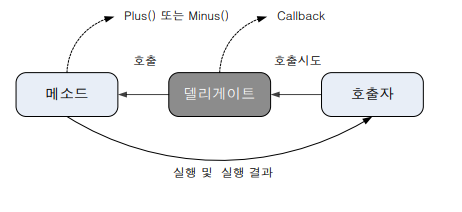
Chapter 11 델리게이터와 람다

01 델리게이트란?

* 콜백이란?
  + 비서처럼 대신 어떤 일을 해 주는 코드, 이 코드가 실행할 세부 코드는 컴파일 시점이 아닌 실행 시점에 부여함
* 델리게이트란?
  + 델리게이트는 대리인, 사절이라는 뜻으로 C#에서의 델리게이트는 메소드에 대한 참조를 말함
  + 델리게이트에 메소드의 주소를 할당한ㄴ 후 델리게이트를 호출하면 델리게이트가 메소드를 호출
* 델리게이트 선언 형식
  + 한정자 delegate 반환형식 델리게이트이름 (매개변수\_목록) ;
* 델리게이트 선언 예
  + Delegate int MyDelegate( int a, int b);
* 델리게이트를 이용하여 콜백을 구현하는 과정

1. 델리게이트를 선언한다.
2. 델리게이트의 인스턴스를 생성한다. 인스턴스를 생성할 때는 델리게이트가 참조할 매소드를 매개 변수로 넘긴다.
3. 델리게이트를 호출한다.



* 델리게이트 선언과 사용 예

Int Plus ( int a, int b ) { return a+b; }

Int Minus (int a, int b ) { return a-b };

MyDelegate Callback;

Callback = new MyDelegate(plus);

Console.WriteLine(Callback(3,4)); //7출력

Callback = new MyDelegate(Minus);

Console.WriteLine(Callback(7,5)); //2 출력

03 일반화 델리게이트

* 델리게이트는 보통의 메소드 뿐 아니라 일반화 메소드도 참조 가능

Delegate int Compare<T> (T a, T b);

Static void BubbleSort<T> (T[ ] DataSet, Campare <T> Comparer

Static int AscendCompare <T> (T a, T b) where T:

IComparable <T>{ return a.CompareTo(b); }

Comparer 라는 이름으로 메소드 사용이 가능하고

이렇게 되어있으면 AscendCompare<T>( ) 메소드를 매개 변수로 넘기면 BubbleSort<T>( )

메소드는 오름차순으로 정렬

04 익명 메소드

* 익명 메소드 : 이름이 없는 메소드
* 두 번 다시 호출할 일이 없는, 다시 말해 1회용 메소드를 선언할 때 이용

델리게이트 인스턴스 = delegate (매개변수\_목록) { //실행하고자 하는 코드 }

//// 선언 형식

////선언 및 호출 예

Public static void Main() { Calculate Calc;

Calc = delegate (int a,int b){ return a+b; }

Console.WriteLine(Calc(3,4));

// 이름을 제외한 메소드의 구현 🡺익명 메소드!

//Calc 을 호출하면 이 코드를 실행

01 델리게이터 관련 용어 소개

* 델리게이터 : 일종의 클래스 선언
* 델리게이터 사용 방법

Delegate void TestDelegate( ); //자료형을 선언하는 방법

TestDelegate testDelegate = <메서드이름, 무명델리게이터, 람다>

TestDelegate 🡺 델리게이터를 자료형으로 변수를 만들고

델리게이터 변수를 초기화 하는 방법이 메서드 이름, 무명 델리게이터, 람다 이다.

02 메서드 이름, 무명 델리게이터, 람다

* 델리게이터 구현
* 델리게이터 : 형식화된 메서드, 특정 형태의 메서드를 매개변수로 전달하라는 의미
* Sort( ) 함수에도 Comparison 이라는 것이 매개변수로 쓰일 수 있는데 여기에서 Comparison 은 델리게이터 이다. 그러므로 sort( ) 함수에 매개 변수로 메서드를 보낼 수 있다.

class Product

{

public string Name { get; set; }

public int Price { get; set; }

}

static void Main(string[] args)

{

// 리스트를 생성합니다.

