

닷넷프로그래밍

과제3 – 6장

2019305059

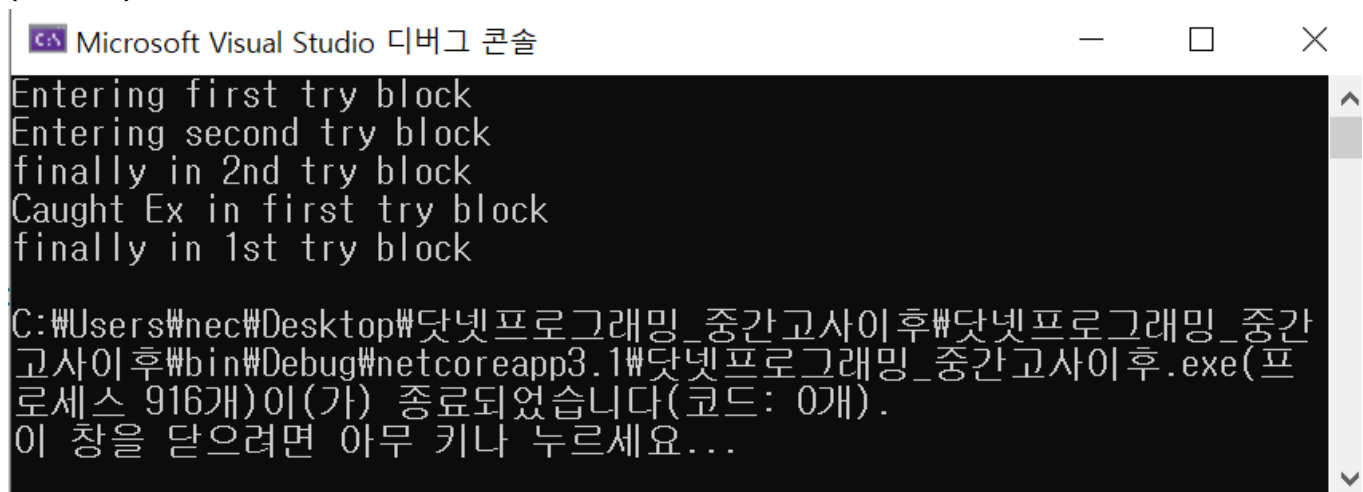
이현수

6.10 - (2)

(코드캡처)

```
1    using System;
2    class Ex : Exception { }
3    class ExerciseCh6_10_2
4    {
5        public static void Main()
6        {
7            Console.WriteLine("Entering first try block");
8            try
9            {
10               Console.WriteLine("Entering second try block");
11               try
12               {
13                   throw new Ex();
14               }
15               finally
16               {
17                   Console.WriteLine("finally in 2nd try block");
18               }
19           }
20           catch(Ex e)
21           {
22               Console.WriteLine("Caught Ex in first try block");
23           }
24           finally
25           {
26               Console.WriteLine("finally in 1st try block");
27           }
28       }
29   }
```

(실행캡처)



Microsoft Visual Studio 디버그 콘솔

Entering first try block
Entering second try block
finally in 2nd try block
Caught Ex in first try block
finally in 1st try block

C:\Users\wne\Desktop\닷넷프로그래밍_중간고사이후\닷넷프로그래밍_중간고사이후\bin\Debug\netcoreapp3.1\닷넷프로그래밍_중간고사이후.exe(프로세스 916개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

(코드설명)

클래스 EX를 만들면서 Exception을 상속한다. 그리고 클래스 ExerciseCh6_10_2 내 Main메소드에서 코드를 시작한다.

(코드 7줄) 에 Console.WriteLine을 통해 Entering first try block을 출력한다. 그리고 try문을 만나게 된다. try문 안에있는 Console.WriteLine을 통해 Entering second try block을 출력한다. 그리고 바로 다음줄에 try블록이 나온다. try블록안에 바로 throw new Ex();를 통해 명시적으로 예외를 발생시킨다. 그 후

(코드15-18) finally블록은 반드시 실행된다. 그래서 finally블록안에 있는 코드를 실행하면 finally in 2nd try block이 출력된다.

