컴퓨터정보과
C# 프로그래밍

4주차 클래스와 제어문 (메소드)

강의 순서

1. C# 환경설치 / C# 기본 구조
2. 클래스 기본(필드) + 변수,자료형
3. 클래스 기본(메소드) + 연산자,수식
4. **클래스 기본(메소드) + 제어문**
5. 배열/리스트/딕셔너리
6. 클래스 기본: 접근제한자 (한정자) + 프로퍼티(속성)
7. 클래스 심화:상속
8. 클래스 심화:인터페이스/추상 클래스
9. 예외처리/일반화
10. 파일처리
11. UI (Winform or WPF)
12. LINQ/Delegate/Lamda/…

복습 및 3주차 추가요소

- 연산자
  - 산술/**문자열**/**비교,논리**/대입/증감/sizeof/비트/etc.
- C#구조
  - namespace
    - class
      - field
      - property
      - method
- method
  - 정의
    - return type
    - method name
    - parameter
    - {statement}
  - 호출
    - return value
    - method name
    - argument
- instance(object) 생성 : new ClassName();
- 선언 위치에 따른 변수
  - class
    - instant field
    - static field
  - method
    - local variable

제어문

프로그래밍의 실행 순서를 조정

제어문

- 제어문에 사용하는 조건식의 결과는 C언어 처럼 0이냐 0이 아니냐로 판별 불가
- 반드시 bool 타입으로 조건식의 결과가 나와야 함.

C#

```
int a = 10;
if (a == 10)
{
    …
}
```

C

```
int a = 10;
if (a)
{
    …
}
```

- 분기문
  - 조건분기문 : if, switch-case
    - 조건연산자(삼항연산자)
    - 무조건분기문 : goto
- 반복문 :
  - while, do-while, for, foreach
    - break, continue

if

- 기본 : 괄호의 bool 표현식 결과가 true이면 실행, false이면 무시하는 구문
- if를 활용한 다양한 if 조건 문장
  - if
  - if – else
  - nested if
  - if - else if - else
- C언어와 다른 점: 조건문은 수치데이터가 아닌 bool형 데이터만 취급한다.

switch

- 비교 값을 이용하여 해당하는 case를 실행하는 조건문
- C언어와 다른 점.
  - 정수 데이터가 아닌 문자열 데이터도 비교 값으로 사용할 수 있다.
    - 최근 C#의 switch문법은 기본 문법 외에 많은 기능과 비교 값 타입이 추가되었다.
    - (해당 기능은 수업 시간에 사용하지 않음)
  - 빈 case문일때만 break를 생략할 수 있음.

while

- 가장 기본적인 반복문
  - 조건이 만족하면 계속 반복해서 실행
  - if와 유사 : if는 조건이 만족하면 한 번만 실행
- for문 대비, 반복의 횟수가 명확하지 않는 경우에 많이 사용.

do-while

- while문 변형
- 보통 반복문은 조건을 검사한 후 실행하는데 반해
  - do-while은 먼저 실행한 후 조건을 검사하여 다시 실행할지 결정
- 주로 console 프로그램에서 메뉴 출력 시 유용하게 사용함.

for

- 가장 구조화된 반복문
- for구문을 통해서 시작값, 조건, 횟수등을 유추할 수 있음.
- while 대비, 반복의 횟수가 명확한 경우 사용

foreach (추후 진행하겠음)

- 제어문 중 C언어 없는 유일한 문법
- 향상된 for문이나 foreach라는 용어로 표현함.
  - python의 for문과 유사함.
- 많이 사용하는 for문의 패턴을 간편화한 문법
  - 패턴: 배열과 같은 연속적인 자료형 구조를 처음부터 끝까지 탐색하는 용도
- 문법 : foreach(자료형 변수 in 컬렉션) { … }

생성자 & 종료자

인스턴스를 생성하고 제거한다.