List<Product> products = new List<Product>()

{

new Product() { Name = "감자", Price = 500 },

new Product() { Name = "사과", Price = 700 },

new Product() { Name = "고구마", Price = 400 },

new Product() { Name = "배추", Price = 600 },

new Product() { Name = "상추", Price = 300 }

};

// 정렬합니다.

products.Sort(SortWithPrice);

// 출력합니다.

foreach (var item in products)

{

Console.WriteLine(item.Name + " : " + item.Price);

}

}

static int SortWithPrice(Product a, Product b)

{

return a.Price.CompareTo(b.Price);

* }
* 이런식으로!!
* 무명 델리게이터 기본
  + 무명 델리게이터 형태

Delegate (<매개변수>,<매개변수>) { /\* 코드 \*/ return /\* 반환 \*/; }

// 정렬합니다.

products.Sort(delegate (Product a, Product b)

{

return a.Price.CompareTo(b.Price);

});

* 이런식으로 사용한다.
* 람다 기본
  + 람다 : 델리게이터를 쉽게 이용 할 수 있게 한 것
  + 람다의 형태

(<매개변수>,<매개변수>) => { /\* 코드 \*/ return /\* 반환 \*/ }

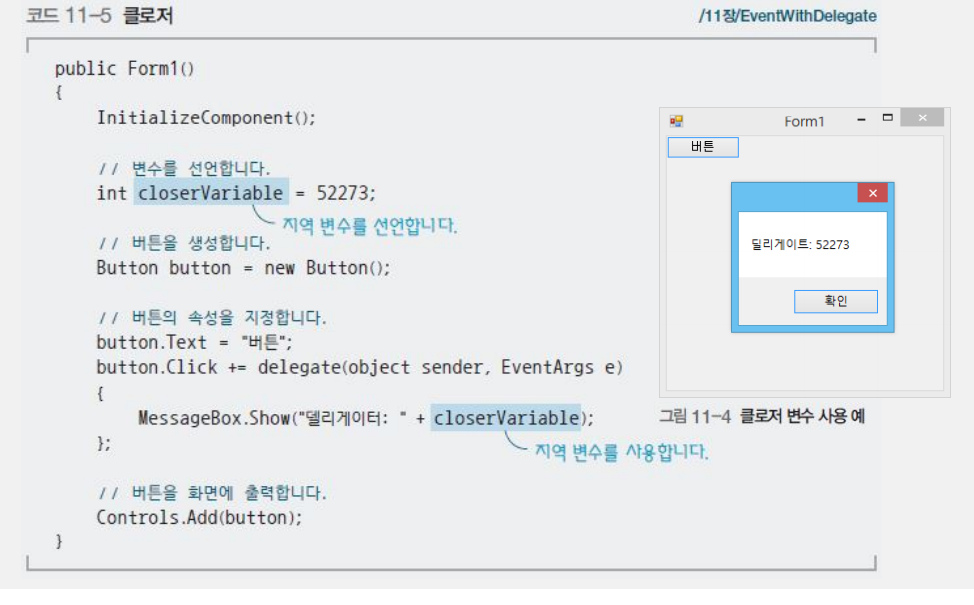
(a,b) => /\* 반환 \*/

products.Sort((a, b) =>

{

return a.Price.CompareTo(b.Price);

});

* 클로저 : 지역 변수가 메서드가 끝나도 사라지지 않고 남는 현상
* 

03 델리게이터 선언

* 델리게이터 형식 지정
  + 이름 있는 델리게이터 : 특정한 형식을 가진 메서드를 자료형으로 선언하는 방법

[ 접근 제한자 ] delegate [ 반환형 ] [ 델리게이터 이름 ]([ 매개변수 ]);

* 클래스 선언하는 위치와 같은 위치라면 어디든지 선언 가능
* 클래스를 선언하는 파일과 같기만 하다면 델리게이터 생성을 클래스 외부에도 할 수 있고 내부에도 할 수 있다.
* 
* 델리게이터 초기화 방법

Public delegate void TestDelegate( );

TestDelegate delegateA = TestMethod; 🡺 메서드 이름을 사용한 초기화 방법

TestDelegate delegateB = delegate( ) { }; 🡺 무명 델리게이터를 사용한 초기화 방법

TestDelegate delegateC = ( ) => { }; 🡺람다를 사용한 초기화 방법

delegateA( );

delegateB( );

delegateC( );

} 🡺 델리게이터는 일반 메서드처럼 호출할 수 있다.

