Contents

[Delegate 2](#_Toc54965367)

[**فراخوانی Delegate** 4](#_Toc54965368)

[**ارسال Delegate به عنوان یک پارامتر** 4](#_Toc54965369)

[**Multicast Delegate** 5](#_Toc54965370)

[آشنایی با Func در زبان سی شارپ 6](#_Toc54965371)

[**Func** 6](#_Toc54965372)

[**به کار بردن Func به همراه متد های بی نام** 8](#_Toc54965373)

[آشنایی با Action در سی شارپ 9](#_Toc54965374)

[**Action** 9](#_Toc54965375)

[**مزایای استفاده از Action و Func** 10](#_Toc54965376)

[معرفی Delegate : 11](#_Toc54965377)

[Use Delegate In FUNCTION: 11](#_Toc54965378)

[Add Method To DELEGATE: 12](#_Toc54965379)

[Remove Method From Delegate : 13](#_Toc54965380)

[Delegate By Return TYPE: 13](#_Toc54965381)

[Use Delegate In Input METHOD: 14](#_Toc54965382)

[Anonymous METHOD: 15](#_Toc54965383)

[Example 1: 15](#_Toc54965384)

[Example 2: 16](#_Toc54965385)

[نکته : 16](#_Toc54965386)

[Example 3: 17](#_Toc54965387)

[Example 4: 17](#_Toc54965388)

[Generic DELEGATE: 18](#_Toc54965389)

[FUNC: 19](#_Toc54965390)

[Just OUT: 19](#_Toc54965391)

[ACTION: 19](#_Toc54965392)

[PREDICATE: 20](#_Toc54965393)

[کاربرد Delegate : 21](#_Toc54965394)

[فرق FirstOrDefault & SingleOrDefault : 22](#_Toc54965395)

[Where (x, Y): 22](#_Toc54965396)

[پیاده سازی First Or Default با دستور Where : 23](#_Toc54965397)

آموزش Delegate

در این بخش میخواهیم با Delegate ها آشنا شویم :

در زبان سی شارپ یک متد میتواند شامل یک و یا چندین پارامتر با نوع های مختلف باشد. اما آیا میتوان از یک متد به عنوان پارامتر برای متدی دیگر استفاده کرد. با استفاده از Delegate میتوان این کار را انجام داد.

Delegate

آیا می توان از یک تابع به عنوان پارامتر تابعی دیگر استفاده کرد؟ زبان سی شارپ چطور توابع callback را مدیریت می‌کند؟ جواب این سوال را می توان در delegate  ها پیدا کرد یک delegate همانند یک اشاره گر به توابع مورد نظر اشاره می کند. یک delegate  ارجاعی از متد مورد نظر را در خود نگهداری میکند. همه جایگزین ها به شکل ضمنی از کلاس System.Delegate مشتق می‌شوند.

یک delegate می‌تواند با استفاده از کلمه کلیدی delegate  تعریف شود. قاعده نوشتاری delegate  در  زیر نشان داده شده است

|  |
| --- |
| **<access modifier> delegate <return type> <delegate\_name>(<parameters>)** |

نمونه مثال زیر یک delegate با نام Print را تعریف می کند :

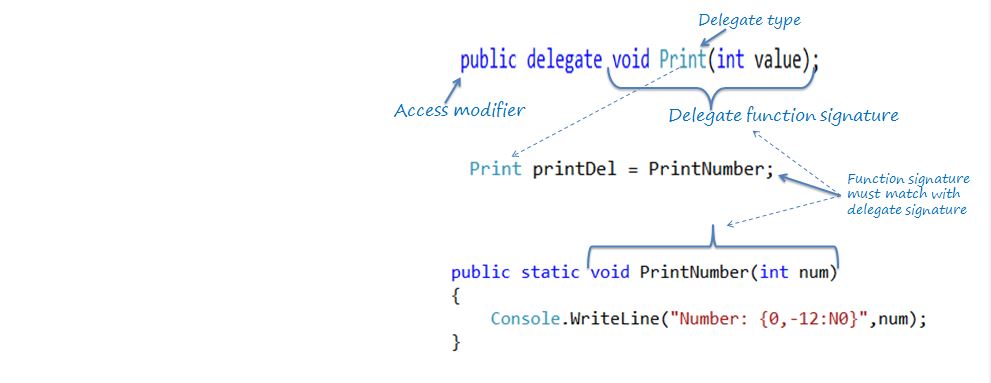
public delegate void Print(int value);

از delegate تعریف شده در مثال بالا می توان برای اشاره به هر متدی که دارای مقدار بازگشتی و پارامترهای یکسان با آن باشد، استفاده کرد:



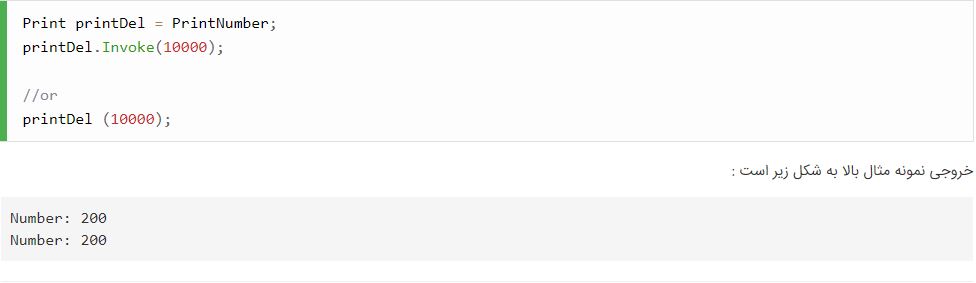
در نمونه مثال بالا یک delegate با نام Print را تعریف کرده ایم که پارامتری با نوع داده ای int را پذیرفته و مقدار بازگشتی آن void است. در متد ()Main متغیری از نوع Print تعریف و با مقدار PrintNumber مقدار دهی شده است.

