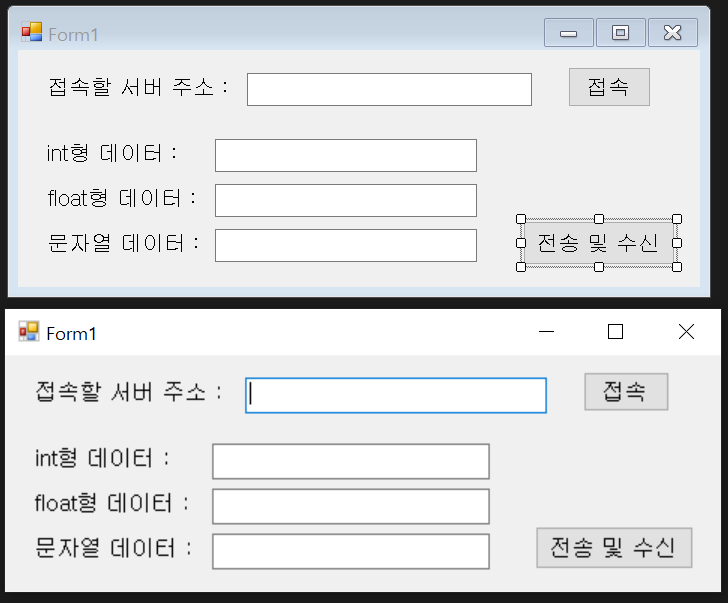
3강 콘솔 클라이언트 코드를 윈도우 코드로 옮기기

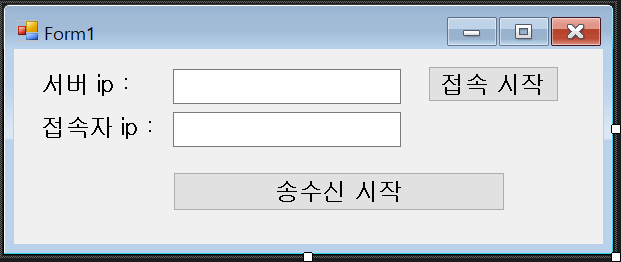
* 디자인



: 접속 버튼으로 tcpClient가 접속할 서버 주소로 연결 요청을 보냅니다.

: 서버에서 송수신 시작 버튼을 누른 후 3가지 타입의 데이터를 입력하여 전송 및 수신 버튼을 누르게 되면 write() 메소드를 실행하여 서버는 데이터를 받아 다시 Client에게 보내주게 됩니다.

: 또한 write 모드에서 read 모드로 전환되면서 Client는 데이터를 받아내는 대기 상태가 됩니다.



: 서버에서 접속 시작 버튼을 누르게 되면 Client의 요청이 수락 되어 연결됩니다.

: 서버에서 송수신 시작 버튼을 누르게 되면 서버는 대기 상태에 들어가 Client에서 보내는 데이터를 기다리게 됩니다.(Read를 실행한 상태)

4강 Thread

1. 스레드 개요

프로세스와 스레드

-스레드는 운영체제의 자원이며 프로그램 안에서 독립적으로 실행할 수 있는 것

(멀티태스킹 운영체제)

-운영체제 안에 이미 스레드가 구현되어 있지만 유저가 코드로서 요청하거나 명령을 내림

-스레드에 의해 독립적으로 실행하고자 하는 코드들의 집합체 = 함수

-멀티태스킹 운영체제 = 하나의 운영체제 위에 여러 개의 윈도우 창을 실행

(사실은 cpu 하나 당 하나의 프로그램을 실행, 그러나 스케쥴러에 의해 빠르게 실행되기 때문에 동시적으로 실행되는 것처럼 보임)

-Q. 프로그램의 구성은 무엇인가?

-A. 코드와 데이터

-사실 운영체제도 코드와 데이터

\*프로세스?

: 운영체제로부터 할당 받은 메모리에 코드와 데이터를 저장 및 CPU를 할당받아 실행 가능한 상태

-코드를 메모리에 올리기 위해 운영체제의 도움을 받음

-CPU를 나누어 쓸 수 있도록 scheduler로 조정함

\*스레드?

: 프로세스를 할당받고 코드를 실행

프로세스(메모리 개념) : 스레드(실행 개념)

\*스레드는 언제 실행되는가?

-프로그램 실행(Main 함수 = 주 Thread, Single Thread)

-코드에 의해(Thread(클래스))

-스레드가 많을수록 여러 자원들이 할당되기 때문에 적절하게 사용하여야 함

\*스레드에 의해 실행되는 함수는 몇 번 실행되는가?

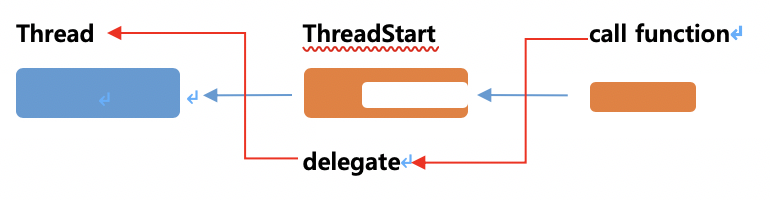
\*스레드 함수의 반복 처리?

-반복문(while, for)

2. 스레드 생성과 사용

스레드 함수 호출 구조

* 호출 구조

****

1. **스레드 함수**를 구현

2. 델리게이트 생성과 스레드 함수 설정

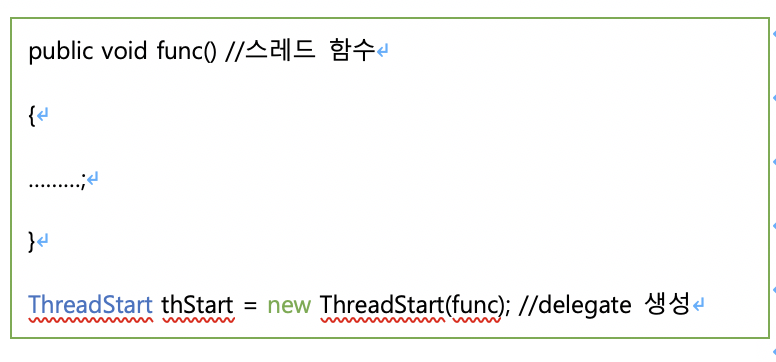
3. 스레드 생성

4. 스레드 실행

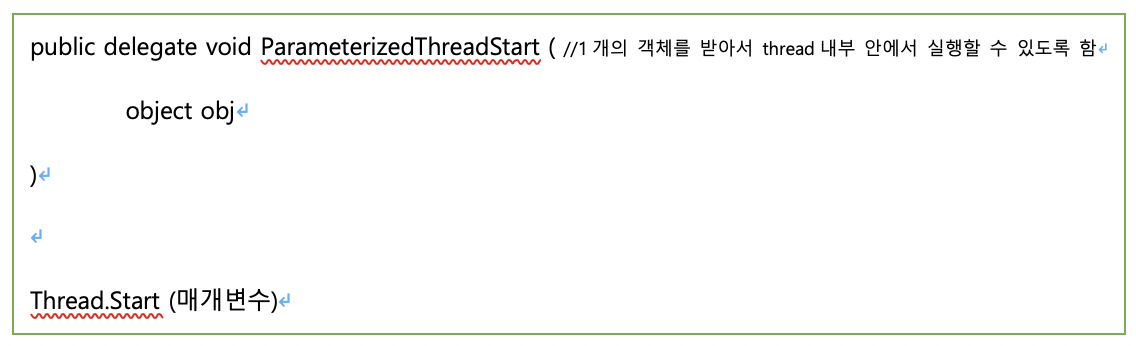
\*using System.Threading

\*delegate 생성과 설정

- public delegate void ThreadStart()

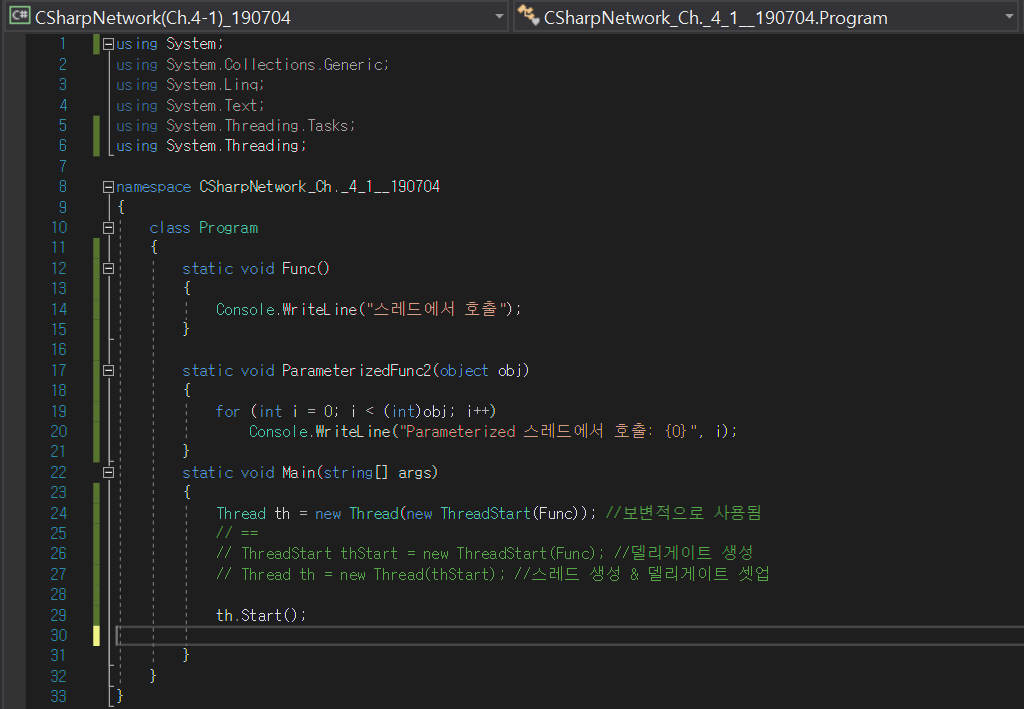


- 매개변수를 참조할 수 있는 스레드 함수의 호출



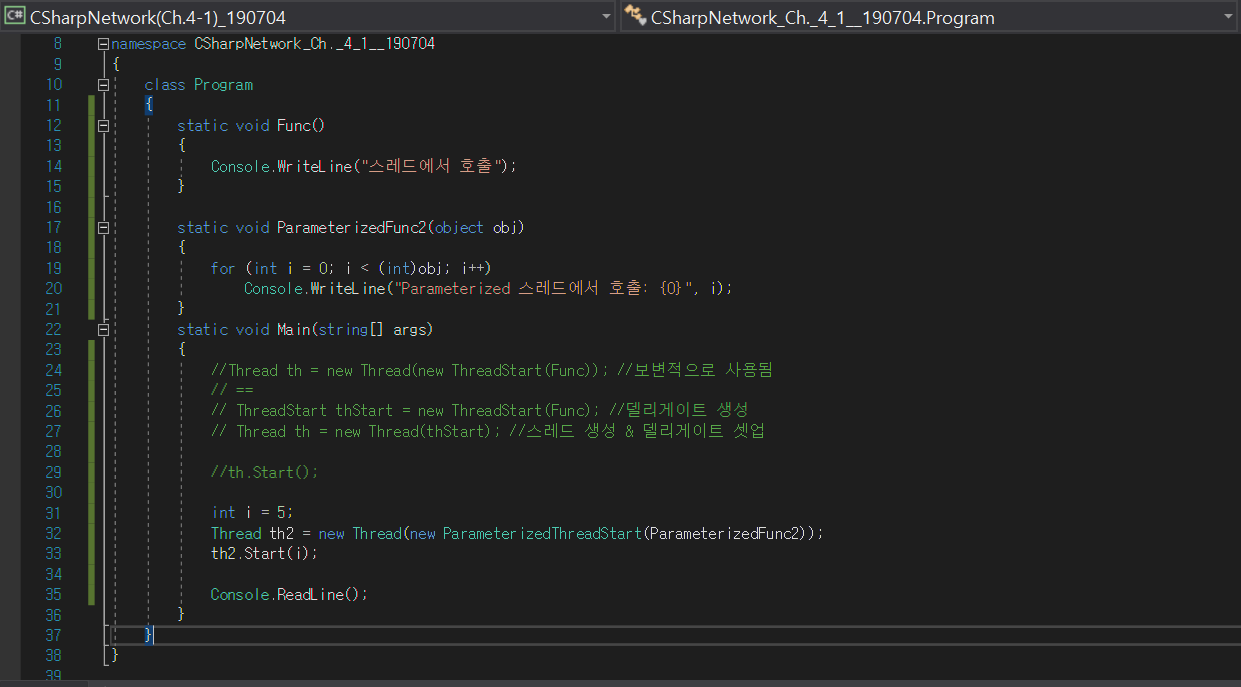
3. 스레드 사용 예

* 스레드 생성 & 실행을 예시로 설명
* 기본 프로그래밍 예
* 스레드 자체가 반복적으로 처리하는 것이 아니라 스레드 함수를 한번만 호출
* 그러나 스레드 함수를 반복적으로 호출하기 위해 반복문을 사용

