스레드와 태스크

이것이 C#이다

Contents

- 프로세스와 스레드
- Task와 Task<TResult>, 그리고 Parallel
- async 한정자와 await 연산자로 만드는 비동기 코드

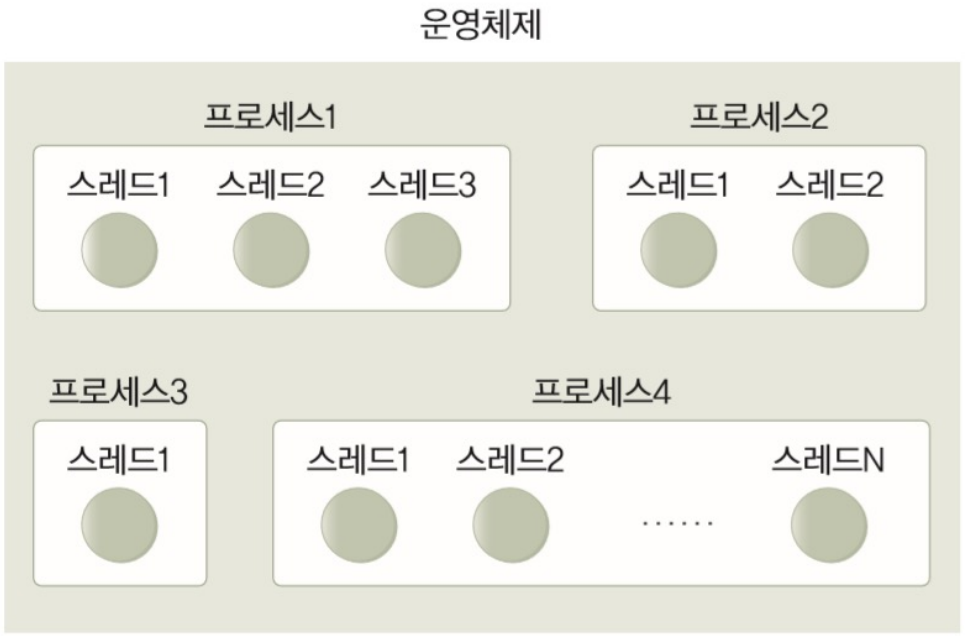
19.1 프로세스와 스레드(1)

❖ 프로세스

- 실행 파일이 실행되어 메모리에 적재된 인스턴스
- 하나 이상의 스레드Thread로 구성

❖ 스레드

- 운영체제가 CPU 시간을 할당 하는 기본 단위

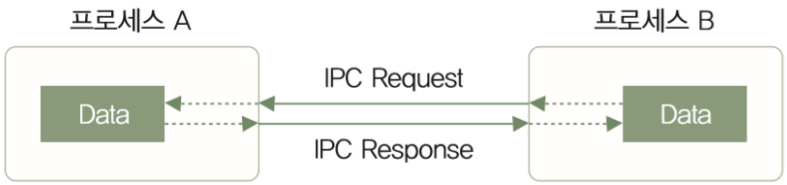


[운영체제와 프로세스, 프로세스와 스레드]

19.1 프로세스와 스레드(2)

❖ 멀티 스레드의 장점

- 사용자 대화형 프로그램(
- 경제성 - 메모리와 자원을
- 멀티 프로세스 방식에 비 다



[IPC를 통한 프로세스 간의 데이터 교환]

[변수를 이용한 스레드 간의 데이터 교환]

❖ 멀티 스레드의 단점

-
- text Switching)
- 향을 끼침

[멀티 프로세스 구조에서 자식 프로세스에 문제가 생긴 경우]