اکنون فراخوانی delegate مربوطه متد PrintNumber را فراخوانی خواهد کرد. به همین طریق اگر delegate مورد نظر با متد PrintMoney مقدار دهی شود، در هنگام فراخوانی آن متد را فراخوانی خواهد کرد. در شکل زیر ساختار یک delegate نشان داده شده است :



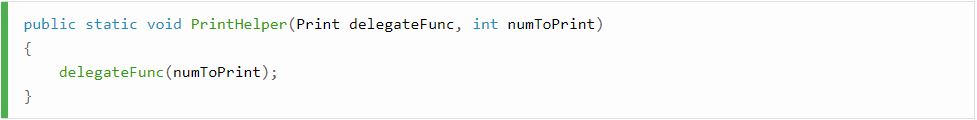
## **فراخوانی Delegate**

یک delegate میتواند همانند یک متد فراخوانی شود و این به خاطر این است که ارجاعی از یک متد را در خود نگه میدارد. فراخوانی یک delegate باعث فراخوانی متدی خواهد شد که delegate مورد نظر به آن ارجاع کرده است. یک delegate میتواند به دو روش فراخوانی شود : با استفاده از عملگر () یا با استفاده از متد ()Invoke . نمونه مثال زیر به خوبی این موضوع را نشان داده است :



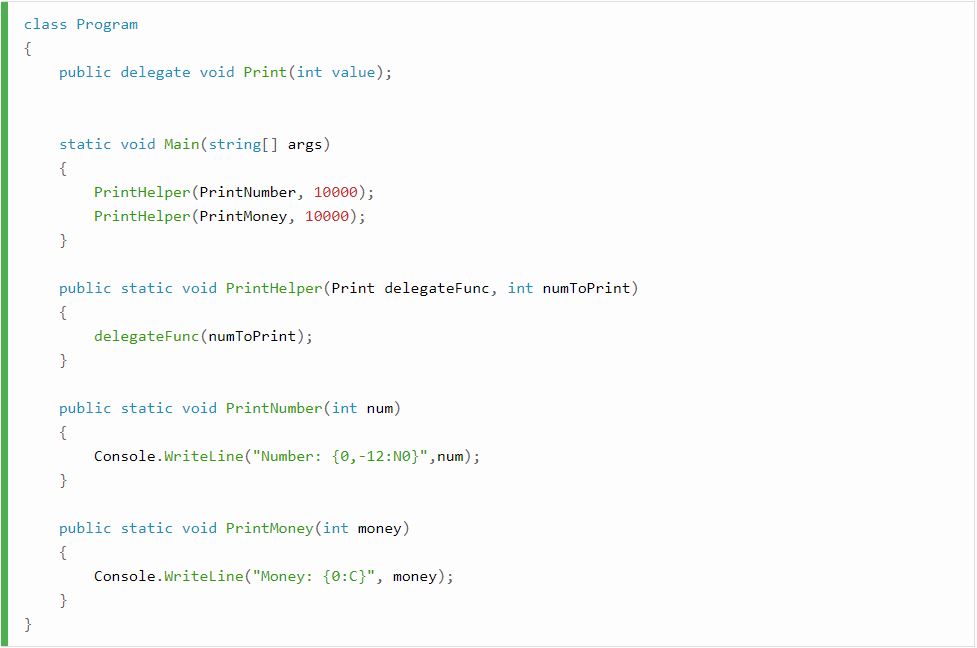
## **ارسال Delegate به عنوان یک پارامتر**

یک متد می تواند پارامتری از نوع delegate را داشته باشد و می تواند در بدنه خود آن را فراخوانی کند :



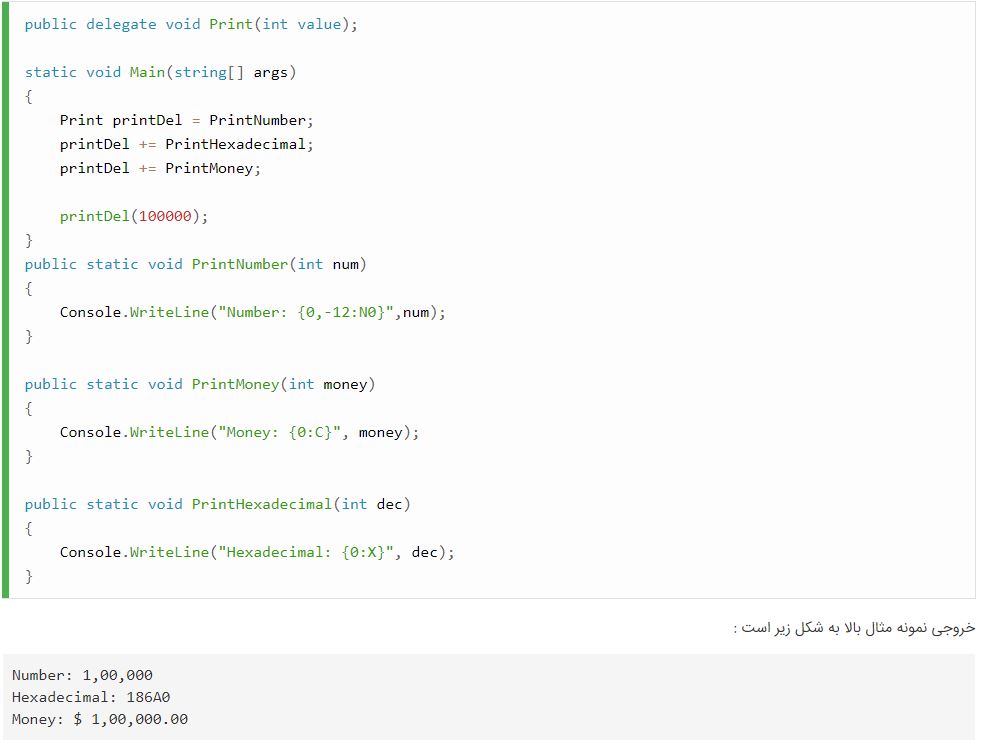
Delegate.Invoke(numToPrint);

