Chapter08 클래스 심화

01 제네릭

* 클래스 내부에서 자료형에 별칭을 지정하는 기능
* Class Wanted<T>{ }

Wanted<string> wantedString = new Wanted<string>("String");

Wanted<int> wantedInt = new Wanted<int>(52273);

Wanted<double> wantedDouble = new Wanted<double>(52.273);

class TestA<T, U>

{

* }
* 위의 예시는 두 개 이상의 제네릭을 사용하는 클래스이다.

\*where 키워드

- 제네릭의 제한

class TestB<T, U>

where T : class

where U : struct

{

* T는 클래스여야하고 U는 구조체여야한다.

}

class TestC<T, U>

where T : IComparable

where U : IComparable, IDisposable

{

}

* T는 IComparable 또는 IComparable 을 상속 받은 것이여야 한다.
* U는 IComparable 과 IDisposable 또는 이런 것들을 상속 받은 것이여야한다.

\*인덱서 선언 예

namespace IndexerBasic

{

class SquareCalculator

{

public int this[int i]

{

get

{

return i \* i;

}

}

}

class Program

{

static void Main(string[] args)

{

SquareCalculator square = new SquareCalculator();

Console.WriteLine(square[10]);

}

}

}

* Public int this[int i] 이렇게 인덱서를 선언하고 사용할 때는 products[10] i 에 10을 넣은 값을 계산한다.

\*out 키워드

- 값을 여러 개 반환하고자 할 때 사용 대표적인 메서드는 TryParse() 메서드

- int.TryParse( ) 메서드의 기본 형태

Public static bool TryParse{ string s, out int result }

* Out 키워드가 붙은 매개변수 입력할 때

int.TryParse(“52273”, out output);

* TryParse() 메서드는 숫자로 바꿀 수 있는 문자열을 매개변수로 넣으면 true 반환
* 바꿀 수 없는 문자열을 매개변수로 넣으면 false 반환
* 문자열을 숫자로 변환한 결과는 반환 않고 매개변수 int result 에 넣은 변수로 반환
* class Program
* {
* static void Main(string[] args)
* {
* Console.Write("숫자 입력: ");
* int output;
* bool result = int.TryParse(Console.ReadLine(), out output);
* if (result)
* {
* Console.WriteLine("입력한 숫자: " + output);
* }
* else
* {
* Console.WriteLine("숫자를 입력해주세요!");
* }
* }
* }

\*out 키워드를 사용하는 메서드 생성

class Program

{

static void NextPosition(int x, int y, int vx, int vy, out int rx, out int ry)

{

// 다음 위치 = 현재 위치 + 현재 속도

rx = x + vx;

ry = y + vy;

}

static void Main(string[] args)

{

int x = 0;

int y = 0;

int vx = 1;

int vy = 1;

Console.WriteLine("현재 좌표: (" + x + "," + y + ")");

NextPosition(x, y, vx, vy, out x, out y);

Console.WriteLine("다음 좌표: (" + x + "," + y + ")");

}

}

* 구조체
* 간단한 객체 만들 때 사용
* 클래스와 거의 동일한 구문 사용, 복사 형식이 다르고 클래스보다 제한 많음
* 구조체는 상속, 인터페이스 구현 불가능, 클래스보다 안정성 높음
* C#의 기본 자료형은 모두 구조체
* 구조체를 만드는 기본 방법은 클래스 만드는 방법과 같음

\*구조체 선언

Struct Point { public int x; public int y; }

class Program

{

struct Point

{

public int x;

public int y;

}

static void Main(string[] args)

{

Point point;

point.x = 10;

point.y = 10;

Console.WriteLine(point.x);

Console.WriteLine(point.y);

}

}

* 구조체의 생성자
* 매개변수 없는 생성자 선언 불가
* 예를 들면 public Point( ){ } 이런 형태!!!

\*매개 변수 넣어 생성자 만들기

struct PointB

{

public int x;

public int y;

public PointB(int x, int y)

{

this.x = x;

this.y = y;

}

}

* 매개변수 없는 생성자가 자동 정의되어 구조체는 매개변수 없는 생성자 만들 수 없음
* 매개변수 있는 생성자 만들어도, 매개변수 없는 생성자 사용 가능

Point point = new Point( ); 이런식으로 만들 수 있음

* 내부 변수는 자동적으로 해당 자료형의 기본 값으로 초기화!

