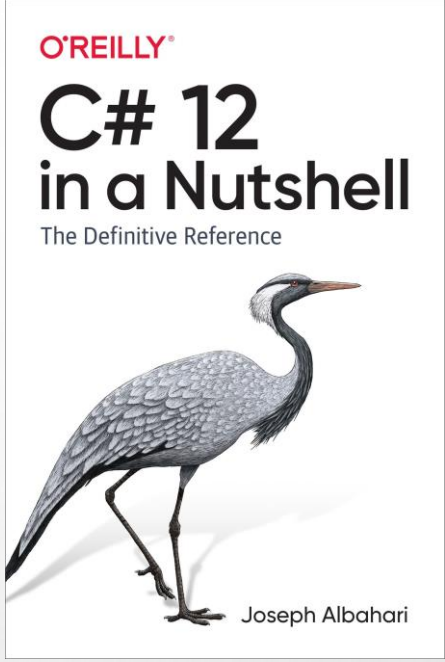
CHAPTER 4 - DELEGATES

- Delegates
- Writing Plug-in Methods with Delegates
- Instance and Static Method Targets
- Multicast Delegates
- Generic Delegate Types
- The Func and Action Delegates
- Delegates Versus Interfaces
- Delegate Compatibility



Delegates

اگه بخواییم یک متد رو به عنوان ورودی به یک متد دیگه بفرستیم از delegate استفاده میکنیم.

```
delegate int Transformer(int x);
```

در اصل delegate یک امضا از اون متدی که قراره ارسال بشه ایجاد میکنه. اینو ببینید:

اینجا transformer با هر متدی که یک ورودی از جنس int داشته باشه و خروجی اونم int باشه سازگاری داره.

خب این یعنی چی؟ توی این مثال یه متد به اسم square داریم که شرایط بالا رو داره.

پس میتونیم آبجکت delegate رو با این متد پر کنیم و هرجا خواستیم اونو صدا بزنیم.

```
1 reference | - changes | -authors, -changes
int Square(int x) => x * x;

Transformer transformer = Square;

Console.WriteLine(transformer(5)); //25
```

Writing Plug-in Methods with Delegates

یک متغیر delegate موقع runtime متد رو به خودش assign میکنه. این بهمون کمک میکنه تا متدهای plug-in بنویسیم.

```
int[] values = { 1, 2, 3 };
Transform(values, Square);      // Hook in the Square method

foreach (int i in values)
    Console.Write(i + "  ");      // 1   4   9

1 reference | - changes | -authors, -changes
void Transform(int[] values, Transformer t)
{
    for (int i = 0; i < values.Length; i++)
        values[i] = t(values[i]);
}

1 reference | - changes | -authors, -changes
int Square(int x) => x * x;
0 references | - changes | -authors, -changes
int Cube(int x) => x * x * x;
```

توی این مثال یه متد داریم به اسم transform که تغییر مد نظر خودشو روی همه خونه های آرایه اعمال میکنه. این متد یه پارامتر ورودی از جنس Delegate داره که میتونید برای مشخص کردن پلاکین transform ازش استفاده کنید.

Writing Plug-in Methods with Delegates

حالا به راحتی فقط با تغییر square به cube نوع transform رو تغییر بدید.

```
values = [1, 2, 3];
Transform(values, Cube);      // Hook in the Cube method

foreach (int i in values)
    Console.Write(i + "  ");      // 1   8   27
```

متد transform یک higher-order function هست، چراکه یک متدی هست که خودش یک متد دیگه رو داره به عنوان ورودی دریافت میکنه. متدی که خروجیش یک نوع delegate باشه هم با همین عنوان شناخته میشه.

Our Transform method is a higher-order function because it’s a function that takes a function as an argument. (A method that returns a delegate would also be a higher-order function.)

Instance and Static Method Targets

A delegate's target method can be a local, static, or instance method.