بریم یه مثال از همین موضوع کار کنیم:

در نمونه مثال بالا متد  PrintHelper  دارای یک پارامتر delegate بوده و آن را همچون یک متد فراخوانی کرده است. نمونه مثال زیر  نشان می‌دهد که چطور می‌توان از متد PrintHelper استفاده نمود :

## **Multicast Delegate**

یک delegate می تواند به چندین متد اشاره داشته باشد. به اینگونه delegate ها ،  multicast  گفته میشود. عملگر جمع(+) یک متد را به یک delegate اضافه کرده و عملگر تفریق(-) یک متد را از آن حذف می‌کند :



همه ی متد های بالا از جنس delegate هستند یعنی امضای آنها یکسان است.

# آشنایی با Func در زبان سی شارپ

## **Func**

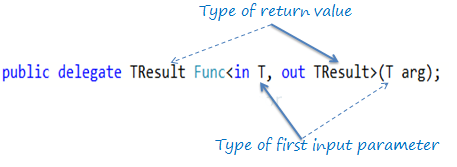
در فصل های قبل مطالبی را در مورد delegate ها آموختیم. یک delegate می تواند به شکلی که در زیر آمده است تعریف شود :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18** | **public delegate int SomeOperation(int i, int j);**  **class Program**  **{**  **static int Sum(int x, int y)**  **{**  **return x + y;**  **}**  **static void Main(string[] args)**  **{**  **SomeOperation add = Sum;**  **int result = add(10, 10);**  **Console.WriteLine(result);**  **}**  **}** | |

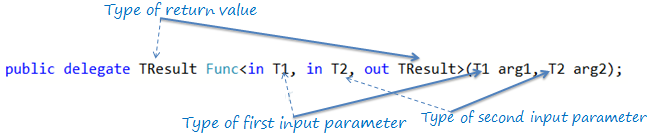
سی شارپ نسخه ی ۳ به بعد شامل یک delegate درونی است که جنریک بوده و Func نام دارد. بنابراین دیگر نیاز نیست به شکلی که در بالا آمده است، delegate دلخواه خود را تعریف کنید. Func در فضای نامی System قرار دارد و می تواند به تعداد صفر و یا بیشتر پارامتر ورودی داشته باشد. Func همچنین دارای یک پارامتر outبرای برگرداندن نتیجه است. برای مثال یک Func که یک پارامتر را به عنوان ورودی دریافت و پارامتری را برای خروجی داشته باشد، به شکل زیر در فضای نامی System قرار دارد :

|  |
| --- |
| **namespace System**  **{**  **public delegate TResult Func<in T, out TResult>(T arg);**  **}** |

آخرین پارامتر در براکت های <> به عنوان نوع بازگشتی در نظر گرفته می شود. دیگر پارامترها پارامتر ورودی محسوب می شوند :



یک Func با دو پارامتر ورودی و یک پارامتر خروجی به شکلی است که در زیر آمده است :



برای مثالی که در ابتدای فصل آمده بود، می‌توان با استفاده از دلیگیت Func بازنویسی را به شکل زیر انجام داد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | **class Program**  **{**  **static int Sum(int x, int y)**  **{**  **return x + y;**  **}**  **static void Main(string[] args)**  **{**  **Func<int,int, int> add = Sum;**  **int result = add(10, 10);**  **Console.WriteLine(result);**  **}**  **}** | |

می‌توانید هر متدی را که دو پارامتر از نوع int داشته باشد و مقداری از نوع int را بازگرداند به Func بالا انتساب داد.

## **به کار بردن Func به همراه متد های بی نام**

میتوان یک متد بی نام را به Func انتساب داد. این کار با استفاده از کلمه کلیدی delegate انجام می پذیرد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5** | **Func<int> getRandomNumber = delegate()**  **{**  **Random rnd = new Random();**  **return rnd.Next(1, 100);**  **};** | |

میتوان در انتساب به یک Func از عبارت های لمبدا استفاده نمود.مثال زیر این مورد را به خوبی نشان می‌دهد :

|  |
| --- |
| **Func<int> getRandomNumber = () => new Random().Next(1, 100);** |

//Or

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1** | **Func<int, int, int> Sum = (x, y) => x + y;** | |

دو تا پارامتر اولی ورودی میباشد و پارامتر سومی خروجی میباشد.

# آشنایی با Action در سی شارپ

Action در سی شارپ یک نوع delegate محسوب میشود و عملکرد مشابهی با دلیگت Func دارد. تفاوت اصلی این دو در این است که از Action به همراه متدهایی استفاده میشود که نوع بازگشتی آنها void باشد. در این فصل با دلیگیت Func در سی شارپ آشنا میشویم.