생성자 Constructor

- 인스턴스를 생성할 때 호출한다.
- 생성시 필요한 초기화 작업을 수행한다
- 클래스(구조체) 당 1개 이상 반드시 존재해야 한다.
  - 정의가 되어 있지 않으면 컴파일러가 자동으로 (매개변수 없고, 실행문이 없는) 기본 생성자 생성
  - 프로그래머가 정의한 생성자가 있는 경우 기본 생성자는 호출할 수 없다.

- 정의 형태
  - 반환 타입 없다
    - return 사용하지 않는다.
  - 클래스(구조체)와 동일한 이름이어야 한다.
  - 그 외에는 메소드와 규칙이 동일하다.
  - 예제
    - public Score() { …}
    - public Score(int kor, int eng, int math) { … }

- (참고)구조체의 경우 명시적으로 정의하려면 하나 이상의 매개변수를 가진 생성자에 한해 정의 가능

종료자 Finalizer

- 타 언어어의 소멸자(destructor)와 동일한 역할
- 인스턴스 종료(소멸)시 필요한 정리 작업이 필요한 경우 사용
  - CLR의 Garbage Collection에 의해 Heap에 할당한 영역을 자동으로 삭제할 때 자동 호출
  - 사용자는 해당 호출 시점을 알 수 없다. 즉, 제어할 수 없다.
- 종료자 정의가 되어 있는 경우 Garbage Collection 작업이 복잡해지면서 성능이 떨어지기 때문에 필요한 경우 외에는 사용하지 않는 것을 권한다.
- 정의형태
  - 접근제한자 사용하지 않는다.
  - 반환 타입 없다.
  - 클래스 명 앞에 ~(Tilde) 기호를 붙인다.
  - 매개변수 정의하지 않는다.
  - 예제
    - ~Score() { … }
- (참고) 구조체는 종료자를 정의할 수 없음.

객체지향 프로그래밍

- 정수형, 실수형, 논리형, 문자형, 문자열형은 단일한 해당 객체를 표현하기 위한 기본 자료형이고… 이외에 프로그래밍을 하기 위해 다른 자료형이 필요해진다.

- 성적처리
  - 학생, 과목, 성적, …
- 도서관
  - 책, 대출자, …
- 주차관리
  - 자동차, 주차 시간, 주차 장소,…
- 학교
  - 학생, 선생, 과목, …
- 회사
  - 고용주, 고용인, …
- 게임
  - 플레이어, 적, 아이템, …

주소록 관리 (AddressBook)

```
AddressBook addrBook1 = new AddressBook();
addrBook1.Name = "김인하";
addrBook1.Address = "인천 미추홀";
addrBook1.Phone = "010-1111-1111";
addrBook1.Group = "";

AddressBook addrBook2 = new AddressBook();
addrBook2.Name = "이인하";
addrBook2.Address = "인천 남미추홀";
addrBook2.Phone = "010-1111-1112";
addrBook2.Group = "친구";

Console.WriteLine(addrBook1.Address);
```

```
class AddressBook
{
    public string Name;
    public string Address;
    public string Phone;
    public string Group;

    public AddressBook()
    {
    }
}
```

국어,영어,수학 점수 관리

```
Score score1 = new Score();
score1.Kor = 10;
score1.Eng = 20;
score1.Mat = 30;

Score score2 = new Score();
score2.Kor = 10;
score2.Eng = 20;
score2.Mat = 30;

Console.WriteLine(score1.Kor);
Console.WriteLine(score1.Average);
Console.WriteLine(score2.Average);
```

```
Score score1 = new Score(10, 20, 30);
Score score2 = new Score(10, 20, 30);

Console.WriteLine(score1.Kor);
Console.WriteLine(score1.Average);
Console.WriteLine(score2.Average);
```

Score

```
class Score
{
    public int Kor;
    public int Eng;
    public int Mat;

    public int Average
    {
        get {
            return (Kor + Eng + Mat) / 3;
        }
    }

    public Score(int kor, int eng, int mat)
    {
        Kor = kor;
        Eng = eng;
        Mat = mat;
    }
}
```