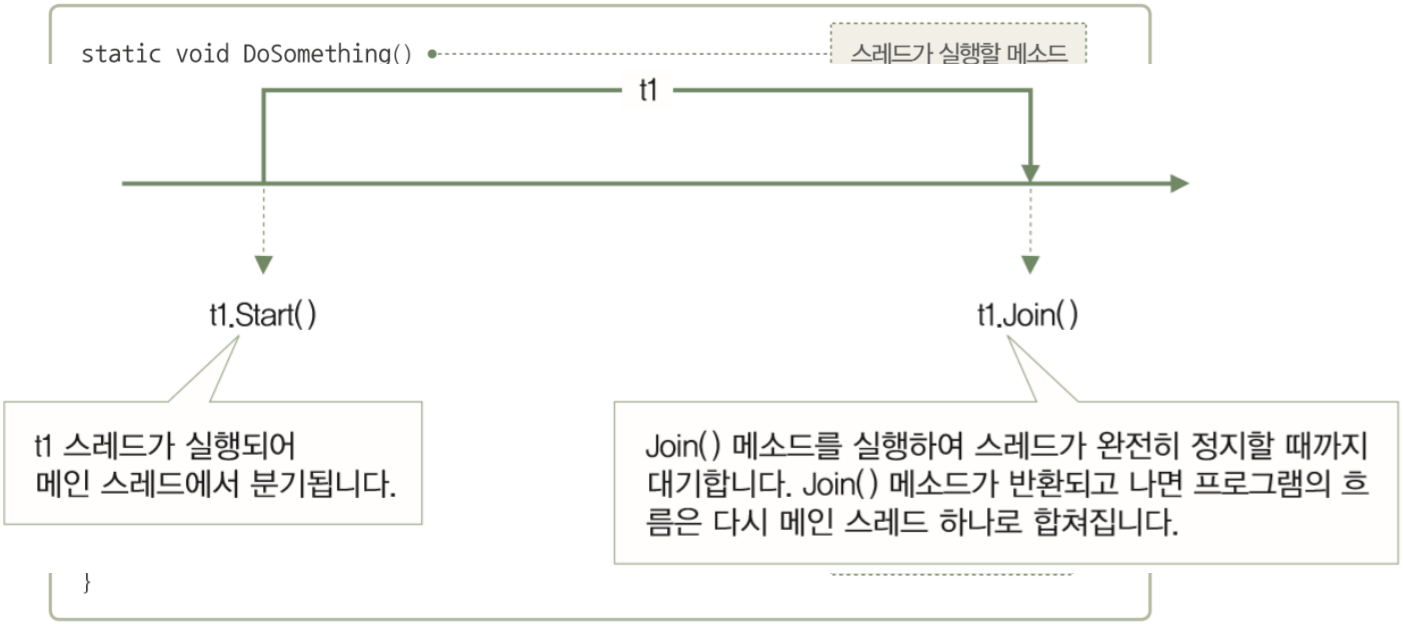
[멀티 스레드 구조에서 자식 스레드에 문제가 생긴 경우]

19.1.1 스레드 시작하기

❖ System.Threading.Thread

- Thread의 인스턴스를 생성. 생성자의 매개 변수로 실행할 메소드 전달
- Thread.Start() 메소드 호출 → 스레드 시작
- Thread.Join() 메소드 호출 → 종료 시 까지 대기

