C#

3주차 1차시

4장 클래스

3주차 학습 내용

* 1차시
  * 2주차 3교시 과제 피드백
  * **4장 클래스**
* 2차시
  * **7장 폼**
    * 윈폼 어플리케이션의 구조
    * 폼 클래스
      * 폼 클래스의 계층도
      * 폼 클래스의 프로퍼티, 메소드, 이벤트
    * 컨트롤 클래스
      * 컨트롤 클래스의 계층도
      * 컨트롤 클래스의 프로퍼티, 메소드, 이벤트
* 3차시
  * **8장 컨트롤(3)**
    * 리스트
      - 리스트 상자
      - 콤보 상자
      - 체크 리스트 상자

4장 클래스

C# 프로그래밍

4. 클래스

객체 지향 프로그램의 개념

- 객체지향 방법이란?
  - 실세계의 객체를 소프트웨어적으로 표현하기 위한 방법
  - **실세계의 객체가 갖는 상태와 행동을 소프트웨어 객체의 변수(variable: 객체 상태)와 메서드(method 또는 function: 객체 행동)로 모델링**

- 객체지향 프로그램의 특징
  - 추상화(abstraction)
    - 사물들이 가지는 특성들을 잘 정리하여 필드와 메서드로 표현하는 과정
  - 캡슐화(Encapsulation)
    - 상태 정보를 저장하고 있는 변수와 상태를 변경하거나 서비스를 수행하는 메서드를 하나의 소프트웨어 묶음으로 묶는 것
    - **높은 모듈성과 정보은닉 등 두 가지 이득을 제공**
  - 다형성(Polymorphism)
    - 여러 개의 클래스가 같은 메시지에 대해 각자의 방법으로 작용할 수 있는 능력
    - 같은 이름을 갖는 여러 가지 형태가 존재
  - 상속(Inheritance)
    - 클래스를 이용하여 다른 클래스를 생성 또는 정의 가능
    - 하위클래스는 상위클래스가 갖고 있는 모든 특성들을 상속하여 사용 가능
    - **상위 클래스를 여러 하위 클래스들이 재사용, 소프트웨어 개발에 드는 비용 감소**

객체 지향 프로그램의 개념

- 객체지향 프로그램의 특징
  - 메시지(Message)
    - 객체와 다른 객체 사이에 통신을 할 수 있도록 도와주는 것
    - 메시지의 구성요소
      - 메시지를 받을 객체, 수행을 요청한 메서드의 이름, 메서드에 의해 필요한 매개변수
  - 클래스(Class)
    - 어떤 특정 종류의 모든 객체들에 대해 일반적으로 적용할 수 있는 메서드를 정의하고 있는 설계도(blueprint) 또는 프로토타입(prototype)
  - 인스턴스(Instance)
    - 클래스를 실제로 사용할 수 있도록 선언하는 것
    - 실제로 메모리 공간을 차지
    - 인스턴스의 메서드를 이용하여 변수들의 값을 설정 및 변경 가능
  - 객체(Object)
    - 클래스는 객체의 상태와 행동을 정의하고, 이 클래스를 실제 사용할 수 있도록 변수를 선언한 것이 인스턴스이며, 이 인스턴스를 객체라 함

클래스와 객체

- 객체 지향 프로그램 구성
  - 클래스 정의
  - 객체 선언
  - 객체 생성
  - 객체의 멤버 접근 및 활용

- 클래스(Class)
  - C# 프로그램의 기본 단위
    - 재사용성(reusability), 이식성(portability), 유연성(flexibility) 증가
  - 객체를 정의하는 템플릿
    - 객체의 구조와 행위를 정의하는 방법
  - 자료 추상화(data abstraction)의 방법
  - 상태(변수, 필드)와 동작(함수,메서드)을 하나의 타입으로 묶어서 선언하는 것

- 객체(Object)
  - 클래스의 인스턴스로 변수와 같은 역할
  - 객체를 정의하기 위해서는 해당하는 클래스를 정의

클래스 멤버

- 클래스 형식

```
class 클래스명
{
        //클래스 멤버
}
```

- 멤버의 세 종류
  - 자료 멤버(필드, 상수, 이벤트)
  - 함수멤버(메서드, 생성자, 소멸자, 속성, 색인자, 연산자)
  - 내포형(클래스형, 구조형, 열거형, 인터페이스형, 대리형)
- 클래스 멤버
  - 필드(Field) : 변수 혹은 멤버변수. 객체의 상태정보 저장
  - 상수(Constant): 변하지 않는 값이나 숫자
  - 이벤트(Event) : 어떤 사건이 발생하였을때 사용자에게 해당사실을 알려주는 방법
  - 메소드(Method): 객체의 행동특성 정의
  - 생성자(Constructor): 클래스나 구조체의 인스턴스를 생성 혹은 멤버초기화
  - 속성(Property): 필드와 같이 상태정보를 저장하며 내부적으로 상태에 대한 접근 가능한 메소드 제공
  - 인덱서(Indexer): 객체의 배열화 한 형태
  - 연산자(Operator): 특정기능의 수행을 위한 기호

