منبع : مرکز اطلاع رساني و پژوهش هاي برنامه نويسي ايران (irAsp.Net)
آموزش : مهدي عسگري
ايميل: mehdideveloper@gmail.com

عنوان: **First article (basics)on CShap (مقاله ۱)**

بسم الله الرحمن الرحیم

سلام

این اولین قسمت از سری آموزشی C# و .NET. است که قراره توی این سری مقالات به ناگفته ها و مطالبی از این دو مبحث بپردازم که زیاد توی کتابا و سایتها ازشون مطلب نمی بینید. اینجا من مطالب ساده و ابتدایی رو ذکر نمیکنم ، چون به نظر من برای یاد گرفتن این چیزا بهتره یه کتاب بگیرید و بخونید .

پس این مطالب برای برنامه نویسای متوسط به بالاست ، نه برای تازه کارها.
مطالب این مقالات نه تنها برای برنامه نویسان C# ، بلکه برای هرکسی که به نحوی با .NETسرو کار داره مفیده (مثل برنامه نویسای .NET. VB و VJ# و(Visual C++ .NET
البته من تخصصی در زمینه .NET. Web , ASP ندارم ،در مورد اونا دوستای دیگه ای هستن تو سایت.
در ضمن سعی میکنم هر چند جلسه هم یه کتاب خوب رو معرفی کنم.
و در آخر : اگه سوال یا مشکلی داشتید یه سر به تالار گفتگو بزنید و مشکلتون رو اونجا مطرح کنید یا میتونین email کنین بهم ، به نظر من آدم با پرسش و پاسخ ، بیشتر یاد می گیره تا با آموزش صرف.
خب پرحرفی بسه ، بریم سر اصل مطلب:

این جلسه به Terminology یا اصطلاح شناسی و همچنین توضیح مفاهیم زیرین .NET. میپردازم ، تا از مقالات بعد به مشکل بر نخوریم . شاید امروز زیاد عملی نباشه و مفاهیم کاملا تئوری رو بگم ، اما ارزششو داره ، تا آخر بخونید.
ببینید به طور کلی چارچوب برنامه نویسی .NET. به دو بخش اساسی تقسیم شده:
1- BCLیا Base Class Library که همون کلاس ها و انواع داده ای و اینها هستن و ما توسط اونا برنامه هامون رو مینویسیم (مثل(...,String , Environment , Math و یکی از دلایل موفقیت .NET. ، مجموعه بزرگ کلاس هاشه ، مثلا ما تو .NET. کلاسهایی برای کار باWindows , Regular Expressions , Text , Remoting , XML ASP .NET , Database , Network وWeb Service و ... داریم.
2- CLRیا Common Language Runtime که از اسمش معلومه ، محیط اجرای مشترک بین تمام زبانهای برنامه نویسیه که وظایفی مثل کامپایل کردن کد و برقراری امنیت اجرای کد و ایجاد ارتباط بین زبان های برنامه نویسی مختلف)البته تحت (.NET. و مدیریت حافظه و خیلی وظایف دیگه رو به عهده داره که به همشون خواهیم رسید.
معمولا برنامه نویسا با BCL آشناترن و ۹۰ درصدشون حتی از وجود CLR هم اطلاعی ندارن دلیلشون هم اینه که " : کاربرد مستقیم نداره" ، درسته ، ولی اگه آشنایی مختصری با اون داشته باشین ، درکتون از برنامه نویسی تحت .NET.چند برابر میشه و در نتیجه برنامه هایی که می نویسید هم بهتر اجرا میشن.

CLSیا Common Language Specification یک سری قوانین رو بیان میکنه که اگه هر زبان

برنامه نویسی اونا رو رعایت کنه میتونه مطمئن باشه که همهٔ زبانهای تحت .NET. میتونن کد نوشته شده در اون رو فراخوانی کنن
)چند تا از این قوانین :
-عدم استفاده ازPointer
-عدم استفاده از انواع داده ای بدون علامت (مثل ulong , uint در(C#
-عدم استفاده از لیست متغیر در آرگومان های توابع (params) یا(ParamArray
-اندیس شروع تمام آرایه ها باید صفر باشد.
interface -ها نمیتونن توابع static داشته باشن.
-یک سازنده قبل از هر کاری باید سازندهٔ کلاس پدر را فراخوانی کند.
.... -
حالا اگه شما این قوانین رو رعایت کنید میتونین مطمئن باشین که هر زبان برنامه نویسی تحت .NET. قادر به فراخوانی کد شما خواهد بود.
مثال:
اگه توی C# یا Managed C++ یه کلاس درست کنید که حاوی یه تابع با نوع برگشتیpointer باشه یا یکی از آرگومان هاش یه pointer باشه ، اون وقت نمیتونین اون تابع رو از .NET. VB فراخوانی کنین(.

CTSیا : Common Type System مشخص کنندهٔ یک سری نوع داده و همچنین قوانینی در مورد این نوع داده هاست که در مورد تموم زبان های تحت .NET. مشترکه.
به عنوان مثال آیا تا به حال فکر کردین چرا وقتی یه تابع نوشته شده در .NET. VB با نوع برگشتی Singleرو در C# فراخوانی میکنین ، اون رو به صورت float می بینین ، دلیلش اینه که در اصل floatو Single یکی ان و هر دوشون در نهایت تبدیل میشن به ساختار System.Single

و البته بعضی چیزا اصلا دست ما نیست ، مثلا طراح) C# آقای(Anders Hejlsberg
بنا به دلایلی نخواسته تو C# بتونیم interface ای با توابع static داشته باشیم ، پس هیچ کاری در این مورد نمیشه کرد یا بر عکس میتونیم از pointer ها در C# استفاده کنیم فقط به این دلیل که طراح اون ، خواسته همچین قابلیتی توی زبان باشه.

MSILیا Microsoft Intermediate Language ، زبانیه که کد برنامه های ما در هر زبان از.NET. که باشن به اون تبدیل خواهند شد ، یعنی اینطور نیست که برنامه هامون رو که کامپایل میکنیم مستقیما به زبان ماشین تبدیل بشن (مثل Pascal , C , C++) بلکه کد ما اول به MSIL تبدیل میشه و بعدا توسط (Just In-Time Compiler) JIT به زبان ماشین تبدیل میشه.

البته روند کامپایل و اجرای برنامه ها در .NET. پیچیده تر از این حرفاست و جلسه بعد کاملا توضیح میدمش ، علاوه بر اون جلسه بعد یه سری نکته میگم و یه کتاب هم معرفی میکنم.

فکر کنم دیگه کافیه ،
امیدوارم چیزی یاد گرفته باشین. اگه از روند آموزش یا مطالب ارائه شده یا حجم مطالب نا راضی هستین ، میتونین پیشنهاد بدین تا تغییر بدمشون.
آدرس من : mehdideveloper@gmail.com
موفق باشید .

عنوان: **Compiling .NET Programs(مقاله ۲)**

بسم الله الرحمن الرحیم
این مقاله ، دومین مقاله در بخش C# و آخرینش در سال ۸۳ است.
امروز میخوام یه سری چیز بگم ، بعد چند نکته و بعدشم معرفی کتاب. تو این مقاله روند کامپایل و در مقاله بعدی ، روند اجرای برنامه های .NET.رو میگم.