그 후 catch문을 만나는데 Ex e가 있다. 아까 throw를 통해 명시적으로 예외가 발생해서 catch블록 내에 있는 코드가 실행되 Caught Ex in first try block이 출력된다. 그다음 finally블록을 만난다. finally블록은 무조건 실행되므로 finally블록안에 있는 코드를 통해 finally in 1st try block이 출력된다.

6.10 - (3)

(코드캡처)

```
1  using System;
2  class FinallyClause
3  {
4      public void MethodA()
5      {
6          try
7          {
8              Console.WriteLine("1");
9          }
10         catch(Exception e)
11         {
12             Console.WriteLine("2");
13         }
14     }
15     public void MethodB()
16     {
17         try
18         {
19             Console.WriteLine("3");
20         }
21         finally
22         {
23             Console.WriteLine("4");
24         }
25     }
26 }
27 class ExerciseCh6_10_3
28 {
29     public static void Main()
30     {
31         FinallyClause fc = new FinallyClause();
32         fc.MethodA();
33         fc.MethodB();
34     }
35 }
```

(실행캡처)

(코드설명)

클래스 FinallyClause를 선언한다. 그 클래스 안에 public으로 MethodA, MethodB 두개의 메소드를 만든다. 두 개 모두 반환형은 void이다.

(코드4-14) public으로 반환형은 void인 MethodA 메소드를 만든다. try블록과 catch블록이 있다. try블록안에는 Console.WriteLine("1");을 통해 1을 출력하고 다음줄로 넘어간다. 그리고 catch블록은 Exception e가 되어있고 그 안에는 2를 출력하게 되었다.

(코드15-25) public으로 반환형은 void인 MethodB 메소드를 만든다. try블록과 finally블록으로 이루어져 있다. try블록안에는 Console.WriteLine("3");을 통해 3을 출력하고 다음줄로 넘어간다. 그리고 예외 유무와 상관없이 무조건 실행되는 finally블록안에는 Console.WriteLine("4");를 통해 4를 출력하고 다음줄로 넘어간다.

(코드29~) 클래스 ExerciseCh6_10_3내에 있는 Main메소드에서 클래스 FinallyClause 객체 fc를 생성한다. 그리고 그 객체 fc로 fc.MethodA(); fc.MethodB();를 통해 메소드 두개를 각각 호출한다.

fc.MethodA();를 통해 MethodA메소드를 호출하면 try블록안에 있는 코드를 통해서 1이 먼저 출력된다. 그리고 예외가 발생할 상황이 없기때문에 catch블록은 건너된다. 그러면 MethodA메소드는 종료된다. 다음은 fc.MethodB();를 통해 MethodB 메소드를 호출하면 try블록을 통해 3이 출력된다. 그다음은 바로 finally블록을 만나는데 finally블록은 예외 유무와 상관없이 무조건 실행되는 특징을 가진다. 그래서 finally구문에 있는 코드를 통해서 4가 출력된다.