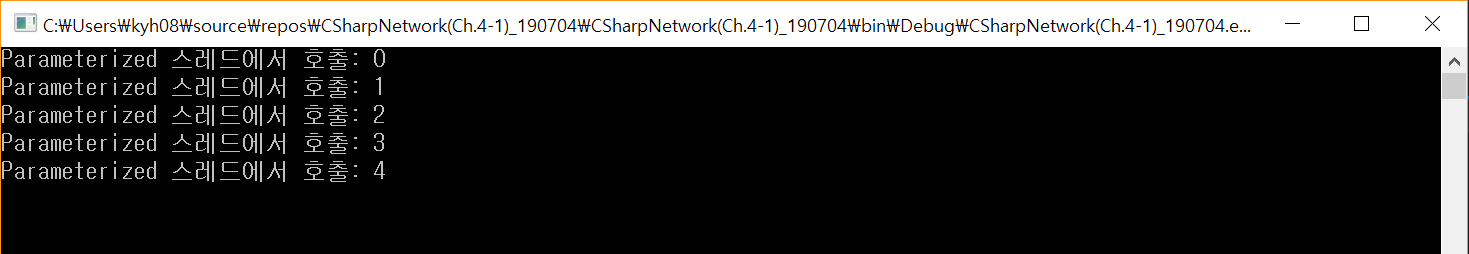


<매개 변수 없는 스레드 호출과 생성>

* ParameterizedThreadStart() 사용 (1) 변수 넘기기

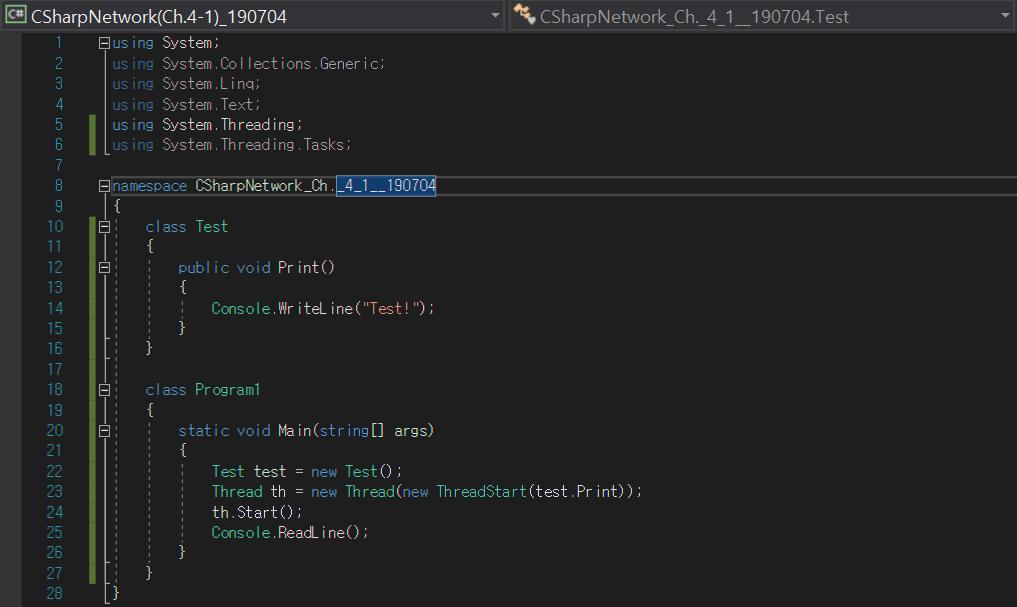


<매개 변수가 있는 스레드 호출과 생성>

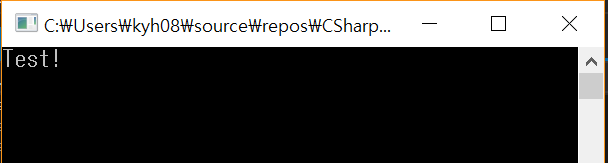


<매개 변수가 있는 스레드의 결과>

* 객체 메소드 넘기기



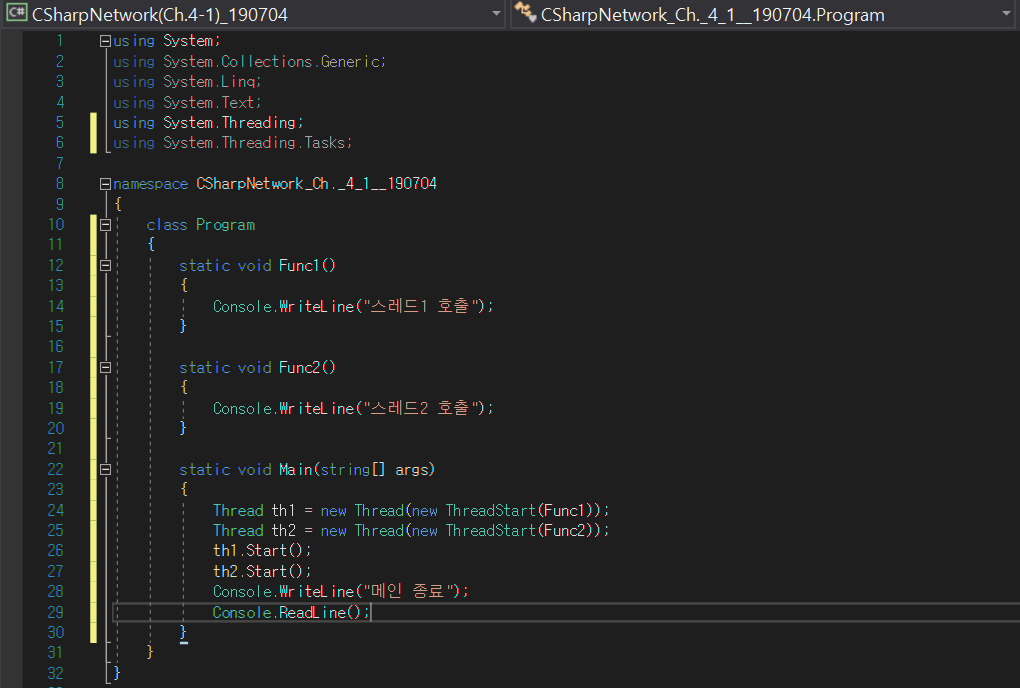
<객체가 있는 스레드에 함수로 호출>



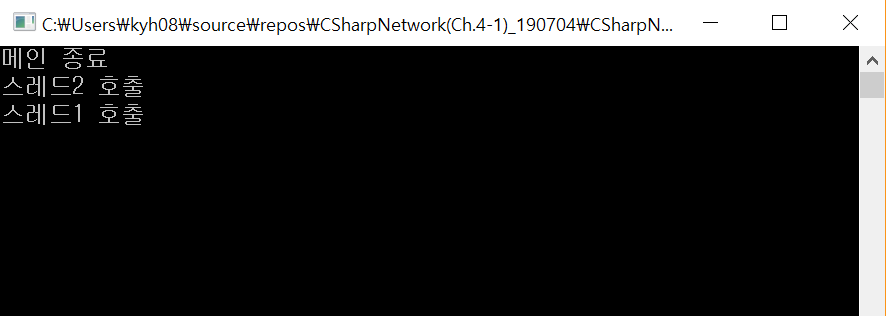
<결과>

스레드 프로그래밍

* 두개 이상의 스레드로 각각의 함수 호출 -> 멀티스레드

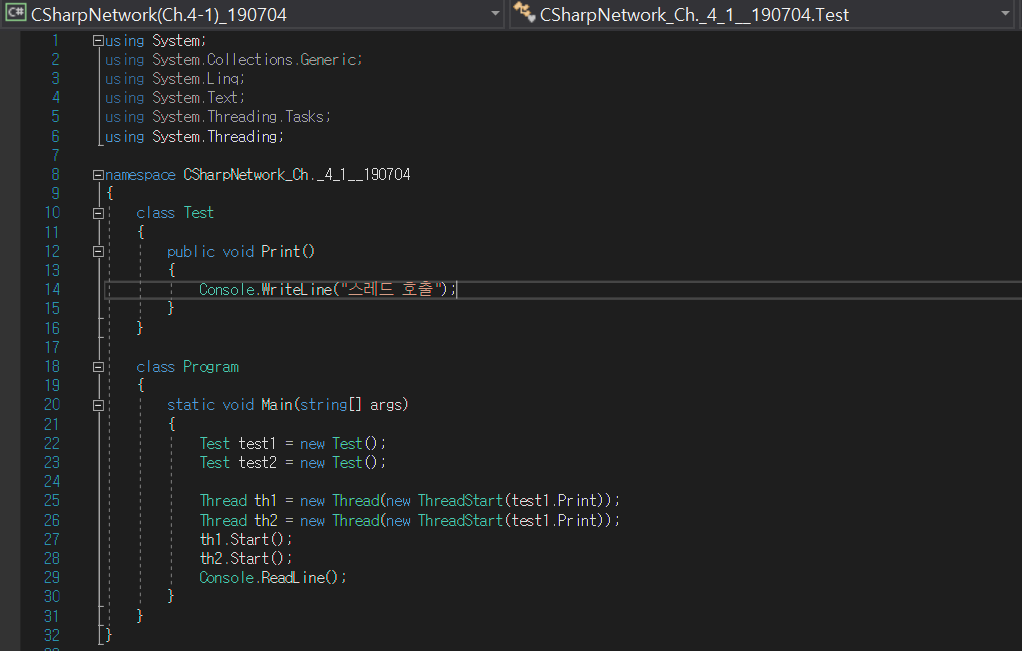


<2개 이상의 스레드로 함수를 호출하는 코드>

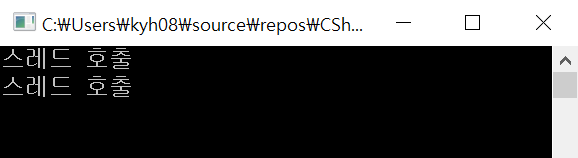


<결과>

* 두개 이상의 스레드로 객체 메소드 호출
* 하나의 객체를 생성 후 그 객체 메소드를 스레드 2개에 같이 넣게 되면 하나의 자원에 스레드 2개가 같이 쓸려고 하여 충돌(일단은 개별로 사용한다는 것을 전제에 두고 실행)
* Test라는 인스턴스는 같으나 test1, test2는 별개의 객체, 서로 간섭하지 않음