روند کامپایل برنامه ها در .NET ، حالا در هر زبانی که باشه به این قراره:
اول که شما کد برنامه تون رو توی یه ویرایشگر مثل Notepad یاVS .NET
می نویسید و اون رو ذخیره میکنید تو یه فایل ، بعد نوبت به کامپایل کردن
این کد میرسه که توسط یه ابزار Command-Line) یا به قولی "صفحه سیاه("
به نام (csc.exe (CSharp Compiler صورت میگیره که در مسیر زیر میتونین
پیداش کنین:
[Windows Drive]:\WINDOWS\Microsoft.NET\Framework\[version]\csc.exe
csc)برای سی شارپه ، برای زبانهای دیگه باید از کامپایلر مخصوصشون استفاده
کنین ، مثلا برای VB .NET از vbc.exe یا برای #J از(vjc.exe
البته وقتی شما از IDE های پیشرفته ای مثل Visual Studio .NET یا
C# Builderبرای کامپایل پروژه هاتون استفاده میکنین ، اونا هم در اصل و
به دور از چشم شما ، از csc.exe استفاده میکنن.
کامپایلر پس از بررسی فایل ایجاد شده توسط شما ، اگه فایلتون اشکال نحوی
(syntax error)و همچنین Reference های غیر معتبر نداشت ، یک فایل با
پسوند dll یا exe) netmodule بسته به آرگومانهای کامپایلر ؛ البته پیش فرض
اینها exe است که همون فایلهای اجرایی خودمونه(
فرض میکنیم اسم فایلمون هست MyFile.cs ، و میخوایم از اون یه فایل اجرایی
بسازیم ، پس مینویسیم:

```
csc MyFile.cs
```

حالا یه فایل اجرایی با نام MyFile.exe توی مسیر جاری ایجاد میشه.
)اگه خواستیم یه کتابخونه بسازیم ، می نویسیم:

```
csc /target:library MyFile.cs (
```

اگه خواستیم اسم فایل خروجی رو تغییر بدیم ، از سویچ out/ استفاده
می کنیم.
مثلا دستورcsc /out:FirstApp.exe MyFile.cs
باعث ایجاد یه فایل اجرایی با نام FirstApp.exe خواهد شد.
)من همه سویچ ها رو نمیگم ، برای مشاهده لیست کامل سویچ ها و نحوه
استفاده از کامپایلر #C ، توی MSDN ، عبارت "C# Compiler Options" رو
Searchکنین).
برای این مقاله و مقاله بعدی از مثال زیر استفاده می کنیم:
Notepadرو باز کنین و کد زیر رو توش تایپ کنین :

```
using System;
public class MyClass
{
public static void Main()
{
Console.WriteLine("Hello World");
Console.ReadLine();
}
}
```

و به عنوان فایل C:\MyCS.cs ذخیره اش کنین. حالا نوبت به کامپایل این فایل
میرسه ، اول برید
Start -> All Programs -> Microsoft Visual Studio .NET 2003
-> Visual Studio .NET Tools
و Visual Studio .NET 2003 Command Prompt رو اجرا کنین.
)اگه Visual Studio .NET رو ندارین ، برین Start -> Run و تایپ کنینcmd
حالا برین به مسیر کامپایلر (پوشه حاوی کامپایلر((
خب حالا تایپ کنین csc /out:C:\MyCS.exe C:\MyCS.cs و Enter رو بزنین.
اگه همه مراحل رو درست رفته باشین ، (و کامپایلر هیچ error ی نده) ، الان
یه فایل به نام MyCS.exe توی درایوC تون ایجاد شده که با اجرای اون عبارت
"Hello World"چاپ شده و منتظر گرفتن یه کلید از شما می مونه.
همین کارها رو برای کد زیر که در VB .NET نوشته شده انجام بدین (اسم
فایل MyVB.vb : ، کامپایلر vbc : و دستور(vbc /out:MyVB.exe C:\MyVB.vb :

```
Imports System
Public Module MyModule
```

```
Sub Main
("Console.WriteLine("Hello World
()Console.ReadLine
End Sub
End Module
```

منظورم از این کار ، مقایسه فایلهای تولید شده است که یکی در VB و دیگری در #C نوشته شده و هر دو یک کار رو انجام میدن (لطفا هر دوشون رو ایجاد کنین ، چون بعدا لازمشون داریم ؛ البته میخواستم همین برنامه رو در #J و++C هم درست کنیم ، ولی دیدم حجم مقاله خیلی زیاد میشه(
ما به برنامه های تحت .NET. میگیم 'Managed' یا مدیریت شده ، یعنی تحت نظارت و مدیریت یک چیزیه ، اسم این چیز هست "CLR" :
یکی از تفاوتهای برنامه های managed با) unmanaged مثل ++C و Pascal و(...
در کدیه که کامپایلر ها تولید می کنن ؛ وقتی یه برنامه با C می نویسیم و اون رو کامپایل می کنیم کامپایلر C یک فایل اجرایی تولید می کنه که شامل دستورالعمل های زبان ماشینه و ما مستقیما میتونیم اون رو اجرا کنیم (نیاز به نصب هیچ چیز اضافی نیست) اما وقتی یه برنامه با #C یاVB .NET می نویسیم ، به جای زبان ماشین ، کد شما به زبان MSIL که نوعی زبان اسمبلیه ، تبدیل میشه. و برای اجرای اون روی یه کامپیوتر دیگه ، باید رو اون سیستم .NET framework. نصب باشه تا کاربر بتونه برنامه ما رو اجرا کنه.
)در واقع برنامه های managed هنگام اجرا هم یک بار کامپایل میشن ، که در مقاله بعد روند اجرای اون ها رو هم توضیح میدم(.
حالا به عنوان آخرین کار امروز ، میخوام شما رو با یکی از مفیدترین و بهترین ابزاری که تو عمرم بهش بر خوردم(البته پس از (Notepad آشنا کنم:
) ILdasmمخفف (Intermediate Language Diassassembler که برنامه های نوشته شده تحت .NET. رو براتون تشریح میکنه (کدشون رو بهتون نشون میده ، یعنی میتونین بفهمین برنامه نویس از چه الگوریتم هایی برای نوشتن برنامه اش استفاده کرده ، منتها فقط به زبان(IL برای پیدا کردن این برنامه ، برین به پوشه ای که .NET. Visual Studio رو نصب کردین ، حالا برین به اینجا:
Bin -> v1.1 -> SDKو برنامه ildasm.exe رو اجرا کنین ،
حالا از منوی File برنامه ، Openرو بزنین و فایل C:\MyFile.exe رو باز کنین.
الان یه مستطیل آبی رنگ با سه تا شاخه به سمت راست می بینین ، که کنارش نوشتهMyClass علامت مثبت کنارش رو بزنین تا باز بشه ، و مربع صورتی رنگ با نام Main رو دوبار کلیک کنید تا یه صفحه جدید باز بشه ؛ حالا در این صفحه ، کدی که شما نوشته بودین به زبان IL تبدیل شده و شما میتونین ببینینش ، (نگران نباشید اگه چیزی ازش نمی فهمین ، قرارم نیست شما IL رو بلد باشید ، فقط همینجوری بخونیدش تا خط آخر (
حالا فایل C:\MyVB.exe رو هم با این برنامه باز کنین ، و همین مراحل رو در اون انجام بدین ، می بینین که دقیقا همون کد هم در این فایل وجود داره ، پس یه نتیجه خیلی مهم می گیریم و اون اینکه دیگه مثل قدیما نیست که بگیم فلان زبان بر اون یکی برتری داره ، چون در دات نت ، همه زبانها در نهایت به IL تبدیل میشن و انتخاب زبان برنامه نویسی صرفا یه موضوع شخصیه.
حالا که چشماتون قلمبه شده ، میخوام یه برنامه بی نظیر معرفی کنم که نه تنها کد IL برنامه ها ، بلکه کد .NET. VB و #C و Delphi اون ها رو هم بهتون میده
برنامه Reflector که میتونین از اینجا دانلودش کنین :
http://www.aisto.com/roeder/dotnet
دیگه این رو توضیح نمیدم ، خودتون باهاش ور برین ، ساده است.
)البته اینو بگم که با وجود بی نظیر بودن این برنامه ، به هیچ وجه نمیتونه جای ildasm رو پر کنه خود من فقط از ildasm استفاده می کنم ؛ دلیلش رو خودتون پس از چند بار استفاده از هردوشون می فهمین(.

