

مرجع آموزشی

# C# 2010

(جلد ۱)

مهندس نادر نبوی

انتشارات پندار پارس



### انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگر جنوبی، کوی رشتچی، شماره ۱۴، واحد ۱۶

تلفن: ۶۶۵۷۲۳۳۵ - تلفکس: ۶۶۹۲۶۵۷۸ همراه: ۰۹۱۲۲۴۵۲۳۴۸

[www.PendarePars.com](http://www.PendarePars.com)      [info@PendarePars.com](mailto:info@PendarePars.com)



نام کتاب : مرجع آموزشی **C# 2010** (جلد ۱)

ناشر : انتشارات پندار پارس ناشر همکار: مانلی

تالیف : نادر نبوی

چاپ اول : زمستان ۹۰

شمارگان : ۱۰۰۰ نسخه

طرح جلد : محمد اسماعیلی هدی

لیتوگرافی : ترام سنچ

چاپ، صحافی : صالحان، خیام

قیمت : ۱۷۵۰۰ تومان با **CD** شابک : ۹۷۸-۶۰۰-۶۵۲۹-۰۵-۹ دوره: ۹۷۸-۶۰۰-۶۵۲۹-۰۷-۳

••••• \* هرگونه کپی برداری، تکثیر و چاپ کاغذی یا الکترونیکی از این کتاب بدون اجازه ناشر تخلف بوده و پیگرد قانونی دارد \* •••••

## پیشگفتار

چندین سال تجربه در امر تدریس برنامه‌نویسی، خصوصا زبان C#، این امر را برایم روشن ساخته است که تعداد افراد واجد استعداد برنامه‌نویسی، بخصوص در میان دانشجویان این رشته، فراوان، ولی تعداد افرادی که حقیقتا به آن علاقه داشته و از سر این علاقه حاضر به صرف ساعات متمادی (معمولا کسل کننده) در کدنویسی، دیباگ برنامه و تست دوباره و چندباره آن باشند، زیاد نیست و باز از میان این افراد، معدودند کسانی که بخواهند یک زبان برنامه‌نویسی را از راه اصولی و درست آن فرا گیرند.

البته کم نیستند افرادی که برنامه‌نویسی "بلند" و حتی برنامه هم نوشته و بالاتر از آن، پروژه هم قبول کرده و به مرحله اجرا نیز می‌رسانند... با این حال برنامه‌نویسی این نیست. چه خود این افراد برای نوشتن برنامه‌ای مشابه با برنامه‌های گذشته خود، به عنوان مثال در جایی که فقط منبع داده‌ها تغییر کند و یا رابط کاربر جدیدی مورد نیاز باشد، از آنجا که از روش‌های درست استفاده نکرده‌اند، مجبور به بازنویسی تمام کدهای سابق خود هستند، و یا از آنجا که اقدام به پیاده‌سازی توابع عام و همه منظوره نکرده‌اند، آنقدر کدهای تکراری در جای جای برنامه آورده‌اند که خود دیگر قادر به بازخوانی آن نیستند چه رسد به تغییر و اصلاح برنامه. مشکلاتی از این دست زیادند، به عنوان مثالی دیگر، برنامه نویس غیر حرفه‌ای به دلیل عدم آشنایی با کاربرد درست واسط‌ها و تفکر جداسازی لایه‌ها، به راحتی رابط کار و منطق برنامه را با منبع داده‌های برنامه آمیخته، در نهایت نرم‌افزاری تهیه می‌کند که به هر چیزی شبیه است، غیر از برنامه کامپیوتری.

لذا با توجه به مطالب فوق، اگر بپذیریم که یادگیری برنامه‌نویسی به زبانی مثل C#، چیزی بیشتر از آشنایی با دستورات و نحو<sup>1</sup> صوری آنهاست، به مرجعی نیاز داریم که نه تنها فرامین و دستورات زبان را مطرح کرده باشد (که در بسیاری موارد با جستجوی ساده در اینترنت نیز قابل دستیابی هستند)، بلکه به جای کاربرد آنها، بر نحوه صحیح ساختارهای صحیح و مطمئن برنامه‌نویسی و روش‌های درست مهندسی نرم‌افزار به عمیق‌ترین معنای آن، تکیه داشته باشد.

