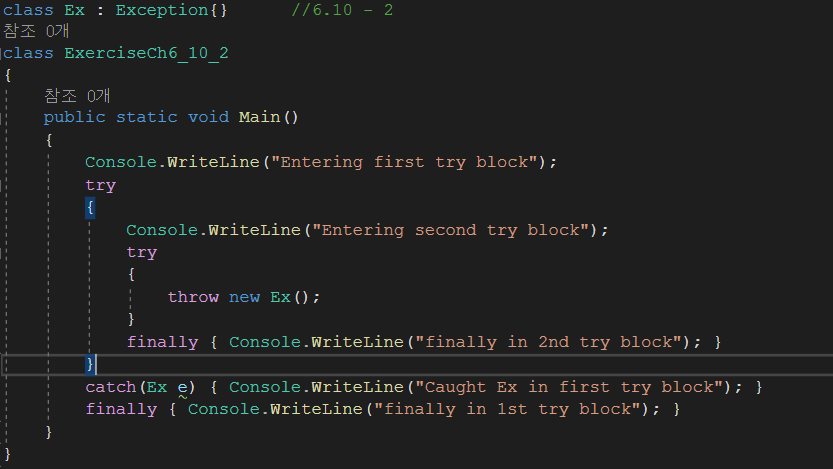


닷넷 프로그래밍 6장 레포트

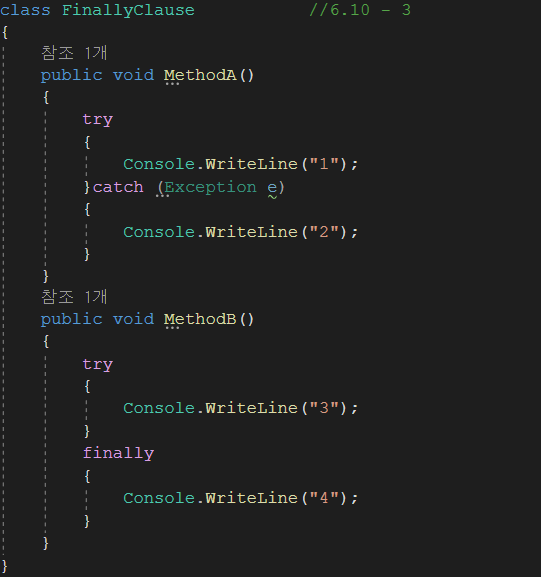
**<6.10>**

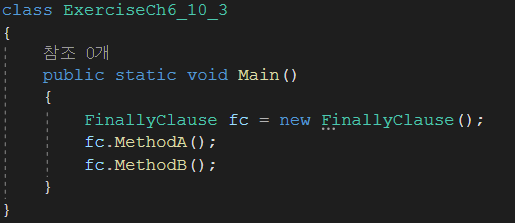
다음 프로그램의 실행 결과를 쓰시오.

(2)

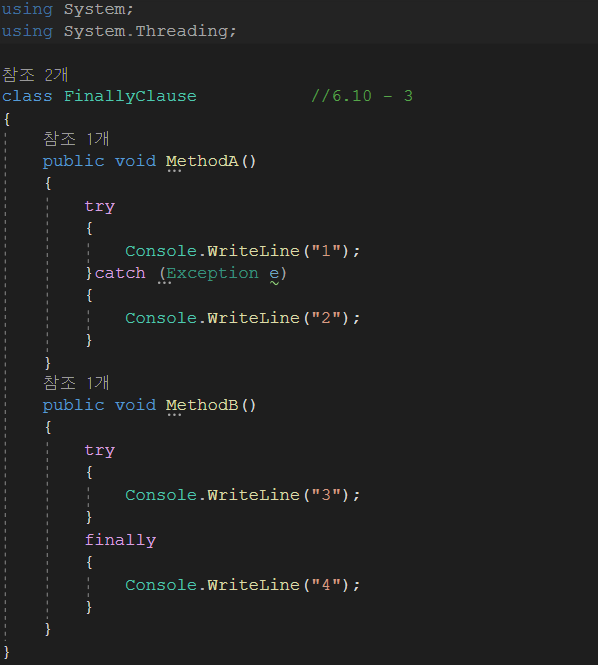


(3)