그래서 프로그램을 실행하면

1

3

4

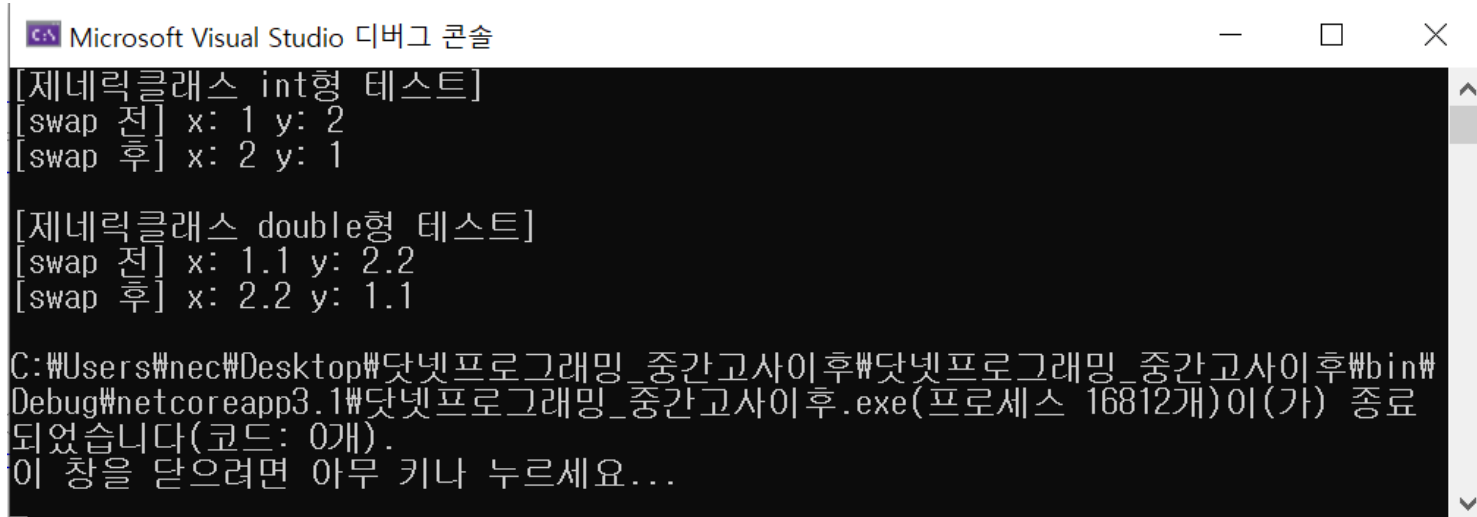
가 출력된다.

6.11

(코드캡처)

```
1    using System;
2    class GenericSwap<T>
3    {
4        public T x, y;
5        public void swap()
6        {
7            T temp;
8            temp = x; x = y; y = temp;
9        }
10   }
11   }
12   public class ExerciseCh6_11
13   {
14       public static void Main(String[] args)
15       {
16           Console.WriteLine("[제네릭클래스 int형 테스트]");
17           GenericSwap<int> i = new GenericSwap<int>();
18           i.x = 1; i.y = 2;
19           Console.Write("[swap 전] ");
20           Console.WriteLine("x: " + i.x + " y: " + i.y);
21           i.swap();
22           Console.Write("[swap 후] ");
23           Console.WriteLine("x: " + i.x + " y: " + i.y);
24
25           Console.WriteLine("\n[제네릭클래스 double형 테스트]");
26           GenericSwap<double> d = new GenericSwap<double>();
27           d.x = 1.1; d.y = 2.2;
28           Console.Write("[swap 전] ");
29           Console.WriteLine("x: " + d.x + " y: " + d.y);
30           d.swap();
31           Console.Write("[swap 후] ");
32           Console.WriteLine("x: " + d.x + " y: " + d.y);
33       }
34   }
```