면적 관리

```
Rect rect1 = new Rect(20.1, 30.5);
rect1.Width = 20.1;
rect1.Height = 30.5;

Rect rect2 = new Rect();
rect2.Width = 2;
rect2.Height = 3;

rect1.ChangeWidth(20.0);
rect1.ChangeHeight(-3);

Console.WriteLine(rect1.Width);
Console.WriteLine(rect1.Area);
Console.WriteLine(rect2.Area);
```

Rect

```
class Rect
{
    public double Width;
    public double Height;

    public double Area
    {
        get {
            return Width * Height;
        }
    }

    public bool ChangeWidth(double size)
    {
        if (Width + size < 0) {
            return false;
        }

        Width += size;
        return true;
    }

    public bool ChangeHeight(double size)
    {
        if (Height + size < 0) {
            return false;
        }
        Height += size;
        return true;
    }

    public Rect() { }

    public Rect(double width, double height)
    {
        Width = width;
        Height = height;
    }
}
```

method overloading

회원 관리

```
Member mem1 = new Member("김인하", 27, true);
Member mem2 = new Member("이인하", 22);

mem1.ChangeGrade();
mem2.ChangeGrade();

Console.WriteLine(mem1.Name);
Console.WriteLine(mem1.Status);
Console.WriteLine(mem2.Status);
```

Member

```
class Member
{
    public string Name;
    public int Age;
    public bool IsRegular;

    public string Status
    {
        get {
            string type = IsRegular ? "정회원" :
"준회원";
            return $"{Name} 회원은 {type} 입니다.";
        }
    }

    public void ChangeGrade()
    {
        IsRegular = !IsRegular;
    }
    public Member(string name, int age,
                  bool isRegualr = false)
    {
        Name = name;
        Age = age;
        IsRegular = isRegualr;
    }
}
```

경기 팀 관리

```
Team team1 = new Team("SSG", "인천");
team1.Coach = "이숭용";
team1.Level = 9;

Team team2 = new Team("삼성","박진만",3,"대구");

team1.IncreaseLevel(2);
team1.DecreaseLevel(2);

Console.WriteLine(team1.Name);
Console.WriteLine(team1.CurrentStatus);
```

Team

```
class Team
{
    public string Name;
    public string Coach;
    public int Level;
    public string Home;
    public static int LowerLevel = 10;

    public string CurrentStatus { … }

    public void IncreaseLevel(int value) { … }

    public void DecreaseLevel(int value) { … }
```

```
    public Team(string name, string home)
    {
        Name = name;
        Home = home;
    }

    public Team(string name, string coach,
       int level, string home) : this(name, home)
    {
        Coach = coach;
        Level = level;
    }
}
```

this : 현재 instance를 가리킴
this() : 현재 instance의 생성자

은행 계좌 관리

```
Account acc1 = new Account("111 - 1111 - 1", "김인하");
Account acc2 = new Account("111 - 1111 - 2", "김인하", 10000000);


acc1.AddBalance(10000);
acc2.SubBalance(100);
Console.WriteLine(acc1.Balance);
```

Account

```
class Account
{
    public string Number;
    public string Owner;
    public decimal Balance;

    public bool AddBalance(decimal money)
    {
        …
    }

    public bool SubBalance(decimal money)
    {
        …
    }
```

```
    public Account(string number, string owner)
        : this(number, owner, 0)
    {

    }

    public Account(string number, string owner
        , decimal balance)
    {
        Number = number;
        Owner = owner;
        Balance = balance;
    }
}
```

직원 관리

```
Employee emp1 = new Employee("김인하", "20240001");
emp1.Depart = "경영지원";
emp1.Salary = 3_600_0000;

Employee emp2 = new Employee("이인하", "20200005");
emp2.Depart = "기술개발";
emp2.Salary = 4_5000_000;

emp1.ChangeSalary(5.6); //5.6% 상승
Console.WriteLine($"월급:{emp1.MonthlySalary}");
```

Employee

```
class Employee
{
    public string Name;
    public string Number;
    public string Depart;
    public decimal Salary;

    public decimal MonthlySalary
    {
        get {
            return Salary / 12;
        }
    }

    public decimal ChangeSalary(double percent)
    {
        Salary *= (decimal)(1.0 + (percent / 100.0));
        return Salary;
    }
```

```
    public Employee(string name, string number)
    {
        Name = name;
        Number = number;
    }
}
```

메소드 오버로딩

동일한 이름의 메소드를 클래스에 정의할 수 있다.