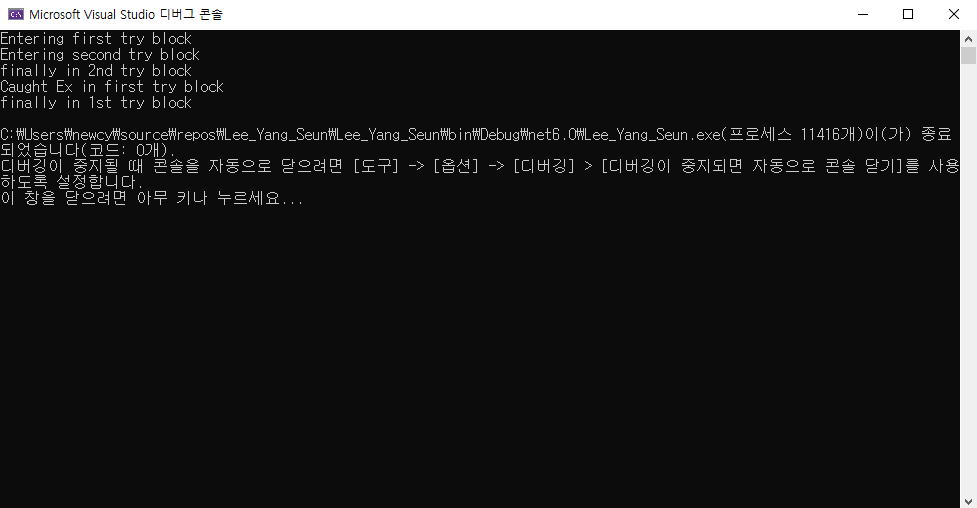


(4)

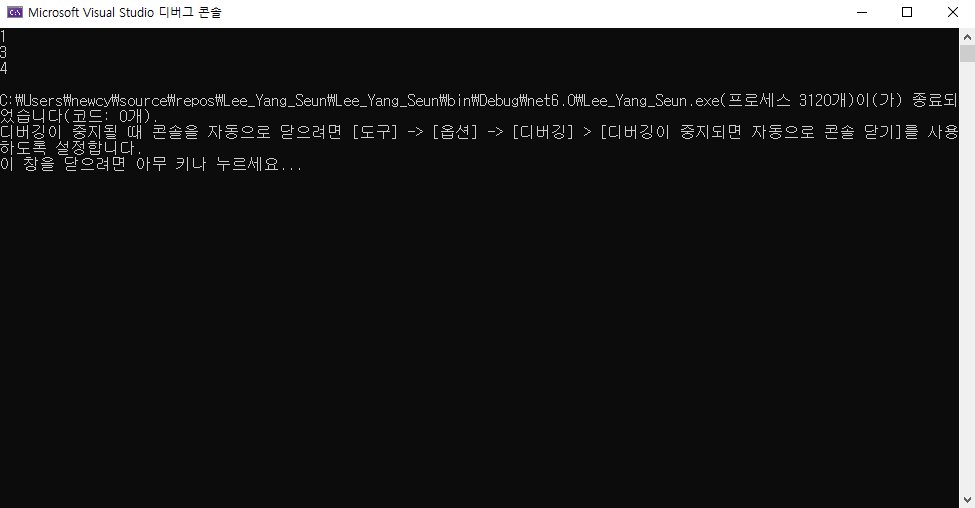


**<결과>**

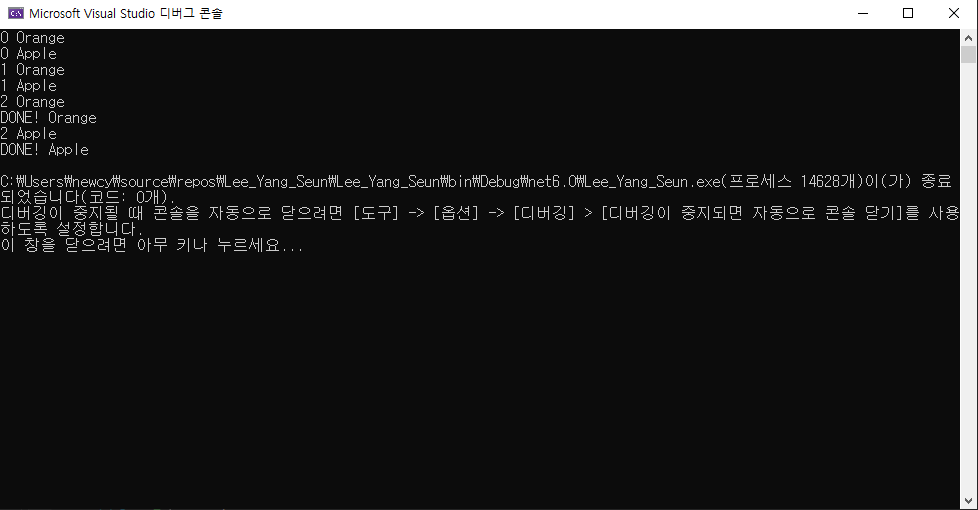
(2)



(3)



(4)



**<설명>**

* (2)번 문제는 try-catch-finally 구문에 대한 내용을 담고 있다. try 블록 안에서는 예외가 검사되고, 만약 예외가 발생한다면 catch 블록에서 예외가 처리된다. finally 블록은 선택적으로 나올 수 있는 부분으로 존재한다면 반드시 실행되는 부분이다.

