عنوان: **آشنایی با جنریک‌ها #3**
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۱/۲۱ ۲۳:۴۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Generics

متدهای جنریک

متدهای جنریک، دارای پارامترهایی از نوع جنریک هستند و بوسیله‌ی آنها می‌توانیم نوع‌های (type) متفاوتی را به متد ارسال نمائیم. در واقع از متد، یک نمونه پیاده سازی کرده‌ایم، در حالیکه این متد را برای انواع دیگر هم می‌توانیم فراخوانی کنیم.

تعریف ساده دیگر

جنریک متدها اجازه می‌دهند متدهایی با نوع هایی که در زمان فراخوانی مشخص کرده ایم، داشته باشیم.

نحوه تعریف یک متد جنریک بشکل زیر است:

```
return-type method-name<type-parameters>(parameters)
```

قسمت مهم syntax بالا، **type-parameters** است. در آن قسمت می‌توانید یک یا چند نوع که بوسیله کاما از هم جدا می‌شوند را تعریف کنید. این typeها در return-value و نوع برخی یا همه پارامترهای ورودی جنریک متد، قابل استفاده هستند. به کد زیر توجه کنید:

```
public T1 PrintValue<T1, T2>(T1 param1, T2 param2)
{
    Console.WriteLine("values are: parameter 1 = " + param1 + " and parameter 2 = " + param2);
    return param1;
}
```

در کد بالا، دو پارامتر ورودی بترتیب از نوع T1 و T2 و پارامتر خروجی (return-type) از نوع T1 تعریف کرده‌ایم.

اعمال محدودیت بر روی جنریک متدها

در زمان تعریف یک جنریک کلاس یا جنریک متد، امکان اعمال محدودیت بر روی typeهایی را که قرار است به آن‌ها ارسال شود، داریم. یعنی می‌توانیم تعیین کنیم جنریک متد چه typeهایی را در زمان ایجاد یک وهله‌ی از آن بپذیرد یا نپذیرد. اگر نوعی که به جنریک متد ارسال می‌کنیم جزء محدودیت‌های جنریک باشد با خطای کامپایلر روبرو خواهیم شد. این محدودیت‌ها با کلمه کلیدی where اعمال می‌شوند.

```
public void MyMethod< T >()
       where T : struct
{
  ...
}
```

محدودیت‌های قابل اعمال بر روی جنریک ها

struct: نوع آرگومان ارسالی باید value-type باشد؛ بجز مقادیر غیر NULL.

```
class C<T> where T : struct {} // value type
```

class: نوع آرگومان ارسالی باید reference-type (کلاس، اینترفیس، عامل، آرایه) باشد.

```
class D<T> where T : class {} // reference type
```

new(): آرگومان ارسالی باید یک سازنده عمومی بدون پارامتر باشد. وقتی این محدوده کننده را با سایر محدود کننده‌ها به صورت همزمان استفاده می‌کنید، این محدوده کننده باید در آخر ذکر شود.

```
class H<T> where T : new() {} // no parameter constructor
```

```
public void MyMethod< T >()
       where T : IComparable, MyBaseClass, new ()
{
  ...
}
```

<base class name>: نوع آرگومان ارسالی باید از کلاس ذکر شده یا کلاس مشتق شده آن باشد.

```
class B {}
class E<T> where T : B {} // be/derive from base class
```

<interface name>: نوع آرگومان ارسالی باید اینترفیس ذکر شده یا پیاده ساز آن اینترفیس باشد.

```
interface I {}
class G<T> where T : I {} // be/implement interface
```

U: نوع آرگومان ارسالی باید از نوع یا مشتق شده U باشد.

```
class F<T, U> where T : U {} // be/derive from U
```

توجه: در مثال‌های بالا، محدوده کننده‌ها را برای جنریک کلاس‌ها اعمال کردیم که روش تعریف این محدودیت‌ها برای جنریک متدها هم یکسان است.

اعمال چندین محدودیت همزمان

برای اعمال چندین محدودیت همزمان بر روی یک آرگومان فقط کافی است محدودیت‌ها را پشت سرهم نوشته و آنها را بوسیله کاما از یکدیگر جدا نمایید.

```
interface I {}
class J<T>
  where T : class, I
```

در کلاس J بالا، برای آرگومان محدودیت **class** و **اینترفیس I** را اعمال کرده‌ایم.

این روش قابل تعمیم است:

```
interface I {}
class J<T, U>
  where T : class, I
  where U : I, new() {}
```

در کلاس J، آرگومان T با محدودیت‌های class و اینترفیس I و آرگومان U با محدودیت اینترفیس I و new() تعریف شده است و البته تعداد آرگومان‌ها قابل گسترش است.

حال سوال این است: **چرا از محدود کننده‌ها استفاده می‌کنیم؟**

کد زیر را در نظر بگیرید:

```
//this method returns if both the parameters are equal
public static bool Equals< T > (T t1, Tt2)
{
  return (t1 == t2);
}
```

متد بالا برای مقایسه دو نوع یکسان استفاده می‌شود. در مثال بالا در صورتیکه دو مقدار از نوع int با هم مقایسه نماییم جنریک متد بدرستی کار خواهد کرد ولی اگر بخواهیم دو مقدار از نوع string را مقایسه کنیم با خطای کامپایلر مواجه خواهیم شد. عمل مقایسه دو مقدار از نوع string که مقادیر در heap نگهداری می‌شوند بسادگی مقایسه دو مقدار int نیست. چون همانطور که می‌دانید int یک value-type و string یک reference-type است و برای مقایسه دو reference-type با استفاده از عملگر ==

تمهیداتی باید در نظر گرفته شود.
برای حل مشکل بالا 2 راه حل وجود دارد:
Runtime casting
استفاده از محدود کننده‌ها

casting در زمان اجرا، بعضی اوقات شاید مناسب باشد. در این مورد، CLR نوع‌ها را در زمان اجرا بدلیل کارکرد صحیح بصورت اتوماتیک cast خواهد کرد اما مطمئناً این روش همیشه مناسب نیست مخصوصاً زمانی که نوع‌های مورد استفاده در حال تحریف رفتار طبیعی عملگرها باشند (مانند آخرین نمونه بالا).