\*구조체

- 생성자에서는 반드시 모든 멤버 변수 초기화 상태로 만들어줘야 함

- 선언과 동시에 멤버 변수 초기화 할 수 없음

/\* struct PointC

{

public int x;

public int y;

public string testA;

public string testB = "init";

public PointC(int x, int y)

{

this.x = x;

this.y = y;

}

} \*/

* testA 는 생성자에서 초기화를 해줘야함
* testB처럼 선언과 동시에 초기화 불가능

\*구조체에서 클래스 인스턴스를 멤버 변수로 선언할 때

- 반드시 초기화!!!!

struct PointE

{

public int x;

public int y;

public Program program;

public PointE(int x, int y)

{

this.x = x;

this.y = y;

this.program = null;

}

}

* 여기에서는 null 로 초기화 해주었다.
* Class 에서 값 복사를 하고 내용을 바꾸면 기존의 값도 같이 바뀌게 되는데 구조체에서 값 복사를 하고 내용을 바꾸면 두개는 따로 존재하기 때문에 내용이 바뀌지 않는다.

((((((( 추상 클래스 ))))))))

* 추상 메소드를 갖는 클래스
* 추상 메소드 : 실질적인 구현을 갖지 않고 메소드 선언만 있는 경우
* 추상클래스 선언 방법

Abstract class AbsractClass{

Public abstract void MethodA( );

Void MethodB( ){ //.. }

* 구현되지 않고, 단지 외형만을 제공
* 추상 클래스는 객체를 가질 수 없음
* 다른 외부 클래스에서 메소드를 일관성 있게 다루기 위한 방법 제공
* 다른 클래스에 의해 상속 후 사용 가능
* Abstract 수정자는 virtual 수정자의 의미 포함
* 추상 클래스는 파생 클래스에서 구현
* Override 수정자를 사용하여 추상 메소드를 재정의
* 접근 수정자 항상 일치

User System;

Abstract class AbstractClass {

Public abstract void MethodA( );

Public void MethodB( ){

Console.WriteLine(“MethodA( )”);

}

}

Class ImpClass : AbstactClass{

Override public void MethodA( ){

Console.WriteLine(“MethodB( )”);

}

}

Class AbstractClassApp{

Public static void Main(){

ImpClass obj = new ImpClass( );

Obj.MethodA( );

Obj.MethodB( );

}

}

Abstract class Figure{

Public abstract void Area();

Public abstract void Girth();

Public abstract void Draw();

}

* 이렇게 있다면 이 class 를 상속 받는 클래스를 하나 만들고 그리고 override 해서 위에 메소드들을 재정의해서 사용한다.

\*제네릭 메소드

- 제네릭 메소드

=> 형 매개변수를 갖는 메소드

- 제네릭 메소드 정의 예

Void Swap<DataType>(DataType x, DataType y) {

DataType temp = x;

X = y;

Y = temp;

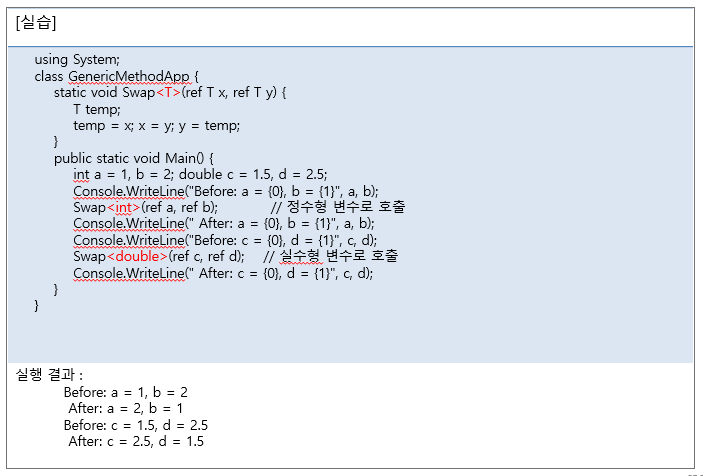
}

* 제네릭 메소드 호출 예

Swap<int>(a,b); // a , b : 정수형

Swap<double>(c,d); //c , d : 실수형

* 형 매개변수의 중첩
* 제네릭 메소드의 형 매개변수의 이름과 제네릭 클래스의 형 매개변수 이름이 같은 경우
* 서로 독립된 형 매개변수의 개념을 가짐
* 제네릭 클래스는 객체 생성 시에 형 매개변수를 전달 받음
* 제네릭 메소드는 호출 시에 유추하여 형 매개변수가 결정됨



* 형 매개 변수의 범위
* 프로그램의 유연성 ⬄ 신뢰성
* 프로그램의 신뢰성을 증진하기 위해 제네릭에 전달 가능한 자료형의 범위를 제한할 필요가 있음
* 제네릭 클래스를 작성시 한정
* Where 키워드를 사용한다.
* <T> where T: S – S 형의 서브클래스형으로 제한한다.