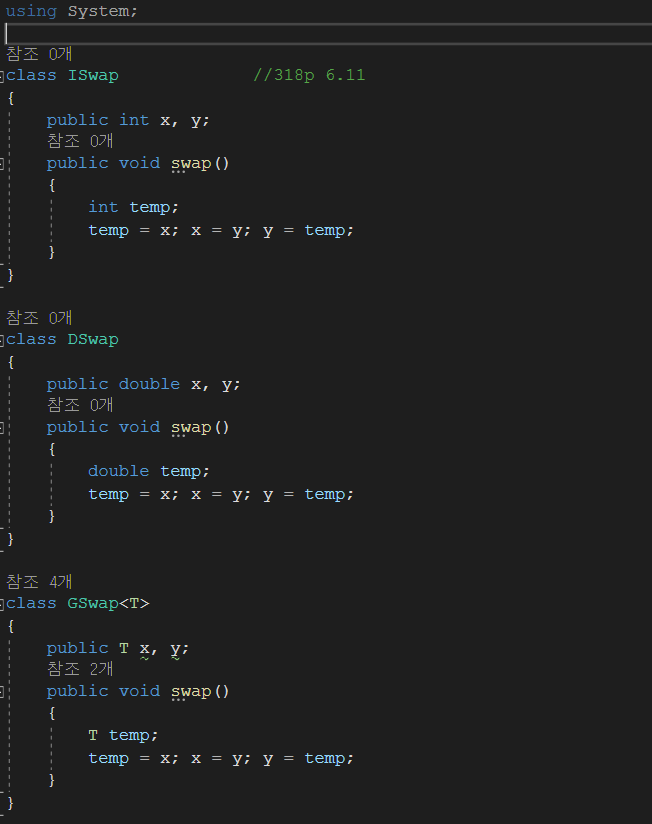
실행 과정을 살펴보면 Main() 안에서 먼저 "Entering first try block" 부분이 출력되고, try 블록이 나타난다. 첫번째 try 블록에 있는 "Entering second try block" 문장이 출력되는데 두번째 try 블록이 나타난다. 이 블록에서는 예외를 일으키는 문장인 throw new Ex(); 이 나타나는데 Ex 예외 객체는 Exception 클래스의 파생클래스이다. 다음으로 나타나는 finally 블록은 반드시 실행되는 부분이므로, "finally in 2nd try block" 문장이 출력된다. 첫번째 try 블록과 두번째 try 블록 모두 끝났고 예외를 처리하는 catch 블록이 존재한다. catch 블록에서는 Ex 예외 객체에 대해 처리를 담당하고 있고 해당 객체에 대한 예외가 발생했을 시 "Caught Ex in first try block" 문장을 출력하는 역할을 한다. 마지막으로 finally 블록안에 "finally in 1st try block" 문장을 출력하고 프로그램은 종료된다.

