public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

//생성자란? 클래스를 생성할때 기본으로 자동 실행되는 함수

//생성자는 리턴타입이 없는 함수이다.

//생성자는 클래스명이랑 똑같아야만 한다. 즉 함수명이 클래스명이랑 똑같다.

TestClass t = new TestClass();

//이때 t.n의 값은? 33

//왜냐하면 new 뒤에 기본 생성자로 생성했기 때문이다.

t.n = 1;

t.c = 'B';

//t.f와 t.s는 private라서 직접 접근이 불가능하다.

t.setFandS(0.3f, "world");

//t라는 객체속 n, f, c, s의 결과값은?

//n: 1, c: B, f: 0.3, s: world

TestClass tc = new TestClass(112);

//tc.n의 값은? 112

//왜냐하면 new 뒤에 숫자 인자값을 넣는 생성자로 생성했기 때문이다.

TestClass tc2 = new TestClass('Z', "abcd");

//tc2.n의 값은? 1234

//왜냐하면 new 뒤에 문자와 문자열 인자값을 넣는 생성자로 생성했기 때문이다.

//변수는 총 3개이다. t, tc, tc2

//클래스는 그저 자료형에 불과할뿐이다

ChildClass c = new ChildClass();

//c.s는 접근이 불가능하다. 왜냐면 s는 protected로 선언했기 때문에

//ChildClass내에서만 사용이 가능할뿐, ChildClass를 객체로 만든 외부에서는

//접근이 불가능하다.

//이때 c.n의 값은? 33 왜냐하면 부모클래스인 TestClass에서의 인자값이 없는

//기본 생성자에서 n=33으로 설정되었기 때문이다.

var v1 = 2; //int

var v2 = 'C'; //char

var v3 = 0.5f; //float

var v4 = "good"; //string

//var는 변수의 자료형을 자동으로 알아서 저장한다.

//단, 클래스의 멤버변수 또는 전역변수로는 쓸 수 없다.

//오직 함수 안에서만 쓰인다.

//단점은 자동으로 알아서 처리하기 때문에 메모리가 더 많이 든다.

//에러가 났을때, 명시적으로 자료형을 선언한것보다 디버깅이 어려워진다.

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

//int[] array = new int[5]; 이렇게 선언한다음 각 항목에 직접 값을 넣어도 된다.

array[0] = 3;

array[1] = 5;

//이 배열의 결과값은? 3,5,3,4,5

for(int i=0; i<5; i++)

{

array[i] = 3;

}

//이 배열의 결과값은? 3,3,3,3,3

for (int i = 0; i < 5; i++)

{

array[i] = i%2; //%는 뒤의수로 나눈 나머지

}

//이 배열의 결과값은? 0,1,0,1,0

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

list.Add("사과");

list.Add("귤");

list.Add("포도");

list.Add("딸기");

list.Add("수박");

int listCount = list.Count();

//listCount의 결과값은? 5

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

{

string row = list[i];

if("포도".Equals(row)) //if(row.Equals("포도") 이것과 같다

{

list[i] = "샤인머스캣";

}

}

//list[2]의 결과값은? 샤인머스캣

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

d.Add("20250001", "홍길동");

d.Add("20250002", "짱구");

d.Add("20250003", "맹구");

d.Add("20250004", "강감찬");

string person = d["20250003"];

//person의 결과값은? 맹구

d["20250001"] = "이순신";

//d["20250001"]의 결과값은? 이순신

}

private void button2\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

//버튼1을 클릭하면 label1의 텍스트를 바꾼다.

label1.Text = "버튼1 클릭";

}

private void button2\_Click\_1(object sender, EventArgs e)

{

//버튼2를 클릭하면 팝업창을 띄운다.

MessageBox.Show("버튼2 클릭");

}

}

public class TestClass

{

public int n = 0;

private float f = 0.1f;

public char c = 'A';

protected string s = "hello";

public void setFandS(float f, string s)

{

this.f = f;

this.s = s;

}

//생성자는 리턴타입이 없이 클래스명과 동일한 함수이다.

public TestClass()

{

this.n = 33;

}

//생성자는 여러개가 될 수 있다.

public TestClass(int n)

{

this.n = n;

this.s = "ddd";

}

//생성자는 여러개가 될 수 있다.

public TestClass(char c, string s)

{

this.c = c;

this.s = s;

this.n = 1234;

}

}

public class ChildClass : TestClass

{

public ChildClass()

{

//부모클래스인 TestClass의 기본생성자 TestClass()에 들어있는

//this.n = 33을 기본으로 세팅을 한다. 왜? 상속받았으니까

//그래서 이 생성자안에는 this.n = 33이 생략되어 있는것이나 다름없다.

//부모클래스인 TestClass에서 s는 protected로 선언되었기 때문에

//외부에서는 접근이 불가능하지만, 자식클래스에서는 접근이 가능하다.

this.s = "tttt";

}

}