❖ 사용 사례

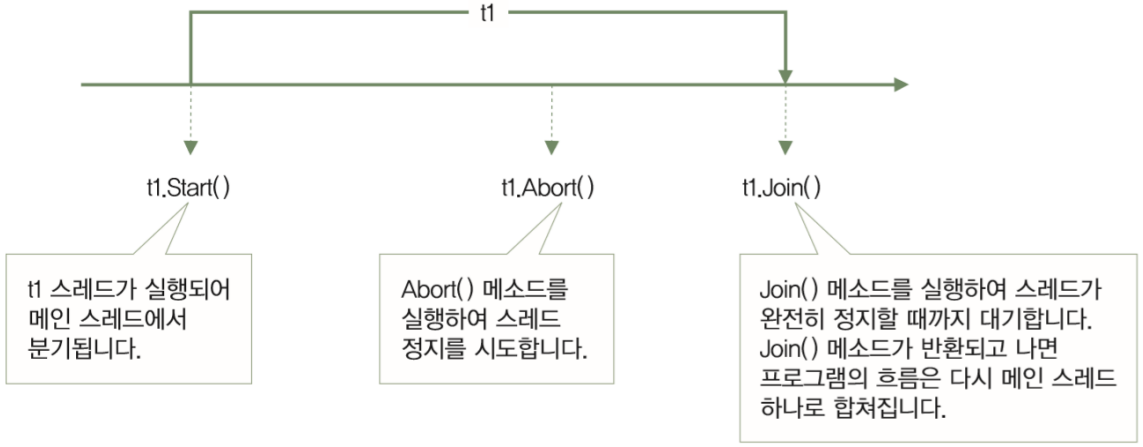


❖ 데모 예제 - BasicThread

19.1.2 스레드 임의로 종료시키기

❖ Thread 객체의 Abort() 메소드 사용

- 실행 중이던 코드에 ThreadAbortException 호출
  - 예외 catch 코드가 있는 경우 finally 블록 실행 후 스레드 종료
- 동작하던 스레드가 즉시 종료된다는 보장은 없음

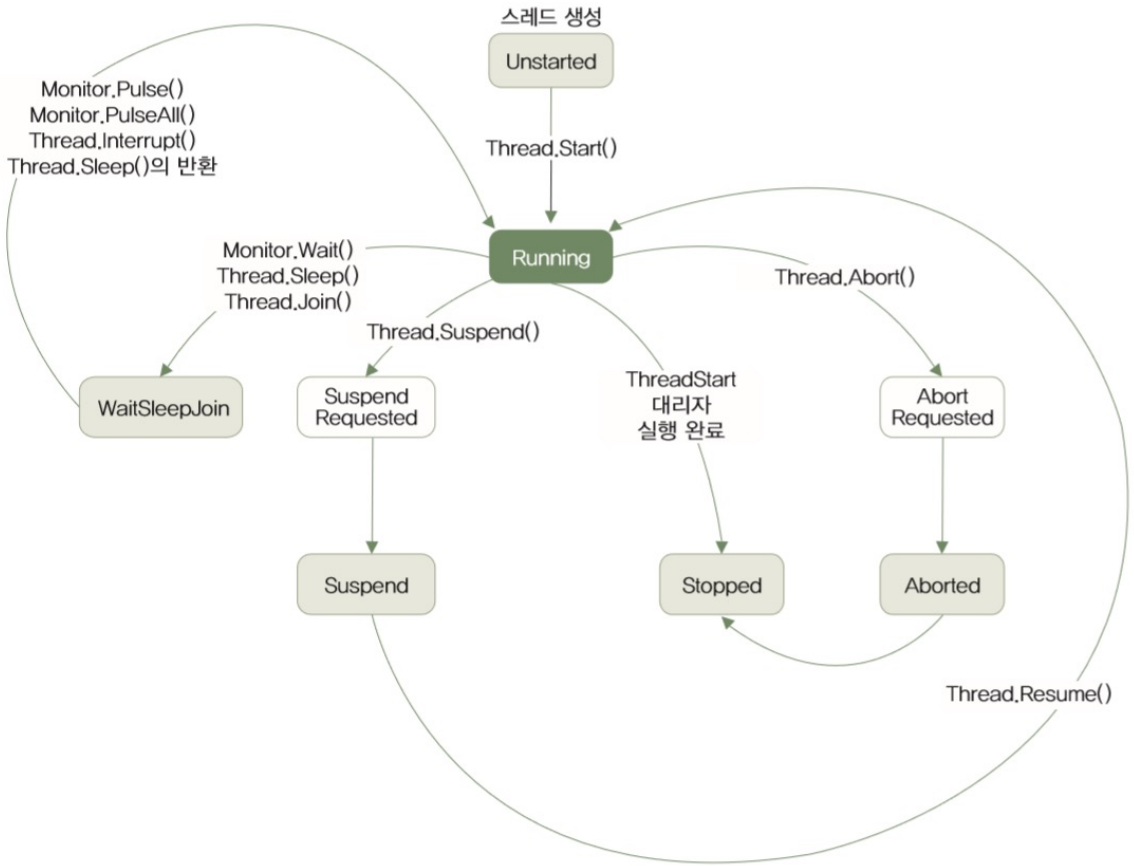


❖ 권장하지 않음

- 자원을 독점한 스레드가 해제 못한 상태로 Abort의 희생양이 되는 문제
- 꼭 해야 한다면 → 도중에 강제로 중단해도 프로세스 자체나 시스템에 영향 주지 않는 작업