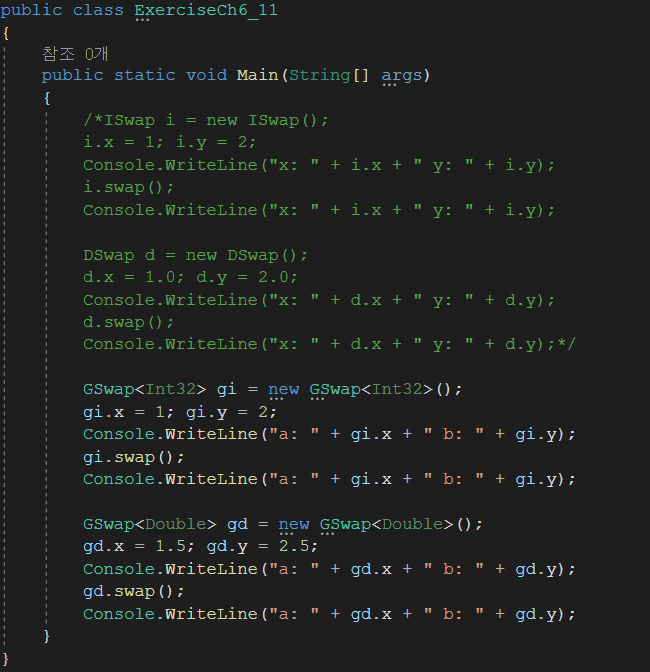
* (3)번 문제도 try-catch-finally 구문에 대한 문제이지만 위의 문제와 다른 점이 있다. 위의 문제에서는 예외 객체를 만들어 예외를 발생시켰지만, 이 문제에서는 예외 객체가 존재하지 않은 상태에서 try-catch-finally 구문이 어떻게 작동하는지를 보여준다. FinallyClause 클래스에는 MethodA() 메소드와 MethodB() 메소드가 존재한다. MethodA() 메소드에는 try 블록과 Exception 에 해당하는 catch 구문이 존재한다. MethodB() 메소드에는 try 블록과 finally 블록이 존재한다. Main() 에서 먼저 FinallyClause 객체인 fc를 생성하였고 fc의 MethodA()와 MethodB() 를 실행시킨다. MethodA()에서는 try 블록의 “1” 출력이 실행되지만 예외 발생이 일어나지 않기 때문에 catch 블록은 실행되지 않는다. 다음으로 MethodB() 의 try 블록의 “3” 출력이 실행되고, finally 블록의 “4”가 출력된다. 이후 프로그램이 종료된다.
* (4)번 문제는 멀티스레드 시스템에 대한 문제이다. 멀티스레드 시스템이란 스레드가 하나의 프로그램 내에 여러 개 존재할 수 있는 시스템이다. 해당 문제에서는 두 개의 스레드를 실행시켜 동시에 실행되는 모습을 보여준다.

먼저 스레드를 생성하기 위해 using.System.Threading 네임스페이스를 작성한다. SimpleThread 클래스에서는 Main() 에서 실행시킬 스레드 몸체에 해당하는 메소드가 담겨 있다. SimpleMethod() 메소드는 난수를 생성하는 클래스인 Random의 randomNumber 객체를 생성한다. 이후, for 반복문을 통하여 3번, i와 해당 스레드의 Name을 출력한다. 그 후, Random 클래스의 Next()메소드로 1부터 5사이의 임의의 수 \* 1000 의 시간동안 중단 상태가 된다. 반복문이 3번 반복되었다면 "DONE! " 문장과 함께 해당 스레드의 Name을 출력한다. Main()에서 먼저 SimpleThread 클래스의 객체인 obj를 생성하고 ThreadStart 델리게이트 ts에 obj객체의 SimpleMethod() 메소드를 연결시킨다. 그 후, worker1 스레드와 worker2 스레드에 ts를 이용하여 스레드를 생성한다. worker1과 worker2의 이름을 각각 “Apple”, “Orange”로 설정하고 Start() 메소드를 실행시키면 두 스레드가 동시에 실행되어 출력되는 모습을 볼 수 있다.

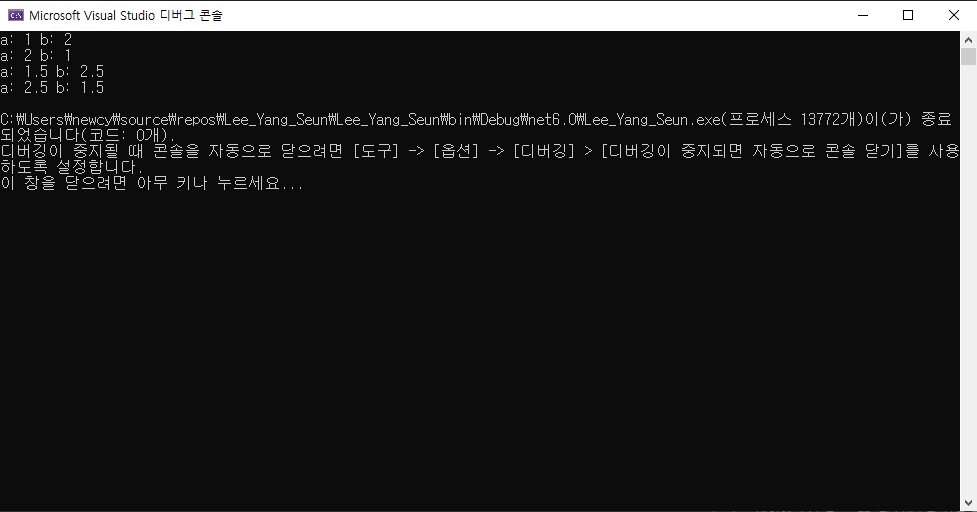
**<6.11>**

다음 프로그램을 제네릭 클래스를 사용하여 다시 작성하시오.