## **Action**

Action یک نوع delegate محسوب می‌شود که در فضای نامی System قرار دارد.  عملکرد Action شبیه Func بوده و تفاوت آنها در این است که Action مقداری را بر نمی گرداند. به عبارت دیگر یک Action می‌تواند برای متد های که دارای مقدار بازگشتی void هستند، مورد استفاده قرار گیرد. برای مثال دلیگیت Print زیر یک مقدار صحیح را چاپ میکند :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | **public delegate void Print(int val);**  **static void ConsolePrint(int i)**  **{**  **Console.WriteLine(i);**  **}**  **static void Main(string[] args)**  **{**  **Print prnt = ConsolePrint;**  **Prnt(10);**  **}** | |

به جای تعریف دلیگت Print در بالا می توان مثال را به شکل زیر بازنویسی کرد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **static void ConsolePrint(int i)**  **{**  **Console.WriteLine(i);**  **}**  **static void Main(string[] args)**  **{**  **Action<int> printActionDel = ConsolePrint;**  **printActionDel(10);**  **}** | |

میتوان یک دلیگیت Action را با استفاده از کلمه کلیدی new و یا با استفاده از یک متد مقداردهی اولیه کرد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1** | **Action<int> printActionDel = ConsolePrint;** | |

//Or

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1** | **Action<int> printActionDel = new Action<int>(ConsolePrint);** | |

میتوان از متدهای بی نام همراه دلیگیت Action استفاده نمود :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** | **static void Main(string[] args)**  **{**  **Action<int> printActionDel = delegate(int i)**  **{**  **Console.WriteLine(i);**  **};**  **printActionDel(10);**  **}** | |

میتوان به همراه دلیگیت Action از عبارتهای لمبدا نیز استفاده نمود :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7** | **static void Main(string[] args)**  **{**  **Action<int> printActionDel = i => Console.WriteLine(i);**    **printActionDel(10);**  **}** | |

## **مزایای استفاده از Action و Func**

مزایای استفاده از دلیگیت های Action و Func به شرح زیر است :

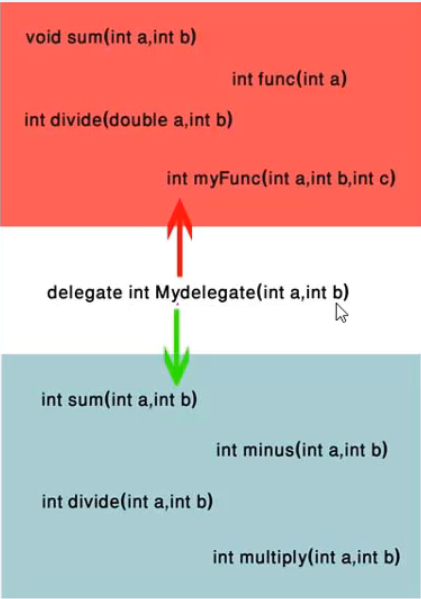
* تعریف  سریع و آسان یک delegate
* کوتاه کردن کد نویسی

شروع دوره جدید Delegate در سایت Top Learn

# معرفی Delegate :

Delegate ها اشاره گر به متد ها هستند که بسیار کاربرد دارد. ما از Delegate ها در Lambda Expression ها زیاد استفاده کرده ایم .

ابتدا میخواهیم یک دید کلی برای Delegate داشته باشیم . تصویر زیر را مشاهده کنید :



تصویر وسط یک Delegate میباشد که با کلمه کلیدی delegate شروع شده است که خروجی آن هم int میباشد و بعدا اسم Delegate پیاده سازی میشود و در مرحله آخر ورودی های Delegate تعیین میشود .

در پایین تصویر امضای متد ها با Delegate برابر است لذا به راحتی میتوان گفت Delegate میتواند به آن متد اشاره کند ولی امضای متد های بالای تصویر به Delegate نمیخورد و نمیتوان متد های بالای عکس را به Delegate نسبت داد .

# Use Delegate In FUNCTION:

میخواهیم در این بخش delegate را داخل فانکشن استفاده کنیم :

ابتدا یک متد تعریف میکنم که وظیفه Sum را برعهده داشته باشد . این متد نوع بازگشتی آن void میباشد و عدد به عنوان ورودی دریافت میکند :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4** | **static void Sum(int a, int b)**  **{**  **Console.WriteLine(a + b);**  **}** | |

حالا بریم یک Delegate تعریف کنیم ، ساختار Delegate به صورت زیر میباشد :

|  |
| --- |
| **<access modifier> delegate <return type> <delegate\_name>(<parameters>)** |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1** | **public delegate void DSum(int num1, int num2);** | |

حالا بریم بررسی کنیم که چگونه متد Sum که در بالا ساختیم به این Delegate نسبت بدهیم :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | **class Program**  **{**  **public delegate void DSum(int num1, int num2);**  **static void Main(string[] args)**  **{**  **DSum sum = new DSum(Sum);**  **sum(10, 20);**  **Console.ReadKey();**  **}**  **static void Sum(int a, int b)**  **{**  **Console.WriteLine(a + b);**  **}**  **}** | |

دقت کنید داخل متد Main یک نمونه جدید از Delegate ساختم که متد Delegate به محض نمونه سازی به من میگه یک ورودی به من بده که نوع بازگشتی آن void باشه و 2 تا عدد به عنوان ورودی دریافت کند دقیقا همین ویژگی را متد Sum دارد .

در اصل من در خط 8 کد بالا خود متد Sum را صدا زده ام و مقدار های 10 و 20 را به آن نسیت داده ام.