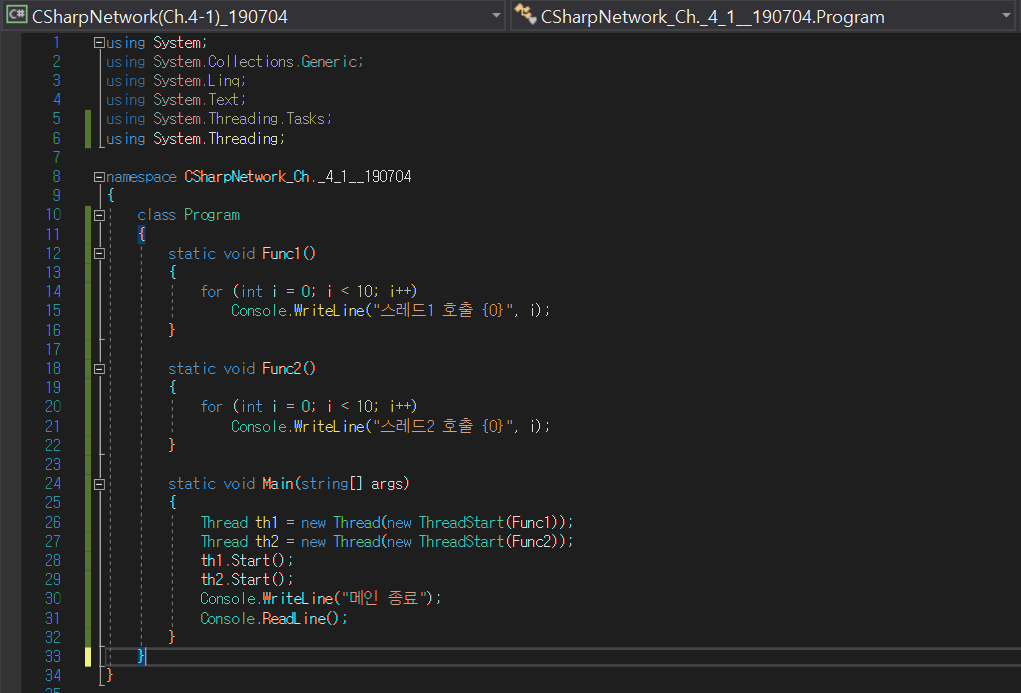


<멀티스레드로 객체 메소드 호출하는 코드>

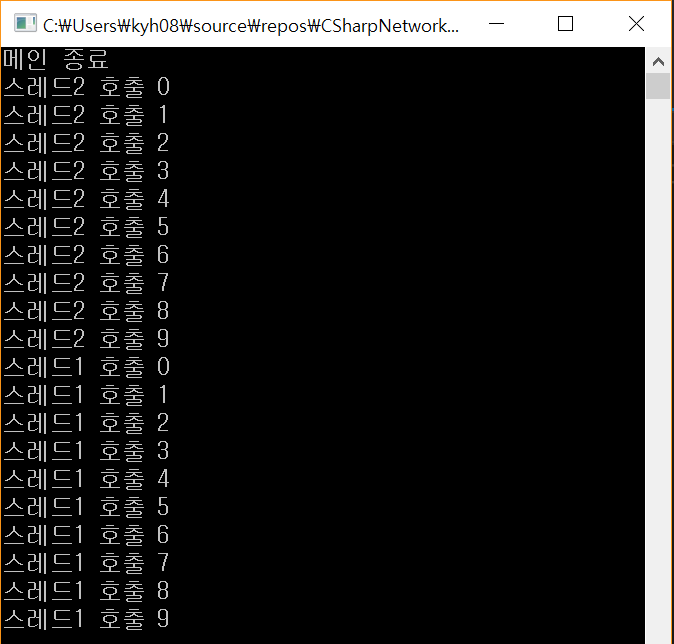


<결과>

* 두개 이상의 스레드 함수가 반복 -> 반복문
* 스레드는 주 스레드랑은 별개로 OS에 의해 독립적으로 돌아감



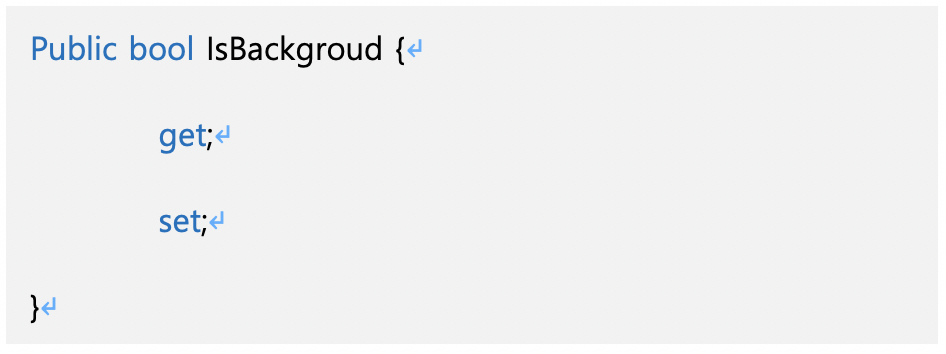
<멀티스레드 반복문 코드>



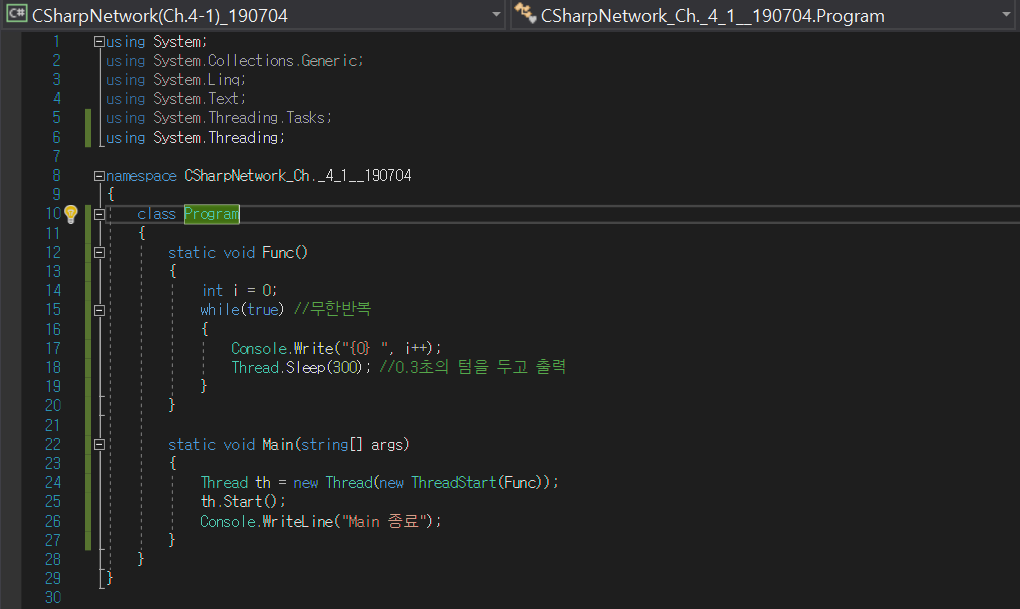
<결과>

4. 스레드 속성

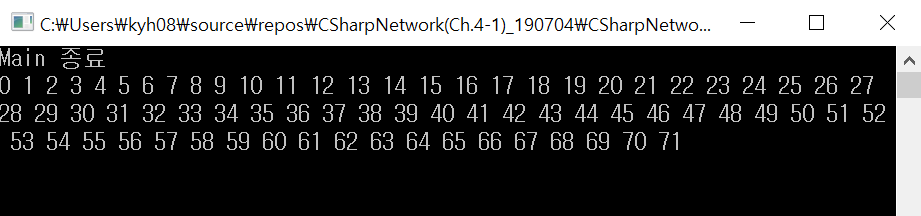
* Name (스레드의 명칭 지정)
* IsAlive (사용하는 스레드가 실제로 작동하는가의 유무 판단, return 값 : true / false)
* IsBackground ()



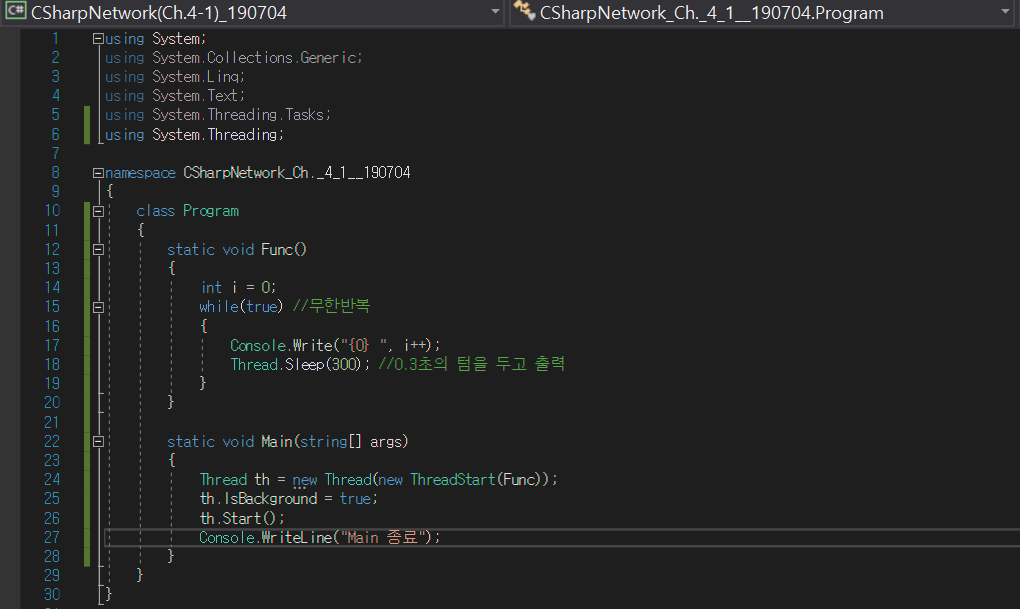
* foreground? background? (foreground : 주 스레드와 관계없이 독립적으로 돌아가는 스레드 ó background : 주 스레드가 종료됨과 동시에 부 스레드들도 종료되는 스레드)



<foreground 예시 코드>



<결과>



<IsBackground 예시 코드>

주 스레드가 종료될 때 부 스레드도 같이 종료되므로 Func 함수가 실행되지 않음을 확인할 수 있습니다.



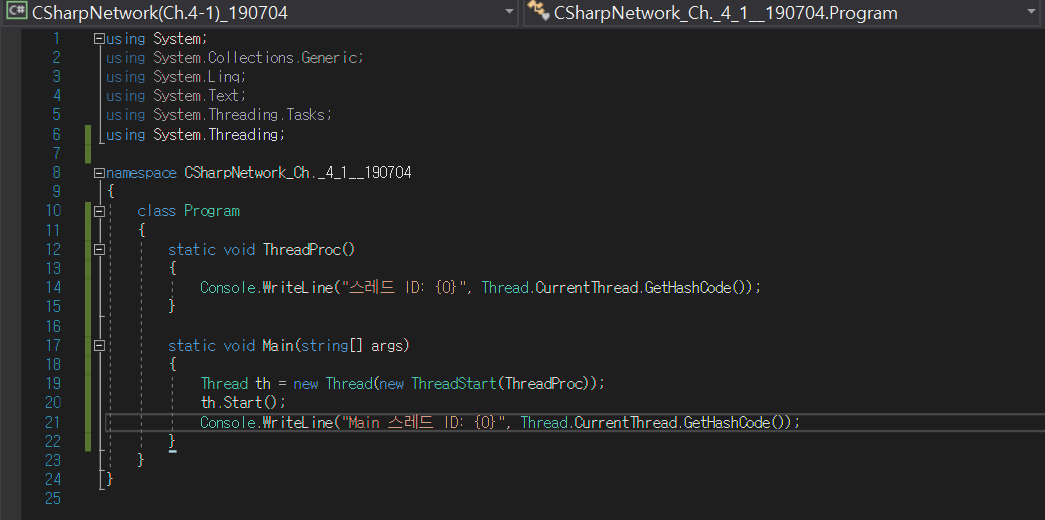
<결과>

* CurrentThread

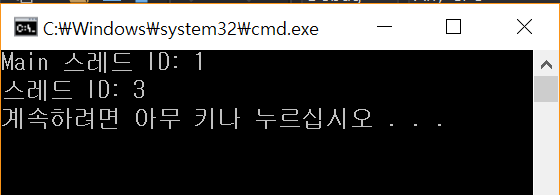


현재 자신이 속해있는 스레드의 오브젝트를 return해주는 함수

\*GetHashCode() : thread의 독립된 ID를 받아오는 함수



<CurrentThread 예시 코드>



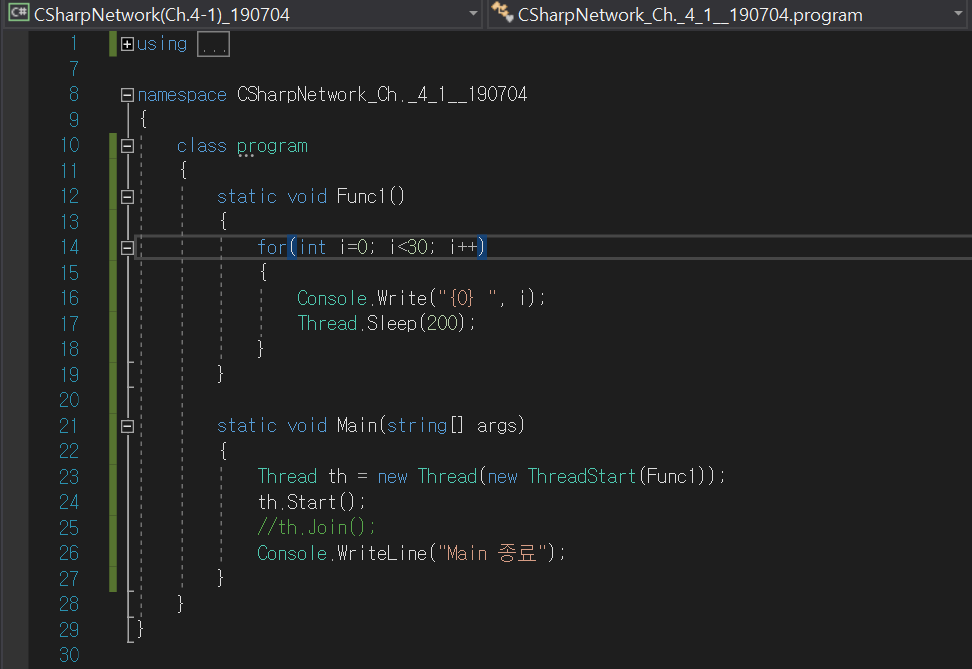
<결과>

5. 스레드 메소드

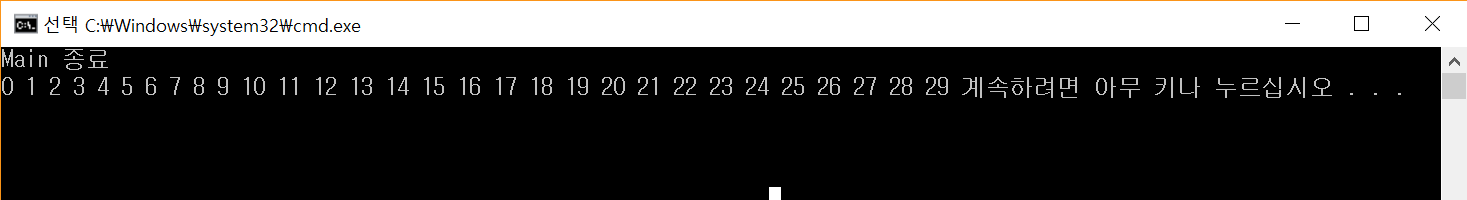
* Start()