**<결과>**



**<설명>**

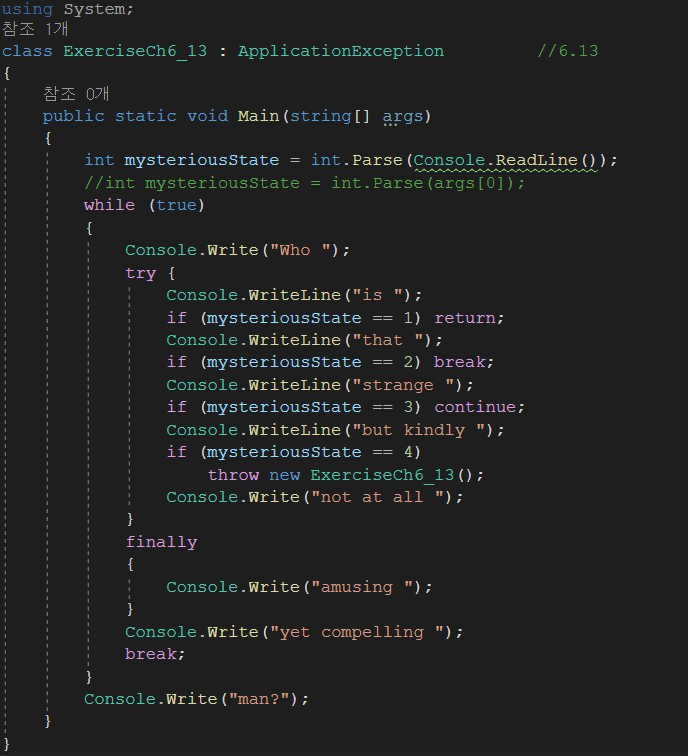
* 문제에는 int 자료형의 변수들을 바꿔주는 ISwap 클래스와 double 자료형의 변수들을 바꿔주는 DSwap 클래스가 작성되어 있다. 제네릭 클래스란 형 매개변수를 가지는 클래스를 말한다. 형 매개변수란 객체 생성 시에 전달받으며 클래스 내에 필드나 메소드를 선언할 때 자료형으로 사용되는 변수를 말한다.

형 매개변수 “T” 를 가지는 제네릭 클래스 GSwap를 만들었다. T 자료형의 변수 x와 y를 선언하고 바꿔줄 수 있는 메소드 swap를 가지고 있다.

문제 속 Main()에서는 ISwap 객체 i 와 DSwap 객체 d 를 생성하여 각 객체의 변수들이 바뀌는 모습을 보여준다. 제네릭 클래스 GSwap 객체를 통해 Int32와 Double 자료형을 전달하고 해당 자료형에 알맞게 swap() 메소드가 작동하는 모습을 출력 결과를 통해 볼 수 있다.

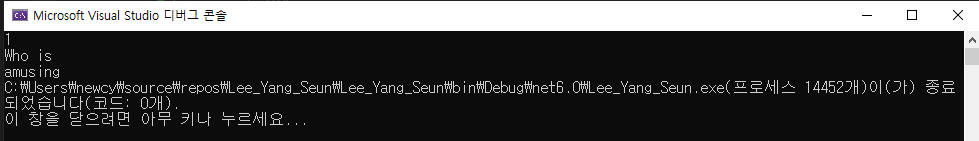
**<6.13>**

다음 프로그램은 명령어 라인 매개변수 값에 따라 다른 형태의 결과를 갖는다. 명령어 라인 매개변수로 1부터 5까지 각각의 값에 따른 결과를 예측하여 보시오.

