**씨샵 핵심 정리**

**///1장**

**1.플랫폼**소프트웨어 응용 프로그램의 실행에 사용되는 소프트웨어와 하드웨어의 집합

**2.라이브러리**

코드를 쉽게 사용하기 위해 미리 짜여진 코드

**2.프레임 워크**

제어 역전이 있는 대규모의 라이브러리

**////2장**

정수/정수 = 정수이다.

나머지 연산자의 부호는 왼쪽 피연산자의 부호를 따온다.

EX> 4%-3 = 1 , -4%-3= -1 , -4 % 3 = -1

실수를 만들려면 . 사용 0= 정수 0.0 = 실수

실수에선 나머지 연산자 예측이 어려워서 사용 잘안함

문자는 ‘가‘ 형식으로 선언

문자열은 “ 집에가고싶다 ㅠ” 형식 문자열[숫자]형식을 통해 문자열안에 문자들 탐색가능 C언어 배열이랑 비슷함

인덱스가 배열범위를 벗어나면 오류 출력함

문자열은 + 를 통해 서로 연결되지만 문자는 +를 이용하면 숫자로 변환된다.( 문자자료형은 숫자랑 가까움)

Long자료형을 나타내는기호 L인데

6534l 경우 l을 1과 혼동하기 쉬움 주의할 것

그래서 L로쓴다 보통

다른 모든 자료형의 원형은 struct로 시작하지만 String만 class로 시작한다

복합 대입연산자 (+=,/=)

증감연산자(++,--) 순서 위치 조심

**자료형 검사**

GetType()메소드를 이용해서 검사한다.

Ex> console.write((273).GetType());

출력의 경우 System.Int32 이런식으로 나옴

**var키워드**

var은 자동으로 자료형을 저장해 주는데

한번 지정된 자료형은 계속 유지된다. 변형이 불가함

두가지 조건을 만족해야 쓸수있다.

지역변수로 선언하고, 변수를 선언과 동시에 초기화 해야한다. Ex> var global=52; 식으로

Parse()를 이용해서 다른 형태로 다 바꿀 수 있다.

Int.parse(“52”);

파란색으로 한 부분 형태가 같아야한다.

Ex> int.parse(“52.321”); 일 경우 오류 출력

다른 자료형을 문자열로 변환할라고 하면

ToString() 사용할것.

Ex> (10).ToString();

**////3장**

**Switch 반복문**

Case 개념에서 c랑은 다르게

Break; 문이 없을경우 훑으면서 내려오는게 아니라

그냥 중복 조건으로 된다

Case 3:

Case 4: console.writeline(4);

일경우에 3과4가 입력되면 4를 출력한다.

**삼항 연산자**

String input = Console.Readline();

Int number = Int.parse();

Console.Writeline( number>0 ? ”자연수입니다”:”자연수가 아닙니다”);

이거 나올 듯 왠지 파란색 참거짓 판별부분은 자료형이 똑같아야지만 실행된다.

**클래스 배열**

Phone[] phone = new Phone[100];

for(int i=0;i<100;i++)

{

phone[i] = new Phone();

}

이런식으로 하자 왠지 나올거같다

**////4장 5장**

**배열 선언 방법**

Int [] a = {2,3,4};

**배열의 길이 알아내기**

Console.WriteLine(a.Length);

배열의 범위를 벗어나는 인덱스에 접근하면 오류가 출력된다

**원하는 크기의 배열 생성**

Int[] array = new int [100];

**와일 반복문**

While(true) 기억하고 while(1)은 아님

**역 for 반복문**

For(int i = array.length – 1;i > = 0;i++)

{

Console.WriteLine(array[i]);

}

**Foreach 반복문**

foreach (var a in array)

{

Console.WriteLine(a);

}

**/////////////외의 잡것들**

String input = “Potato”;

Console.WriteLine(input.ToUpper());//전부 대문자로

Console.WriteLine(input.ToLower());//전부 소문자로

출력할 때 바꿔줘야함 애네는 비파괴적 메소드라서 자기자신을 변형하지않고 출력할때만 바꿔줌

**문자열 자르기**

Split() 메소드를 사용하면된다

String input = "감자 고구마 토마토";

String[] inputs = input.Split(new char[] { ' ' });

foreach(var a in inputs)

{

Console.WriteLine(a);

}

**문자열 양옆 공백 제거**

Trim() 메소드를 사용하면 된다.

