**GrammerNote**

**상수**: 바꿀 수 없는 데이터. 절대 바꾸지 말아야 한다는 것을 표시해 두는 용도

const int nunber = 0;

**enum**: 열거형 새로운 타입

enum Choice { Rock, Paper, Scissors }

(int) Choice.Scissors (enum Choice의 2번 값)

**메소드**: (private) (static) 반환형식 메소드명(매개변수1, 매개변수2...) {return result}

**Ref**: 복사(짝퉁) 가 아닌 참조(진퉁) 데이터 원본이 바뀌니까 사용에 주의

int num1 = 0;

static int AddOne(ref int number) { return number + 1}

사용: AddOne(ref num1); 결과값:num1 = 1;

**Out:** 메서드 매개변수에 돌려보낼값 넣는거 리턴값이 여러개 있을때 사용

static void AddOne(int a, out int b)

int result1 = 0; 아웃으로 받아올 값을 넣을 변수

사용: AddOne(10, out result1)

bool 인벤토리빈공간찾기(out index)

{

if (찾으면)

{

index = 찾은번호;

return true;

}

}

**오버로딩:** 같은 이름의 함수 재사용

static int Add(int a, int b) { return a + b}

static float Add(float a, float b) { return a + b}

**오버라이딩(재정의)**

**abstract**는 클래스도 반드시 abstract여야 한다. 메서드 틀만 만들 수 있음 메서드 내부 구현불가 하위 클래스는 메서드를 반드시 구현해야한다.

**virtual** 은 속성이나 메서드를 정의할 수 있다 하위 클래스는 메서드를 재정의 해도 되고 안 해도 된다

public **override** int Add() { }

**Interface**: 붕어빵 틀의 틀

인터페이스는 다중 상속 가능, 전부 abstract 뭐가 있어야 되는지 만 규정해 놓은 기능 없는 클래스 틀을 가진

**배열**

**iterator** 반복자는 객체 지향적 프로그래밍에서 배열이나 그와 유사한 자료 구조의 내부의 요소를 순회(traversing)하는 객체이다.

배열 b는 iterator를 가지고 있으므로 foreach문에서 순회가 가능하다.

int[] b(배열이름) = new int[5(배열크기)]

int[] b = { 1, 2, 3, 4, 5 };

b[3] = 1; (b의 3번 자리에 정수 1 대입)

foreach(int a in b {Debug.Log(a)}

int[,] array1 = new int[2, 2] { { 0, 0 }, { 1, 1 } }; (2차원배열 선언과 사용)

int[][] array2 = new int[2][]

{

new int[2] { 0, 0 },

new int[4] { 1, 2, 3, 4 }

};

**List**

using System.Collections.Generic; 달아줘야함

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

list.Add(1); 리스트 마지막에 1 추가

list.Insert(2, 999) 리스트의 2번째 자리에 정수 999를 넣기 기존 리스트의 숫자들은 한칸씩 밀림

list.Count 리스트의 총 길이

list.Remove(삭제할 값) 앞에서 찾기 시작해서 맨 첫번째 삭제할 값 삭제

list.RemoveAt(2) 2번째 자리의 리스트 값 지우기

list.Clear() 리스트 전체 비우기

**Dictionary:** 키 -> 값 해시테이블: 상자들 나눠 담기.

class Monster {public int id;

public Monster(int id) { this.id = id} }(생성자 메서드)

Dictionary<int, Monster> dic = new Dictionary<int, Monster>(); (int를 키로 가지고 Monster 클래스 타입을 값으로 가지는 Dictionary 키 값 쌍)

dic.Add(1, new Monster(1)); 1번 아이디를 가진 몬스터를 넣어줌 dic[1] = new Monster(1); 과 같음

Monster mon;

bool found = dic.TryGetValue(1, out mon); TryGetValue는 값을 찾았는지 bool값을 반환함

dic.Remove(키) 키 값으로 찾아서 하나 지움 , dic.Clear(); 전체 삭제

**Generic:** 타입이 함수를 만들 때 정해지는 것이 아니라, 실행할 때 타입이 정해지는 함수가 제네릭 함수라고 함

<>꺽쇠 안에 원하는 타입을 적어서 실행 ,ㅣ부등호 안에 들어가있는 글자를 원하는 타입으로 치환

T IsFight<T, A>(T t, A a, out A flag1) where T : Component where A : Component

{

flag1 = a.GetComponent<A>();

return t.GetComponent<T>();

}

**Property**

get이나 set에는 접근한정자를 적을 수 있는데 **Property**자체의 접근 한정자보다는 작아야 한다. **Property**가 private이었다고 해보면 set public이라도 밖에서 어차피 못봄.

