컴퓨터정보과
C# 프로그래밍

5주차 Collection (배열/리스트/딕셔너리/…)

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 4주차 추가요소

- 제어문
  - 조건의 결과는 반드시 bool 형!
  - 선택문
    - if
    - switch
  - 반복문
    - while / do-while
    - for
    - foreach
  - 기타
    - break
    - continue
- method overloading
  - 한 영역에 다른 메소드와 동일한 이름으로 메소드를 만들 수 있는 방법
    - 기준 :매개변수의 개수나 형
- 특수 method
  - 생성자
    - 인스턴스를 생성할 때 필요한 초기작업을 수행할 때 정의하여 사용
    - method overloadin으로 여러 개 갖을 수 있음
    - 기본 생성자(default constructor) : 매개변수도 없고, 아무런 실행 문장이 없는 생성자
      - 생성자를 프로그래머가 정의하지 않으면 컴파일러는 기본 생성자를 강제로 넣는다.
  - 종료자
    - 인스턴스 소멸시 필요한 마무리 작업을 수행할 때 정의하여 사용
    - 호출 시점을 알 수 없다. (CLR의 GC 작업 시 호출)
      - 종료자 정의를 하면 GC작업 단계가 늘어나 불필요하게 정의하지 않도록 한다.
    - 오직 하나만 정의할 수 있음

배열

동일한 자료형의 모임

배열 Array

- Array : 동일한 데이터 형식의 요소가 메모리에 순서대로 나열된 데이터 구조
  - 크기는 선언될 때 정적으로 정의되며, 실행 중에 크기를 변경할 수 없습니다(고정길이)

1차원 배열 Array

- 데이터형식[ ] 배열이름 = new 데이터형식[크기];
  - int[] array = new int[5];
  - 인덱스(index) :[0~ n-1]
    - array[1] = 2;
    - int a = array[3];
- 초기화
  - 초기화 하지 않으면, 기본 값으로 초기화 됨
    - 값타입(0, 0.0, false) vs. 참조타입(null)
  - int[] array = new int[3]{1,2,3};
  - int[] array = new int[]{1,2,3};
  - int[] array = {1,2,3};

예제

```
string[] arrName = new string[] { "1", "2", "3" };

Console.WriteLine("{0}차원", arrName.Rank);

for (int i = 0; i < arrName.Length; i++) {
    Console.WriteLine(arrName[i]);
}

foreach (var name in arrName) {
    Console.WriteLine(name);
}

for (int i = 0; i < arrName.GetLength(0); i++) {
    Console.WriteLine(arrName[i]);
}
```

다차원 배열 Array

- 데이터형식[,] 배열이름 = new 데이터형식[2차원길이(행), 1차원길이(열)];
  - int[,] array = new int[2,3];
  - 인덱스(index) : [0~ i-1 , 0~j-1]
    - array[0,1] = 2;
    - int a = array[1,2];
- 초기화
  - 초기화 하지 않으면, 기본 값으로 초기화 됨
    - 값타입(0, 0.0, false) vs. 참조타입(null)
  - int[,] array = new int[2,3]{ {1,2,3} , {4,5,6} };
  - int[,] array = new int[] { {1,2,3} , {4,5,6} };
  - int[,] array = { {1,2,3} , {4,5,6} };

예제

```
string[,] arrName = new string[,]
{
    { "1", "2", "3" },
    { "11", "12", "13" },
    { "21", "22", "23" }
};

Console.WriteLine("{0}차원", arrName.Rank);

//for (int i = 0; i < arrName.Length; i++) {
//    Console.WriteLine(arrName[i]);
//}

foreach (var name in arrName) {
    Console.WriteLine(name);
}

for (int i = 0; i < arrName.GetLength(0); i++) {
    for (int j = 0; j < arrName.GetLength(1); j++) {
        Console.WriteLine(arrName[i, j]);
    }
}
```

Array관련 메소드와 프로퍼티

- 정적 메소드
  - Clear() : 배열의 모든 요소 초기화
  - Sort() : 배열을 정렬
  - IndexOf() : 배열에서 특정 데이터의 인덱스를 반환
- 인스턴스 메소드
  - **GetLength**(n) : 배열에서 지정한 차원의 길이 반환, 다차원 배열에서 유리
    - 1차원 0 , 2차원 1, 3차원 2, …
- 인스턴스 변수(멤버 변수)
  - **Length** : 배열의 전체 길이 반환
  - Rank : 배열의 차원 반환
- 가변배열 (jagged array)
  - 배열이 요소인 배열
    - int[][] test = new int[2][];

Collection

같은 성격을 갖는 데이터의 모음

Collection

기본 Collection	Generic Collection
ArrayList	List<T>
Queue	Queue<T>
Stack	Stack<T>
Hashtable	Dictionary<TKey, TVvalue>
모든 자료형(object)을 담을 수 있다.	T형의 자료형만 담을 수 있다.