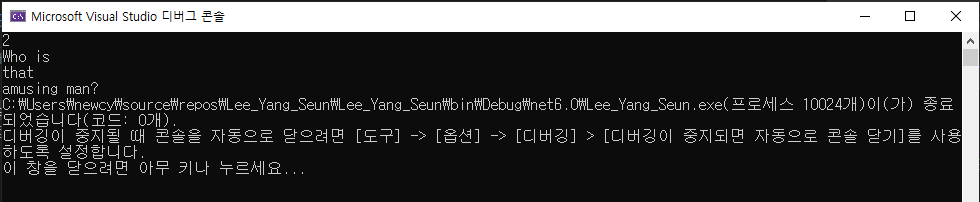


**<결과>**

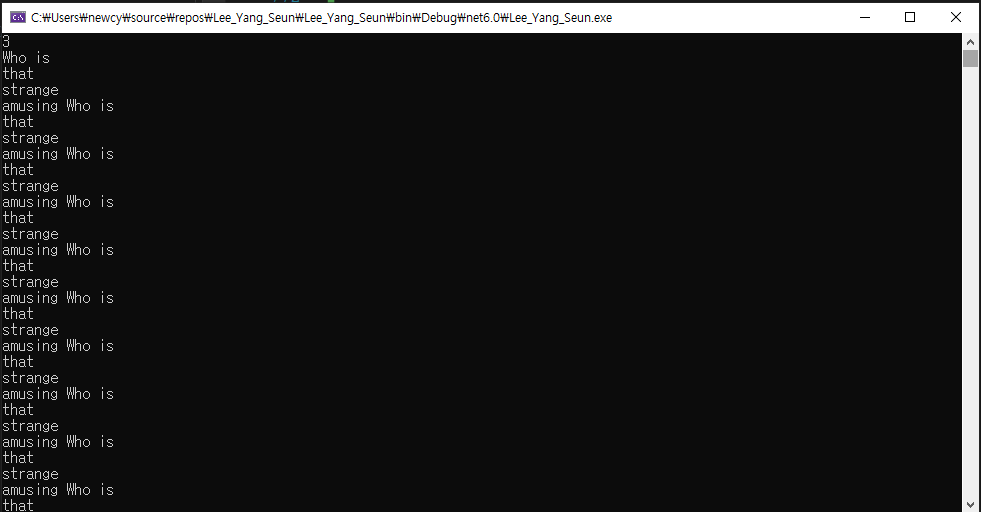
1의 경우)



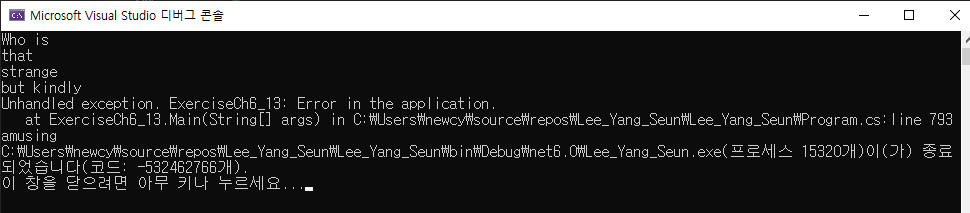
2의 경우)



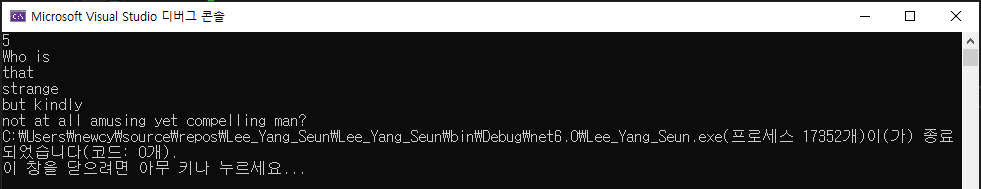
3의 경우)



4의 경우)



5의 경우)



**<설명>**

* 이 문제는 명령어 라인 매개변수의 값에 따라 출력이 다르게 나오는 것을 나타내고 있다. ExerciseCh6\_13 클래스는 ApplicationException 클래스를 상속받아 예외 클래스로 정의되었고, Main()에는 명령어 라인 매개변수인 string[] args가 있다. 이 매개변수는 정수형 변수 mysteriousState에 저장된다.

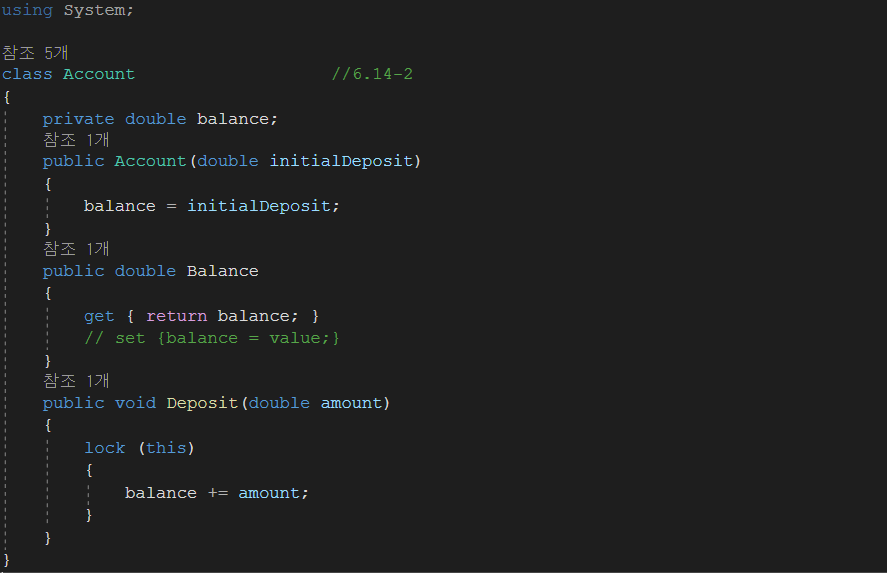
while 반복문에는 먼저 "Who " 를 출력하는 문장이 있고, try 블록에는 "is "를 출력하는 문장, 변수 mysteriousState가 1일 경우에는 return 하라는 문장, "that " 을 출력하는 문장, mysteriousState가 2일 경우 break하여 반복문을 중지하는 문장, "strange "을 출력하는 문장, mysteriousState가 3일 경우 continue하여 다음 반복을 실행하는 문장, "but kindly " 을 출력하는 문장, mysteriousState가 4일 경우 ExerciseCh6\_13 예외를 발생시키는 문장, "not at all " 을 출력하는 문장으로 구성되어 있다.