클래스의 멤버

- 클래스의 선언 형태

*클래스 멤버의 종류

-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

public, internal,
abstract, static, sealed

```
[class-modifier] class ClassName {
        // member declarations
        // access modifier
        [field-modifier] DataType fieldNames;
        [method-modifiers] returnType MethodName(parameterList) {
                // method body
        }
        …
}
```

필드, 메소드, 프로퍼티,
인덱서, 연산자 중복, 이벤트

클래스의 멤버

[클래스 멤버의 종류]
-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

[객체 지향 프로그램 구성]
-클래스 정의
-객체 선언
-객체 생성
-객체의 멤버 접근 및 활용

예제4-1 [w3_ex4_1_xxxx] Fraction클래스

- Fraction – 클래스형
  - 필드 2개, 메소드 계통의 멤버 2개

```
using System;
namespace FractionApp
{
    class Fraction
    {
        int numerator;                          // 분자 필드
        int denominator;                        // 분모 필드
        public Fraction(int num, int denom) {       // 생성자
            numerator = num;
            denominator = denom;
        }
        public void PrintFraction() {               // 출력 메소드
            Console.WriteLine(numerator + "/" + denominator);
        }
    }
    class Program {
        static void Main(string[] args) {
            Fraction f = new Fraction(1, 2);   //객체 선언과 객체 생성
            f.PrintFraction();                 //객체의 멤버 접근
        }
    }
}
```

실행 결과 :

객체 선언과 객체 생성

- 객체 선언
  - 클래스형의 변수 선언
  - 예) Fraction f1, f2;
    - f1, f2 – 객체를 참조(reference)하는 변수 선언
- 객체 생성
  - f1 = new Fraction();
  - Fraction f1 = new Fraction();
- 생성자
  - 객체를 생성할 때 객체의 초기화를 위해 자동으로 호출되는 루틴
  - 클래스와 동일한 이름을 갖는 메소드
- 객체의 멤버 참조
  - 객체 이름과 멤버 사이에 멤버 접근 연산자인 점 연산자(dot operator) 사용
  - 참조 형태
    - 필드 참조: f1.numerator
    - 메소드 참조: f1.Add(f2)
    - 연산자 중복: 직접 수식 사용
  - 멤버의 참조 형태

```
objectName.MemberName
```

클래스의 수정자(class modifier)

- 수정자(modifier)
  - 부가적인 속성을 명시하는 방법
- 클래스 수정자(class modifier)
  - class 앞에 액세스 수준을 정의
  - public
    - 다른 프로그램에서 사용 가능
  - internal
    - 같은 프로그램에서만 사용 가능
    - 수정자가 생략된 경우**(default)**
  - static
    - 클래스의 모든 멤버가 정적 멤버
    - 객체 단위로 존재하는 것이 아니라 클래스 단위로 존재
    - 예) 정적 필드 , 정적 생성자 (교재 예제 4-2, 예제4-11 ,참고)
  - abstract, sealed - 파생 클래스 사용 => 상속
  - protected - 상속
  - new – 중첩 클래스에서 사용되며 베이스 클래스의 멤버를 숨김
  - **private사용 할 수 없음!**

접근 수정자 (access modifier)

- 접근 수정자(access modifier)
  - 다른 클래스에서 필드의 접근 허용 정도를 나타내는 속성

접근 수정자	동일 클래스	파생 클래스	네임스페이스	모든 클래스
private	O	X	X	X
protected	O	O	X	X
internal	O	X	O	X
protected internal	O	O	O	X
public	O	O	O	O

  - 접근 수정자의 선언 예

액세스 수준	의 미
Public **(논리적인 액세스 지정자: 클래스 계층 구조 기준)**	어떤 코드에서도 멤버에 액세스 가능
Private **(논리적인 액세스 지정자)**	현재 클래스 안에서 멤버를 액세스 가능 아무것도 선언하지 않으면 기본 값으로 이 값이 저장**(default)**
Protected **(논리적인 액세스 지정자)**	현재 클래스 또는 현재 클래스로부터 파생된 클래스에서만 액세스 가능
internal **(물리적인 액세스 지정자)**	현재 멤버가 정의된 프로젝트 안에서만 액세스 가능 **같은 네임스페이스 내에서 자유롭게 접근**
protected internal	현재 멤버가 정의된 프로젝트 안에서만 또는 파생 클래스에서만 액세스 가능 **파생 클래스와 동일 네임스페이스 내에서도 자유롭게 접근**