- 일반화 프로그래밍 (Generic Programming)
  - 특정한 데이터 유형을 결정하지 않고 일반화된 형태로 구현하는 프로그래밍 기법
  - 형식 매개변수를 이용해 사용하고자 하는 데이터의 유형을 결정한다.
- 일반화 콜렉션
  - 제네릭 프로그래밍 개념에 따라 구현된 컬렉션 자료 구조
  - 기본 콜렉션은 모든 데이터를 object로 형을 변환해서 저장한다.
    이를 다시 사용하려면 원래 형태로 형변환을 진행해야한다. 이에 따른 부하가 발생

List<T>

- ArrayList의 generic 버전
- 배열과 가장 닮은 콜렉션
  - 요소 접근시 [index] 이용
- 크기를 미리 지정할 필요가 없음
  - 가변길이 (Count)
- 주요 메소드
  - Add(), RemoveAt(), Insert()
- 생성 예제
  - List<int> listNumber = new List<int>();
  - T의 자리에 저장하고자 하는 데이터 형 int를 지정한다.
    - listNumber 리스트는 int의 데이터만 저장한다.
  - 나머지는 코드로 확인

Dictionary<TKey, TVal>

- Hashtable의 generic 버전
- 키(key)와 값(value)의 쌍으로 이루어진 데이터
  - key는 중복될 수 없다.
- 크기를 미리 지정할 필요가 없음
  - 가변길이 (Count)
- key를 기준으로 value를 찾을 수 있다.
  - 요소 접근시 [] 이용
  - []안에는 인덱스가 아니라 key를 사용한다.
- 주요 메소드
  - Add(), RemoveAt(), Insert()
- 생성 예제
  - Dictionary<string, int> dictNumber = new Dictionary<string, int>();
  - key는 string 형, value는 int형
  - 나머지는 코드로 확인

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";

AddressBook[] addressBooks = { addrBook1, addrBook2 };

for(int i=0; i < addressBooks.Length; i++) {
    Console.WriteLine(addressBooks[i]);
}
```

주소록 관리 (AddressBook)

```
class AddressBook
{
    public override string ToString()
    {
        StringBuilder builder = new StringBuilder();

        builder.Append("이름:").Append(Name).Append(Environment.NewLine);
        builder.Append("주소:").Append(Address).Append(Environment.NewLine);
        builder.Append("전화:").Append(Phone).Append(Environment.NewLine);
        var group = string.IsNullOrEmpty(Group) ? "없음" : Group;
        builder.Append("그룹:").Append(group).Append(Environment.NewLine);

        return builder.ToString();
    }
}
```

국어,영어,수학 점수 관리

```
Dictionary<string, Score> scores = new Dictionary<string, Score>();
int count = 0;
while(count < 3) {
    Console.Write("학번:");
    var number = Console.ReadLine();
    if(string.IsNullOrEmpty(number) || number.Length < 5) {
        continue;
    }

    if (scores.ContainsKey(number)) {
        continue
    }

    Console.Write("국어:");
    if (false == int.TryParse(Console.ReadLine(), out int kor)) {
        continue;
    }
```

국어,영어,수학 점수 관리

```
    Console.Write("영어:");
    if (false == int.TryParse(Console.ReadLine(), out int eng)) {
        continue;
    }

    Console.Write("수학:");
    if (false == int.TryParse(Console.ReadLine(), out int mat)) {
        continue;
    }

    scores[number] = new Score(kor, eng, mat);

    count++;
}

foreach(var score in  scores) {
    Console.WriteLine("학번:{0} 평균:{1}", score.Key, score.Value.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

```
List<Rect> rects = new List<Rect>();
rects.Add(new Rect(20.1, 30.5));

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

rects[0].ChangeWidth(20.0);
rects[1].ChangeHeight(-3);

foreach(Rect rect in rects) {
    Console.WriteLine(rect.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

```
Dictionary<int, Member> members = new Dictionary<int, Member>();
members[1] = new Member("김인하", 27, true);
members.Add(2, new Member("이인하", 22));

members[1].ChangeGrade();
members[2].ChangeGrade();

foreach (var member in members) {
    Console.WriteLine("등록번호:{0} {1}", member.Key, member.Value.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[] teams = new Team[10];
teams[0] = new Team("SSG", "인천");
teams[0].Coach = "이숭용";
teams[0].Level = 9;

teams[1] = new Team("삼성","박진만",3,"대구");

teams[0].IncreaseLevel(2);
teams[1].DecreaseLevel(2);

for(int i = 0; i < teams.Length; i++) {
    Console.WriteLine("{0}-{1}",teams[i].Name, teams[1].CurrentStatus);
    //왜 에러가 날까요?
    //처리할 수 있는 방법은?
}
```

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

```
List<Account> accounts = new List<Account>();
accounts.Add(new Account("111 - 1111 - 1", "김인하"));
accounts.Add(new Account("111 - 1111 - 2", "김인하", 10000000));

accounts[0].AddBalance(10000);
accounts[1].SubBalance(100);
for(int i=0; i <accounts.Count; i++) {
    Console.WriteLine("{0}:{1}원", accounts[i].Number, accounts[i].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

```
List<Employee> employees = new List<Employee>();
employees.Add(new Employee("김인하", "20240001"));
employees[0].Depart = "경영지원";
employees[0].Salary = 3_600_0000;

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

employees[0].ChangeSalary(5.6); //5.6% 상승

foreach(var emp in  employees) {
    Console.WriteLine($"{emp.Name} - {emp.Salary:F0}원");
}
```

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;
    }
}
```