و اما نکته ها :
نکته ۱ : برای ارسال ایمیل از برنامه هاتون:
System.Diagnostics.Process.Start("mailto:[address]");

معرفی کتاب:
– 1آموزش #C در ۲۱ روز
ترجمه : پریسا گوهری
انتشارات نص
552صفحه ، ۳۵۰۰ تومن

من این کتاب رو نخوندم اما فکر میکنم بهترین کتاب فارسی باشه در این زمینه.

2 – Inside C#
نویسندهTom Archer :
انتشارات Microsoft
این کتاب خوبیه ، خوندمش ، برای کسانیکه آشنایی قبلی با یک زبان برنامه نویسی دارن خوبه.

به نظر من تا جایی که می تونین از کتابهای زبان اصلی و مرجع استفاده کنین تا کتابای ترجمه شده (خود من تا حالا در زمینه .NET. و C# کتاب فارسی نخوندم (.

امیدوارم چیز تازه ای یاد گرفته باشید.
سال نو رو خدمت همه تون تبریک عرض می کنم ، موفق و سر بلند باشید.
تا سال آینده ، خدا نگهدار.

عنوان: **Running .NET Programs(مقاله ۳)**

بسم الله الرحمن الرحیم
سلام ، این هم اولین مقاله در سال جدید ، امیدوارم که سال خوبی باشه برای همه.
جلسه قبل ، روند کامپایل برنامه ها رو گفتم ، در این درس میخوام روند اجرای برنامه های تحت .NET. رو بگم.
اگه به نظرتون سطح این مقالات بالاست ، بگین تا یه خورده آسونتر و کاربردی ترش کنم؛ در ضمن اگه میخواید در مورد یه موضوعی در C# یا .NET. بیشتر سر در بیارید ، بگین تا در مورد اون بنویسم .

ببینین ، هر برنامه ای در هر زبان برنامه نویسی ، باید یه نقطه شروع (یا (Entry Point داشته باشه تا وقتی ما اون برنامه رو اجرا کردیم ، سیستم عامل از اونجا ، اجرای برنامه رو شروع کنه . (در C/C++ بهش میگن تابع main ؛ در C# میگن(Main
)بعضی ها فکر میکنن مثلا .VB .NET تابع Main رو نداره ، در حالی که در اشتباهن ، در برنامه های Console ، .VB .NETتابع Main داره و در برنامه های ویندوز در .VB .NET و VB6 هم تابع main وجود داره ولی قابل دیدن نیست و توسط کامپایلر به طور خودکار در فایل نهایی گنجونده میشه(
البته فایلهای کتابخانه (DLL) هم یه نقطه شروع دارن ، که معمولا از دید برنامه نویس پنهان)بیشتر به درد برنامه نویسای حرفه ای C++ میخوره(
خب ، حالا چطوری سیستم عامل ، برنامه ما رو اجرا میکنه ؟
در یک برنامه تحت .NET. ، خود سیستم عامل قادر به اجرای مستقیم برنامه نیست و نیاز به یک محیط اجرا (runtime) داره به نام CLR که جزء اصلی .NET framework. هست.
) .البته فعلا اینطوریه ، ولی قراره از Windows Longhorn ،همه ویندوز ها همراه با .NET. باشن .بعضی از دوستام میگن Windows 2003 هم net. داره ، ولی من خودم ندیدم(.
سیستم عامل قبل از اجرا ، یه بخش مخصوصی رو در فایل چک میکنه ؛ در برنامه های تحت net. این بخش اشاره میکنه به فایل MSCoree.dll در پوشه System32
)این فایل ، مغز .NET. است ، و در واقع اگه در یه سیستم در پوشه System32 این فایل رو دیدید ، به احتمال ۹۹,۹۹۹۹ درصد ، .NET.روی اون سیستم نصب شده)
حالا یه تابع در این فایل به نام _CorEXEMain اجرا میشه ، این تابع هم CLR رو اجرا میکنه و سپس CLR هم میره entry point یا همون تابع Main برنامه رو اجرا میکنه.
خب حالا نوبت به اجرای کد شما میرسه ، کدی که در زبان برنامه نویسی مورد علاقتون نوشته بودین حالا تبدیل شده به کد IL ، ولی کد IL باید تبدیل بشه به زبان ماشین CPU تا قابل اجرا باشه ، اینجاست که یه نفر به اسم (JIT (Just In Time Compiler می یاد و کدهای IL رو کامپایل میکنه به زبان ماشین و CPU هم اونا رو اجرا میکنه.
دقت کنین که JIT هر تابع رو فقط یه دفعه کامپایل میکنه ، یعنی اگه شما یه تابع داشته باشین

که هزار بار فراخوانی میشه در کدتون ، فقط در اولین فراخوانی کد اون کامپایل میشه به زبان ماشین و در دفعات بعدی (یعنی ۹۹۹ بار بعد) ، به جای کامپایل مجدد ، همون کد کامپایل شده دفعه اول که در حافظه ذخیره شده بود ، اجرا میشه و این یعنی سرعت بالا.
)در JAVA هر دفعه که تابعی اجرا میشه ، کد اون کامپایل یا تفسیر میشه ، برای همینم کنده(
اما سوالی که ممکنه برای شما پیش بیاد ، اینه که اصلا چه نیازی هست به این که برنامه ها به IL تبدیل بشن ؟ چرا مستقیم به زبان ماشین تبدیل نشن تا سرعت کار بالاتر بره ؟
جواب : مزایای این کار عبارتند از
JIT – 1هنگام کامپایل IL به زبان ماشین ، میتونه CPU ی سیستم رو تشخیص بده و کد مخصوص به اون CPU رو تولید کنه تا سرعت اجرا بالاتر بره.
JIT – 2میتونه بسته به سیستمی که برنامه در اون اجرا میشه ، کد رو بهینه (optimize) کنه.
IL – 3مستقل از محیط و ریز پردازنده است ؛
شما اگه با C++ یه برنامه برای CPU های X86 بنویسین ، رو بقیه CPU ها نمیتونین از اون برنامه استفاده کنین ولی در .NET. اینطور نیست ، همچنین اگه با C++ یه برنامه گرافیکی برای ویندوز بنویسین ، تحت لینوکس اجرا نخواهد شد ، چونکه موتور گرافیکی شون با هم فرق میکنه ، اما تحت .NET. از این مشکلا نداریم ، یه کتابخونه گرافیکی هست مخصوص .NET. ، حالا هر سیستم عاملی که باشه ، به ما ربطی نداره و اون دیگه مشکل خود دات نته .
در واقع الان .net framework. برای سیستم عامل های زیر موجوده:
Windows 98/ME/2000/XP/2003 , Linux (SUSE , Redhat) , UNIX Free BSD , Windows CE
و احتمالا برای Macintosh هم یکی درست کنن (شایدم کردن و ما خبر نداریم(.
– 4در حین کامپایل برنامه های تحت .NET. ، CLRکد رو چک میکنه که به این عمل میگن Verification؛ مثلا چک میکنه که:
هر تابع با تعداد درست پارامترها فراخوانی شده باشه.
از هیچ خانه حافظه ای چیزی خونده نمیشه ، مگر اینکه قبلا چیزی توش نوشته شده باشه
هر متدی یک return داشته باشه.
و خیلی چیزای دیگه ، که باعث میشه برنامه هامون قابل اطمینان و قوی باشن.

