**پنج تغییر اساسی در3.0 #c**

---

**C# 2.0شرکت مایکروسافت ویژگی‌هایی را ارایه نمود که بلافاصله مورد استقبال و استفاده برنامه‌نویسان قرار گرفت، از قبیل متدهای عمومی (Generic) و متدهای بی‌نام .(Anonymous) مایکروسافت در نسخه 3.0 (سال 2007) ویژگی‌های جدیدی را ارایه می‌کند که برنامه‌نویسان می‌توانند با استفاده از آنها انعطاف‌پذیری و قابلیت نگهداری نرم‌افزارهای خود را افزایش دهند. این مقاله به پنج ویژگی برتر C# 3.0 می‌پردازد.**
**کلید واژه:**

**Anonymous , Generic, Extension**

**-1متغیرهای محلی نوع‌دار ضمنی**

**C# 3.0کلیدواژه جدیدی به نام var را ارایه می‌کند. این کلیدواژه به برنامه‌نویس اجازه می‌دهد که متغیری را بدون تعیین نوع آن تعریف کند. به عنوان نمونه، خط زیر برای یک نمونه از یک رشته می‌تواند به کار رود:**

```
var myData = "This is my data";
```

**توجه داشته باشید که در این دستور هیچ اشاره‌ای به رشته بودن myData نمی‌شود؛ هر چند که این امر در C# 2.0 ضروری بود. با وجود این که C# 3.0 به شما اجازه می‌دهد که از تعیین نوع متغیر طفره بروید، ولی این قضیه به ماهیت نوع‌دار بودن #C لطمه‌ای نمی‌زند. چرا که کلیدواژه var تنها متغیر را تعریف می‌کند ولی نوع متغیر به محض تخصیص اولین مقدار تعیین می‌شود و برنامه‌نویس پس از آن دیگر قادر نخواهد بود که نوع متغیر را تغییر دهد. به عنوان مثال کد زیر درست کار نخواهد کرد:**

کد:

```
var myDate = DateTime.Now;

myDate = "Hello.";
```

**یکی از مزایای کلیدواژه var این است که به برنامه‌نویس امکان می‌دهد از تعریف مکرر نوع متغیرهای اجتناب کند. به عنوان مثال، برای این که شی Customer را در C# 2.0 تعریف کنید، نیاز به کدی مشابه نمونه زیر دارید:**
کد:

```
Customer myCustomer = new Customer();
```

**در صورتی که با کلیدواژه جدید var می‌توانید کد فوق را به صورت زیر بنویسید:**

کد:

```
var myCustomer = new Customer();
```

**از دیگر ویژگی‌های دیگر var این است که برنامه‌نویس را از تغییر فراخوانی متدی که نوع خاصی را باز می‌گرداند، فارغ می‌کند. مثلا، اگر در C# 2.0 نیاز به فراخوانی متدی داشته باشید که شیرا باز می‌گرداند، باید کدی مشابه زیر بنویسید** Customer **:**

کد:

```
Customer myCustomer =  GetByName("Zach");
```

**حال اگر بنا به هر دلیلی متد GetByName بخواهد شی‌ای از هر نوع دیگری را باز گرداند، برنامه کامپایل نخواهد شد. در این صورت اگر از کلیدواژه var در تعریف متغیری که مقدار بازگشتی متد به آن تخصیص می‌یابد استفاده کنید و در ادامه نوع بازگشتی متد GetByName به شی‌ای از نوعتغییر کند، هیچ مشکلی پیش نخواهد آمد** Person **.**

کد:

```
var myData = GetByName("Zach");
```

**-2متدهای گسترش(Extension)**

**در #C نمی‌توان نوع‌هایی را که با پیرایندە[1 sealed ]نشان‌گذاری شده‌اند، به ارث برد یا گسترش داد. در صورتی که در 3.0 #C به‌راحتی می‌توان با استفاده از متدهای Extension این کار را با هر کلاسی، حتی اگر با sealed علامت‌گذاری شده باشد، انجام داد. برای مثال، اگر بخواهیم متدی به نامرا به نوع رشته (String) اضافه کنیم، کافی است کدی مشابه نمونه زیر را به کار بگیریم** NoSpaces() **:**
کد :
کد:

```
namespace MyExtensionMethods {

public static class Extension {

public static void NoSpaces(this string data) {

return data.Replace(" ", "");

}

}

}
```