필드

- 필드(field)
  - 객체의 구조를 기술하는 자료 부분
  - 변수의 선언으로 구성
- 필드 선언 형태

```
[field-modifier] DataType  fieldNames;
```

- **필드 수정자(field modifier)에는 접근 수정자와 new, static, readonly, volatile등이 있다.**
- 필드 선언 예

```
int anInterger, anotherInteger;
public string usage;
static long idNum = 0;
public static readonly double earthWeight = 5.97e24;
```

메소드

- 객체의 행위를 기술하는 방법
  - 객체의 상태를 검색하고 변경하는 작업
  - 특정한 행동을 처리하는 프로그램 코드를 포함하고 있는 함수의 형태

```
[method-modifiers] returnType MethodName(parameterList) {
        // method body
}
```

- 메소드 선언 예

```
class MethodExample {
    int SimpleMethod() {
      //...
    }
    public void EmptyMethod() { }
}
```

메소드 수정자

- 메소드 수정자: 총 11개
- 접근 수정자: public, protected, internal, private
- static
  - 정적 메소드
  - 전역 함수와 같은 역할
  - 정적 메소드는 해당 클래스의 정적 필드 또는 정적 메소드만 참조 가능
  - 정적 메소드 호출 형태

```
ClassName.MethodName();
```

- abstract / extern
  - 메소드 몸체 대신에 세미콜론(;)이 나옴
  - abstract – 메소드가 하위 클래스에 정의
  - extern – 메소드가 외부에 정의
- new, virtual, override, sealed – 5장 상속 참조

매개변수

- 매개변수
  - 메소드 내에서만 참조될 수 있는 지역 변수
- 매개변수의 종류
  - 형식 매개변수(formal parameter)
    - 메소드를 정의할 때 사용하는 매개변수
  - 실 매개변수(actual parameter)
    - 메소드를 호출할 때 사용하는 매개변수
- 매개변수의 자료형
  - 기본형, 참조형

```
void parameterPass(int i, Fraction f) {
        // ...
}
```

매개변수

- 클래스 필드와 매개변수를 구별하기 위해 this 지정어 사용
  - this 지정어 - 자기 자신의 객체를 가리킴

```
class Fraction {
    int numerator, denominator;
    public Fraction(int numerator, int denominator) {
        this.numerator = numerator;
        this.denominator = denominator;
    }
}
```

매개변수 전달

- 값 호출(call by value)
  - 실 매개변수의 값이 형식 매개변수로 전달 – 예제 4.4
- 참조 호출(call by reference)
  - 주소 호출(call by address)
  - 실 매개변수의 주소가 형식 매개변수로 전달
  - C#에서 제공하는 방법
    - 매개변수 수정자 이용 – 예제 4.5
    - 객체 참조를 매개변수로 사용 – 예제 4.6
- 매개변수 수정자
  - ref – 매개변수가 전달될 때 반드시 초기화
  - out – 매개변수가 전달될 때 초기화하지 않아도 됨

매개변수 전달

예제4-4 [w3_ex4_4_xxxx] 값 호출

```
using System;
class CallByValueApp {
    static void Swap(int x, int y) {
        int temp;
        temp = x; x = y; y = temp;
        Console.WriteLine("  Swap: x = {0}, y = {1}", x, y);
    }
    public static void Main() {
        int x = 1, y = 2;
        Console.WriteLine("Before: x = {0}, y = {1}", x, y);
        Swap(x, y);
        Console.WriteLine(" After: x = {0}, y = {1}", x, y);
    }
}
```

실행 결과 :

매개변수 전달

예제4-5 [w3_ex4_5_xxxx] 주소 호출=참조 호출

```
using System;
class CallByReferenceApp {
    static void Swap(ref int x, ref int y) {
        int temp;
        temp = x; x = y; y = temp;
        Console.WriteLine("  Swap: x = {0}, y = {1}", x, y);
    }
    public static void Main() {
        int x = 1, y = 2;
        Console.WriteLine("Before: x = {0}, y = {1}", x, y);
        Swap(ref x, ref y);
        Console.WriteLine(" After: x = {0}, y = {1}", x, y);
    }
}
```

실행 결과 :