در ضمن زیاد نگران سرعت نباشین ، هر کدی فقط یک بار کامپایل میشه و دفعات بعدی از همون کامپایل شده هه ، استفاده میشه که این چندان چیزی نیست.

خب اینم از این ،
امیدوارم به دردتون خورده باشه.
لطفا هر پیشنهاد ، انتقاد ، نظر و ... که دارین بگین تا کیفیت کار بالاتر بره.
موفق باشین.

عنوان: نکاتی برای همه(مقاله ۴)

به نام خدا
سلام
خب از این مقاله دیگه میریم سراغ جنبه های کاربردی تر .net ؛ امروز میخوام یه مبحث خیلی مهم رو بگم که دونستنش واسه هر برنامه نویس دات نت لازمه .

اول از همه باید اینو بدونین که هر چیزی در .net. از کلاس System.Object مشتق میشه)اشتقاق یا inheritance یا ارث بری ، یعنی یک کلاس متد ها و ویژگی های یک کلاس دیگه رو به ارث ببره و چون مثلا کلاس Object یه متد عمومی داره به اسم ToString پس تمامی کلاس ها و struct ها و خلاصه هر چیزی ، یه متد ToString داره ؛ البته لازم نیست حتما یه کلاس مستقیما از Object مشتق بشه ، مثلا اگه کلاس B از کلاس A مشتق بشه ،در اون صورت از کلاس System.Object هم به طور غیر مستقیم مشتق میشه ، چونکه A از System.Object مشتق میشه(
در .net. یک کلاس فقط از یه کلاس دیگه میتونه مشتق بشه ، اما در عوض میتونه از هر چندتا Interfaceکه میخواد ، مشتق بشه.

Assemblyو: Namespace
یک اسمبلی در .NET. ، کوچکترین واحد مستقله که میتونه نصب بشه ، به عنوان مثال یه فایل DLLیا exe که ایجاد می کنیم در .NET. ، یه اسمبلیه. (البته اسمبلی ها میتونن چند فایلی

هم باشن ، اما معمولا ما با اسمبلی های تک فایلی سر و کار داریم(.
اما) namespace فضای نام) یه جور جداسازی منطقی بین کلاسهاست ، مثلا ما تموم کلاسها و enum های مربوط به کار با فایلها رو در فضای نام System.IO داریم یا هر چیز مربوط بهXML داخلSystem.Xml قرار داره که باز داخل اون ، فضای نام های دیگه ای مثل System.Xml.XPath و System.Xml.Schema و ... داریم که این جداسازی کار ما رو راحتتر میکنه ، چون اولا تمام کلاسهای مربوط به هم داخل یک فضای نام قرار میگیرن و این یک جور انسجام به وجود می یاره ، ثانیا ما میتونیم دو تا کلاس همنام با هم ولی در دو فضای نام مختلف داشته باشیم ، که این بدون وجود namespace ها شدنی نیست. (به عنوان مثال کلاس Timer هم توی System.Timers هست و هم توی System.Windows.Forms ، که به دلیل تفاوت در فضای نامشون ، مشکلی ایجاد نمیکنه ، شما هم میتونین در هر فضای نام دیگه ای غیر از این دوتا ، یه کلاس با نامTimer داشته باشید(
بعضی ها فرق بین این دو رو خوب متوجه نمیشن ، اولا بدونین که namespace یه جداسازی منطقیه و assembly یه جداسازی فیزیکی .
ثانیا : ما میتونیم داخل یک اسمبلی چندین namespace داشته باشیم ، به عنوان مثال اسمبلی System.dllشامل namespace های Microsoft.CSharp و System.CodeDom و چند تای دیگه است که کاملا داخل این اسمبلی هستن و بالعکس کلاسهای یه namespace میتونن داخل چند اسمبلی جدا از هم قرار داشته باشن ، مثلا قسمتی از System.IO داخل System.dll و قسمت دیگرش داخل mscorlib.dll است (اصلا فایل (اسمبلی System.IO.dll (وجود نداره و کل کلاسهای System.IOداخل System.dll و mscorlib.dl قرار دارن(
)اینجاست که متوجه فیزیکی و منطقی بودن میشید ، در اصل
چیزی به نام namespace وجود خارجی نداره و فقط یک مفهومه اما اسمبلی وجود خارجی داره و همین System.dll و mscorlib.dll که در فولدر [version]\Microsoft.NET\Windows\ قرار دارن هر کدوم یه اسمبلی ان(.
وقتی ما توی C# میگیم

```
using System.IO;
```

یا در VB .NET میگیم:

```
Imports System.IO
```

در واقع میخوایم مستقیما از کلاسهای داخل System.IO استفاده کنیم ، یعنی اگه نیاز به کلاس System.IO.File داریم ، نیاز نیست این همه تایپ کنیم ، کافیه از اسم File استفاده کنیم.

```
File.Create("C:\\myFile.txt");
```

در صورت استفاده نکردن از (using (Imports ، باید مسیر کامل کلاس رو تایپ می کردیم:

```
System.IO. File.Create("C:\\myFile.txt");
```

حالا اسمبلی کجاست ؟ در پروژه تون ، برید به قسمت Solution Explorer و داخل References رو نگاه کنید ، اونجا اسمبلی هایی رو که استفاده کردین ، مشاهده می کنین و میتونین اونا رو اضافه یا کم کنید.
برای ایجاد یک: namespace
سی شارپ:

```
namespace Mehdi
{
class MyClass {...}
enum MyEnum {....}
}
```

VB :

```
Namespace Mehdi
Class MyClass
...
End Class

Enum MyEnum
...
End Enum
End Namespace
```

حالا برای استفاده از کلاس MyClass دو راه داریم:
راه اول :

```
Mehdi.MyClass mc = new Mehdi.MyClass();
// Dim mc As New Mehdi.MyClass()
```
راه دوم :
```
using Mehdi ; // Imports Mehdi
...
...
MyClass mc = new MyClass();
// Dim mc As New MyClass ()
```

چطور اسمبلی درست کنیم ؟
وقتی یه پروژه ویندوز با نام مثلا WindowsApplication1 درست می کنین و اون رو Build یا کامپایل می کنین ، یه فایل تو پوشه bin\Debug ایجاد میشه با نامWindowsApplication1.exe که این فایل یه اسمبلیه و برای دیدن محتویاتش میتونین از ildasm استفاده کنین (رک درس ۲( )فایلهای dll یا Class Library هم همینطور)
)معمولا نام یک اسمبلی با نام فایلش یکیه ، یعنی فایل abc.exe شامل اسمبلی abc است ، گرچه میتونین نام فایل رو تغییر بدین ، اما کار درستی نیست ، تو بعضی موارد به مشکل بر میخورین به جاش ، اسم اسمبلی تون رو به اسم دلخواه تغییر بدین و دوباره برنامه رو Build کنین تا نام فایلتون هم همون بشه ؛
برای تغییر نام اسمبلی:
در محیط Visual Studio .NET از منوی Project گزینه Properties رو انتخاب کنین و در همون صفحه اول که می یاد ، سمت راست بالا ، گزینه Assembly Name رو تغییر بدین(
خلاصه:

– 1کلاس System.Object مادر تمام کلاسها در دات نته.
– 2در NET. هر کلاس ، فقط و فقط از یه کلاس دیگه میتونه ارث ببره ، اما در مورد ارث بری از interfaceها محدودیتی وجود نداره.
– 3اسمبلی ، جداسازی فیزیکی کلاسها و namespace ، جداسازی منطقی اونهاست.