public int a { get; private set; } //get, set은 변수 자체의 보호수준을 기본값으로 따라 감

**Property**멤버 변수와 메소드의 중간 단계다

변수를 접근하는 것처럼 그대로 접근을 할 수 있어

(변수에 접근할 때 특별한 일을 하고 싶은 경우의 예시)

private float curHealth = 30;

private float maxHealth = 30;

public float healthRate

{

get { return curHealth / maxHealth; } (get 뒤에 return 붙여야 함)

set { curHealth = value \* maxHealth; } (value는 예약어)

}

Debug.Log(myCharacter.healthRate);

myCharacter.healthRate = 0.3f;

**읽기 전용 필드**: Property인데 get만 있고 set이 없음

**Delegate** 함수 포인터: 함수의 시작점을 가리키는거

delegate int IntegerCalculator(int left, int right); (반환이 int 매개변수 int 두 개인 이렇게 생긴 함수들을 저장할 수 있게끔 공간을 마련함, 델리게이트에 메서드를 더하려면 반환형 매개변수 두개 다 똑같아야 함)

public delegate int IntegerCalculator(int left, int right);

public IntegerCalculator currentCalculator = Take;

public static int Take(int left, int right)

{

Console.WriteLine($"환장하겠네{left}");

return left;

}

문법메모 문법 = new 문법메모();

문법.currentCalculator += 문법메모.Take;

문법.currentCalculator += 문법메모.Take;

문법.currentCalculator = 문법.currentCalculator + 문법메모.Take;

문법.currentCalculator(1, 1);

**Action 액션:** 반환값 void이면 Action**,** public Action<int> a;

public void b(int c) (반환값이 void고 매개변수가 int인 델리게이트**)**

{

Console.WriteLine(c);

}

문법.a += 문법.b;

문법.a += 문법.b;

문법.a.Invoke(2);

**Function:** 펑션은 액션과 똑같은데 맨 마지막이 반환값 타입

public Func<int, int, int> myFunc;

public virtual int a더하기b(int a, int b)

{

return a + b;

}

public virtual int a곱하기b(int a, int b)

{

return a \* b;

}

문법.myFunc += 문법.a더하기b;

문법.myFunc += 문법.a더하기b;

문법.myFunc += 문법.a더하기b;

문법.myFunc += 문법.a곱하기b;

Console.WriteLine(문법.myFunc.Invoke(1, 1)); (이 경우 맨 마지막 a곱하기b의 리턴값만 출력됨)

**Event:** delegate인데 다른 클래스들은 event에 함수 추가는 가능, 근데 실행은 안됨**.** 실행은 event를 가지고 있는 클래스 본인만 호출 가능.다른 클래스에서 구독 가능**.** 남들이 함부로 실행할 수 없도록 제한해 놓은 것

예를 들면 public set private get과 비슷함

public delegate int 함수명(int 매개변수1, int 매개변수2);

public event 함수명 이벤트명;

public int 함수명2(int a, int b)

{

이벤트명 += 어떤메서드;

이벤트명(a, b);

return a + b;

}

**Lambda:** 코드에서 함수를 만드는 것. 델리게이트 펑션 액션 이벤트 에서만 사용 가능

public class Button

{

public Action OnClick;

}

문법메모.Button attackButtom = new 문법메모.Button();

attackButtom.OnClick = () =>

{

Console.WriteLine("아야");

};

문법메모.Button jumpButton = new 문법메모.Button();

jumpButton.OnClick = () =>

{

Console.WriteLine("y축 좌표로 1이동");

};

attackButtom.OnClick();

jumpButton.OnClick();

**Reflection:** 보고 싶은 프로그램 내의 정보를 엿볼 수 있다

public int a = 1;

public Type GetType1()

{

return Type.GetType("문법메모");

}

public int sum(int a, int b, int c)

{

return a + b + c;

}

Type b = 문법.GetType1();

b.GetMethod("Equals");

Console.WriteLine(b.GetMethod("Equals"));

Console.WriteLine(b.GetMethod("sum").GetParameters()[0]);

Console.WriteLine(b.GetMember("a"));

**Nullable:** int 같은 null을 쓸 수 없는 자료형에 null을 사용할 수 있게 된다.

static int? a;

int d = a ?? 1;

int? FindLocation()

{

return null;

}

void StateSomething()

{

if (FindLocation() != null)

{

}

}

**Tuple:** 함수 하나에 여러 리턴값 밀어넣는거 out이 빠르니 out을 쓰자

public (int, float) sum(int a, int b, int c)

{

return (a + b + c, a + b);

}

(int, float) a = 문법.sum(2, 2, 3);

Console.WriteLine(a.Item1);