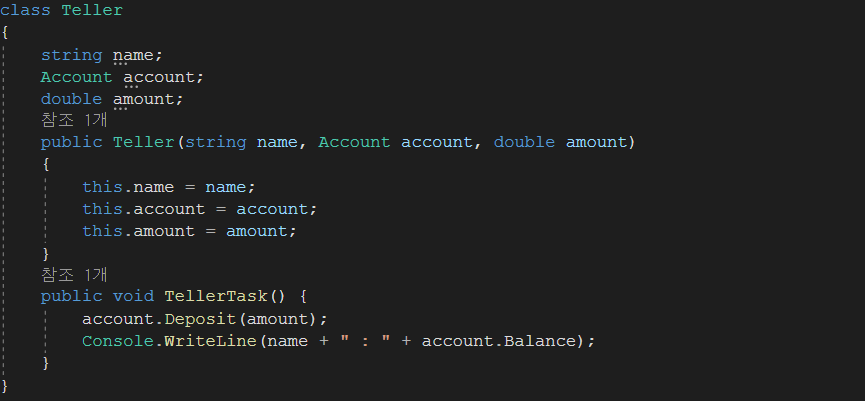
finally 블록은 "amusing " 을 출력하는 문장으로 구성되어 있고, "yet compelling " 을 출력하는 문장과 break를 통해 반복문을 종료시키는 문장을 끝으로 while 반복문이 구성되어 있다. 그 후, "man?" 을 출력하는 문장을 끝으로 Main()은 종료된다.

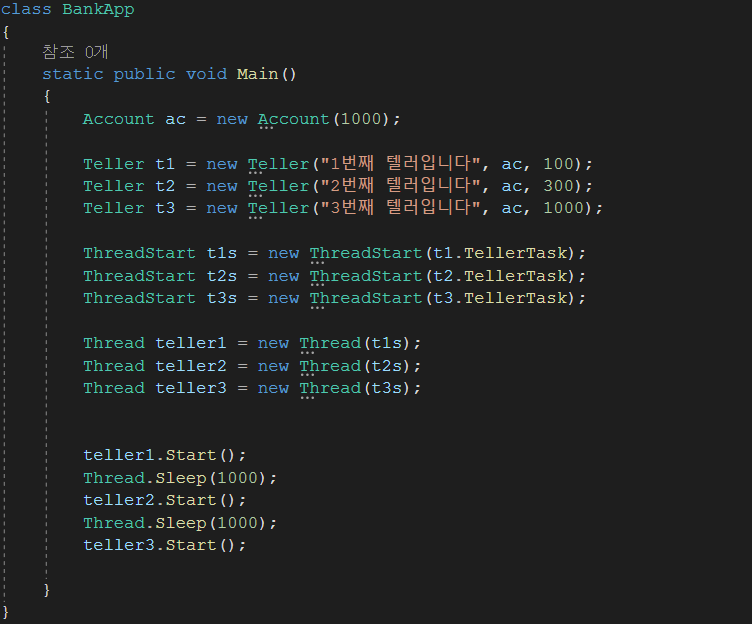
* 입력에 따라 달라지는 출력을 확인하기 위해 int mysteriousState = int.Parse(Console.ReadLine()); 문자을 추가하였다. 입력이 1인 경우 while문의 출력문장 "Who " 와 try 블록의 "is " 가 출력된 후 mysteriousState가 1이므로 retrun으로 인해 프로그램이 종료되어야 하지만, finally 블록이 실행되어야 하므로 "amusing " 가 출력되고 프로그램이 종료된다.
* 입력이 2인 경우 while문의 출력문장 "Who " 와 try 블록의 "is ", "that " 가 출력되고 mysteriousState가 2이므로 break로 인해 while 문이 종료된다. 그 후, "man?" 을 출력하고 프로그램은 종료된다.
* 입력이 3인 경우 while문의 출력문장 "Who " 와 try 블록의 "is ", "that ", “strange “ 가 출력되지만 mysteriousState가 3이므로 continue를 통해 finally 블록에 "amusing " 가 출력되고 다시 반복문을 실행한다. 이러한 과정을 반복하여 무한 루프가 발생한다.
* 입력이 4인 경우 while문의 출력문장 "Who " 와 try 블록의 "is ", "that ", “strange “, "but kindly " 가 출력되고 mysteriousState가 4이므로 예외를 발생시키는데 이에 해당하는 예외를 처리하는 catch 블록이 없기 때문에 디폴트 예외처리기가 발생하고 finally 블록에 "amusing " 가 출력 후 프로그램은 종료된다.
* 입력이 5인 경우 어떤 조건문도 실행되지 않기 때문에 while문의 출력문장 "Who ", “yet compelling“ 와 try 블록의 "is ", "that ", “strange “, "but kindly ", “not at all “ 가 출력되고 finally 블록에 "amusing " 가 출력 후 "man "의 출력을 끝으로 프로그램이 종료된다.