نکته شماره ۲:
اگه میخواین بدونین net. چطور درست شده ، و میخواین بیشتر ازش سر در بیارین یا تحقیق کنین)یا شایدم خودتون یکی بسازین) ، بهتره بدونین الان دو تا منبع خوب وجود داره تو دنیا که کد اون رو هم بهتون میدن (البته اونا همین net. که ما استفاده می کنیم نیستن ، چون این یه نسخه تجاریه ، اما برای مصارف آموزشی و تحقیق فوق العاده ان چون شباهت زیادی به نسخه تجاری دارن(

Shared Source CLI – 1یا Rotor که مایکروسافت اونو درست کرده و اون طور که گفته میشه ، شباهت خیلی زیادی به نسخه تجاری داره و بزرگترین مجموعه کدیه که تا به حال توسط مایکروسافت منتشر شده و فقط بر روی سیستم عاملهای Windows XP و UNIX Free BSD قابل استفاده است. و شامل کد کامپایلر های #C و Jscript و یه سری ابزار مثل ildasm و خیلی چیزای دیگه است و شامل چند میلیون خط کده که اکثرا در C و ++C نوشته شدن.
مسیر دانلود:
http://msdn.microsoft.com/net/sscli
حجم فایل زیپ شده : MB ۱۵,۷

Mono – 2که یه نسخه رایگان از دات نته و برای Linux Redhat , SUSE , Netware , Windows , ساخته شده.
مسیر دانلود :
http://www.go-mono.com/
خود من فقط اولی رو دانلود کردم ، چون مال خود مایکروسافته و مطمئنا شباهت بیشتری داره به net.ی که ما استفاده می کنیم.

عنوان: **(مقاله ۵)(net. انواع در) Types in .NET**

سلام
این مقاله هم مثل مقاله های قبلی ، فعلا ربطی به زبان #C نداره و مربوط به خودnet framework. میشه و علت اینکه در قسمت #C آوردمش ، اینه که مثالها رو به #C مینویسم و ثانیا ، زبان اصلی خودم سی شارپه و ثالثا ، بعدها میخوام برم سراغ خود زبان سی شارپ.
اما این مطالب پایه است برای دات نت ، حالا هر زبانی که باشه و من میخوام
اول مفاهیم زیرین رو کامل درک کنیم بعد بریم سراغ خود زبان و تا اون موقع ، تمامی برنامه نویسای net) .در هر زبانی VB .NET : و #C و #J و Delphi .NET و ...) میتونن از این مطالب استفاده کنن.
بریم سر اصل مطلب:
در net. ، تمامی موجودات ! به دو دسته کلی تقسیم میشن:
Reference Typesو Value Types که تو این مقاله اونا رو به اختصار ، RTو VT مینامیم.
کلاسها (کلاسهای معمولی ، string ، آرایه ها ، interfaceها و delegate ها) از نوع RT هستن و ساختارها (struct) و انواع داده ای معمولی (مثل int و bool و (double و enum ها) از نوع. VT
RTها مستقیما از کلاس System.Object مشتق میشن ، اما VT ها به جای System.Object از System.ValueTypeمشتق میشن (که اونم از System.Object مشتق میشه(
خب ، حالا فرق این دو در چیه ؟
ما وقتی یک شیء از نوع) RT مثلا یه کلاس یا آرایه) تعریف می کنیم (با کلمه (new ، CLRباید هنگام اجرای اون قسمت ، در قسمت خاصی از حافظه به نام heap ، مقدار حافظه لازم برای اون متغیر رو ایجاد کنه و یه سری کارای داخلی هم انجام میده ؛
در حالی که هنگام تعریف یه متغیر از نوع) VT مثلا یه (char ، نیازی به عملیات ویژه نیست ، بلکه در قسمت خاصی از حافظه به نام stack ، اون متغیر رو ایجاد میکنه ؛
ایجاد و تخصیص و انتساب حافظه در stack ، خیلی سریعتر از heap انجام میشه ، در ضمن ، GC Garbage Collection)یا جمع آوری زباله (هم فقط بر روی heap انجام میشه.
GC)یکی از ویژگیهای خیلی خوب و مهم دات نته و به این معنیه که پس از ایجاد متغیر ها و وقتی که کارمون باهاشون تموم شد ، CLRخودش حافظه اونا رو آزاد میکنه و در اختیار Runtime قرار میده تا بازم بشه از اون مکانها استفاده بشه. به این میگن مدیریت خودکار حافظه که کار برنامه نویس رو خیلی راحت میکنه (البته چیز جدیدی نیست ، VBهم مدیریت خودکار حافظه داره ، و Java هم یه GC داره که البته هیچ کدوم از نظر سرعت و الگوریتمهای به کار رفته ، به پای net GC. نمیرسن.
اما برنامه نویسای ++C مجبور بودن اشیایی رو که با new ایجاد شده بودن با delete پاک کنن و در صورت فراموش کردن اینکار ، یا انجام دادن اون بیش از یه دفعه ، برنامه شون مشکل پیدا میکنه (

تفاوتها و شباهتها:
– 1کار با RT ها راحتتره ، چون مثلا وقتی یه RT رو مساوی اون یکی قرار میدیم ، یا مثلا به عنوان پارامتر یه تابع ارسال میکنیم ، فقط آدرس حافظه اون کپی میشه ، یعنی:

```
ArrayList arr1 = new ArrayList();
arr1.Add(10);
arr1.Add(20);
ArrayList arr2 = arr1;
arr2[0] = 36;
```

در اینجا دو شیء داریم که هر دو به یه مجموعه از داده ها اشاره می کنن و تغییر در هر کدوم ، بر روی اون یکی تاثیر میذاره . در آخر ، عضو اول هر دو متغیر میشه ۳۶.
اما در VT ها ، یه کپی از متغیر ایجاد میشه و اونه که ارسال میشه ، و در صورت زیاد بودن حجم اون ، سرعت برنامه تحت تاثیر قرار میگیره.(معمولا VT ها زیر ۱۶ بایت هستن(
مثال :

```
int a = 10;
int b = a;
b = 20;
```

پس از اجرای این قطعه کد ، مقدار متغیر b میشه ۲۰ و مقدار a همون 10 باقی میمونه ، چون در خط دوم ، فقط یه کپی از متغیر a در b قرار میگیره.
پس تغییر در یکی از این دو نوع VT ، اون یکی رو تغییر نمیده.
RT – 2ها نیاز به مدیریت حافظه و GC و غیره دارن و این یعنی ایجاد و از بین بردن یک RT ، کندتره نسبت به VT ها (البته این کندی که میگم در حد صدهزارم ثانیه است(.