**اگر این کلاس در هر کلاسی import شود، برنامه‌نویس می‌تواند متد ()NoSpace را برای هر رشته‌ای که درون آن کلاس باشد، فراخوانی کند. پارامتر اول متد گسترش نوعی را معین می‌کند که این متد باید برای آن قابل دسترسی باشد. در مورد مثال فوق this string data مشخص می‌کند که نوعی که متد با آن سروکار خواهد داشت" رشته" خواهد بود. اگر پارامتر به صورت this object data تعیین می‌شد، متد ()NoSpace برای هر نوع متغیری قابل استفاده می‌بود. برای تعیین استفاده کردن یک کلاس از متدهای گسترش، باید حکم[2 using ]را به کار برد. به عنوان مثال، برای استفاده از متد گسترش فوق باید به صورت زیر عمل نمود:**
کد:
کد:

```
using MyExtensionMethods;

namespace MyNamespace {



public class MyClass {

public MyClass() {

string data = "this is my data";

// nospaces will contain "thisismydata".

string nospaces = data.NoSpaces();

}

}
```

**}**

**دقت کنید که متد گسترش نسبت به متدهای نمونه[3] ساخته شده از کلاس از اولویت کمتری برخوردار است. بنابراین، در صورت هم‌نامی، متدی که در شی ساخته شده وجود دارد اجرا خواهد شد.**
کد:

```
[1] modifier

[2] directive

[3] instance

[[ ]]
```

**-3مقداردهی اولیه[4]**

**در C# 2.0 برنامه‌نویسان مجبور بودند برای مقداردهی اولیه نمونه‌ای از یک کلاس، از متد سازنده[5] استفاده کنند. به نمونه‌های زیر توجه کنید:**

**-*کلاسی که به Customer دسترسی دارد:**

**Customer myCustomer = new Customer("Zach", "Smith");**

**-*متد سازنده کلاس :Customer**
نقل قول:

```
public Customer(string firstName, string lastName) : this() {

this.FirstName = firstName;

this.LastName = lastName;
```

**C# 3.0امکان جدیدی را ارایه می‌کند، بدین صورت که می‌توان شی را در زمان ساخت نمونه از کلاس مقدار دهی نمود. برای مثال، نمونه زیر را به عنوان بازنویسی کد فوق مورد توجه قرار دهید:**

**-*کلاسی که به Customer دسترسی دارد:**

کد:

```
Customer myCustomer = new Customer{ FirstName = "Zach",  LastName = "Smith"
                                                                    };
```

**·متد سازنده کلاس:Customer**

**public Customer**

کد:

```
()
```

**{ }**

**در کد C# 3.0 هیچ متد سازنده‌ای که به مقداردهی صفات مربوط باشد وجود ندارد. این امر برنامه‌نویس را از اجبار به ایجاد سازنده‌های مختلف برای مجموعه متفاوتی از صفات رها می‌سازد. اثر جانبی این روش مقداردهی افزایش خوانایی کد است. به عنوان مثال، گرچه در قطعه‌کد زیر مشخص است که یک شی Car ساخته می‌شود، ولی معلوم نیست که مقادیر به چه صفاتی تخصیص می‌یابند.**

کد:

```
Car car = new Car(18, 10, 550);
```

**در صورتی که نمونه کد زیر، با این که تایپ بیشتری لازم دارد، ولی خواناتر است:**

کد:

```
Car car = new Car { WheelDiameter = 18, WheelWidth = 10,  Horsepower = 550 };
```

**-4انواع بی‌نام[6]**

**همان طور که در" C# 2.0 متدهای بی‌نام" معرفی شدند در C# 3.0 نیز "انواع بی‌نام" در دسترس برنامه‌نویسان قرار گرفتند. انواع بی‌نام از این نظر مانند متدهای بی‌نام هستند که "در خط" تعریف می‌شوند و نیازی به نام ندارند. برای تعریف یک نوع بی‌نام باید از دو مفهوم مقداردهی اولیه شی و متغیرهای محلی نوع‌دار ضمنی (که در بندهای پیشین تشریح شدند) استفاده شود. کد زیر نمونه‌ای از یک نوع بی‌نام را نشان می‌دهد :**

کد:

```
var myType = new { Length = 79, Width = 30  };
```

**حوزه یک متغیر با نوع بی‌نام همانند دیگر متغیرهای تعریف شده است. مثلا، نمونه cobra در کد زیر تنها در بلوک تابع Speed قابل دسترسی است:**
کد:

```
private void Speed() {

var cobra = new { Horsepower = 550, Torque = 570  };

}
```