# Add Method To DELEGATE:

خوب حالا میخواهم متد هایی که امضای آنها با Delegate برابر است را به Delegate اضافه کنم یعنی وقتی Delegate اجرا شد 2 تا متد را صدا بزند :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21** | **class Program**  **{**  **public delegate void DSum(int num1, int num2);**  **static void Main(string[] args)**  **{**  **DSum sum = new DSum(Sum);**  **sum += Minus;**  **sum(10, 20);**  **Console.ReadKey();**  **}**  **static void Sum(int a, int b)**  **{**  **Console.WriteLine(a + b);**  **}**  **static void Minus(int a, int b)**  **{**  **Console.WriteLine(a - b);**  **}**  **}** | |

با توجه به کد بالا به راحتی به delegate sum توانستم متد Minus را با علامتگر + به آن اضافه کنم .

# Remove Method From Delegate :

حالا فرض کنید یک Delegate داریم که چندین متد را در زمان فراخوانی اجرا میکند حالا من میخواهم وقتی از Delegate در جایی دیگر استفاده نمودم ، یک متد را از آن حذف کنم و آن اجرا نشود دقیقا مثل مثال بالا باید متد مورد نظر را از آن کاهش بدهید .

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22** | **class Program**  **{**  **public delegate void DSum(int num1, int num2);**  **static void Main(string[] args)**  **{**  **DSum sum = new DSum(Sum);**  **sum += Minus;**  **sum -= Minus;**  **sum(10, 20);**  **Console.ReadKey();**  **}**  **static void Sum(int a, int b)**  **{**  **Console.WriteLine(a + b);**  **}**  **static void Minus(int a, int b)**  **{**  **Console.WriteLine(a - b);**  **}**  **}** | |

مثال بالا بارنگ قرمز این موضوع را بیان کرده است .

# Delegate By Return TYPE:

حالا بریم سراغ اینکه یک Delegate بسازیم که مقدار بازگشتی دارد . ابتدا یک متد میسازم که نوع بازگشتی داشته باشد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4** | **static int Divid(int num1, int num2)**  **{**  **return num1 / num2;**  **}** | |

اگر کمی دقت کنید متد بالا به امضای Delegate من نمیخورد لذا باید Delegate ای تعریف کنیم که به امضای آن همخوانی داشته باشد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | **public class ReturnType**  **{**  **public delegate int DDivid(int num1, int num2);**  **static void Run()**  **{**  **DDivid divid = Divid;**  **var result = divid(10, 20);**  **Console.WriteLine(result);**  **}**  **static int Divid(int num1, int num2)**  **{**  **return num1 / num2;**  **}**  **}** | |

با دقت به مثال بالا من متدی را به delegate نسبت دادم که امضای آنها با هم دیگه هم خوانی داشت .

# Use Delegate In Input METHOD:

در این بخش میخواهیم یک Delegate را به ورودی یک متد بدهیم و آن را صدا بزنیم :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23** | **class Program**  **{**  **public delegate void SumDelegate(int a, int b);**  **static void Main(string[] args)**  **{**    **SumDelegate dl = Sum;**  **Test(dl, 10, 20);**  **}**  **static void Sum(int a, int b)**  **{**  **Console.WriteLine(a + b);**  **}**  **static void Test(SumDelegate sum, int a, int b)**  **{**  **sum(a, b);**  **Console.WriteLine(a \* b);**  **}**  **}** | |

به متد Test دقت کنید داره میگه ورودی من یک Delegate میگیرم از جنس همان Delegate باید بهش بدهیم .

حالا من اومدم اون بالا یک Delegate تعریف کرده ام و بعدش اومدم داخل متد Main یک نمونه جدید از Delegate ساخته ام .

داخل همان متد Main اومدم متد Test را صدا زده ام و نمونه Delegate را به متد Test به عنوان ورودی داده ام.

به همین راحتی یک Delegate را به عنوان ورودی به یک متد ارسال کردم .

توجه کنید در متد Test من Delegate را صدا زده ام و مقدار های ورودی را بهش داده ام .همین Delegate اومده متد Sum را به خودش داده است .لذا با صدا زدن Delegate من عملا متد Sum را صدا زده ام.

**توضیح کلی** : پروژه ابتدا Main را اجرا میکند به سراغ Delegate می آید که من بهش گفتم متد sum را پوشش بوده یعنی میتوانیم چندین متد که امضای آنها با delegate یکسان هست را بهش بدهم . میاد پایین تا میرسه به متد Test اون متد یک delegate میگیره و 2 تا ورودی وقتی وارد متد Test میشه delegate را اجرا میکند هرچی متد من به delegate نسبت دادده ام را اجرا میکند یعنی متد Sum در متد Test صدا زده میشود .

# Anonymous METHOD:

متدهای بی نام چه چیزی هستند؟ متد های بی نام در اصل جایی تعریف نشده اند و در دل کد قرار گرفته اند پس این متد ها را در جایی دیگر نمیتوان صدا زد چون اسم ندارند.

## Example 1:

اولین مثالی که میخواهیم حل کنیم این است که چگونه یک متد بی نام را به یک Delegate نسبت بدهم ؟؟

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | **class Program**  **{**  **delegate int DLG1(int a, int b);**  **static void Main(string[] args)**  **{**  **DLG1 dl1 = delegate (int a, int b)**  **{**  **a \*= 2;**  **return a + b;**  **};**  **int result = dl1(10, 20);**  **Console.WriteLine(result);**  **Console.ReadKey();**  **}**  **}** | |

در کد بالا من یک متد بی نام را به delegate نسبت دادم که دقیقا امضای متد بی نام با امضای delegate برابر است و با نوشتن return و گذاشت براکت میتوانم بدنه برای این متد بنویسم.

پس میتوان گفت ساختار کد بالا با استفاده از کلمه کلیدی delegate صورت گرفته و 2 تا وروردی int گرفته و با 2 تا براکت بدنه تابع را ساختم و در نهایت مقداری را return کرده ام.