String input = " test \n";

Console.WriteLine(input.Trim());

**클래스 작성**

Car car = new Car();

**Random 클래스 작성**

Random random = new Random();

Console.WriteLine(random.next(10,100)); // 10에서 100사이에 정수 난수 지정문

Console.WriteLine(random.nextDouble()); // 0.0에서 1.0사이에 실수 난수 지정문

Console.WriteLine(random.nextDouble()\*10); // 0.0에서 10.0사이에 실수 난수 지정문

**List 클래스 작성**

리스트 클래스는 크기가 가변이다 계속 변화가능

List <int> list = new List<int>() ;

추가하고싶을땐 Add 를쓰고 지우고싶을땐 Remove를 쓰자

list.Add(52);

list.Add(48);

제거하고싶을땐

list.Remove(52)

길이를 재고싶을땐

Console.WriteLine(list.count);

**추상화 모델클래스 작성**

class my

{

public String name;

public int grade;

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApp1

{

class my

{

public String name;

public int grade;

}

class Program

{

static void Main(string[] args)

{

List<my> list = new List<my>();

list.Add(new my() {name="김신욱",grade =1 }); /// 클래스를 이용한 리스트 구현

list.Add(new my() { name = "킹정욱", grade = 2 });

list.Add(new my() {name = "김킹킴", grade = 3 });

list.Add(new my() { name = "감자킴", grade = 1 });

for (int i = list.Count - 1; i >= 0; i--)///제거는 역반복문으로해야함 foreach는 불가능 정반복문 경우에 요소가 지워지면서 인덱스가 밀려서 오류가 발생함

{

if (list[i].grade > 1)

{

list.RemoveAt(i);

}

}

foreach(var a in list)

{

Console.WriteLine(a.name + " " + a.grade);

}

}

}

}

**////6장**

메소드 형태

Ex> public void get(int a,int b)  
{

}

매개변수는 여러 개를 가질수 있다.

**클래스 메소드**

Ex>

Class car

{

public static int wash (int input)

{ }

}

Static 선언시에는 메인에서 클래스 생성을 안해도

( ex>new car) car.wash()이러한 형태의 메소드를 쓸수있다.

**메소드 오버로딩**

단순하게 이름은 같고 매개변수는 다른 메소드를 만드는 것으로 정의된다.

이때 중요한 점은 이름이 같고 매개변수가 달라야하고 반환값이 다르다고해서 오버로딩이 일어나진 않는다.

**접근제한자**

대표적으로 public과 private가 있다.

Main은 기본적으로 private메소드이고 접근 제한자 입력을 안하면 자동으로 private 제한자 설정이된다.

Private 는 자신의 클래스 내부에서만 사용가능함

**생성자**

-이름은 클래스 이름과 같아야한다.

-접근제한자는 pubilc으로 해야함

-반환과 관련된 선언 x

Ex> class product

{

Public Int a;

Public product (int a)

}

**소멸자**(아마 안나올거같지만 불안하니 보긴하자)

Ex>~product()

{

}

• 이름은 클래스 이름 앞에 ~ 기호 붙은 것

• 접근 제한자 사용 않음

• 반환과 관련된 선언 하지 않음

• 매개변수와 관련된 선언 하지 않음

• 하나의 클래스에는 하나의 소멸자만

**상수**

선언할 때 const 를쓰면된다

Ex> public const double pi=3.14;

상수는 변경할수없다.

**캡슐화**

Private를 통해서 접근을 못하게 하는 것

**속성**

**겟터,셋터를 통한 값설정**

Ex>

private int width;

private int height;

public int Width

{

get { return width; }

set

{

if (value > 0) { width = value; }

else Console.WriteLine("너비는 자연수만");

}

}

public int Height

{

get { return height; }

set

{

if (value > 0)

{

height = value;

}

else

{

Console.WriteLine("높이도 자연수만");

}

}

}

속성은 자바에 없는 개념이므로 이해가 힘들다 ㅅㅂ  
왜쓰는지도모르겠고 그냥 set,get 메소드 만들어 쓰는게 빠를거같은데 그냥 외우자일단

**깊은복사(참조복사) 얕은복사(값복사)**

얕은복사의 경우

static void Change(int input)

{

input = 20;

}

static void Main(string[] args)

{

int a = 10;

Change(a);

Console.WriteLine(a);