خلاصه اینکه متدی که به متد delegate معرفی میکنیم میتونه استاتیک

باشه یا یه متد توی یه کلاس دیگه باشه

```
2 references | 0 changes | 0 authors, 0 changes
internal class SampleClass
{
    1 reference | 0 changes | 0 authors, 0 changes
    public static int Square(int x) => x * x;
    1 reference | 0 changes | 0 authors, 0 changes
    public int Cube(int x) => x * x * x;
}
```

```
Transformer t1 = SampleClass.Square;
Transformer t2 = new SampleClass().Cube;

Console.WriteLine(t1(2) + "     " + t2(2));  // 4   8
Console.WriteLine(t1.Target);  // null
Console.WriteLine(t2.Target);  // sampleClass
```

Multicast Delegates

همه آبجکت های delegate قابلیت multicast رو دارند، یعنی یک متد delegate میتونه چندتا target-method داشته باشه. عملگرهای + و =+ میتونند این امکان رو برای ما ایجاد کنند:

```
Transformer t = SampleClass.Square;
t += new SampleClass().Cube;
Console.WriteLine("Multicast Delegates: " + t(2));
Console.WriteLine(string.Join(", ", t.GetInvocationList().Select(q => q.Method.ToString())));  // Int32 Square(Int32), Int32 Cube(Int32)
```

عملگرهای – و =– هم میتوانند یک متد را از لیست حذف کنند که البته باید حواسمون باشه ممکنه null بشه لیست متدها:

```
t -= SampleClass.Square;
Console.WriteLine(string.Join(", ", t.GetInvocationList().Select(q => q.Method.ToString())));  // Int32 Cube(Int32)
```

Multicast Delegates

عملگرهای + و =+ میتونند با متد delegate که مقدارش null هست هم کار کنند و به مشکلی برنخورند.

همچنین عملگرهای – و –= ممکن است باعث null شدن متغیر delegate شود.

```
Transformer transformer = null;
transformer += SampleClass.Square;
Console.WriteLine(string.Join(", ", transformer.GetInvocationList().Select(q => q.Method.ToString())));  // Int32 Square(Int32)
transformer -= SampleClass.Square;
Console.WriteLine(transformer is null);  // true
```

Multicast Delegates

Delegates are immutable, so when you call += or -=, you're in fact creating a new delegate instance and assigning it to the existing variable.

خلاصه اینکه وقتی از این عملگرهای =+ یا =- استفاده میکنیم، ازونجایی که این متدها immutable هستند، در اصل یک نمونه جدید از اون متد delegate ساخته میشه و به متغیر قبلی assign میشه.

Multicast Delegates

اگه یه multicast delegate خروجی داشته باشه وقتی فراخوانی بشه فقط مقدار آخرین متدی که بهش assign شده برمیگرده، البته که عملیات داخل همه متدها به درستی انجام میشه ولی خروجی آخرین متد فقط برگردانده میشود و بقیه خروجی ها از بین میروند.

برای اینکه این مشکل به وجود نیاد، در اکثر مواقع از multicast delegate ها فقط برای مواردی که خروجی ندارند یا به اصطلاح nonvoid نیستند استفاده میشه.

Multicast Delegates

همه delegate ها از کلاس System.MulticastDelegate مشتق میشوند که خودش از System.Delegate ارث میبره. و در اصل وقتی از عملگرهای +، =+، – و یا =– استفاده میکنیم تابعهای استاتیک remove یا combine از کلاس delegate فراخوانی میشند:

```
Transformer tt = null;
var testDelegate = Delegate.Combine(tt, SampleClass.Square);
testDelegate = Delegate.Combine(testDelegate, new SampleClass().Cube);
testDelegate = Delegate.Remove(testDelegate, new SampleClass().Cube);
```

Generic Delegate Types

یک delegate میتونه به صورت جنریک تعریف بشه و type-parameter داشته باشه.

```
public delegate T Transformer<T>(T arg);
```

و اینجوری هم میشه ازش استفاده کرد.

```
int[] intValues = [1, 2, 3];
string[] stringValues = ["Mohammad", "Ali"];

Util.Transform(intValues, Square);      // Hook in Square
foreach (int i in intValues)
    Console.Write(i + "  ");           // 1   4   9

1 reference | - changes | -authors, -changes
string sayHello(string s) => $"Hello, {s}";
Util.Transform(stringValues, sayHello);      // Hook in Square
foreach (string i in stringValues)
    Console.Write(i + "  ");           // Hello, Mohammad  Hello, Ali
```

The Func and Action Delegates

با استفاده از generic delegate ها میشه یه سری انواع delegate ساده داشت که با هر نوع خروجی و تعداد مشخصی ورودی کار کنند. این delegate ها عبارتند از Func ها و Action ها که در فضای نام system قرار دارند.