스레드 시작

스레드가 주 스레드와 별개로 작동되는 모습을 확인할 수 있음



<Start 예시 코드>

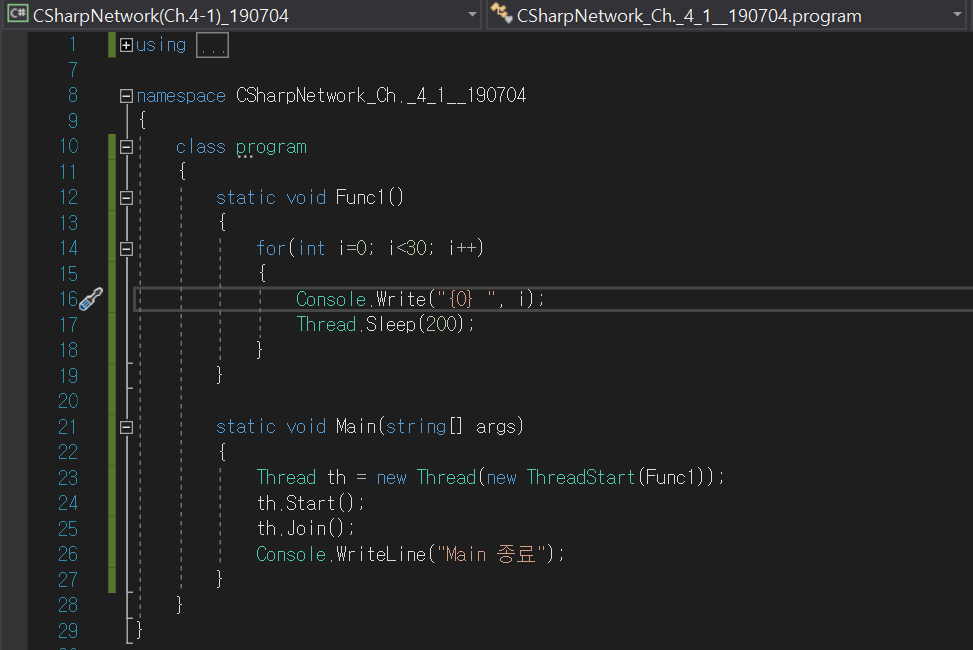


<결과>

* Join()

스레드가 종료될 때까지 대기

스레드가 완전히 실행되고 난 후에야 Main 스레드가 실행되고 종료됨을 확인할 수 있음



<Join 예시 코드>



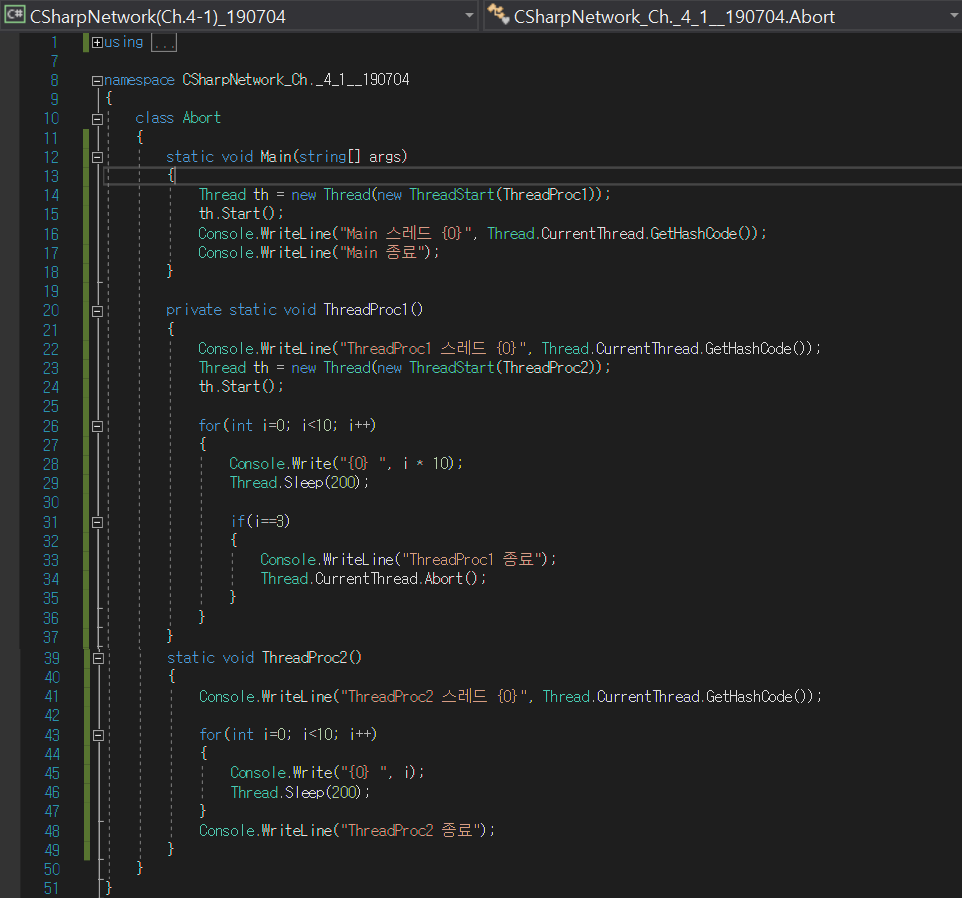
<결과>

* Abort()

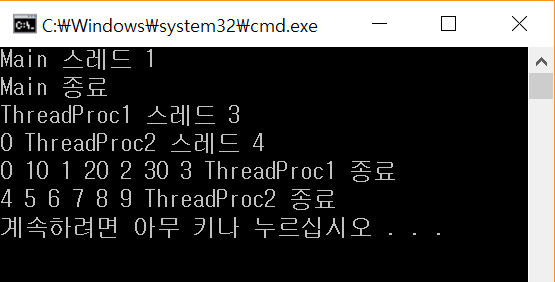
이 함수를 호출한 곳의 현재 스레드를 중지

ThreadAbortException 예외 발생 -> try ~ catch(객체에 대한 해제) 구문 필요

CurrentThread의 정적(static) 속성을 이용하여 자신의 스레드를 알아낼 수 있고 Abort()를 이용해서 호출하게 되면 자기 자신을 정지시킬 수 있음



<Abort 예시 코드>



<결과>

부 스레드 안에 부 스레드를 실행시킬 수 있음

(순서 Main 스레드 시작 > Main 스레드 종료 > 부 스레드1 시작(i=3이면 종료) > 부 스레드2 시작(반복문 끝나면 종료) > 부 스레드1 종료 > 부 스레드2 종료

* Resume(), Suspend()는 .Net 2.0 version 이후부터 사용하지 않는 메서드이며 제거됨

6. 윈도우 네트워크 프로그래밍에 스레드 적용

* 서버

서버 코드 안에서 스레드의 대상?

TCPListener : Client의 연결 상태(대기, 수락)관리, TCPClient 객체 생성

* 스레드 대상
* 클라이언트의 연결 수락 부분

TcpClient tcpClient = tcpListener.AcceptTcpClient();

* 데이터를 주고받는 부분



<Server> <Client>

Server : BinaryReader와 writer부분에서 스레드 사용

Client : 사용자가 스레드 역할, 스레드가 필요 없음

* 서버 시작과 함께 스레드 시작

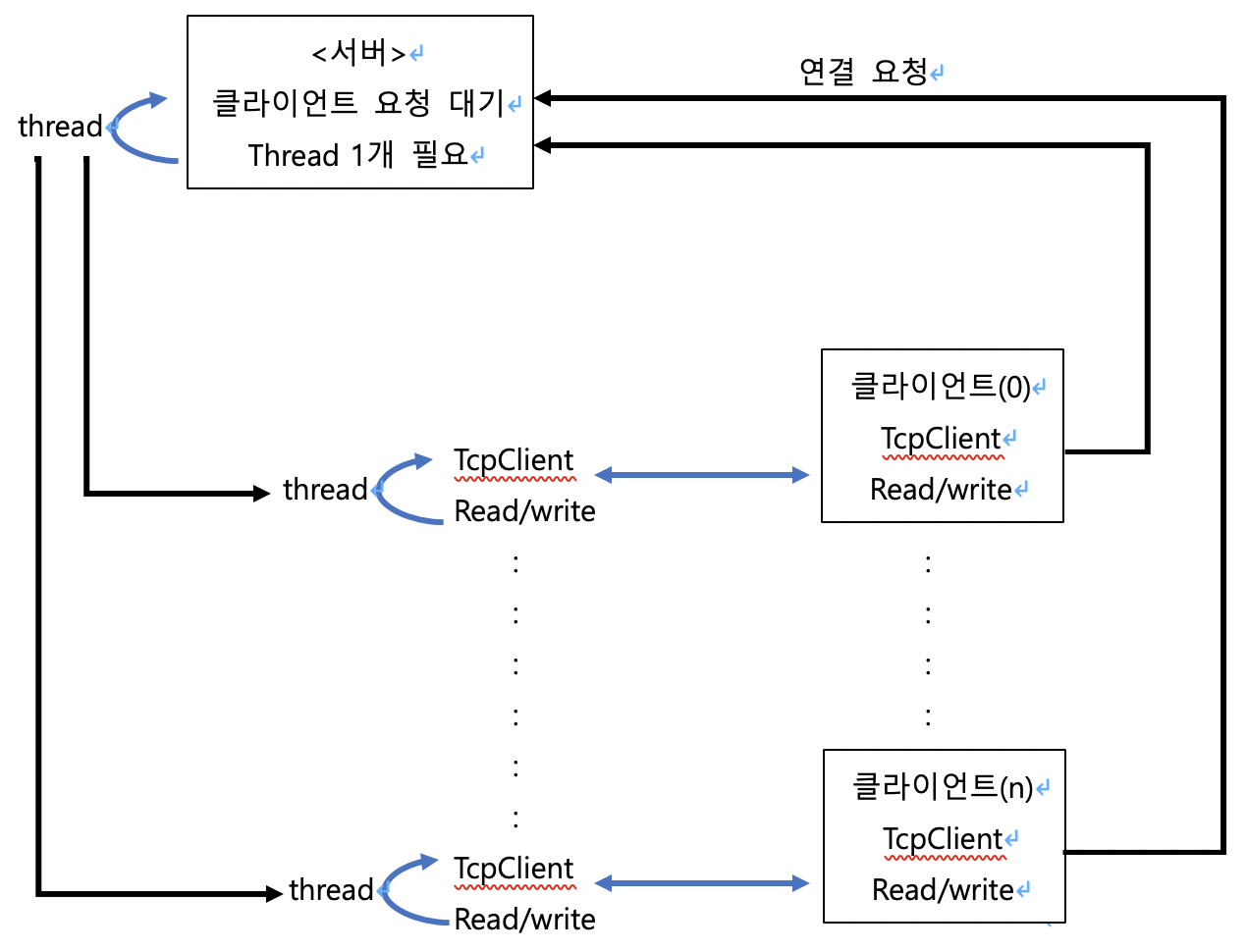
서버 시작이라는 버튼을 누름으로써 아래의 코드(스레드에 적용된)가 실행됨

thread

TCPClient tcpClient = tcpListener.AcceptTcpClient();

* TcpClient 일대일 구조 스레드로 생성

* 서버 스레드 구조 정리



5강 동기화

네트워크에서 동기화의 목적은 다수의 스레드가 공유 자원을 동시에 사용하고자 할 때 발생하는 문제들을 동기화를 통해 해결합니다.

1. 개요

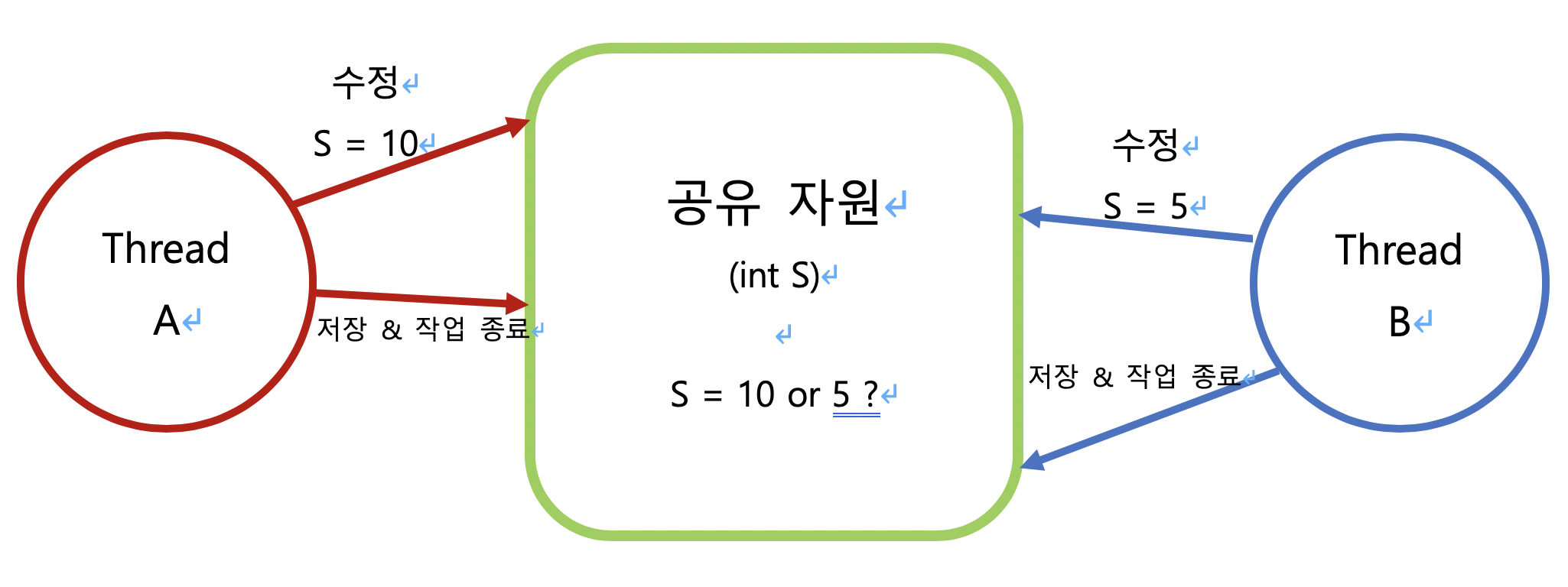
* 기본 개념

“동기화”란?