(실행캡처)



```
Microsoft Visual Studio 디버그 콘솔
[제네릭클래스 int형 테스트]
[swap 전] x: 1 y: 2
[swap 후] x: 2 y: 1

[제네릭클래스 double형 테스트]
[swap 전] x: 1.1 y: 2.2
[swap 후] x: 2.2 y: 1.1

C:\Users\nec\Desktop\닷넷프로그래밍_중간고사이후\닷넷프로그래밍_중간고사이후\bin\
Debug\netcoreapp3.1\닷넷프로그래밍_중간고사이후.exe(프로세스 16812개)이(가) 종료
되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

(코드설명)

제네릭 클래스를 만들어준다. Class GenericSwap<T>{ } 형식으로 만들어 준다. 제네릭 클래스 안에는 public으로 T형의 변수 x, y를 선언한다.

그 후 public으로 반환형이 void인 swap메소드를 만들어준다. 변수 x,y를 교환하는 기능을 한다. 우선 T형의 temp변수를 선언하고, temp에 x를 값을 저장하고, x에 y를 저장하고, y에 temp 값을 저장하면 x, y의 값이 바뀌어진다.

public으로 된 클래스 ExerciseCh6_11 내에 있는 Main메소드에서

(코드17) GenericSwap<int> i = new GenericSwap<int>();를 통해 제네릭클래스 GenericSwap를 자료형 int로 만든 객체 i를 생성한다. 그리고 i객체의 x, y에 각각 1, 2를 저장한다. 그리고 x, y 값을 출력한다. 그 후 i.swap();을 통해 객체 i에 있는 필드 x, y의 값을 교환해준다. 그 후 다시 출력해준다.

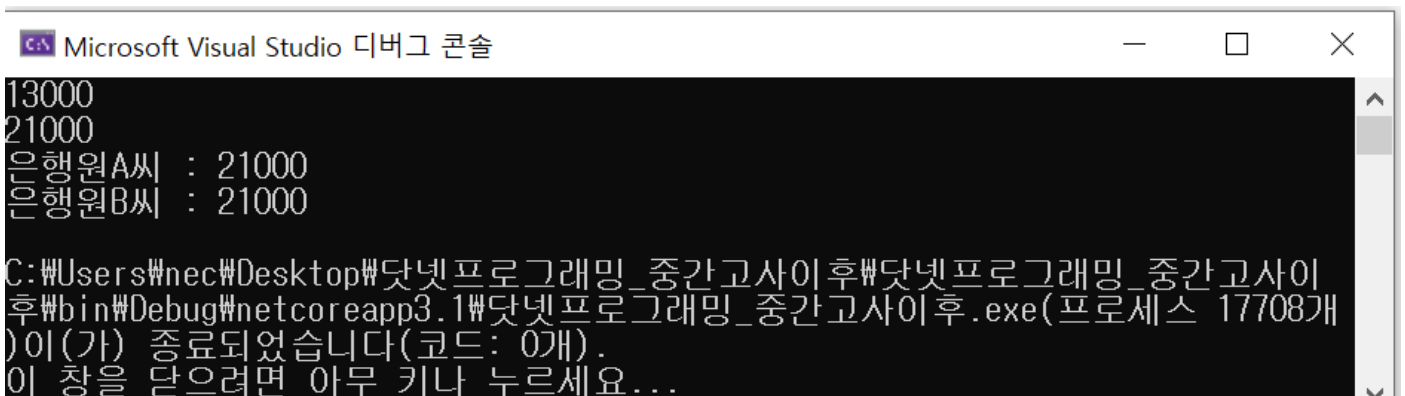
(코드25) GenericSwap<double> d = new GenericSwap<double>();를 통해 제네릭클래스 GenericSwap를 자료형 double로 만든 객체 d를 생성한다. 그리고 d객체의 x, y를 각각 1.1, 2.2로 초기화한다. 그 후 x, y값을 출력한다. 그 후 d.swap();을 통해 객체 d의 필드 x, y 값을 교환한다. 결과를 확인하기 위해서 출력한다.

6.14 - (2)

(코드캡처)

```
1  using System;
2  using System.Threading;
3  class Account
4  {
5      private double balance;
6      public Account(double initialDeposit)
7      {
8          balance = initialDeposit;
9      }
10     public double Balance
11     {
12         get { return balance; }
13     }
14     public void Deposit(double amount)
15     {
16         lock (this)
17         {
18             balance += amount;
19             Console.WriteLine(balance);
20         }
21     }
22 }
23 class Teller
24 {
25     string name;
26     Account account;
27     double amount;
28     public Teller(string name, Account account, double amount)
29     {
30         this.name = name;
31         this.account = account;
32         this.amount = amount;
33     }
34     public void TellerTask()
35     {
36         account.Deposit(amount);
37         Console.WriteLine(name + " : " + account.Balance);
38     }
39 }
40 class ExerciseCh6_14
41 {
42     public static void Main()
43     {
44         Account a = new Account(10000);
45         Teller t1 = new Teller("은행원A씨", a, 3000);
46         Teller t2 = new Teller("은행원B씨", a, 8000);
47
48         ThreadStart ts1 = new ThreadStart(t1.TellerTask);
49         ThreadStart ts2 = new ThreadStart(t2.TellerTask);
50
51         Thread th1 = new Thread(ts1);
52         Thread th2 = new Thread(ts2);
53
54         th1.Start();
55         th2.Start();
56     }
57 }
```


(실행캡처)



```
Microsoft Visual Studio 디버그 콘솔

13000
21000
은행원A씨 : 21000
은행원B씨 : 21000

C:\Users\wneec\Desktop\닷넷프로그래밍_중간고사이후\닷넷프로그래밍_중간고사이후\bin\Debug\netcoreapp3.1\닷넷프로그래밍_중간고사이후.exe(프로세스 17708개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

(코드설명)

클래스 Account를 선언한다. Account 클래스 내에는 private인 double형 balance 변수를 선언한다. balance변수는 계좌 잔액을 의미한다.