**<6.14>**

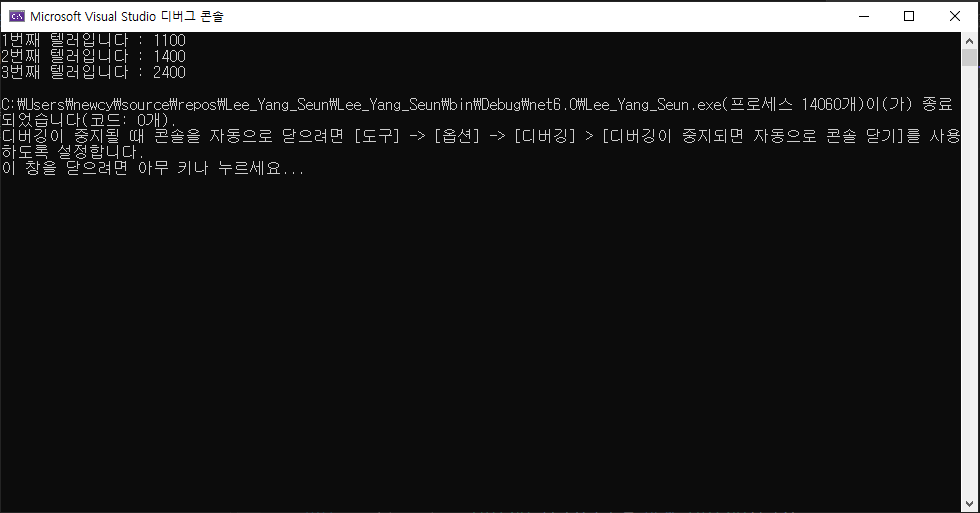
(2) 다음은 lock 문을 사용하여 은행 예금 문제를 해결한 클래스이다. Main() 메소드가 있는 테스트 클래스를 작성하여 프로그램을 완성하시오.







**<결과>**



**<설명>**

* 위 문제는 lock 문을 사용하여 동시에 예금이 되는 문제를 방지하기 위한 코드 작성 문제이다. 스레드를 사용하기 위해 using System.Threading 네임스페이스를 추가하였다.
* Account 클래스는 잔액을 나타내는 변수 balance, 초기 예금을 매개변수로 받는 생성자 Account, 현재 잔액을 반환하는 프로퍼티, 외부에서 추가 예금을 받아 계좌 잔액에 추가하는 메소드 Deposit() 등으로 이루어져 있다. Deposit() 메소드는 lock 문은으로 이루어져 있어 여러 객체가 동시에 실행할 때 스레드 중첩을 방지할 수 있다.
* Teller 클래스는 은행원 이름 변수 name, 처리할 계정 account 객체, 예금액 변수 amount, 위의 변수와 객체를 매개변수에서 받는 Teller 생성자, Account 클래스의 Deposit() 메소드에 생성자 호출시 받은 변수 amount를 보내주고, Teller 객체의 이름과 계좌 잔액을 출력하는 TellerTask() 메소드로 이루어져 있다.
* Main() 에서는 Account 클래스의 초기예금을 1000으로 설정하여 생성한 객체 ac, Teller 클래스의 객체 t1, t2, t3, 스레드를 실행시키는 Threadstart 객체 t1s, t2s, t3s에 t1, t2, t3의 TellerTask() 메소드를 연결시키고, Thread 객체 teller1, teller2, teller3 를 생성시켜 1초간 지연시키며 출력하도록 작성하였다. 1000의 초기 에금에서 시작하여 100, 300, 1000의 추가 예금액을 더한 모습이 출력된다.