Static void TestMethod( ){ }

* 델리게이터 활용
* 대표적인 방법 : 콜백 메서드
  + 매개변수로 전달하는 메서드
* 콜백 메서드의 형태

Public delegate void CustomDelegate( ); 🡺델리게이터를 선언한다.

Public void Method(CustomDelegate customDelegate) { customDelegate( ); }

🡺매개변수로 전달된 델리게이터(메서드)를 호출한다.

class Student

{

public string Name { get; set; }

public double Score { get; set; }

public Student(string name, double score)

{

this.Name = name;

this.Score = score;

}

public override string ToString()

{

return this.Name + " : " + this.Score;

}

}

class Students

{

private List<Student> listOfStudent = new List<Student>();

public delegate void PrintProcess(Student list);

public void Add(Student student)

{

listOfStudent.Add(student);

}

public void Print()

{

Print((student) =>

{

Console.WriteLine(student);

});

}

public void Print(PrintProcess process)

{

foreach (var item in listOfStudent)

{

process(item);

}

}

}

class Program

{

static void Main(string[] args)

{

Students students = new Students();

students.Add(new Student("윤인성", 4.2));

students.Add(new Student("연하진", 4.4));

students.Print();

students.Print((student) =>

{

Console.WriteLine();

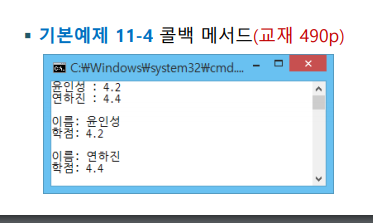
Console.WriteLine("이름: " + student.Name);

Console.WriteLine("학점: " + student.Score);

});

}

}



04 델리게이터 연산

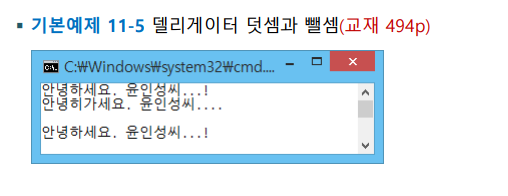
//버튼을 생성한다.

* Button button = new Button( );

//버튼의 속성을 지정한다.

* button.Text = “버튼”;
* button.Click += button\_Click; 🡺메서드를 더했다.

// 버튼을 화면에 출력했다.

* Controls.Add(button);
* 델리게이터 변수에 적용하는 연산자
* + : 델리게이터를 더한다.
* - : 델리게이터를 뺍니다.
* using System;
* using System.Collections.Generic;
* using System.Linq;
* using System.Text;
* using System.Threading.Tasks;
* namespace DelegateOperator
* {
* class Program
* {
* public delegate void SendString(string message);
* static void Main(string[] args)
* {
* SendString sayHello, sayGoodbye, multiDelegate;
* sayHello = Hello;
* sayGoodbye = GoodBye;
* multiDelegate = sayHello + sayGoodbye;
* multiDelegate("윤인성");
* Console.WriteLine();
* multiDelegate -= sayGoodbye;
* multiDelegate("윤인성");
* }
* public static void Hello(string message)
* {
* Console.WriteLine("안녕하세요. " + message + "씨...!");
* }
* public static void GoodBye(string message)
* {
* Console.WriteLine("안녕히가세요. " + message + "씨....");
* }
* }
* }
* 
* -------스레드 실행
* class Program
* {
* static void Main(string[] args)
* {
* Thread threadA = new Thread(() => {
* for (int i = 0; i < 1000; i++)
* {
* Console.Write("A");
* }
* });
* Thread threadB = new Thread(() => {
* for (int i = 0; i < 1000; i++)
* {
* Console.Write("B");
* }
* });
* Thread threadC = new Thread(() =>
* {
* for (int i = 0; i < 1000; i++)
* {
* Console.Write("C");
* }
* });
* threadA.Start();
* threadB.Start();
* threadC.Start();
* }
* }