درسته که بیشتر با RT ها سر و کار داریم ، ولی VT ها هم خیلی مفیدن ، فکر کنید اگه با هر تعریف یک int ، بخواد یه عمل تخصیص حافظه صورت بگیره ، چی میشد ؟؟؟
3- شیئ گرایی فقط در مورد RT ها صدق میکنه ، یعنی inheritance و اینا در مورد VT ها معنی نداره ، ما نمیتونیم یه struct رو از یه کلاس یا struct دیگه مشتق کنیم.
4- فقط RT ها میتونن null باشن ؛ int i = null;) معنی نداره(.

در جلسه بعد ، درباره Boxing و UnBoxing و چند نکته جالب از .NET. و C# ، میخوام صحبت کنم.

نکته شماره ۳ :
برای عملیات معمولی بر روی رشته ها ، کلاس String ، عالیه ، ولی در صورتی که عملیاتی مثل Replace و Insert و Remove یا مثلا چسبوندن و تغییر بعضی کاراکترهای داخل رشته رو زیاد (مثلا بیش از ۵ بار) میخواید انجام بدین ، پیشنهاد میکنم از System.Text.StringBuilder استفاده کنین ؛ چون هر بار که یه رشته از نوع string رو تغییر میدین ، رشته قبلی از بین رفته و یه رشته جدید به وجود می یاد . مثلا:

```
string s = "Hello";
s = s + " World";
```

اینجا تو خط دوم ، متغیر قبلی پاک شده و یه متغیر جدید ایجاد میشه و در صورتی که زیاد میخواین یه رشته رو تغییر بدین ، StringBuilderخیلی سریعتره چون رشته جدید ایجاد نمیکنه و با همون رشته اولیه ، کار میکنه. (از این کلاس ، بیشتر با دو متد Append و ToString سر و کار خواهید داشت ، دیگه اونا رو توضیح نمیدم ، خودتون یاد بگیرین(

معرفی کتاب:
C# and VB .NET Conversion Pocket Reference
انتشارات O'Reilly ، سال ۲۰۰۲
144صفحه
مولفJose Mojica :
این کتاب درباره تفاوتها و شباهتهای دو زبان C# و VB .NET بحث میکنه ، به درد کسایی میخوره که میخوان فقط یکی از این دو زبان رو انتخاب کنن و شک دارن.

امیدوارم این مقاله چیز جدیدی براتون داشته باشه.

عنوان: **Boxing and UnBoxing(مقاله ۶)**

به نام خدا
خب اینم اولین مقالهٔ تابستونی ما ، ان شاالله از این به بعد هر هفته یک مقاله جدید خواهیم داشت.
امروز میخوایم دربارهٔ Boxing و UnBoxing صحبت کنیم ، یکی از مفاهیم مهم در .NET. که به درد همه برنامه نویسا می خوره ،
اگه یادتون باشه ، تو مقالهٔ قبلی Reference و Value Type ها رو توضیح دادم ؛ برای فهمیدن مطالب این مقاله باید مطالب مقاله قبل رو کامل بلد باشید.
Boxingاز نظر لغوی ، معنی "بسته بندی" یا "داخل جعبه گذاشتن" رو میده و تعریف برنامه نویسیش میشه" : هرگاه یک متغیر از نوع RT را مساوی یک مقدار یا متغیر Value Type قرار دهیم ، عمل Boxing صورت می گیرد. عکس این عمل ، یعنی تبدیل یک RT به یک VT را UnBoxing گوییم".
)البته من گفتم RT ولی بیشتر با Object سروکار داریم (.

مثال:

```
int myVT = 10 ;
object myRT = myVT ;
```

```
myRT = 20 ;
```

در اینجا یک متغیر از نوع System.Int32 نوع عددی ۴ بیتی) ایجاد کردیم با مقدار اولیه ۱۰)که یک VT هست(و سپس یک متغیر از نوع System.Object که یک RT هست ایجاد کردیم و مساوی متغیر قبلی قرار دادیم. حالا متغیر myVT بر روی Stack ایجاد شده ، اما متغیرmyRT بر روی Heap ؛ یعنی myVT مستقیما حاوی مقدار ۱۰ هست ، اما myRT به خونه ای از حافظه اشاره میکنه که حاوی مقدار ۱۰ هست ؛ در خط دوم عمل Boxing صورت گرفته ؛
حالا سوال اینه که آیا در خط سوم ، مقدار myVT هم تغییر کرده یا نه ؟
نه ، چرا ؟ چون myRT به خونه ای از حافظه اشاره میکنه که مقدار ۱۰ رو داره و حالا هم مقدار اون خونه رو کرده ۲۰ ، پس myVT دست نخورده باقی مونده.
خب دیدیم که در عمل Boxing ، نیاز به تبدیل نوع (Casting ; Conversion) نیست ، یعنی نیاز نیست
بنویسیم :

```
object myRT = (object)myVT ;
```

چونکه همونطور که میدونین نوع object ، مادر تمام کلاسها در دات نته ، بنابراین میتونه هر چیزی رو در خود نگه داره.
خب حالا اگه فرضا بخواهیم مقدار موجود در myRT رو بدونیم و داخل یک متغیر int قرار بدیم ، باید عمل UnBoxing انجام بشه ،

```
int b = (int) myRT ;
```

حالا متغیر b حاوی مقدار موجود در) myRT مثلا ۲۰ (است.
نکتهٔ مهم : برای UnBoxing باید عمل تبدیل نوع انجام بگیرد.
)برای کسانی که C# بلد نیستن :
(int)یعنی مقدار بعد از من رو به نوع int تبدیل کن و برگردون ؛ در واقع عملگر () ، علاوه بر فراخوانی
متد ها ، در تبدیل نوع هم استفاده داره. در VB .NET به جای (int) myRT ، از CInt(myRT) استفاده
کنین(.
در خط بالا اول CLR چک میکنه ببینه myRT درون خودش یک int داره یا نه ، وگرنه یک خطا از نوع InvalidCastExceptionاتفاق می افته .
پس قبل از UnBoxing باید مطمئن بشیم که از یک نوعن.
برای این کار از کلمه کلیدی is استفاده می کنیم :

```
int b = 0;
if(myRT is int)
b = (int)myRT;
```

در این کد اگر myRT حاوی یک متغیر از نوع int باشه ، bمساوی اون مقدار ، وگرنه مساوی صفر خواهد
بود.
)البته بعدها یک مقاله مفصل و کامل دربارهٔ تبدیل نوع داده ها به همدیگه و کلمات کلیدی as و is و عملگر () و کلاس Convert و متد Parse و ... می نویسم(.