}

결과: 10출력

결과적으로 변수 a와 변수 input은 독립적인 변수이므로 서로가 서로를 변경해도 영향을 주지않음

깊은 복사의 경우

class MyClass

{

public int value = 10;

}

class MainApp

{

static void Change(MyClass input)

{

input.value = 20;

}

static void Main(string[] args)

{

{

MyClass myclass = new MyClass();

myclass.value = 10;

Change(myclass);

Console.WriteLine(myclass.value);

}

}

}

이런식으로 객체안에 메소드를 이용해서 바꿔줘야 한다.

using System;

namespace DeepCopy

{

class MyClass

{

public int MyField1;

public int MyField2;

public MyClass DeepCopy()

{

MyClass newCopy = new MyClass();

newCopy.MyField1 = this.MyField1;

newCopy.MyField2 = this.MyField2;

return newCopy;

}

}

class MainApp

{

static void Main(string[] args)

{

Console.WriteLine("Shallow Copy");

{

MyClass source = new MyClass();

source.MyField1 = 10;

source.MyField2 = 20;

MyClass target = source;

target.MyField2 = 30;

Console.WriteLine("{0} {1}", source.MyField1, source.MyField2);

Console.WriteLine("{0} {1}", target.MyField1, target.MyField2);

}

Console.WriteLine("Deep Copy");

{

MyClass source = new MyClass();

source.MyField1 = 10;

source.MyField2 = 20;

MyClass target = source.DeepCopy();

target.MyField2 = 30;

Console.WriteLine("{0} {1}", source.MyField1, source.MyField2);

Console.WriteLine("{0} {1}", tagret.MyField1, target.MyField2);

}

}

}

}

교수님 코드

핵심은 기본 자료형은 값이 들어있고 객체는 참조가 들어있다 이다.

**Dictionary 클래스**

Dictionary<string, string> hoons = new Dictionary<string, string>();

<key,value> 임

이런식으로 선언한다

자바의 HashMap 과 매우 비슷

using System;

namespace DeepCopy

{

using System;

using System.Collections.Generic;

namespace Jinsoo.Examples

{

class DictionaryTest

{

static void Main(string[] args)

{

Dictionary<string, string> hoons = new Dictionary<string, string>();

Console.WriteLine("-------------------------------------------");

// 사전에 추가

hoons.Add("HS", "HOONS");

hoons.Add("JS", "JINSU");

hoons.Add("YS", "YOUNGSU");

hoons.Add("SY", "SUYOUNG");

Console.WriteLine(hoons["JS"]);

// 키가 존재하면 값이 대체 됩니다.

hoons["JS"] = "JISUNg park";

Console.WriteLine(hoons["JS"]);

Console.WriteLine("-------------------------------------------");

// 사전에서 제거합니다.

hoons.Remove("JS");

// 지정한 키가 없다면 예외가 발생합니다.

try

{

Console.WriteLine(hoons["JS"]);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

// 키의 존재 우뮤 판단으로 안전하게 사전에 추가할 수 있습니다.

if (!hoons.ContainsKey("DJ"))

{

hoons.Add("DJ", "DONGJIN");

}

else

{

hoons["DJ"] = "DONGJIN";

}

// 설정/검색이 가능한 키/값 쌍을 나열합니다.

foreach (var h in hoons)

{

Console.WriteLine("Key = {0}, Value = {1}", h.Key, h.Value);

}

Console.WriteLine("-------------------------------------------");

// 사전 키를 나열합니다.

foreach (var hKey in hoons.Keys)

{

Console.WriteLine(hKey);

}

Console.WriteLine("-------------------------------------------");

// 사전 값을 나열합니다.

foreach (var hValue in hoons.Values)

{

Console.WriteLine(hValue);

}

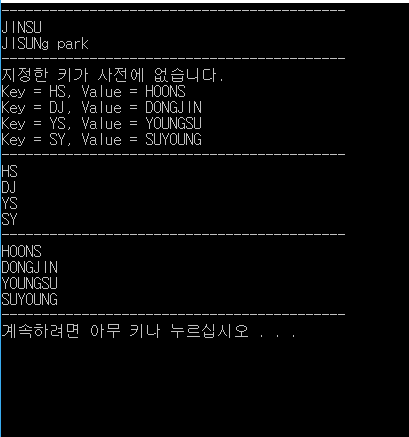
Console.WriteLine("-------------------------------------------");

}

}

}

}



이 코드가 그대로 나올거같으니 그냥 통째로 외우자.

**////7장**