❖ 데모 예제 – AbortingThread

9.1.3 스레드의 일생과 상태 변화



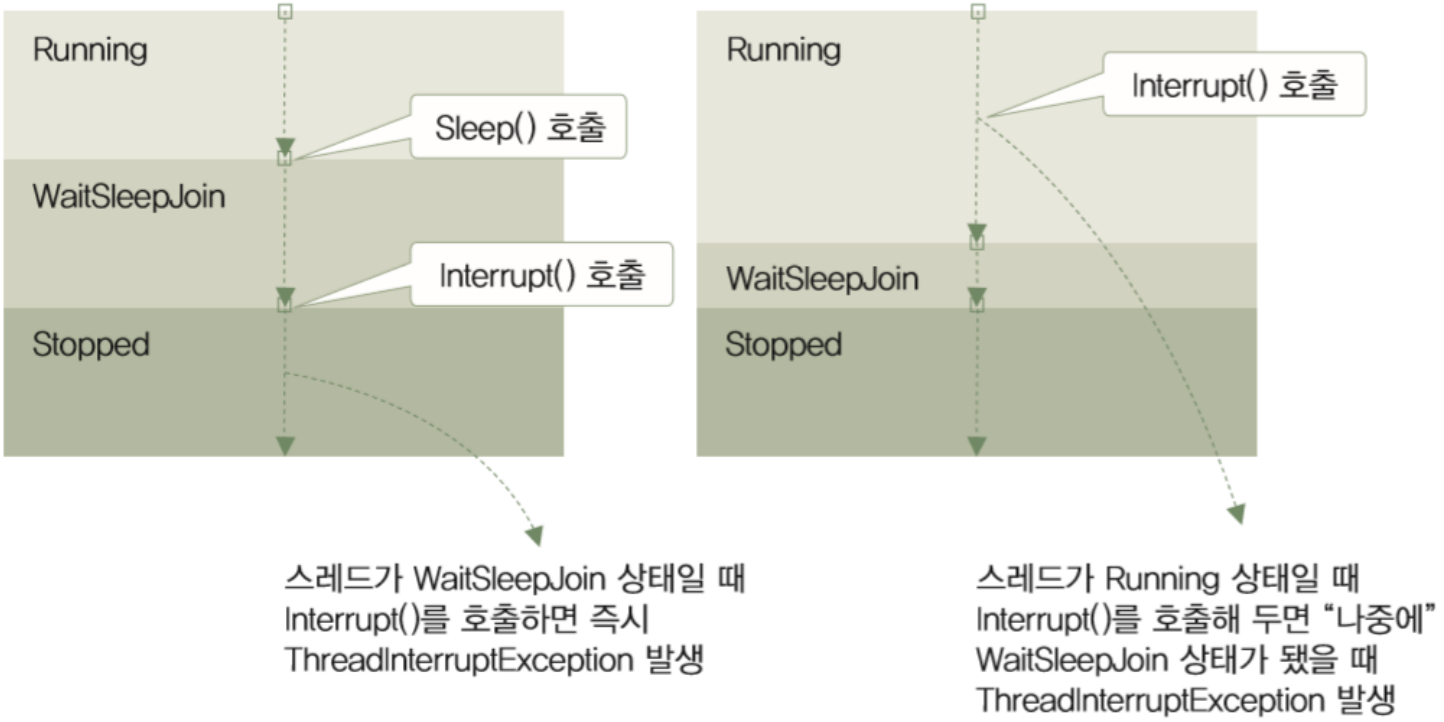
상태	십진수	이진수
Running	0	000000000
StopRequested	1	000000001
SuspendRequested	2	000000010
Background	4	000000100
Unstarted	8	000001000
Stopped	16	000010000
WaitSleepJoin	32	000100000
Suspended	64	001000000
AbortRequested	128	010000000
Aborted	256	100000000

