.WriteLine("Thread " + id + ": 시작" );          try{              for(int i=0; i<10; i++){                  Console.WriteLine("SayHello:" + i);                  Thread.Sleep(100);              }              Console.WriteLine("Thread " + id + ": try 작업종료" );          }catch(ThreadAbortException e){              Console.WriteLine("Thread " + id + ": catch" );              Console.WriteLine(e);              Thread.ResetAbort();          }finally{              Console.WriteLine("Thread " + id + ": finally" );          }          Console.WriteLine("Thread " + id + ": 종료" );      }  }//class  /\*\*\*  Thread 3: 시작  SayHello:0  SayHello:1  SayHello:2  Thread 3: catch  System.Threading.ThreadAbortException: 스레드가 중단되었습니다.     위치: System.Threading.Thread.SleepInternal(Int32 millisecondsTimeout)     위치: Top.SayHello()  Thread 3: finally  Thread 3: 종료    [참고] ResetAbort() 호출이 없다면 ThreadAbortException이 다시 발생하기 때문에  catch에서 프로그램이 멈춰버린다.  \*\*\*/ |

소스를 보면 Top 클래스에서 예외를 처리하기 위해서 try~catch~finally 블록을 설정하고 있으며, 실행 클래스에서 스레드를 실행시킨 후 300ms가 지난 뒤 Abort() 함수를 호출하고 있다. Abort() 함수가 호출되면 다음과 같은 절차에 의해서 스레드 종료가 진행된다.  
  
**▒ 프로그램의 스레드 종료 절차**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 1. 스레드 시작  2. 300ms후 Abort() 함수 호출  3. ThreadAbortException 발생  4. catch 블록에서 ThreadAbortException 처리  5. ThreadAbortException 예외가 다시 발생하는 것을 막기 위해 ResetAbort() 호출  6. finally 실행  7. Join에 의해 스레드가 종료할 때까지 대기 |

결과를 보면 Abort() 함수로 인해서 해당하는 스레드가 중단되고, catch에서 ResetAbort()가 호출된 후 프로그램이 안전하게 종료되는 것을 확인할 수 있다.