## Example 2:

حالا میخواهیم کد بالا را به صورت خلاصه بنویسیم :

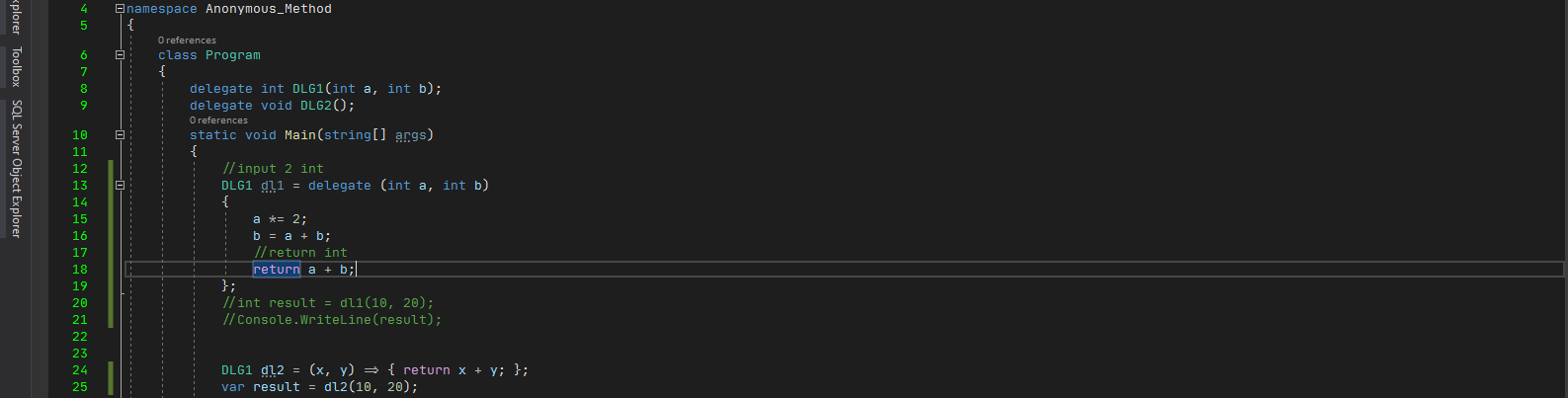
|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13** | **class Program**  **{**  **delegate int DLG1(int a, int b);**  **static void Main(string[] args)**  **{**  **DLG1 dl2 = (x, y) => x + y;**  **var result = dl2(10, 20);**  **Console.WriteLine(result);**  **Console.ReadKey();**  **}**  **}** | |

به متد بی نام بالا دقت کنید من اصلا x,y را تعریف نکردم ولی به من گیر نداده چرا ؟ از کجا متوجه شده است که مقدار x,y میتواند int باشد ؟

این متد بی نام اصلا return ندارد !!! بله دوستان این متد بی نام مقدار های تعریف شده ی خودش را از امضای delegate خودش متوجه میشود .

یعنی نوع x,y را از امضای Delegate متوجه میشود و اینکه آیا مقدار x,y بازگشتی است یا نه ، هم از Delegate متوجه میشود چون Delegate خودش مقدار int را بازگشت میدهد .

## نکته :



2 تا متد بالا را باهم مقایسه کنید .متد اولی کلمه کلیدی delegate را دارد و 2 تا ورودی و بعد از آن 2 تا براکت داره و مقدار return برای بازگشت .

حالا در متد دومی کلمه delegate حذف شده است و فقط 2 تا ورودی مانده است و با استفاده از lambda براکت را نوشته ایم تا بدنه به تابع بدهیم و مقداری را return کنیم .

## Example 3:

حالا فرض کنید من یک Delegate تعریف کردم که مقدار ورودی ندارد !!! چگونه متد های بی نام را به این Delegate نسبت بدهم ؟؟

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14** | **class Program**  **{**  **delegate void DLG2();**  **static void Main(string[] args)**  **{**  **DLG2 dl3 = () =>**  **{**  **string test = "Ali";**  **Console.WriteLine("Hello" + " " + test);**  **};**  **dl3();**  **}**  **}** | |

به متد بی نام بالا دقت کنید چون delegate هیچ نوع بازگشتی نداشت مجبور شدیم در زمان تعریف متد بی نام 2 تا پرانتز خالی را بنویسیم و با نوشتن 2 تا براکت میتوانم بدنه برای این متد بی نام بنویسیم .

## Example 4:

مثال زیر شامل یک بدنه برای متد بی نام میباشد و هم مقدار بازگشتی دارد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17** | **class Program**  **{**  **delegate int DLG1(int a, int b);**  **static void Main(string[] args)**  **{**  **DLG1 dl3 = (x, y) =>**  **{**  **x \*= 2;**  **return x + y;**  **};**  **var result = dl3(10, 20);**  **Console.WriteLine(result);**  **Console.ReadKey();**  **}**  **}** | |

متد بی نام بالا هم بدنه دارد و هم مقدار بازگشتی دارد .

# Generic DELEGATE:

در ابتدا من یک Delegate مینویسم که به صورت جنریک میباشد :

|  |
| --- |
| **<access modifier> delegate <return type> <delegate\_name>(<parameters>)** |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13** | **public class Program**  **{**  **public delegate TResult Dlgt1<in T1, in T2, out TResult>(T1 a, T2 b);**  **private static void Main(string[] args)**  **{**  **Dlgt1<double, int, double> dlgt1 = (a, b) => a / b;**    **//1**  **var result = dlgt1(1.3, 2);**  **Console.WriteLine(result);**  **}**  **}** | |

در مثال بالا من داخل جنریک <> مقدار های ورودی و خروجی را نوشتم چی هست ، نوشتم که از جنس جنریک هستی .