❖ 데모 예제 - ThreadState

19.1.4 스레드를 임의로 멈추는 또 하나의 방법: 인터럽트

❖ 스레드 작업의 강제 중단이 시스템에 악영향을 미칠 경우 대안

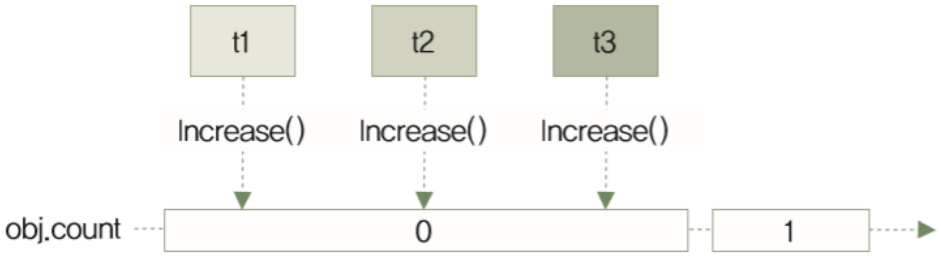
- Thread.Interrupt() 메소드
- →WaitJoinSleep 상태에서 ThreadInterruptedException 예외 던짐



❖ 데모 예제 – InterruptingThread

19.1.5 스레드 간의 동기화 - lock 으로 동기화

- 스레드들이 순서를 갖춰 자원을 사용하게 하는 것
- 자원을 한번에 하나의 스레드가 사용하도록 보장



- 코드 영역을 한 번에 한 스레드만 사용하도록 보장

```
public void Increase()
{
    lock ( thisLock )
    {
        count = count + 1;
    }
}
```

lock 키워드와 중괄호로 둘러싼 이 부분은 크리티컬 섹션이 됩니다. 한 스레드가 이 코드를 실행하다가 lock 블록이 끝나는 괄호를 만나기 전까지는 다른 스레드는 절대 이 코드를 실행할 수 없습니다.

  - 외부 코드에서도 접근할 수 객체를 lock의 매개 변수로 사용 금지
    - lock(this), lock(typeof (SomeClass))나 lock(obj.GetType()), lock("abc")

- 데모 예제 - AnonymousType

19.1.5 스레드 간의 동기화 - Monitor로 동기화

❖ Monitor 클래스의 스레드 동기화 (정적)메소드

- Monitor.Enter() - 크리티컬 섹션 만들기
- Monitor.Exit() - 크리티컬 섹션 제거하기

❖ Lock vs. Monitor 클래스의 메소드

lock

```
public void Increase()
{
    int loopCount = 1000;
    while (loopCount-- > 0)
    {
        lock (thisLock)
        {
            count++;
        }
    }
}
```

Monitor.Enter( )와 Monitor.Exit( )

```
public void Increase()
{
    int loopCount = 1000;
    while (loopCount-- > 0)
    {
        Monitor.Enter(thisLock);
        try
        {
            count++;
        }
        finally
        {
            Monitor.Exit(thisLock);
        }
    }
}
```