메소드 오버로딩

- method overloading
  - class/struct 안에 정의하는 메소드를 동일한 이름으로 여러 개 정의할 수 있는 방법
    - 생성자도 동일하게 적용
    - 보통 한 영역 안에 동일한 이름을 갖는 요소는 구별의 어려움으로 인해 불가능 하다.
    - 대표적인 C#의 예: Console.WriteLine();
  - 규칙
    - 메소드의 이름이 동일하면서 매개 변수의 수가 다르다.
    - 메소드의 이름이 동일하면서 매개 변수의 타입이 다르다.
    - 이 외의 규칙은 없음.
  - 연습

```
class Test
{
        public void TestMethod() { … }
        public int TestMethod() { … }
        public double TestMethod(int a) { … }
        public double TestMethod(int b) { … }
        public void TestMethod(int a, int b) { … }
        public void TestMethod(int a, int b) { … }
        public void TestMethod(int a, double b) { … }
}
```

3주차

변수의 종류…

Score (1)

```
class Score
{
    public static double MidtermRate = 0.35;
    public static double FinalRate = 0.35;
    public static double AttendRate = 0.15;
    public static double HomeworkRate = 0.15;

    public double Midterm;
    public double Final;
    public double Attend;
    public double Homework;

    public double Sum()
    {
        var summary = Midterm + Final + Attend + Homework;
        return summary;
    }
}
```

정적 필드 – 클래스 당 하나만 생성하는 변수
(프로그램 실행시 생성, 프로그램 종료시 사라지는 변수)
외부에서 접근하고자 할 때, 인스턴스가 아닌 클래스로 접근

인스턴스 필드-클래스의 인스턴스 별로 생성하는 변수
(인스턴스 생성 시 생성되며, 인스턴스 소멸시 사라지는 변수)
외부에서 접근하고자 할 때, 인스턴스로 접근

지역변수 – 메소드 실행시에만 생성하고, 메소드 실행이 종료되면 사라지는 변수
외부에서 접근할 방법 없음
예외) ref, out 사용시 …

Score (2)

```
class Score
{
    public const double MidtermRate = 0.35;
    public const double FinalRate = 0.35;
    public const double AttendRate = 0.15;
    public const double HomeworkRate = 0.15;

    public double Midterm;
    public double Final;
    public double Attend;
    public double Homework;

    public double Sum()
    {
        var summary = Midterm + Final + Attend + Homework;
        return summary;
    }
}
```

상수 필드 (+정적 필드의 의미도 같이 있음)

추후. 읽기전용 필드와의 관계와 비교해야 함.

Score

```
class Program
{
    static void Main(string[] args)
    {
        var rate1 = $"중간:{Score.MidtermRate * 100}%";
        var rate2 = $"기말:{Score.FinalRate * 100}%";
        var rate3 = $"출석:{Score.AttendRate * 100}%";
        var rate4 = $"과제:{Score.HomeworkRate * 100}%";

        Console.WriteLine(rate1);
        Console.WriteLine(rate2);
        Console.WriteLine(rate3);
        Console.WriteLine(rate4);

        Score score1 = new Score();
        Score score2 = new Score();
```

```
        Score score1 = new Score();
        Score score2 = new Score();

        score1.Midterm = 100;
        score1.Final = 90;
        score1.Attend = 90;
        score1.Homework = 100;
        Console.WriteLine(score1.Sum());

        score2.Midterm = 90;
        score2.Final = 80;
        score2.Attend = 80;
        score2.Homework = 100;
        Console.WriteLine(score2.Sum());
    }
}
```