در خط 7 اومدم مقدار های ورودی و خروجی را بهش دادم و بعدا ازش استفاده کردم.

حالا اگر بخواهیم ورودی چند تا بگیریم باید چیکار کنیم ؟؟ فرض کنید یک بار Delegate تعریف میکنم که 2 تا ورودی داره دفعه بعدی میخواهم 3 تا ورودی داشته باشه !! اینجوری که من باید 100 تا Delegate تعریف کنم !

برای رفع این مشکل خود Microsoft یک دونه Delegate تعریف کرده که 16 تا ورودی میتواند داشته باشد و حتما یک خروجی که بهش Func میگویند :

## FUNC:

این نوع delegate حتما یک خروجی دارد اگر داخل ورودی هاش فقط یک ورودی وجود داشت یعنی اون خروجی است و اگر 2 تا ورودی داشت یعنی 1 ورودی دارد و یک خروجی دارد به همین صورت برای 16 تا ورودی و 1 خروجی ادامه خواهد داشت .

حالا اگر بخواهیم مثال بالا را به صورت Func بنویسیم به صورت زیر خواهد بود :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18** | **public class Program**  **{**  **public delegate TResult Dlgt1<in T1, in T2, out TResult>(T1 a, T2 b);**  **private static void Main(string[] args)**  **{**  **Dlgt1<double, int, double> dlgt1 = (a, b) => a / b;**  **//1**  **var result = dlgt1(1.3, 2);**  **Console.WriteLine(result);**  **//Convert 1 To Func**  **Func<double, int, double> func1 = (double a, int b) => a / b;**  **Console.WriteLine(func1(1.3, 2));**  **}**  **}** | |

در Func بالا من در ابتدا 2 تا ورودی به متد داده ام و یک خروجی را دریافت کرده امکه دقیقا جلو همین Func اقدام به تعریف متد بی نام نموده ام.

### Just OUT:

حالا فرض کنید من یک Func میخواهم بنویسم که فقط خروجی دارد و ورودی ندارد !!! این را ه میدانیم که Func باید و باید خروجی داشته باشد !!!

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3** | **//3 No Input just return**  **Func<string> func2 = () => "Ali";**  **Console.WriteLine(func2);** | |

متد بالا فقط خروجی از جنس string خواهد داشت .

## ACTION:

حالا فرض کنید من نمیخواهم خروجی داشته باشم و فقط ورودی میخواهم در این صورت دیگر نمیتوانیم از Func استفاده کنیم باید از Action استفاده کنیم این نوع delegate فقط 16 تا ورودی دارد و فاقد خروجی میباشد .

در اصل میتوان گفت action یک delegate میباشد که ورودی دارد ولی خروجی آن Void میباشد .

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2** | **Action<string> myAction = x => Console.WriteLine(x);**  **myAction("Ali");** | |

مثال بالا فقط ورودی میگیرد و آن را چاپ میکند.

## PREDICATE:

این نوع delegate به این گونه است که یک ورودی میگیرد و خروجی آن از جنس bool میباشد ولی نوشته نمیشود . مثال :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2** | **Predicate<int> myPredicate = x => x == 5;**  **Console.WriteLine(myPredicate(5));** | |

همین مثال بالا را میتوان به راحتی به صورت Func هم نوشت که به صورت زیر میباشد :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3** | **//5 Convert Predicate To Func**  **Func<int, bool> myFunc = x => x == 5;**  **Console.WriteLine(myFunc(5));** | |

حالا همین مثال بالا که من به صورت lambda اومدم نوشتم را میخواهم به صورت متد برایش بنویسم که به صورت زیر خواهد بود :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35**  **36**  **37**  **38**  **39**  **40**  **41**  **42**  **43**  **44**  **45**  **46** | **public class Program**  **{**  **public delegate TResult Dlgt1<in T1, in T2, out TResult>(T1 a, T2 b);**  **private static void Main(string[] args)**  **{**  **Dlgt1<double, int, double> dlgt1 = (a, b) => a / b;**  **//1**  **var result = dlgt1(1.3, 2);**  **Console.WriteLine(result);**  **//Convert 1 To Func**  **Func<double, int, double> func1 = (double a, int b) => a / b;**  **Console.WriteLine(func1(1.3, 2));**  **//3 No Input just return**  **Func<string> func2 = () => "Ali";**  **Console.WriteLine(func2);**  **//4**  **Action<string> myAction = x => Console.WriteLine(x);**  **myAction("Ali");**  **//5**  **Predicate<int> myPredicate = Test;**  **Console.WriteLine(myPredicate(5));**  **//5 Convert Predicat To Func**  **Func<int, bool> myFunc = x => x == 5;**  **Console.WriteLine(myFunc(5));**  **}**  **static bool Test(int x)**  **{**  **if (x == 5)**  **{**  **return true;**  **}**  **return false;**  **}**  **}** | |

به مثال بالا دقت کنید من اومدم دقیقا عبارت lambda را پاک کردم و به جایش متد Test را قرار دادم این متد کار همان عبارت را انجام میدهد.