: 작업들 사이의 수행 시기를 맞추는 것 ➝ 다수의 스레드가 동시에 공유 자원을 사용할 때 순서를 정하는 것

: 동기화를 하지 않을 경우 다수의 스레드가 동시적으로 공유 자원의 정보를 수정하게 되고 이는 결과값 오류로 이어지게 됩니다.

ex) Thread A가 먼저 공유 자원의 값을 변경하고 있는 과정에서 Thread B가 중간에 들어와 A의 작업을 변경할 경우 공유 자원의 값이 정확하지 않게 됩니다.



* 동기화 키워드와 클래스

: lock만 동기화 키워드, 나머지는 모두 동기화 클래스입니다.

: lock, Monitor, Mutex는 기본적으로 사용하는 동기화 키워드 & 클래스입니다.

- lock (동기화 키워드)

- Monitor

- Mutex

- Interlocked

- AutoResetEvent

- ManualResetEvent

- ReaderWriterLock

* 동기화 대상
* 공유 자원에 대한 접근이 예상되는 스레드

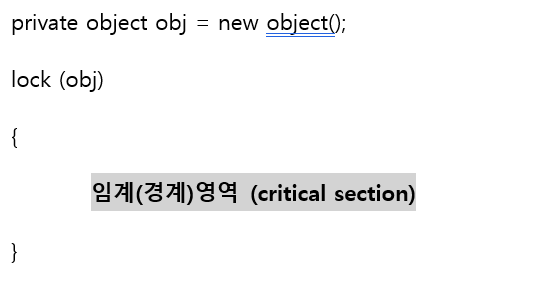
(= 스레드 함수 안에 공유 자원이 있는 경우)

⤋

* 한 객체를 다수의 스레드가 사용되는 때

2. lock

* lock 형식
* 기본 형식



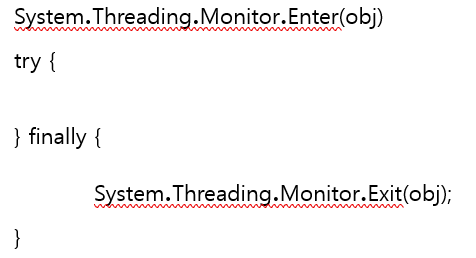
: lock에는 동기화하고자 하는 객체를 명시하도록 되어있습니다.

: obj는 값 형태는 쓸 수 없으며 반드시 객체 type으로 되어야 합니다.

: 임의의 객체(private object)를 선언하여 new에서 생성 후, 문법적으로 lock에 객체를 사용하여야 되기 때문에 대체할 수 있는 부분을 임의로 지정(obj)하는 방법입니다.

: critical section은 실제 thread가 독점하여 사용할 수 있는 영역을 의미합니다.

* 다른 형식



: lock의 기본 형식을 실제로 compile하게 될 경우 치환되는 코드입니다.

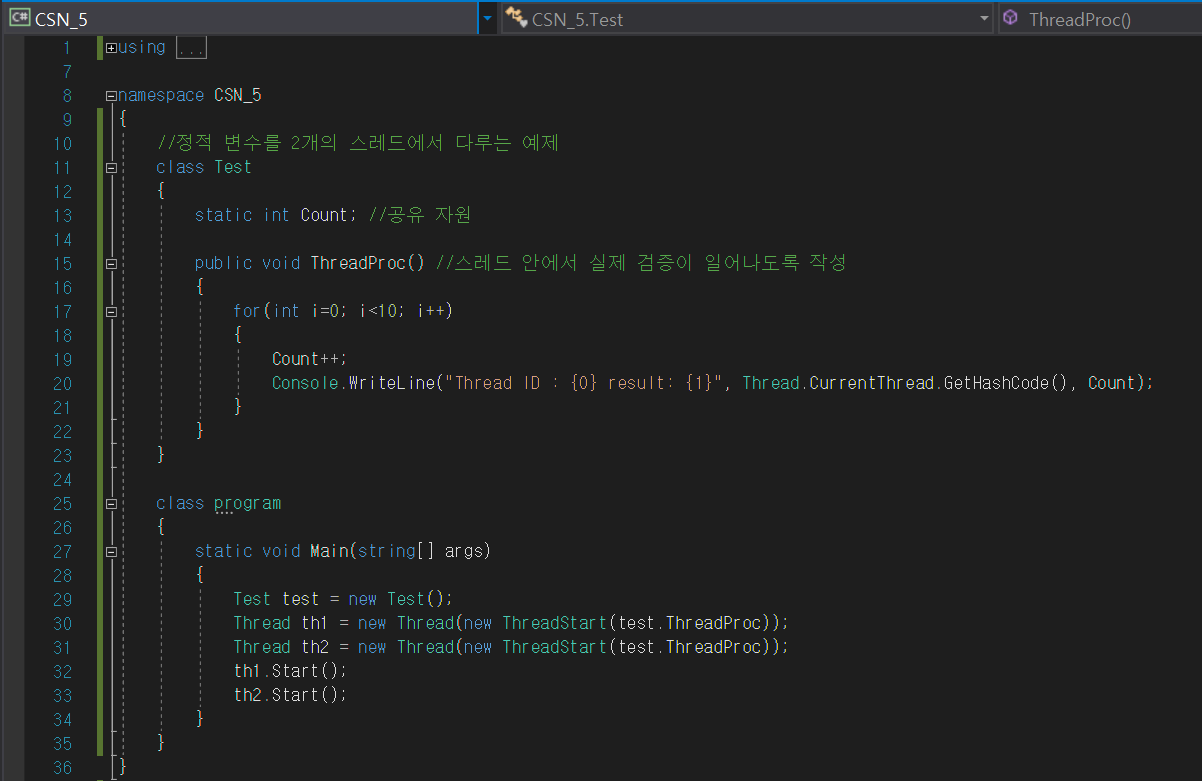
: lock이라는 키워드의 형태가 Monitor의 형태로 치환되어 실행됨을 확인할 수 있습니다.

* lock 주의할 점
* lock (this)
* lock (typeof(클래스명))

: 위의 두 부분을 사용할 수 있지만 지양하는 것이 좋음.

* lock 사용 예시
* 공유 자원(변수)을 스레드가 사용하는 예

1. lock을 사용하지 않을 경우

****

<공유 자원을 2개의 스레드에서 실행하는 코드>

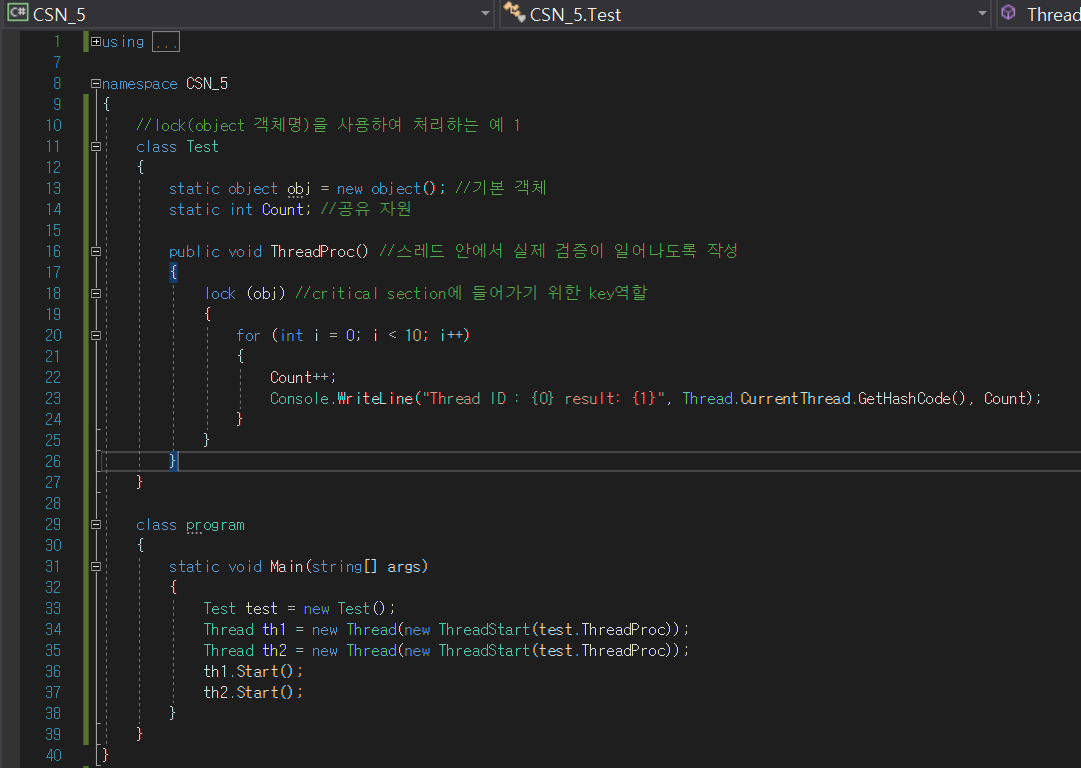
****

<결과값>

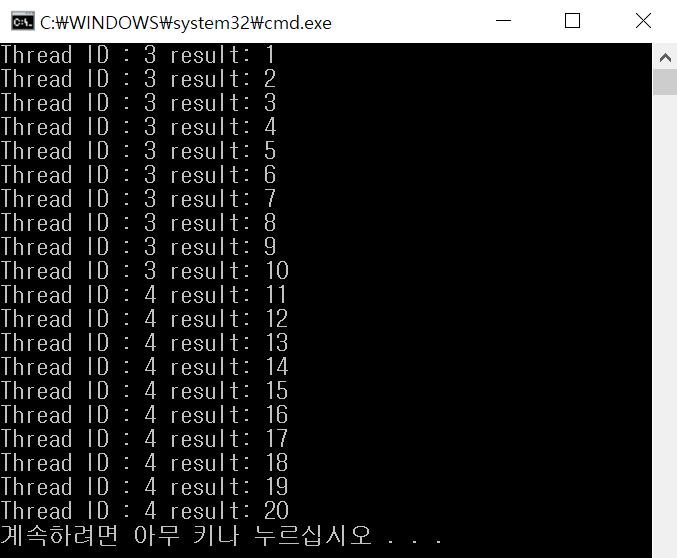
: 스레드에 상관없이 점유한 순간에 공유 자원 값이 변경됨을 확인할 수 있습니다.

(thread 3 : 1 / thread 4 : 2 / thread 3 : 3 / thread 4 : 4 ~ 12 / thread 3 : 13 ~ 20)

2. lock을 사용하는 경우



<공유 자원을 2개의 스레드에서 실행하는 코드>

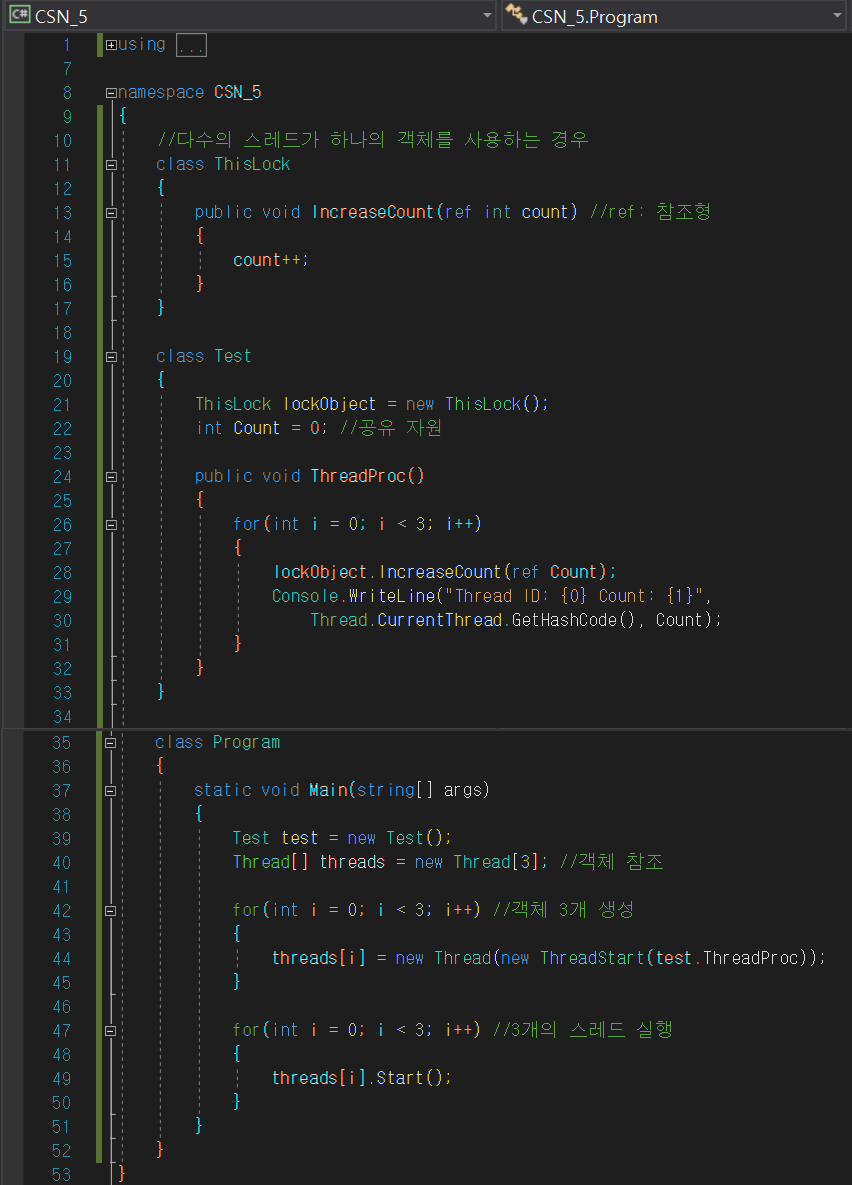


<결과값>

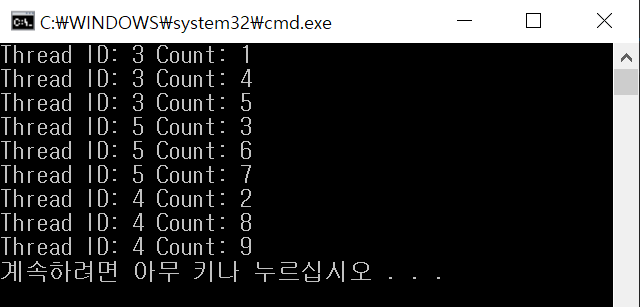
: lock으로 인해 스레드의 점유 순서에 따라 공유 자원 값이 변경됨을 확인할 수 있습니다.