❖ Lock은 Monitor의 Enter()와 Exit() 메소드를 바탕으로 구현

❖ 데모 예제 - UsingMonitor

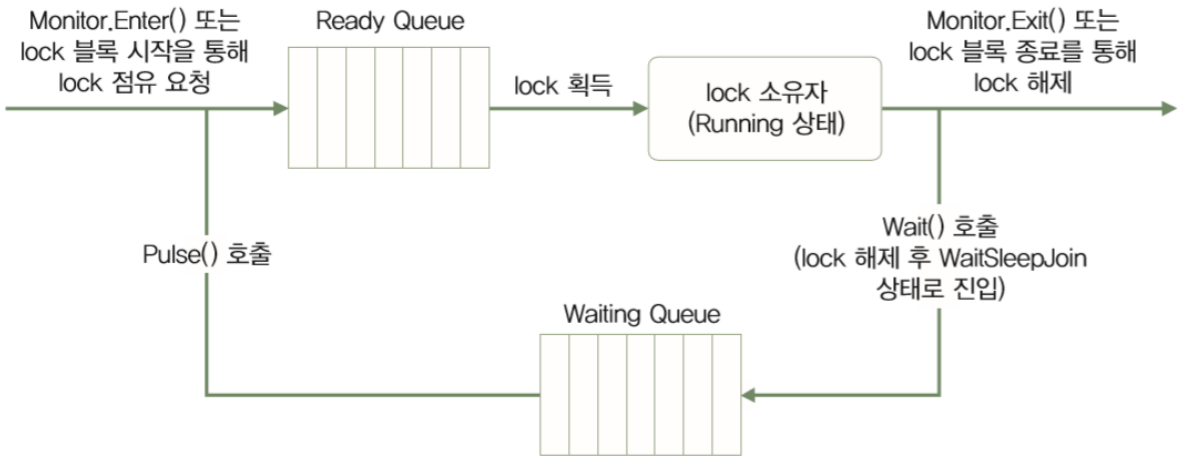
19.1.5 스레드 간의 동기화 – Monitor.Wait( )와 Monitor.Pulse( )로 하는 저수준 동기화

❖ Monitor 클래스를 사용해야 하는 시나리오

- 보다 섬세한 멀티 스레드 동기화 → 성능 향상

❖ 사용 방법

- 반드시 lock 블록 안에서 호출
- Monitor.Wait() - 스레드를 WaitSleepJoin 상태로 만들고 Waiting Queue에 입력
- Monitor.Pulse() - Waiting Queue 첫 요소 스레드를 꺼내 Ready Queue에 입력



❖ 데모 예제 – WaitPulse

19.2 Task와 Task<TResult>, 그리고 Parallel

- **멀티 코어 시대 고성능 소프트웨어 개발**
  - 병렬 처리 기법과 비동기 처리 기법
- **마이크로소프트의 대응**
  - .NET 프레임워크 4.0
  - System.Threading.Tasks 네임스페이스
  - →Task와 Task<TResult>, 그리고 Parallel 클래스

19.2.1 System.Threading.Tasks.Task 클래스

❖ 동기 코드 vs 비동기 코드

- 동기 코드 - 메소드 호출 후 실행이 종료 (반환)되야 다음 메소드 호출
- 비동기 코드 - 메소드 호출 후 종료를 기다리지 않고 다음 코드 실행

❖ Task 클래스를 사용한 비동기 구현

- 인스턴스를 생성할 때 Action 대리자를 받는다

```
var myTask = Task.Run( ()=>
    {
        Thread.Sleep(1000);
        Console.WriteLine(" Printed asynchronously.");
    }
);
Console.WriteLine("Printed synchronously.");

myTask.Wait();

/* 결과는
Printed synchronously.
Printed asynchronously. */
```

Task의 생성과 시작을 단번에 합니다. 덤으로 Task가 실행할 Action 대리자도 무명 함수로 바꿔봤습니다.

❖ 데모 예제 - UsingTask

19.2.2 코드의 비동기 실행 결과를 주는 Task<TResult>

- 코드의 비동기 실행 결과를 손쉽게 얻게 한다.
- Task 클래스의 사용법과 같으나 다른 점
  - 비동기로 수행할 코드를 Func 대리자로 받음
  - 결과를 반환 받을 수 있음
- 사용 사례

```
var myTask = Task<List<int>>.Run(
    () =>
    {
        Thread.Sleep(1000);
        List<int> list = new List<int>();
        list.Add(3);
        list.Add(4);
        list.Add(5);

        return list;
    }
);
var myList = new List<int>();
myList.Add(0);
myList.Add(1);
myList.Add(2);

myTask.Wait();
myList.AddRange(myTask.Result.ToArray());
```

Task<TResult>는 TResult 형식의 결과를 반환합니다.

myList의 요소는 0, 1, 2, 3, 4, 5가 됩니다.