# کاربرد Delegate :

در این بخش میخواهیم کاربرد Delegate ها را بررسی کنیم به مثال اول دقت کنید :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23** | **public class Program**  **{**  **public static void Main(string[] args)**  **{**  **int[] array = new[] { 1, 2, 3, 4, 5, 7, 8, 3 };**  **Func<int, bool> myFunc = Test;**  **var query = array.Where(myFunc);**  **foreach (var i in query)**  **{**  **Console.WriteLine(i);**  **}**  **Console.ReadKey();**  **}**  **private static bool Test(int x)**  **{**  **return x == 3;**  **}**  **}** | |

در بالا Where یک Func میگیره ! میگه به من میتونی int ورودی بدی و من خروجی bool بازگشت میدهم.حالا من به جای اینکه از دستور linq استفاده کنم اومدم واسش یک Func نوشتم که ورودی یک x میگیره و مقدار bool را بازگشت میدهد که همان متد Test میباشد .

در کد بالا من where زدم که ان را To list نکردم یعنی Enumerable تشکیل شده است لذا در خط 9 متد Test را صدا نمیزند وقتی وارد foreach شد حالا شروع میکند به صدا زدن Test که این کار را چند بار انجام میدهد یعنی میره داخل حلقه چاپ میکنه دوباره برمیگرده متد Test را صدا میزند.

حالا همین کد بالا را تبدیل کنید به Tolist در این صورت در جا اجرا میشود و در همان خط 9 متد Test را صدا میزند و بعدا چاپ میکند پس این فرق بین Tolist و Ienamarble میباشد .

نکته : در کد بالا قبل از where من source را دارم و داخل where یک Func را صدا زده ام که ورودی int میگیره و مقدار bool برمیگرداند.

ورودی Test از کجا تامین شده است ؟؟؟دونه دونه از Source به داخل Func من ارسال میشود .

نکته : در کد بالا من اومدم یک Func ساختم و Test را بهش دادم اگر همین Test را به صورت جدا گانه به Where بدهیم هم دراست است اگر Func داخل Where را رعایت کرده باشیم یعنی حتما ورودی از جنس Source باشد و خروجی از جنس bool باشد .

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19** | **public class Program**  **{**  **public static void Main(string[] args)**  **{**  **int[] array = new[] { 1, 2, 3, 4, 5, 7, 8, 3 };**  **string[] arrStrings = new string[] { "Ali", "Reza", "Ahmad" };**  **//1**  **Func<int, bool> myFunc = Test;**  **var query = array.Where(delegate (int x)**  **{**  **if (x == 3)**  **{**  **return true;**  **}**  **return false;**  **});**  **}** | |

به مثال بالا دقت کنید where یک Func delegate میگیره یعنی یک ورودی میگیره و خروجی از جنس bool برمیگرداند .

حالا منم به جای اینکه از دستورات lambda استفاده کنم اومدم بهش گفتم من ورودی بهت int میدهم و خروجی از جنس bool میباشد .

زمانی که شما مینویسید x=> x==3 این همان delegate است فقط فرقش به این است که خلاصه شده است .

# فرق FirstOrDefault & SingleOrDefault :

First وقتی به اولین دیتای مورد نظر رسید از اون میاد بیرون و دیگه کل دیتاها را بررسی نمیکند ولی دستور Single میره کل دیتا ها را بررسی میکند که آیا چنین چیزی یکتا هست یا خیر!!!

پس بهتر است زمانی که دنبال یک دیتا میگردیم از First استفاده کنیم تا کل دیتابیس را جستجو نکند.

# Where (x, y):

تا الان هرچی داخل دستورات Where انجام دادیم فقط یک ورودی داشت مثلا where(x=>x.) حالا یک Extension دیگر هم داره که میگه ورودی 2 تا به من بده و خروجی اش bool باشد یعنی Func <int,int,bool> میگیرد.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1** | **var query3 = array.Where((x, y) => x == y);** | |

در دستور بالا مقدار y چی هست ؟؟ مقدار x چی هست ؟؟ برای اینکه به این موضوع پی ببریم شروع میکنم یک متد مینویسم مثل امضای خود where :

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13** | **public static void Main(string[] args)**  **{**    **var query3 = array.Where(Test1);**  **Console.ReadKey();**  **}**  **static bool Test1(int a, int b)**  **{**  **return a == b;**  **}** | |

اگر مثال بالا را بررسی کنید متوجه میشوید که مقدار y مقدار Index یا همان اندیس مورد نظر میباشد .

# پیاده سازی First Or Default با دستور Where :

میخواهیم در این بخش دستور First Or Default را این بار بدون استفاده از اون با Where داخل یک حلقه foreach پیاده سازی کنیم !

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23** | **public static void Main(string[] args)**  **{**  **int[] array = new[] { 1, 2, 3, 4, 5, 7, 8, 3 };**    **//FirstOrDefault IN Where**  **Func<int, bool> myFunc1 = Test;**  **var query5 = array.Where(myFunc1);**  **foreach (var i in query5)**  **{**  **Console.WriteLine(i);**  **break;**  **}**  **Console.ReadKey();**  **}**  **private static bool Test(int x)**  **{**  **return x == 3;**  **}** | |

دستور بالا چون tolist برای where نداریم لذا تا حلقه foreach متد Test صدا زده نمیشود خوب کار ادامه پیدا میکند تا برسیم به Foreach داخل اون متد Test اونقدر صدا زده میشود تا return True را به Foreach بدهد وقتی وارد اون شد و مقدار را چاپ کرد سریع به دلیل Break از حلقه خارج میشود دقیقا مثل FirstOrDefault میباشد .

توضیح کا مل : در کد بالا در خط 10 من array را در خود دارم ولی هنوز Test را صدا نزده ام .

حالا میرسیم به foreach داخل اون Test را صدا میزنیم اینقدر این حلقه تکرار میشود بین array تا مقدار true را return کند .حالا وقتی به مقدار 3 رسید مقدار i== 3 خواهد بود و با دستور break از حلقه خارج میشود دقیقا مثل FirstOrDefault خواهد بود.