* lock과 private object형을 사용한 동기화

1. lock을 사용하지 않을 경우



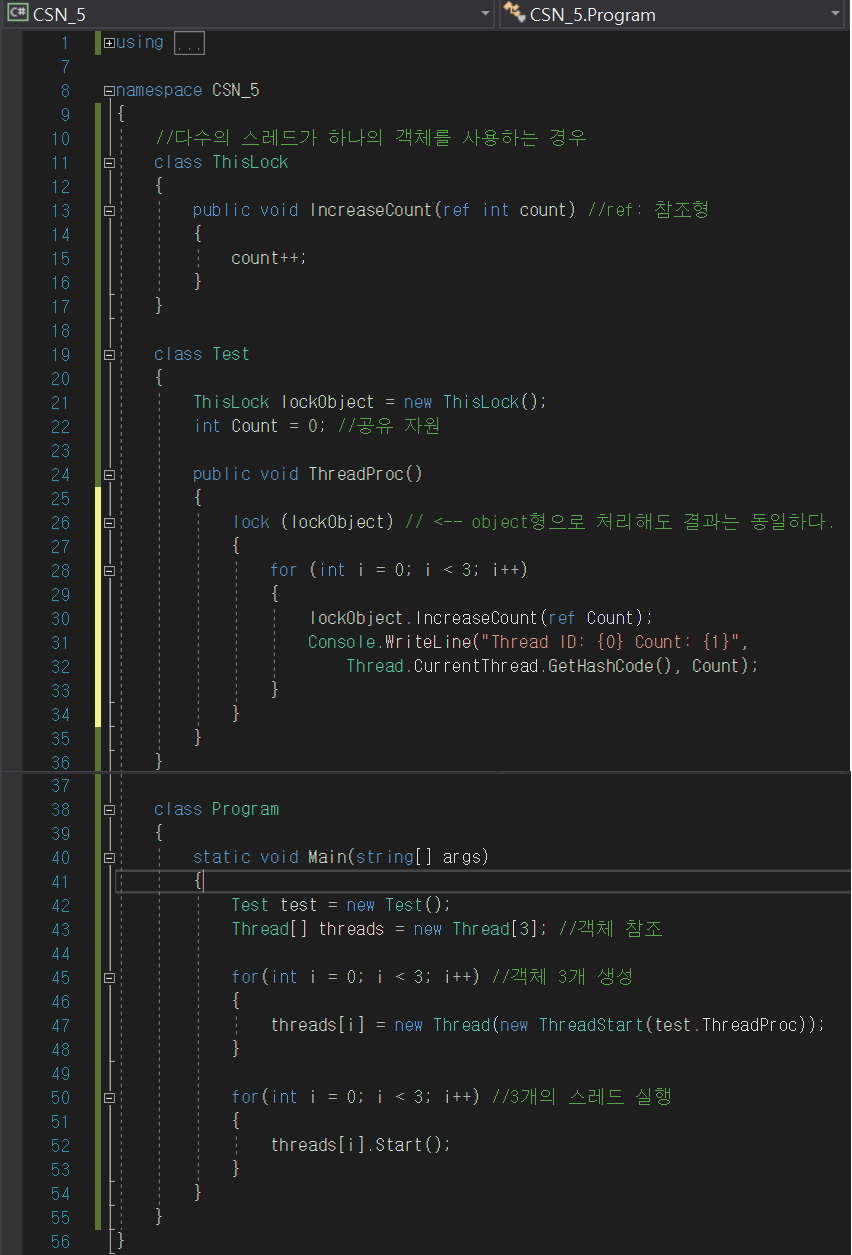
<공유 자원을 3개의 스레드에서 실행하는 코드>



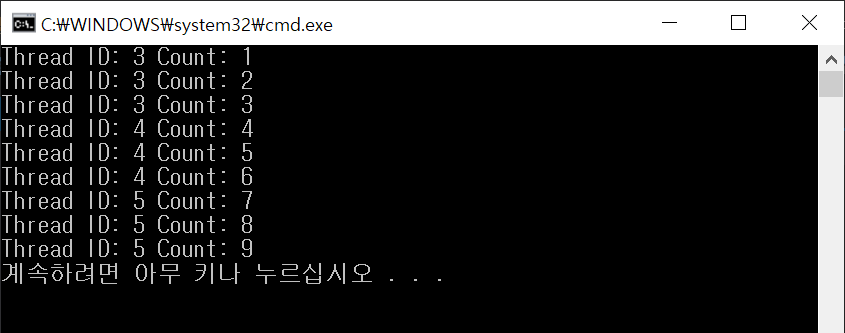
<결과값>

: 스레드에 상관없이 점유한 순간에 공유 자원 값이 변경됨을 확인할 수 있습니다.

2. lock을 사용할 경우



<공유 자원을 3개의 스레드에서 실행하는 코드>



<결과값>

3. Monitor

* 구성
* 클래스

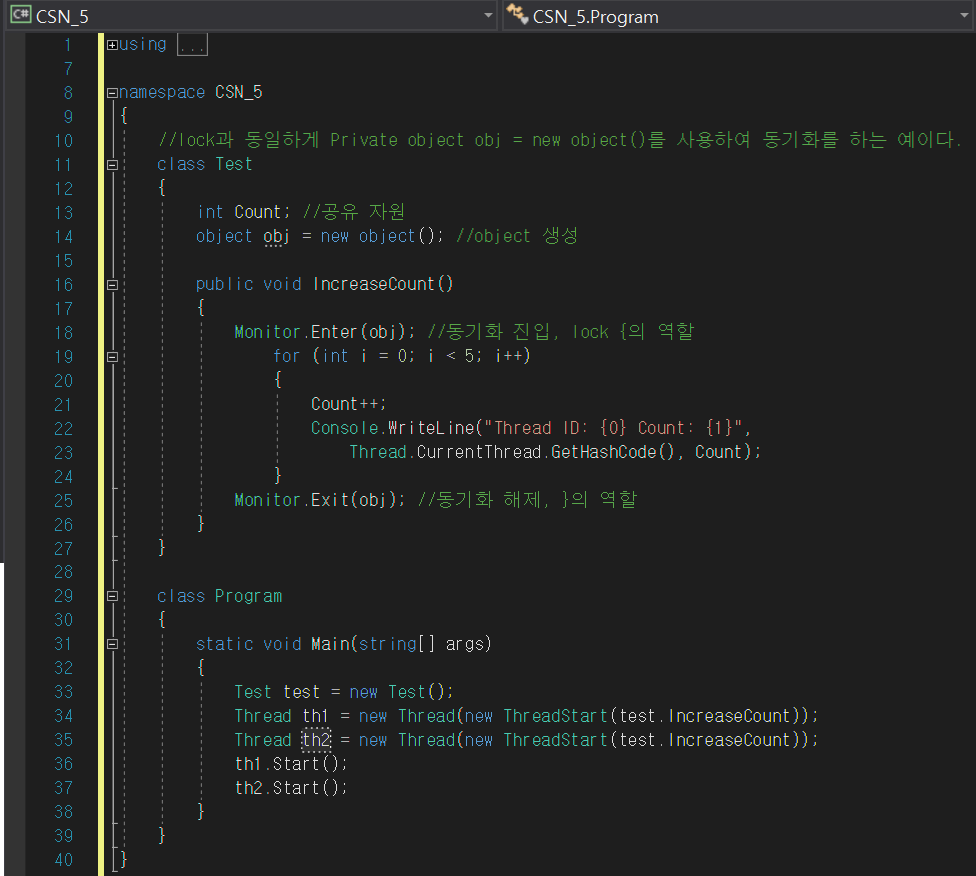
public static class Monitor

* 동기화 진입과 해제 메서드

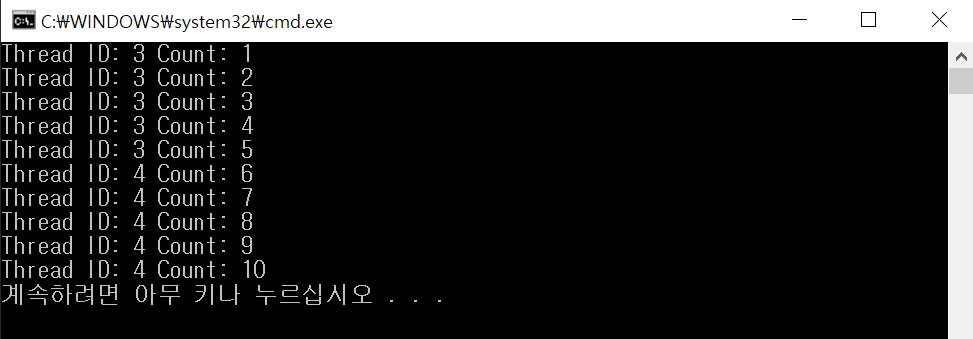
public static void Enter (object obj)

public static Exit (object obj)

* Monitor 사용 예시
* private object형을 사용한 동기화



<공유 자원을 2개의 스레드에서 실행하는 코드>



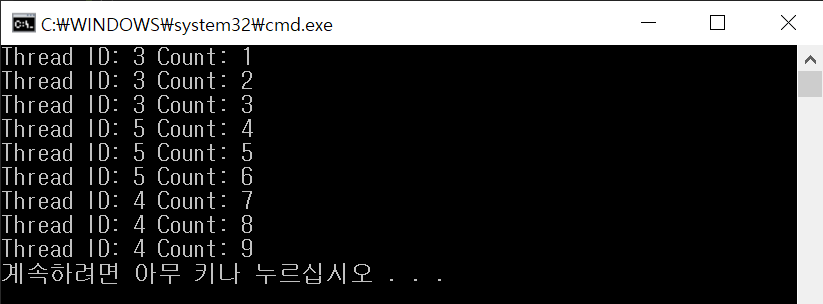
<결과값>

: 2개의 스레드가 순차적으로 실행됨을 확인할 수 있습니다.

* 객체 안에서 Monitor로 동기화하는 예



<공유 자원을 3개의 스레드에서 실행하는 코드>



<결과값>

4. Mutex

* 구성
* 클래스

public sealed class Mutex : waitHandle

: 상속을 할 수 없는(sealed), 단독으로 사용되는 클래스입니다.

: Monitor와 달리 new를 통해 클래스를 생성하여 사용해야 됩니다.

* 동기화 진입과 해제 메서드

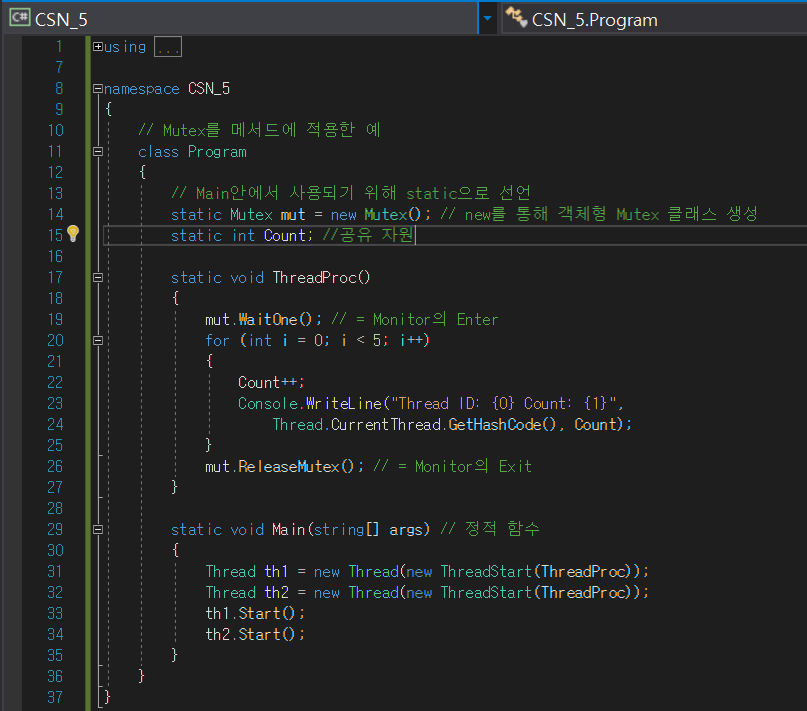
public virtual bool WaitOne()

: critical section 진입을 호출하는 메서드

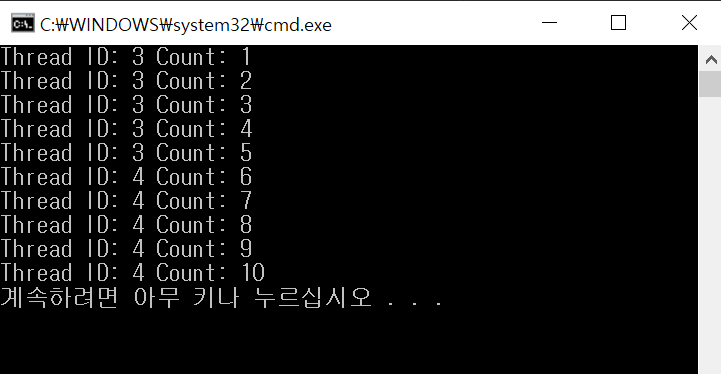
public void ReleaseMutex()

: critical section 해제를 호출하는 메서드

* Mutex 사용 예시
* Mutex를 사용한 메서드 동기화



<공유 자원을 2개의 스레드에서 실행하는 코드>

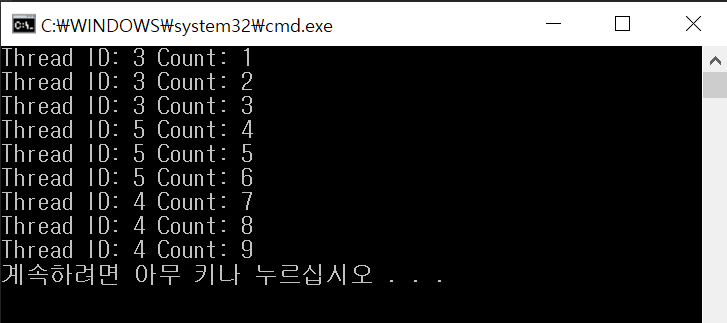


<결과값>

* 객체 안에서 Mutex로 동기화하는 예



<공유 자원을 3개의 스레드에서 실행하는 코드>



<결과값>

5. 정리

* 어디에 사용할 것인가?