매개변수 전달

예제4-6 [w3_ex4_6_xxxx] 객체 참조를 매개 변수로 사용

```
using System;
class Integer {
    public int i;
    public Integer(int i) {
        this.i = i;
    }
}
class CallByObjectReferenceApp {
    static void Swap(Integer x, Integer y) {
        int temp = x.i; x.i = y.i; y.i = temp;
        Console.WriteLine("  Swap: x = {0}, y = {1}",x.i,y.i);
    }
    public static void Main() {
        Integer x = new Integer(1);  Integer y = new Integer(2);
        Console.WriteLine("Before: x = {0}, y = {1}",x.i,y.i);
        Swap(x, y);
        Console.WriteLine(" After: x = {0}, y = {1}",x.i,y.i);
    }
}
```

실행 결과 :

메소드 중복

- 시그네처(signature)
  - 메소드를 구분하는 정보
    - 메소드 이름
    - 매개변수의 개수
    - 매개변수의 자료형
    - 메소드 반환형 제외
- 메소드 중복(method overloading)
  - 메소드의 이름은 같은데 매개변수의 개수와 형이 다른 경우
  - 호출시 컴파일러에 의해 메소드 구별
- 메소드 중복 예

```
void SameNameMethod(int i) { /* ... */ }        // 첫 번째 형태
void SameNameMethod(int i, int j) { /* ... */ } // 두 번째 형태
```

생성자

- 생성자(constructor)
  - 객체가 생성될 때 자동으로 호출되는 메소드
  - 클래스 이름과 동일하며 반환형을 갖지 않음
  - 주로 객체를 초기화하는 작업에 사용
  - 생성자 중복(overloading) 가능

```
class Fraction {
    // ....
    public Fraction(int a, int b) {        // 생성자
        numerator = a;
        denominator = b;
    }
}
// ...
Fraction f = new Fraction(1, 2);
```

소멸자

- 소멸자(destructor)
  - 클래스의 객체가 소멸될 때 필요한 행위를 기술한 메서드
  - **컴파일러에 의해 자동으로 불리어짐**
  - 소멸자의 이름은 생성자와 동일하나 이름 앞에 ~(tilde)를 붙임
  - 매개변수를 가질 수 없음

- Finalize() 메서드
  - 컴파일 시 소멸자를 Finalize() 메서드로 변환해서 컴파일
  - 프로그래머가 Finalize() 메서드 재정의할 수 없으며 반드시 소멸자를 통해서만 가능함
  - 객체가 더 이상 참조되지 않을 때 GC(Garbage Collection)에 의해 호출

```
Override protected void Finalize(){
Try{
    Console.WriteLine("In the destructor...");
} finally {
   base.Finalize();
}
}
```

- Dispose() 메서드
  - CLR에서 관리되지 않은 자원을 직접 해제할 때 사용
  - 자원이 scpoe를 벗어나면 즉시 시스템에 의해 호출
  - System.Idisposable인터페이스를 구현한 Dispose()메서드를 반드시 갖고 있어야 한다.

프로퍼티

- 프로퍼티(property)
  - 클래스의 private 필드를 형식적으로 다루는 일종의 메서드.
  - 멤버 필드에 값을 할당하는 방법(Set과 Get형식의 함수를 일반화한 형태)
  - 셋-접근자 - 값을 지정
  - 겟-접근자로 - 값을 참조
  - 겟-접근자 혹은 셋-접근자만 정의할 수 있음.
- 프로퍼티의 정의 형태

```
[property-modifiers] returnType PropertyName {
        get {
                // get-accessor body
        }
        set {
                // set-accessor body
        }
}
```

[클래스 멤버의 종류]
-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

[객체 지향 프로그램 구성]
-클래스 정의
-객체 선언
-객체 생성
-객체의 멤버 접근 및 활용

프로퍼티

- 프로퍼티 수정자
  - 수정자의 종류와 의미는 메소드와 모두 동일
  - 접근 수정자(4개), new, static, virtual, sealed, override, abstract, extern – 총11개
- 프로퍼티의 동작
  - 필드처럼 사용되지만, 메소드처럼 동작.
  - 배정문의 왼쪽에서 사용되면 셋-접근자 호출.
  - 배정문의 오른쪽에서 사용되면 겟-접근자 호출.
- set 접근자의 매개 변수 value
  - set함수 내에서 value라는 매개변수를 사용하며 외부에서 들어오는 값을 얻어 낼 수 있다.
  - value는 set접근자가 갖는 디폴트 매개 변수이다.