- 데모 예제 - TaskResult

19.2.3 손쉬운 병렬 처리를 가능케 하는 Parallel 클래스

- Task<TResult>로 직접 구현했던 병렬 처리를 손쉽게 구현
- 사용 사례

```
void SomeMethod( int i )
{
    Console.WriteLine( i ) ;
}

// …

Parallel.For( 0, 100, SomeMethod );
```

- 데모 예제 - ParallelLoop

19.3 async 한정자와 await 연산자로 만드는 비동기 코드

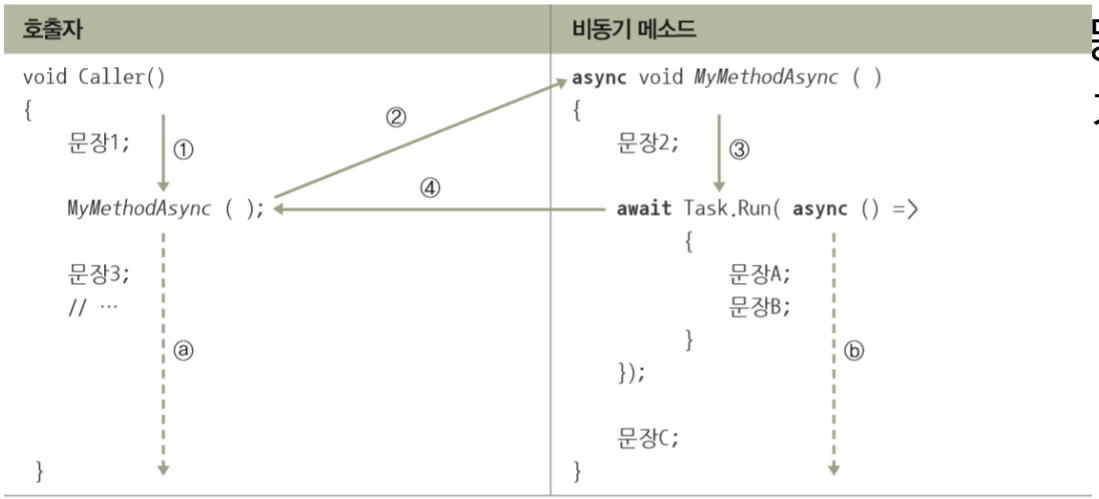
❖ async 한정자.

- 메소드, 이벤트 처리기, 태스크, 람다식 등 수식
- 메소드나 태스크를 수식하기만 하면 비동기 코드 생성

❖ 반환 형식 제약

- Task나 Task<Tresult> - 작업이 완료될 때까지 기다리는 메소드
- Void - 실행하고 잊어버릴(Shoot and Forget) 작업을 담고 있는 메소드

❖ Await 연산자



동기로 실행
기술된 곳에서 호출자에게

❖ 데모 예제 - Async

19.3.1 .NET 프레임워크가 제공하는 비동기 API 맛보기

❖ Microsoft의 비동기 사랑

- C# 언어의 비동기 프로그래밍 패러다임 지원
- .NET 프레임워크에서 비동기 버전 API 추가

❖ System.IO.Stream의 읽기/쓰기 메소드의 동기 및 비동기 버전

동기 버전

비동기 버전

```
async Task<long> CopyAsync(string FromPath, string ToPath)
{
    using (
        var fromStream = new FileStream(FromPath, FileMode.Open))
    {
        long totalCopied = 0;
        using (
            var toStream = new FileStream(ToPath, FileMode.Create))
        {
            byte[] buffer = new byte[1024];
            int nRead = 0;
            while ((nRead  =
            await fromStream.ReadAsync(buffer, 0, buffer.Length)) != 0)
            {
                await toStream.WriteAsync(buffer, 0, nRead);
                totalCopied += nRead;
            }
        }
```

async로 한정한 코드를 호출하는 코드도 역시 async로 한정되어 있어야 합니다. 반환 형식은 Task 또는 void 형이어야 하고요.

ReadAsync()와 WriteAsync() 메소드는 .NET 프레임워크에 async로 한정되어 있습니다. 이들을 호출하려면 await 연산자가 필요합니다.

❖ 데모 예제 - AsyncFileIO

Thank You !

이것이 C#이다