: 동기화를 배웠다고 해서 모든 부분에 동기화를 사용할 것은 아니기에 동기화를 어디에 사용할 것인가를 생각해야 합니다.

* 다수의 스레드가 공유 자원을 사용하는가?

: 다수의 스레드가 실행될 경우, 분명히 공유 자원이 발생하게 되기에 공유 자원을 쓸 때 어떤 순서로 진행을 할 것인가를 동기화를 통해 해결해야 합니다.

* 필요에 따라 사용해 보고 기본 동기화 외의 클래스를 적용해 볼 것(MSDN에서 설명)
* Interlocked
* AutoResetEvent
* ManualResetEvent
* ReaderWriterLock

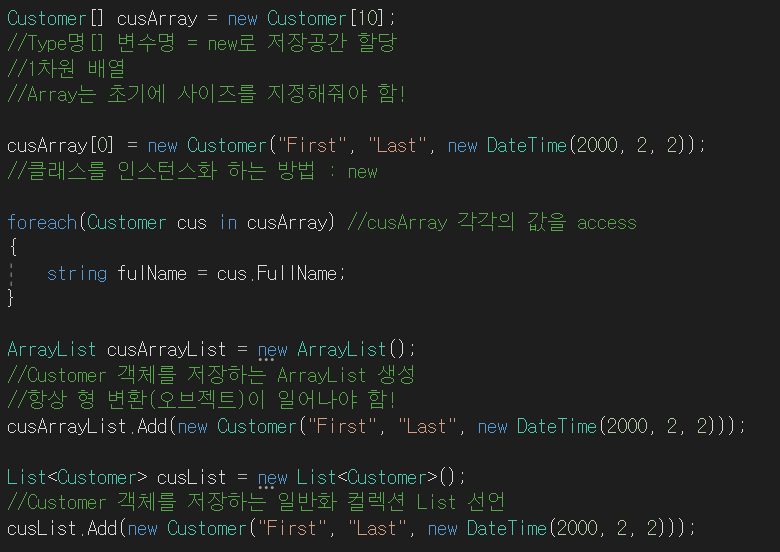
**C# 초보 강좌 예제로 배우는 C# 11**

**: 클래스 상속, 가상 메서드, as/is 연산자**

* 복습

: 배열 선언, ArrayList 선언(기본 Collection), List<>(일반화 Collection)

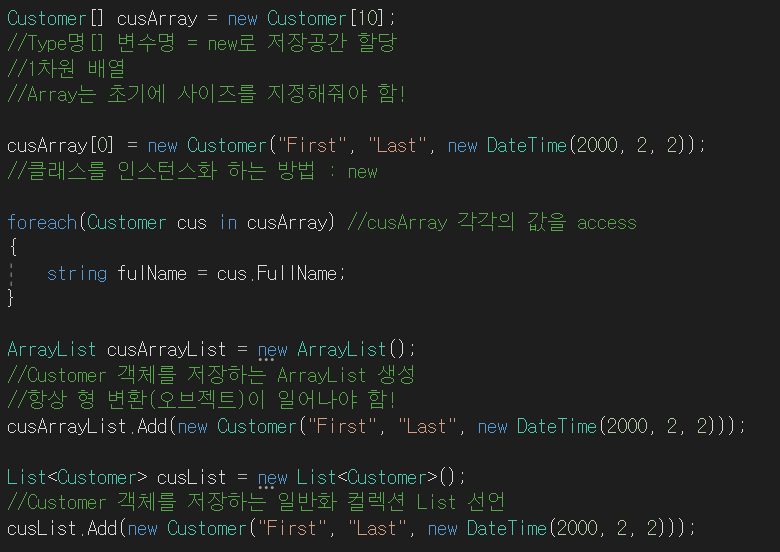
<배열 선언>

****

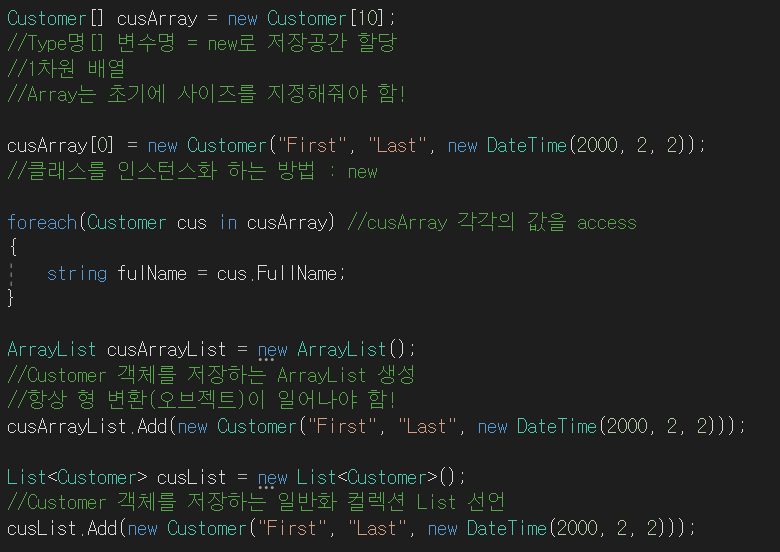
: Type명[] 변수명 = new 생성자로 저장공간 할당하여 배열을 생성합니다.

: 위의 배열은 1차원 배열이며 다중 배열을 생성할 수 있습니다.

: 단점으로 Array는 생성할 때 사이즈(크기)를 지정해줘야 합니다.

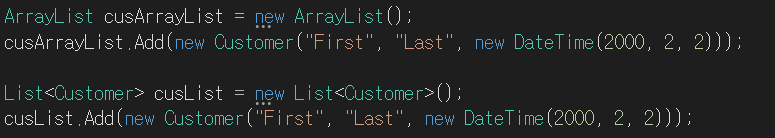
****

: 클래스를 인스턴스화 하는 방법으로는 new 생성자를 이용하는 것입니다.

****

: 배열 안의 값을 개별적으로 모두 access하고 싶은 경우에는 foreach문을 사용합니다.

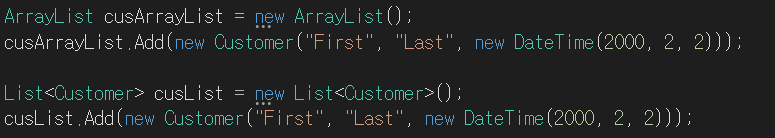
<ArrayList 선언>

****

: Customer 객체를 저장하는 기본 Collection인 ArrayList를 생성합니다.

            : 단점으로는 항상 오브젝트 타입으로 형 변환이 일어나야 합니다.

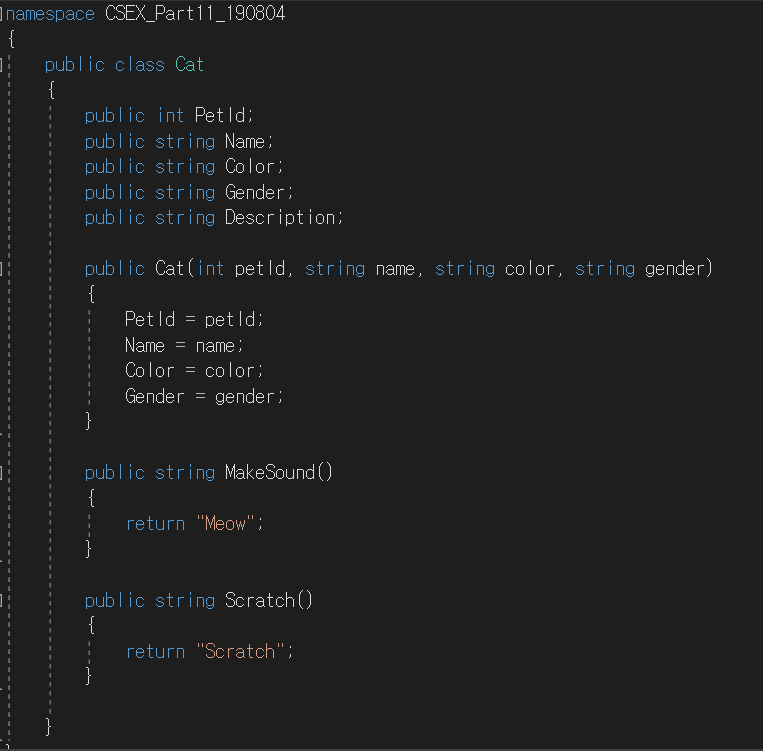
<List<> 선언>

****

: Customer 객체를 저장하는 일반화 Collection인 List<>를 생성합니다.

* 새로운 클래스 생성

<Cat.cs 클래스 생성>

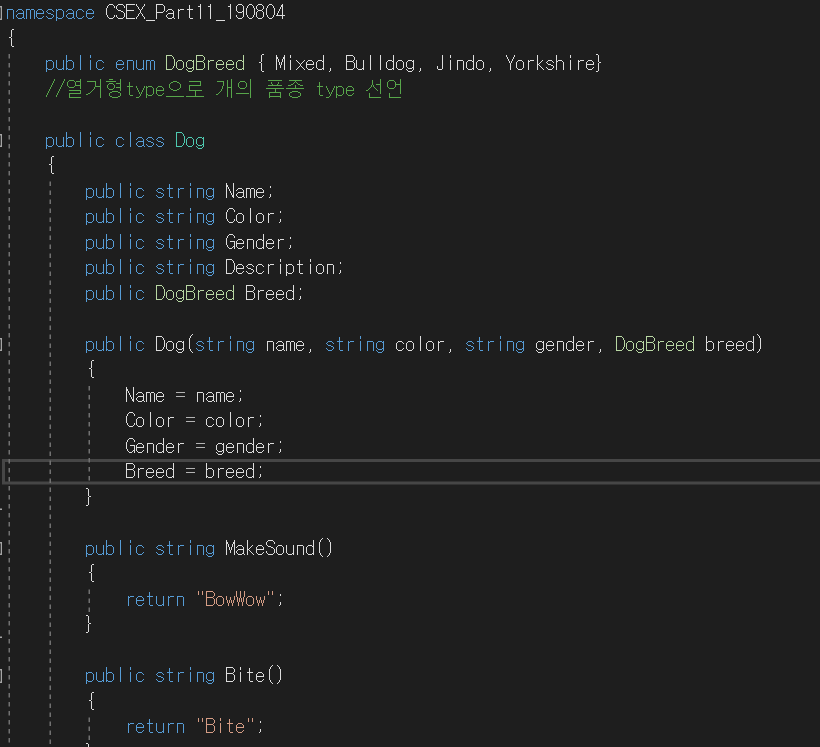


: 입양 동물 중 하나인 고양이에 대한 클래스입니다.

: 생성자 Cat을 생성하여 Customer 클래스에서 Cat을 호출 시 Cat 안의 변수들(PetId, Name, Color, Gender, Description) 을 사용할 수 있도록 합니다.

: Cat 클래스만의 메소드인 Scatch()를 생성합니다.

<Dog.cs 클래스 생성>



: 입양 동물 중 하나인 강아지에 대한 클래스입니다.