ا

```
delegate TResult Func <out TResult>                ();
delegate TResult Func <in T, out TResult>          (T arg);
delegate TResult Func <in T1, in T2, out TResult>  (T1 arg1, T2 arg2);
... and so on, up to T16

delegate void Action                 ();
delegate void Action <in T>          (T arg);
delegate void Action <in T1, in T2>  (T1 arg1, T2 arg2);
... and so on, up to T16
```

The Func and Action Delegates

مثلا توی مثال قبل ما میتونیم transformer رو که خودمون ایجاد کردیم با Func به شکل زیر بازنویسی کنیم:

ت

```
2 references | 0 changes | 0 authors, 0 changes
public static void TransformWithFunc<T>(T[] values, Func<T, T> transformer)
{
    for (int i = 0; i < values.Length; i++)
        values[i] = transformer(values[i]);
}
```

نها حالاتی که نمیشه با این دو delegate پیاده سازی کرد delegate های دارای پارامترهای ref/ out و یا پارامترهای پوینتر هست.

Delegates Versus Interfaces

مشکلی که با delegate حل بشه رو میشه با interface هم حل کنیم؛ بیاییم مثال قبلی رو با اینترفیس پیاده سازی کنیم:

```
1 reference | 0 changes | 0 authors, 0 changes
public interface ITransformer
{
    1 reference | 0 changes | 0 authors, 0 changes
    int Transform(int x);
}
```

```
0 references | 0 changes | 0 authors, 0 changes
class Squarer : ITransformer
{
    2 references | 0 changes | 0 authors, 0 changes
    public int Transform(int x) => x * x;
}
```

```
0 references | 0 changes | 0 authors, 0 changes
public static void TransformAll(int[] values, ITransformer t)
{
    for (int i = 0; i < values.Length; i++)
        values[i] = t.Transform(values[i]);
}
```

```
Util.TransformAll(values, new Squarer());
foreach (int i in values)
    Console.WriteLine(i);
```

Delegates Versus Interfaces

استفاده از طراحی delegate بهتر از استفاده از طراحی اینترفیس است اگر یکی یا چندتا از شرایط زیر برقرار باشه:

- اینترفیس فقط یه متد داشته باشه.
- نیاز به Multicast داشته باشیم.
- subscriber ها نیاز داشته باشند که اینترفیس رو چندین بار پیاده سازی کنند.

در Itransformer ما نیازی به Multicast نداشتیم، اگرچه اینترفیس فقط یه متد داشت و همه subscriberها نیاز داشتند تا این اینترفیس رو برای خودشون پیاده سازی کنند مثل square و cube. با طراحی اینترفیس ما مجبوریم به ازای هر نوع transform یک نوع جدید بسازیم و این خیلی پرهزینه تر از استفاده از delegate هستش.

Delegate Compatibility (Type compatibility)

```
delegate void D1();
delegate void D2();
```

همه delegate ها باهم ناسازگارند حتی اگه از نظر ساختاری مثل هم باشند:

```
D1 d1 = Method1;
D2 d2 = d1;                          // Compile-time error
```

(local variable) D1 d1

CS0029: Cannot implicitly convert type 'CSharp12Nutshell.Chapter04.Delegates.D1' to 'CSharp12Nutshell.Chapter04.Delegates.D2'

البته این ساختار میتونه اتفاق بیفته:

```
D2 d2 = new D2(d1);
```

Delegate Compatibility (Type compatibility)

delegate ها اگر target-method هاشون یکی باشه باهم برابر هستند. همینطور Multicast delegatesها هم اگه متدهای هر دوتاشون به یک ترتیب باشه باهم برابرند

```
Transformer transformer1 = Square;
Transformer transformer2 = Square;
Transformer transformer3 = Cube;
Transformer transformer4 = transformer1 + Cube;
Transformer transformer5 = transformer2 + Cube;

Console.WriteLine(transformer1 == transformer2); // True
Console.WriteLine(transformer1 == transformer3); // False
Console.WriteLine(transformer4 == transformer5); // False
```

Delegate Compatibility (Parameter compatibility)

مثل یه متد معمولی که پارامترهای ورودی میتونند upcast بشند اینجا هم این امکان وجود داره و کد زیر بدون مشکل کار میکنه:

```
delegate void StringAction(string s);
```

```
StringAction sa = new StringAction(ActOnObject);
sa("hello");

1 reference | - changes | -authors, -changes
void ActOnObject(object o) => Console.WriteLine(o);   // hello
```

Delegate Compatibility (Return type compatibility)

موضوع اسلاید قبل در مورد خروجی توابع delegate هم صادق هستش.

```
delegate object ObjectRetriever();
```

```
ObjectRetriever o = new ObjectRetriever(RetrieveString);
object result = o();
Console.WriteLine(result);      // hello

1 reference | - changes | -authors, -changes
string RetrieveString() => "hello";
```