(코드6-9) 클래스 Account 의 생성자함수를 만들어준다. 매개변수 initialDeposit(int형)을 받아 잔액을 의미하는 변수 balance에 저장한다.

(코드10-13) 프로퍼티를 만들어준다. Balance에 대해서 겐-접근자만 만들어준다.

(코드14-21)public으로 반환형이 없는 Deposit메소드를 만든다. 매개변수로 amount를 입력받아 amount만큼 현재 balance 값에 amount를 더하는 기능을 한다. 이때 동일한 계좌객체에 동시에 두 사람이 다른 직원 두명을 통해 잔액을 조회하고 예금액을 더하는 작업을 할 때 한 직원의 amount만큼 잔액에 더하는 작업이 무시되는 것을 막기위해서 lock문을 이용한다. lock(this){ }블록안에 balance+=amount;를 통해 잔액에 amount만큼 추가하고 정상적으로 동작하는지 확인하기위해 Console.WriteLine(balance);문장을 추가했다.

클래스 Teller을 선언한다. 은행원 이름을 나타내는 string형 변수 name, 은행원이 다룰 계좌클래스를 나타내는 Account클래스 형 account변수, 은행원이 처리할 금액을 나타내는 double형 amount변수를 선언한다.

(코드28-33) 클래스 Teller의 생성자 함수를 만든다. 매개변수 3개를 받아서 각각 this지정어를 사용해 클래스 Teller에 있는 name, account, amount를 초기화한다.

(코드.34-38) public으로 반환형이 없는 TellerTask함수를 만든다. 이 함수는 Account클래스 객체 account 계정에 있는 Deposit 메소드를 호출해 amount만큼 일을 처리 후 Console.WriteLine을 통해 은행원 이름과 account계정의 잔액을 출력한다.

클래스 ExerciseCh6_14내에 있는 Main메소드에서 Account객체 a를 만들어주고 생성자함수를 통해 초기잔액은 10000으로 초기화한다.

그리고 클래스 Teller객체 t1, t2를 만든다. 각각 생성자함수를 이용해 t1은 name은 "은행원A씨", 처리할 Account클래스 객체는 a, 처리할 금액 amount는 3000으로 설정한다. t2는 name은 "은행원B씨", 처리할 Account클래스 객체는 a, 처리할 금액 amount는 8000으로 설정한다.

그리고 Teller 클래스의 객체 t1, t2의 메소드 TellerTask메소드를 각각 ThreadStart 델리게이트 ts1, ts2에 각각 연결한다. 그리고 생성된 델리게이트를 이용해 스레드 객체를 th1, th2를 생성한다. 이때 델리게이트 객체는 Thread클래스 생성자의 매개변수가 된다. 그리고 th1.Start(); th2.Start();를 통해 스레드를 활성화한다.

실행결과 은행원A씨가 먼저 10000원에 3000원을 더해 13000원이 되었다. 실행결과 창에 lock블록 안에 있는 Console.WriteLine(balance);를 통해 13000이 정상적으로 뜨고 뒤이어 은행원 B씨가 13000원에 8000원을 더해 21000이 정상적으로 출력된다. 그 후 TellerTask메소드 내 Console.WriteLine(name + " : " + account.Balance);를 통해 상담원A씨 : 21000, 상담원B씨 : 21000이 정상적으로 출력된다. lock문은 Account클래스 내 Deposit메소드에 만 걸려있고 워낙 간단한 동작이라 순식간에 일어나서 두개 모두 21000으로 출력된다.