اینم از این مقاله.
نمیدونم آیا تونستم مطلب رو خوب بفهمونم یا نه ، لطفا اگه وقت کردید نظرات و انتقادات خودتون رو

قرار بدین تا بتونم مقالات بهتری ارائه بدم

نکتهٔ شمارهٔ ۴:
اگه زیاد با فایل های Html ، XMLو ASP .NET Web Forms کار می کنید ، مطمئنا تجربه کردین که در
فایلهای حجیم ، چقدر پیدا کردن یه قسمت خاص مشکله و چه خوب میشد اگر یک نمای منطقی و مرتب از تمامی تگ ها (و کنترل ها) در دست داشتین ، خب چارهٔ کار اینه:
در Visual Studio .NET از منوی View ، گزینهٔ
Other Windows -> Document OutLineرو انتخاب کنید. (یا(Ctrl + Alt + T

معرفی کتاب:
Applied Microsoft .NET Framework Programming
نوشتهJeffrey Richter
انتشاراتMicrosoft

تقریبا ۵۰۰ صفحه

این کتاب فقط در مورد net framework. نه زبان ها یا کلاسها ، فقط مفاهیم اساسی(خود) بحث میکنه و فوق العاده است. در واقع نه تنها بهترین کتاب در مورد net. ، بلکه بهترین کتابیه که تا حالا در زمینه های مربوط به کامپیوتر خوندم ؛ خیلی خوب و روون و دقیق با ذکر تمام جزئیات ؛
البته این رو هم بگم که بیشتر روی مفاهیم و اصول کار کرده و بعضی فصل هاش خیلی پیشرفته ان ، اما برای یک برنامه نویس متوسط به بالا که net. کار میکنه ، اولین و آخرین کتابیه که توصیه میکنم بخونه.

موفق باشید.

عنوان: قسمت دوم مقالهٔ شمارهٔ ۶

سلام
تو مقالهٔ ۶ قسمت اول یه نکته رو یادم رفت بگم ، که الان میگم.
اول اینکه تا اونجا که ممکنه سعی کنید در کدتون boxing و unboxing صورت نگیره ، البته unboxing چندان کار
سنگینی نیست ، اما اگه بیش از حد در کدتون boxing صورت بگیره ، میتونه روی سرعت اجرای برنامه تون تاثیر
محسوس داشته باشه Managed C++ .یه چیز جالبی داره و اونم اینه که از برنامه نویس میخواد که عمل boxing
رو به طور آشکار با کلمهٔ کلیدی box__ ، اعلام کنه ، به کد زیر
توجه کنید:

```
C# : //
int i = 45 ;
Console.WriteLine("I'm {0}" , i);

MC++: //
int i = 45;
Console::WriteLine("I'm {0}",__box(i));
```

در این دو کد ، میخوایم سن شخص رو چاپ کنیم ، منتها سن در متغیر i قرار داره (قسمت {۰} داخل رشته ، هنگام
اجرا با محتویات i عوض میشه ، {۰} یعنی اولین آرگومان ؛ بعد ها یه مقاله دربارهٔ اینا (Format String) قرار میدم(.
از طرفی متد WriteLine از کلاس Console ، تمام آرگومانها رو از نوع object در نظر میگیره؛ یعنی عددی که (i)میخوایم
به عنوان آرگومان به این متد ارسال کنیم ، ابتدا box میشه ، سپس متد ToString اون فراخوانی میشه تا مقدار رشته ایش
به عنوان آرگومان و به جای {۰} ارسال بشه ، (اگه i از نوع String بود ، عمل boxing صورت نمی گرفت)
خب توی C# و VB .NET ، کامپایلر خودش برامون عمل boxing رو انجام میده ، خوبی این کار اینه که

برنامه نویس کارش
راحتتر میشه و سریعتر کد می نویسه ، اما چند درصد برنامه نویسا میدونن کجا ها عمل boxing صورت می گیره ؟
خداییش چند نفر از شما (اونایی که MC++ بلد نیستن) ، میدونستن ؟
اما در MC++ ، شما باید با کلمهٔ کلیدی __box این رو به کامپایلر صریحا بگید وگرنه برنامه تون کامپایل نمیشه.
خب حالا قدم آخر :
یه پروژهٔ Console Application در) C# یا (VB .NET ایجاد کنید و این دو خط کد رو در اون بنویسید (در متد Main(و برنامه
رو کامپایل کنید ؛
Ildasmرو که یادتون نرفته ، این بهترین دوستم رو در مقالهٔ ۲ معرفی کردم. برنامهٔ اجرایی حاصل از پروژه تون رو باIldasm
باز کنید ، برید به متد Main از کلاس Class1 و روی آیکون بنفش رنگ Main دوبار کلیک کنید تا کد IL اون باز بشه ؛
خوب نگاه کنید ، توی یکی از خطوط عبارت box رو می بینید ، قبل اون هم نوشته شده) ldloc.0 یعنی متغیر محلی
شمارهٔ صفر (i) رو load کن و بعد box کن .(در IL متغیرها توسط شمارهٔ عددی شناخته میشن ، نه اسمشون((
و این به ما میگه که یک بار عمل box در تابع Main صورت گرفته .

خب حالا یه تمرین برای جلسهٔ بعد ،
در کد زیر چند بار عمل box صورت گرفته ؟

```
DateTime dt = DateTime.Now;
object o = dt;
string s = "Time is: " + dt;
Console.WriteLine(dt);
```

فقط خواهش میکنم به Ildasm نگاه نکنید ، اول خودتون خوب کد رو موشکافی کنید ، بعد با دلیل بگین کجاها و چرا عمل
boxingصورت گرفته ، بعد به Ildasm نگاه کنین. تو مقالهٔ بعد ، جواب این سوال رو با تشریح علت ، میگم .

البته این مقاله فقط به عنوان ضمیمه ای بر مقاله شمارهٔ ۶ نوشته شده ، برای همینم به عنوان ۷ ، قرار نمیدمش.
موفق باشید

عنوان: این درس عنوان نداره. (مقاله ۷)

سلام
خب اول جواب سوالی رو که به عنوان تمرین در مقالهٔ قبل داده بودم ، تشریح می کنم.
در این قطعه کد ۳ بار عمل box صورت می گیره. بار اول وقتیه که متغیر o رو مساوی dt قرار میدیم (خط ۲) ؛ چون o از نوع Reference Type است (Object) و dt از نوع(DateTime) Value Type
پس یک عمل boxing در اینجا صورت می گیره. بار دوم در خط سومه ، وقتی ما یک رشته رو با چیز

دیگری
)مثل یک رشته یا یک عدد یا یک کاراکتر) جمع می کنیم ، در واقع کامپایلر این +"" رو به متد String.Concat
تبدیل میکنه .(کار این متد چسبوندن دو object به همدیگه و برگردوندن رشتهٔ حاصل است.) خب چون آرگومانهای
این متد از نوع object هستن پس یه box هم اینجا داریم و آخرین box هم در خط آخره ؛
ما اگه یه char یا int یا double یا هر نوع اولیه ای رو به عنوان تنها آرگومان متد Console.WriteLine ارسال کنیم
عمل box صورت نمیگیره ، چون متد Console.WriteLine ، overloadشده (یا بار اضافی داده شده) تا هر آرگومان از
این انواع رو قبول کنه ، اما متاسفانه یه همچین آرگومانی برای نوع DateTime وجود نداره ، بنابراین dt رو بهobject
تبدیل کرده و چاپ می کنه .
همهٔ این وقایع رو میتونین توسط ildasm مشاهده کنین .
)نکته برای تازه کار ها :
Method Overloadingیا بار اضافه دادن به متد ها یعنی چند متد با نام یکسان ولی آرگومانهای متفاوت داشته باشیم.
مثلا :

```
void Show(int x)
{
Console.WriteLine("I am the int version");
}

void Show(double x)
{
Console.WriteLine("I am the double version");
}

void Show(long x)
{
Console.WriteLine("I am the long version");
}
```

ما سه متد با نام Show تعریف کردیم که پارامتر متفاوت از هم دارن. مثلا اگه شما متد Show رو با یک مقدار double فراخوانی کنین ،
عبارت "I am the double version" بر روی کنسول چاپ خواهد شد.

```
Show(25);
Show((long)25);
Show(25.0);
```