**اگر نوع بی‌نامی در بلوکی تعریف شود، در صورتی که نوع بی‌نام دیگری پیشتر در همان بلوک تعریف شده باشد، به طوری که امضای‌شان[7] یکسان باشد، نوع دوم نوع "نوع" اول را به خود می‌گیرد. برای نمونه، در کد زیر cobra و mustang هر دو از یک نوع‌اند و می‌توانند به همدیگر تخصیص داده شوند:**
کد:
کد:

```
private void Speed() {

var cobra = new { Horsepower = 550, Torque = 570 };

var mustang = new { Horsepower = 300, Torque = 300 };

mustang = cobra;

// or you could say cobra = mustang

}



[4] initializer

[5] constructor

[6] Anonymous

[7] signature

[[ ]]
```

**5- LINQ**

**در نسخ قبلی #C برنامه‌نویس‌ها باید از زبان‌های پرس‌وجوی[8] متفاوتی برای دسترسی به منابع داده‌ای مختلف استفاده می‌کردند؛ مثلا، XPathرا برای پرس‌وجوی یک مستند XML و زبان SQL را برای یک پایگاه داده مبتنی بر SQL به کار می‌بردند. این روش دسترسی به داده‌های منابع مختلف گرچه تا**

**به حال بیشتر مورد استفاده بوده ، ولی نواقصی هم به همراه داشته است. از جمله مشکلات این روش می‌توان به استفاده از زبان‌های ناهمگون برای کار با منابع داده‌ای متفاوت اشاره کرد. مشکل دیگر این است که برنامه‌نویس باید نتیجه گرفته شده از یک زبان دیگر، مانندSQL ، را به شی متناسب در #C تبدیل کند تا بتواند در کد خود مورد استفاده قرار دهد.**

**شرکت مایکروسافت برای رفع این معضل، در C# 3.0 خود فناوری جدیدی به نام LINQ[9] ارایه کرده است. با استفاده از این فناوری، برنامه‌نویس می‌تواند پرس‌وجویی استاندارد را بنویسد که برای هر نوع منبع داده IEnumerable<T> کار کند. بنابراین به جای این که از TSQL برای دسترسی به پایگاه داده SQL Server و از XPath برای XML استفاده کنید، LINQرا به کار بگیرید .نمونه کد زیر یک پرس‌وجوی استاندارد است که لیست مشتریانی را که بیش از ده سفارش دارند، باز می‌گرداند.**
کد:
کد:

```
using System;

using System.Query;

using System.Collections.Generic;

public class SampleClass {

static void Main() {

List<Customer> customers = GetCustomers();

// Write our query to retrieve customers who have more than

// 10 orders.

IEnumerable<Customer> queryResult = from customer in customers

where customer.OrderCount > 10

orderby customer.ID

select customer;

}

}
```

**بر خلاف SQL و XPath پرس‌وجوهای LINQ به زبان #C نوشته می‌شوند و کد غریبه‌ای نیستند. این امر پرس‌وجوها را از نظر نوع داده‌ها امن و برنامه‌نویس را از تبدیل نتایج بازگردانده شده به یک شی #C بی‌نیاز می‌کند. عمل تبدیل توسط رابط برنامه‌نویسی LINQ و به صورت خودکار انجام می‌گیرد. به زبان ساده، پروژه LINQ به عنوان یک راه حل "نگاشت شی‌گرا - رابطه‌ای"[10] توکار عمل می‌کند .بنابراین، گستره وسیعی را در بر می‌گیرد. اطلاعات جامعی از این فناوری را می‌توانید در وب‌سایت MSDN بیابید.**

**مترجم: به عنوان یک نظر شخصی تغییراتی که در C# 3.0 صورت پذیرفته، همگی با استقبال مواجه نخواهند شد. بدون شک بزرگترین تغییر linq خواهد بود اما اینکه بتواند جای query نویسی را بشکل کامل بگیرد، جای بحث بسیار دارد. و بعضی از تغییرات با موافقت برنامه نویسان آنچنان همراه نبوده است.**

**شاید بتوان گفت پس از مدتها و با این تغییرات، شناختی که ما از دنیای برنامه نویسی داریم، تغییر خواهد کرد.**

**منبع : sayan.ir**