프로퍼티

[클래스 멤버의 종류]
-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

[객체 지향 프로그램 구성]
-클래스 정의
-객체 선언
-객체 생성
-객체의 멤버 접근 및 활용

퀴즈4_1_1 [w3_quiz4_1_1_xxxx] TimeGetSet

```
using System;
class Time
{                   private int hour, min, sec;
                    public Time(int h, int m, int s)
                              { SetHour(h); SetMin(m); SetSec(s); }

                    public int GetHour() { return hour; }
                    public void SetHour(int aHour) { if (aHour < 24) hour = aHour;
}
                    public int GetMin() { return min; }
                    public void SetMin(int aMin) { if (aMin < 60) min = aMin; }

                    public int GetSec() { return sec; }
                    public void SetSec(int aSec) { if (aSec < 60) sec = aSec; }

                    public void OutTime()
                    {
                     Console.WriteLine("현재 시간은 {0}시 {1}분 {2}초이다.",
                                                              hour, min, sec);
                    }
}
class CSTest
{                   static void Main()
                    {
                                        Time Now;
                                        Now = new Time(12, 30, 45);
                                        Now.OutTime();
                                        Now.SetHour(55);
                                        Now.OutTime();
                    }
}
```

실행 결과 :

퀴즈4_1_2 [w3_quiz4_1_2_xxxx] TimeProperty

```
using System;
class Time
{                   private int hour, min, sec;
                    public Time(int h, int m, int s)
                                        { Hour = h; Min = m; Sec = s; }
                    public int Hour
                    {                   get { return hour; }
                                        set { if (value < 24) hour = value; }}

                    public int Min
                    {                   get { return min; }
                                        set { if (value < 60) min = value; } }

                    public int Sec
                    {                   get { return sec; }
                                        set { if (value < 60) sec = value; }  }

                    public void OutTime()
                    {
                       Console.WriteLine("현재 시간은 {0}시 {1}분 {2}초이다.",
                                                              Hour, Min, Sec);    }
}
class CSTest
{                   static void Main()
                    {                   Time Now;
                                        Now = new Time(12, 30, 45);
                                        Now.OutTime();
                                        Now.Hour = 55;
                                        Now.OutTime();
                    }
}
```

실행 결과 :

프로퍼티

예제4-1 [w3_ex4_1_xxxx] Fraction클래스

```
using System;
namespace FractionApp
{
    class Fraction
    {
        int numerator;                              // 분자 필드
        int denominator;                            // 분모 필드
        public Fraction(int num, int denom) {       // 생성자
            numerator = num;
            denominator = denom;
        }
        public void PrintFraction() {               // 출력 메소드
            Console.WriteLine(numerator + "/" + denominator);
        }
    }
    class Program {
        static void Main(string[] args) {
            Fraction f = new Fraction(1, 2);   //객체 선언과 객체 생성
            f.PrintFraction();                 //객체의 멤버 접근
        }
    }
}
```

실행 결과 :

=> 예제4-14 처럼 속성을 이용하여 프로그램을 작성하면 캡슐화의 특징인 높은 모듈성과 정보 은닉을 제공한다.

예제4-14 [w3_ex4_14_xxxx] Fraction 클래스 속성

```
using System;
class Fraction {
    private int numerator;
    private int denominator;
    public int Numerator {
        get { return numerator; }
        set { numerator = value; }
    }
    public int Denominator {
        get { return denominator; }
        set { denominator = value; }
    }
    override public string ToString() {
        return (numerator + "/" + denominator);
    }
}
class PropertyApp {
    public static void Main() {
        Fraction f = new Fraction(); int i;
        f.Numerator = 1;                // invoke set-accessor in Numerator
        i = f.Numerator+1;              // invoke get-accessor in Numerator
        f.Denominator = i;              // invoke set-accessor in Denominator
        Console.WriteLine(f.ToString());
    }
}
```

실행 결과 :

3주차 학습 내용 정리

* 1차시
  * 2주차 3교시 과제 피드백
  * **4장 클래스**
* 2차시
  * **7장 폼**
    * 윈폼 어플리케이션의 구조
    * 폼 클래스
      * 폼 클래스의 계층도
      * 폼 클래스의 프로퍼티, 메소드, 이벤트
    * 컨트롤 클래스
      * 컨트롤 클래스의 계층도
      * 컨트롤 클래스의 프로퍼티, 메소드, 이벤트
* 3차시
  * **8장 컨트롤(3)**
    * 리스트
      - 리스트 상자
      - 콤보 상자
      - 체크 리스트 상자

[클래스 멤버의 종류]
-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

[객체 지향 프로그램 구성]
-클래스 정의
-객체 선언
-객체 생성
-객체의 멤버 접근 및 활용