ما در اینجا به ترتیب نسخهٔ int ، longو double رو فراخوانی کردیم . ۲۵ یک ثابت صحیح هست و چون مقدارش از
) 2147483647بزرگترین مقداری که int میتونه نگه داره) کوچکتره ، از نوع int در نظر گرفته میشه ، در خط دوم ما این ۲۵ رو به نوع long تبدیل کردیم و در خط آخر چون گفتیم ۲۵,۰ ، یعنی یک ثابت اعشاری که به طور
پیش فرض از نوع double در نظر گرفته خواهد شد.
اگه در Visual Studio .NET این کار رو انجام بدین ، موقع فراخوانی Show می بینین که یه کادر مستطیل شیری رنگ
کنار Show باز میشه که سمت راست اون نام تابع و سمت چپ نوشته : ۱ of 3 همراه با دو فلش رو به بالا و پایین.
این یعنی که ما ۳ متد با نام Show داریم و با فلش ها میتونیم اون یکی ها رو هم ببینیم.
مثلا Console.WriteLine ، 19تا Overload داره.
البته نیاز نیست حتما تعداد پارامترها در متدهای overload شده ، یکسان باشه ،
مثال:

```
void Show(int x)
{
Console.WriteLine("I have 1 parameters.");
```

```
}

void Show(int x , float y)
{
Console.WriteLine("I have 2 parameters.");
}
```

بسته به اینکه چند تا آرگومان به تابع ارسال کنین ، متد مربوطه فراخوانی خواهد شد.
دقت کنین که نمیتونیم دو تابع رو فقط بر اساس نوع برگشتیشون overload کنیم ،
به عنوان مثال ، قطعه کد زیر کاملا درسته و کار میکنه :

```
int Add(long x , long y)
{
return x + y;
}

double Add(double x , double y)
{
return x + y;
}
```

اما قطعه کد زیر مشکل داره و کامپایل نخواهد شد :

```
int Show(int x)
{
return 200;
}

long Show(int x)
{
return 400;
}
```

چونکه امضاء (Signature) دو تابع عین همه (امضا یعنی نام تابع و نام و تعداد و نوع پارامترها(

خب حالا برگردیم به اولین مثالمون (سه تابع Show با آرگومانهای int و long و(double
میخوام توسط اون سه تابع ، مفهوم) Conversion Implicit تبدیل ضمنی) رو براتون توضیح بدم.
اگه به تابع Show ، یه آرگومان از نوع byte یا sbyte یا short یا ushort یا int یا char بفرستید ، نسخهٔ intفراخوانی خواهد شد.
مثال :

```
byte x = 10;
csc.Show(x);//"I am the int version"
```

زیرا کوچکترین نوعی که میتونه این نوع ها رو در بر بگیره ، نوع int هستش (البته مسلما long و doubleهم میتونن مقدار
این نوع ها رو در بر بگیرن ، چونکه بزرگتر از int هستن ، ولی ما اینجا با کوچکترین نوع سر و کار داریم(.
حالا اگه به تابع Show ، یک مقدار uint یا long ارسال کنیم ، نسخهٔ long فراخوانی خواهد شد.
و در صورت ارسال یک مقدار ulong یا float یا double ، نسخهٔ double فراخوانی خواهد شد.
)همینجور نخونید مقاله رو ، هر چی میگم عملا اجرا کنین تا ببینین(
البته implicit conversion فقط هنگام ارسال پارامترها اتفاق نمی افته ، به مثال زیر توجه کنید.

```
int i = 2500;
long l = s;
```

در اینجا هم i به نوع long تبدیل خواهد شد ، بدون اینکه ما چیزی بگیم یا کاری بکنیم.
اما مواقعی هم هست که میخوایم مقدار یک متغیر از نوع بزرگتر رو در یه متغیر از نوع کوچکتر قرار بدیم ، اینجا دیگه
باید صریحا تبدیل نوع رو انجام بدیم(Explicit Conversion) .
مثال :

```
int i = 800;
short s = i;
```

این قطعه کد کامپایل نمیشه ، زیرا i از نوع int و ۴ بایتی هست اما s از نوع short و دوبایتی است ، و کامپایلرerror
میده که: implicitly convert type 'int' to 'short' Cannot :
اصلاح شده :
```
int i = 800;
short s = (short)i;
```
البته در این مثال مقدار i ، 800بود و مقدار s هم ۸۰۰ خواهد بود ، اما اگر مقدار i بیش از ظرفیت short بود ، اون موقع
چی ؟؟؟؟؟
```
int i = 70000;
short s = (short)i;
Console.WriteLine(s);
```
مقدار ۷۰۰۰۰ از حداکثر مقدار قابل نگه داری در(32767) short بزرگتره و نتیجه غلط خواهد بود (در اینجا مقدار s خواهد
شد ۴۴۶۴).
)جهت اطلاع علاقه مندان : مقدار ۷۰۰۰۰ در مبنای ۱۶ میشه : ۰۰۰۱۱۱۷۰ ، در تبدیل این عدد به نوع short، باید فقط
دو بایت کوچکتر در نظر گرفته بشه ، یعنی ۱۱۷۰ در مبنای ۱۶ که میشه ۴۴۶۴ در مبنای ۱۰)
در کل تبدیل از نوع بزرگتر به کوچکتر ، باید Explicit و توسط برنامه نویس انجام بشه و امکان خطا هم در اون هست.
اندازهٔ نوع داده ها بر حسب بایت :
1 : bool ، sbyte،byte
2 : char ، short، ushort
4 : int ، uint، float
8 : long ، ulong ، double

مقدار byte از ۰ تا ۲۵۵ است ، اما sbyte یعنی signed byte) علامتدار) و از -۱۲۸ تا ۱۲۷
مقدار : short از -۳۲۷۶۸ تا ۳۲۷۶۷ ؛ ushort یعنی) unsigned short بدون علامت) و از ۰ تا ۶۵۵۳۵
بقیه رو خودتون حساب کنید.
)
این هم یک نکتهٔ کوچولو !!! برای تازه کارها .

خب این مقاله کمی طولانی شد پس بحث دربارهٔ is و as و checked/unchecked بمونه برای بعد.

معرفی کتاب :
Inside C#
(Second Edition)
نویسندهTom Archer :
انتشاراتMicrosoft :

کتاب خوبیه ، جزو معدود کتابهاییه در جهان که نمرهٔ بالای ۱۵ بهش میدم. خوب نوشته شده و نکات زیادی رو هم ازC#
و هم از کتابخانه های .NET. پوشش داده. این کتاب جزو کتابهای Architectural Reference مایکروسافت قرار داره.
چاپ ۲۰۰۲ و ۲۳ فصل داره ، و شامل ۳ بخش کلی میشه » فصل ۱ تا ۷C# Class Fundamentals :
فصل ۸ تا ۱۵ Writing Code : و ۱۶ تا ۲۳C# Advanced :
این کتاب رو به کسانی که حداقل یه کتاب دربارهٔ C# خوندن ، یا اینکه یه زبان دیگه بلدن و میخوان C# یاد بگیرن ، پیشنهاد
میکنم. (البته چند فصل آخر کتاب خیلی پیشرفته است(.

موفق باشید.

Copyright © ۲۰۰۵ -۲۰۰۴ IrAsp.Net