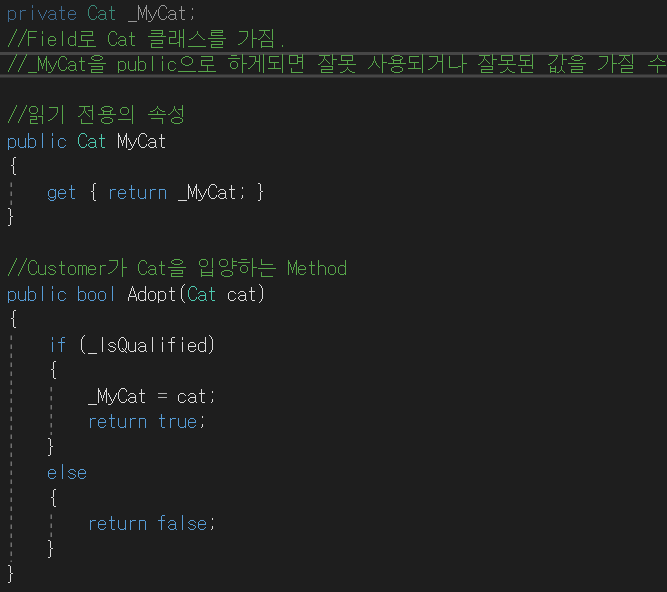
: 열거형으로 개의 품종들을 저장한 DogBreed라는 데이터 type을 정의합니다.

: DogBreed라는 데이터 type을 Dog 클래스가 가질 수 있도록 Breed라는 이름의 field를 생성합니다.

: 생성자 Dog를 생성하여 Customer 클래스에서 Dog를 호출 시 Dog 안의 변수들(PetId, Name, Color, Gender, Breed) 을 사용할 수 있도록 합니다.

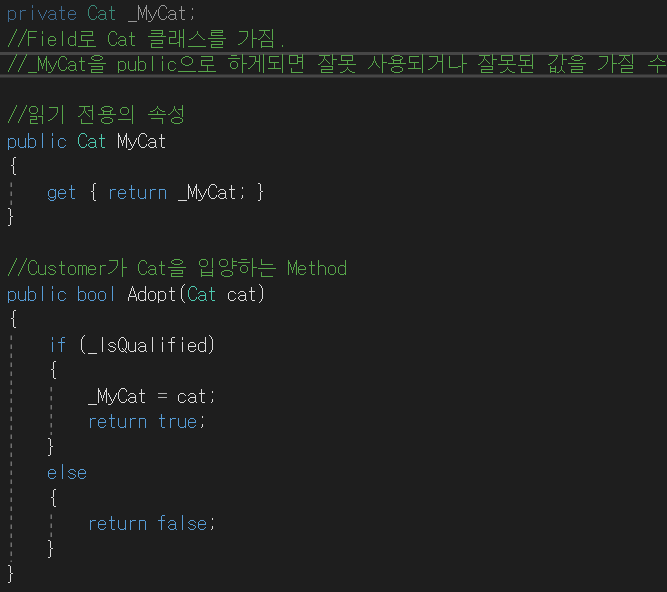
: Dog 클래스만의 메소드인 Bite()를 생성합니다.

* Customer.cs 클래스 수정



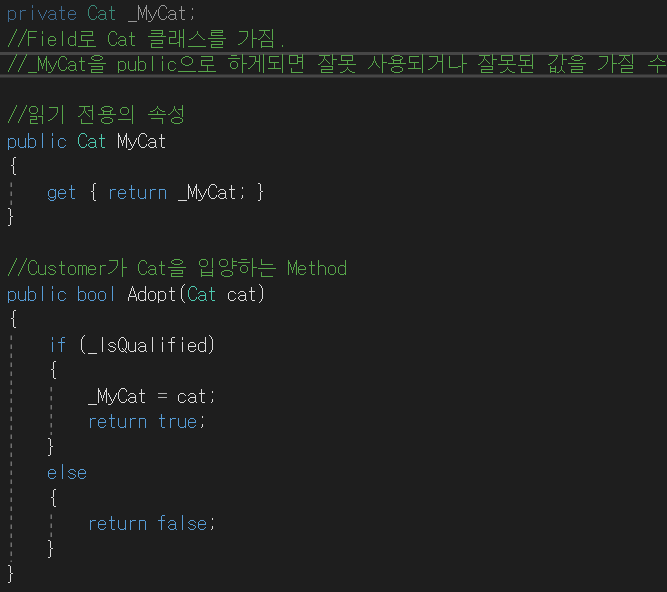
: Field로 Cat 클래스를 가지는 \_MyCat을 선언합니다.

: public의 경우는 잘못 사용되거나 잘못된 값을 가질 수 있으므로 객체가 스스로 모순이 없도록 private로 합니다.



: 속성으로 Cat 클래스를 가지는 MyCat을 생성합니다.

: get만 존재하므로 읽기 전용의 특성을 가지게 됩니다.

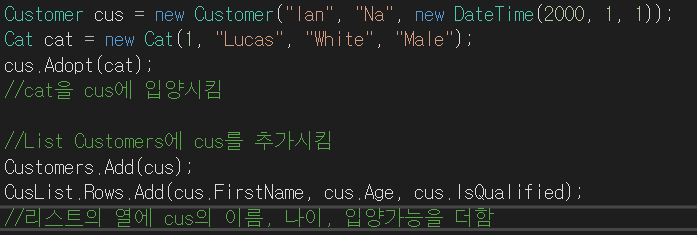


: Customer가 Cat을 입양하는 메소드인 Adopt를 생성합니다.

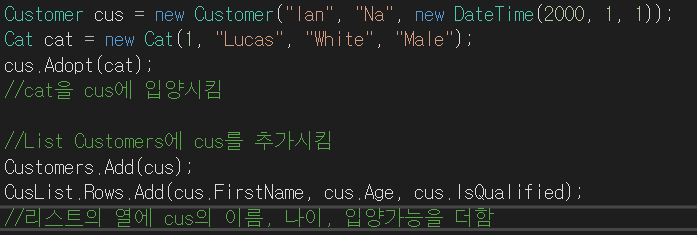
: 이때 Adopt의 type을 bool형으로 설정하여 return 값이 true/false가 될 수 있도록 합니다.

: \_IsQualified로 Customer가 입양이 가능한지의 여부를 판단 후, 가능하면 \_MyCat에 cat을 할당 후 return 값으로 true를, 불가능하면 return 값으로 false를 돌려줍니다.

* Form1.cs 클래스 수정

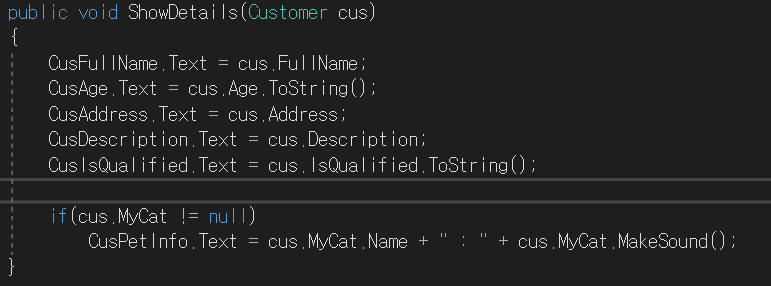


: 새로운 Customer와 Cat을 생성후 cat을 cus에 입양시킵니다.

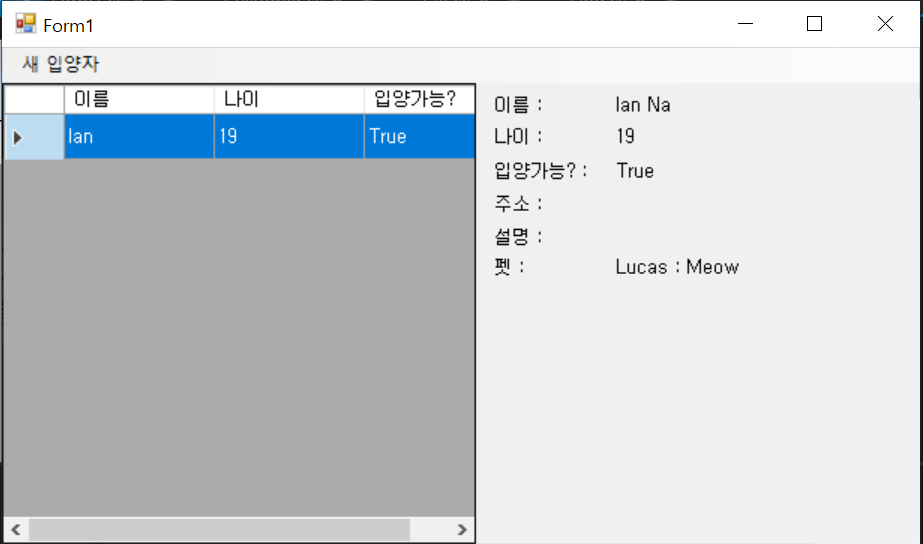


: List Customers에 cus를 추가시킵니다.

: CusList의 열에 cus의 정보(이름, 나이, 입양가능)를 추가합니다.



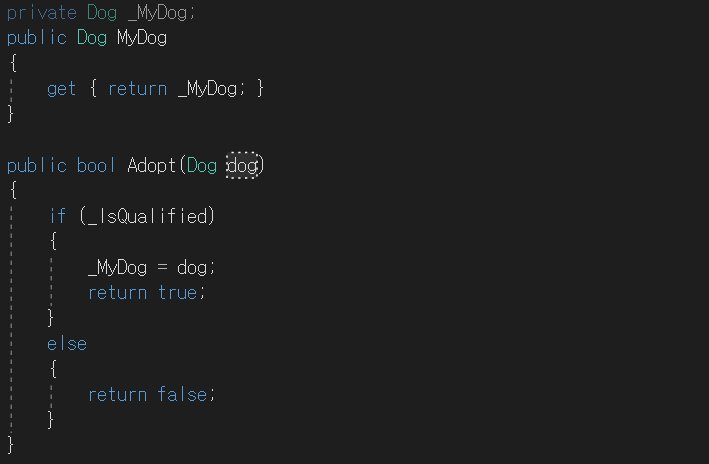
: cus의 정보들을 보여주는 ShowDetails 함수에 Customer가 고양이를 가지고 있는지 판별하여 있는 경우에만 고양이 정보를 줄 수 있도록 조건문을 생성합니다.



<중간 실행 결과>

: 초기에 cus의 정보들이 CusList안에 들어가 있는 것을 확인할 수 있으며 클릭시 cus가 입양한 고양이의 정보도 나타나게 됩니다.

* Customer.cs 클래스 수정



: 위의 Cat을 생성하였을 경우와 같은 것을 이름만 바꾸어서 Dog의 경우를 생성합니다.

: Adopt라는 동일한 명의 함수가 Cat, Dog의 경우로 2개 생성되는데 이런 경우의 함수를 Overloading이라고 합니다.

⋇ Overloading : 매개 변수를 다르게 해서 같은 이름의 함수를 2개 이상 선언하는 것

* Form1.cs 클래스 수정

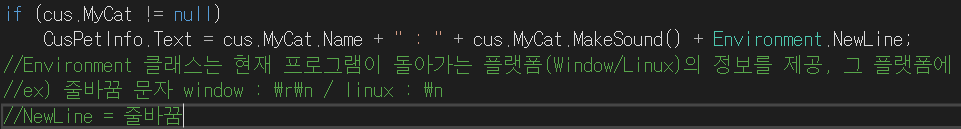




: 위의 Cat을 생성하였을 경우와 같은 것을 이름만 바꾸어서 Dog의 경우를 생성합니다.

: 그러나 Dog의 경우는 Breed를 새로 생성하여야 합니다.

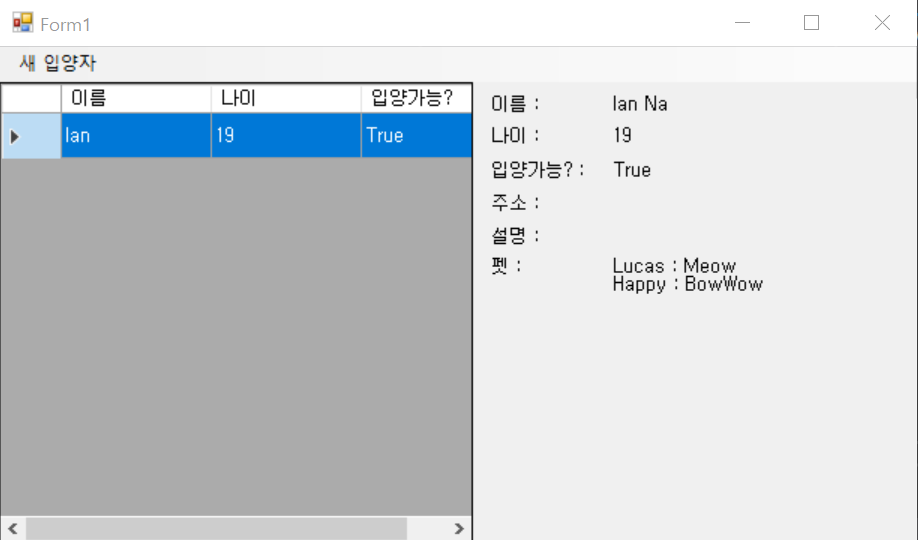
: 또한 Cat의 Information과 중첩되지 않기 위해 ‘+=’로 표시해줍니다.



: Environment 클래스는 현재 프로그램이 돌아가는 플랫폼(Window/Linux)의 정보를 제공하고 그 플랫폼에 맞는 값을 제공합니다.

: 예시로 줄바꿈 문자의 경우, Window는 “\r\n”이지만 Linux는 “\n”으로 나타냅니다.

: NewLine은 줄바꿈을 해줍니다.



<중간 실행 결과>

: Customer의 정보가 있는 열을 클릭하게 되면 Custome에 저장되어있던 Cat과 Dog의 정보가 줄바꿈 형태로 나오는 것을 확인할 수 있습니다.