Console.WriteLine(a.Item2);

**Struct**: 구조체 C#에서는 클래스와 똑같음 C++에서도 똑같음. C에서는 여러 변수를 한번에 들고 다니기 위해 사용됨. 사람들이 관용적으로 struct를 클래스인데 메소드를 만들지 않는 클래스로 사용. 그러나 메서드 만들 수 있음 그러나 상속은 안됨 인터페이스는 됨. 핵심: 변수 정리용

struct StructA

{

}

**메소드 숨기기:** new 키워드로 부모 클래스(기반 클래스)의 메소드를 숨기고 자식 클래스(파생 클래스) 메소드 호출 가능. 자식 클래스에서 new 쓰면 부모 클래스의 메서드는 사용되지 않음 가급적 오버라이딩 하시오

public new int sum(int a, int b)

{

return a + b;

}

**Sealed:** 오버라이딩 봉인하기. 본인은 부모 함수를 오버라이딩 했는데 자식한테는 오버라이딩 하지 말라고 함

public sealed override void Something()

**중첩 클래스:** 이 클래스는 어떤 클래스와 관계 있는 클래스다 하고 네임스페이스처럼 쓰는 것. 중첩 클래스를 사용하면 두 클래스의 멤버들을 서로 쉽게 사용할 수 있고 외부에는 불필요한 관계 클래스를 감춤으로써 코드의 복잡성을 줄일 수 있다는 장점

**분할 클래스:** 클래스가 너무 클 경우 쪼갠다

public partial class Pclass

{

public int c = 0;

}

partial class Pclass

{

public int b = 1;

}

Pclass a = new Pclass();

a.b = 1;

a.c = 2;

**확장 메소드**:

static class ExtensionMethod

{

public static int sum(this int a, int b)

{

Return a + b;

}

}

Console.WriteLine(3.sum(5));

Console.WriteLine(46.sum(5));

Console.WriteLine(ExtensionMethod.sum(3, 5));

예시2: public static void PizzaFight(this GameObject Player, GameObject Enemy)

**스택 메모리:** 지역변수 매개변수

왜 선입후출이 가장 현명했을까? 어쩔 수 없었기 때문에, 함수는 무조건 선입후출이어야 함

public int Printf(int a)

{

Console.WriteLine(a);

return a - 1;

}

문법.Printf(문법.Printf(문법.Printf(문법.Printf(4))));

먼저 들어온 함수를 먼저 실행할 수 없음 위에 쌓여있는 함수들 먼저 처리 해야 맨 처음 함수 처리 가능

**힙 메모리:** 인스턴스변수. 동적할당된 것들 다 힙 메모리에 들어감 유저가 뭘 찾을지 모름. CPU가 가지는 캐시에서 캐시 미스가 나도 최대한 빨리 찾아야 함 가장 빨리 찾으려면 완전 이진트리여야함

**네임스페이스**

namespace MathSpace

{//수학 기능 클래스와 메서드들}

MathSpace.Sum.sum();

**Extern:** 외부 파일 사용

using System.Runtime.InteropServices;

public extern static int a(int b); //이름만 써놓은 함수 다른 곳 외부에서 구현 dll(dynamic linked library 동적 링크드 라이브러리)이나 라이브러리 lib(정적 라이브러리) 같은 외부 파일 함수 미리 이름만 적어 둠

[DllImport("C:/Program Files/Microsoft Visual Studio/2022/Community/Common7/IDE/Rebex.Security.dll")]

public extern static byte[] Encrypt(byte[] rgb);

dll파일들 쪼개 놓고 관리해서 수정 용이

**readonly:** 초기화 값만 들고 있는 변수 수정불가

public readonly int sum = 0;

**unsafe:** 포인터 사용 가능

public unsafe int b()

{

자료형\* 변수명 : 포인터 선언

\*변수명 : 포인터가 가리키고 있는 주소의 값

& :주소

int test = 0;

Console.WriteLine((int)&test);

int\* a = &test;

\*a = (int)&a; (\*a는 a가 가리키고 있는 주소의 값에다가 a의 주소를 넣음test가 36을 가지게 됨)

test = (int)&a;

Console.WriteLine(test);

return \*a;

}

주소 36 28

값[]->[28]->[0]

변수명 a test

test의 주소는 28; 28이 가리키는 값은 0;

28 주소를 인트로 형변환해서 출력

int 자료형를 가리키는 포인터 a를 만들었음 a의 주소(&a)는 36이고 a 안에(값에) test의 주소 28을 넣었음

a가 가리키고 있는 주소 안에 들어있는 값 0이 들어있던 위치 에다가 a의 주소 36을 대입

**애트리뷰트**