کتاب حاضر با توجه به این رویکرد به نگارش درآمده است. قسمت اعظم متنی که در دست دارید، ترجمه کتاب Pro C# 2010 and the .NET platform 4.0 نوشته Andrew Troelsen از انتشارات Apress است. با ذکر این مطلب در اینجا، قصد تاکید بر این نکته را دارم که ترجمه کتاب به سیاق کتاب‌های ادبی و داستانی انجام نشده بلکه نکات اشاره شده در بالا، در گزینش مطالب و یا حتی پاراگراف‌ها و نکاتی که به متن اصلی کتاب افزوده شده‌اند، همواره مد نظر بوده‌اند.

مطالعه کتاب به‌ویژه برای دانشجویان رشته نرم‌افزار به عنوان کتاب کمک درسی و همچنین سایر کسانی که می‌خواهند زبان C# را از پایه بیاموزند و از آن به عنوان ابزار اصلی برنامه‌نویسی استفاده کنند، می‌تواند آموزنده باشد. با این حال دارا بودن زمینه‌ای در C (و نه حتی ++C) به درک بهتر مطالب آن کمک خواهد کرد.

---

<sup>1</sup> syntax

فصل‌های اول و دوم کتاب به شرح اجزای کلیدی محیط NET. پرداخته است، که برای فهم آن به قدری زمینه‌های قبلی در مورد اصول شیء‌گرایی، چگونگی عملکرد کامپایلرها، کتابخانه‌های کد و ساختارهای فایل‌های .exe و .dll، نیاز است. این مطالب در فصل‌های بعدی با ذکر جزئیات، به صورت مشروح مورد بررسی قرار گرفته و روشن‌تر می‌شوند، با این حال توصیه می‌کنم اگر دارای چنین زمینه‌ای نیستید، مطالعه را از فصل ۳ آغاز کرده و بعد از آن به فصل‌های اول و دوم بازگردید.

سایر فصل‌های کتاب، از فصل ۳ به بعد، دارای نظم منطقی هستند که از نظر اصول آموزش این زبان، در مطابقت کامل با اکثر سرفصل‌های به کار گرفته شده در مراکز آموزشی معتبر می‌باشند. بر این اساس، جلد اول کتاب از فصل ۱ تا فصل ۱۴ به بررسی ساختارهای بنیادین زبان و بالاخره کاربرد مطالب بررسی شده در رابطه با پایگاه‌های داده، با استفاده از روش متصل (connected layer)، می‌پردازد.

جلد دوم کتاب که به خواست خداوند به زودی در دسترس دانشجویان محترم قرار خواهد گرفت، به بررسی مطالب پیشرفته‌تر در رابطه با پایگاه‌های داده، پروژه‌های فرم‌های ویندوز، پروژه‌های WPF و WCF، ASP.NET و کاربرد Entity Framework خواهد پرداخت.

در خاتمه باید از دانشجویان عزیزم خانم مهندس مینا کربلایی و خانم مهندس ترانه رنجبر که قسمت‌هایی از متن را مطالعه و ویرایش کرده، با پیشنهادات خود کمک زیادی در نگارش کتاب کرده‌اند، از صمیم قلب تشکر کنم.

کتاب را تقدیم می‌کنم به پسرم سامان که در ابتدای این راه طولانی و پر فراز و نشیب گام گذاشته است، باشد که اصول ذکر شده در ابتدای این مقدمه را همواره مد نظر قرار دهد.

متن کتاب هنوز هم جای ویرایش دارد. از خوانندگان محترم تقاضا می‌کنم پیشنهادات خود را با آدرس nabavijobmail@yahoo.com در میان بگذارند تا در ویرایش‌های بعدی به خواست خداوند، مرتفع گردیده و یا در جایی که لازم باشد، به متن اضافه شوند.

نادر نبوی

دی ماه ۱۳۹۰

	فصل ۱: آشنایی با میانی .NET
۱۳	۱-۱ نگاهی به تاریخ قبل از .NET
۱۴	۱-۱-۲ ۱-۱-۱ کدنویسی واسط برنامه‌نویسی C/Windows
۱۴	۱-۱-۳ ۱-۱-۱ کدنویسی C++/MFC
۱۴	۱-۱-۴ ۱-۱-۱ کدنویسی ویژوال بیسیک 6.0
۱۵	۱-۱-۵ ۱-۱-۱ کدنویسی به عنوان برنامه‌نویس Java
۱۵	۱-۱-۶ ۱-۱-۱ راه حل .NET
۱۶	۱-۲ معرفی پایه‌های اصلی .NET
۱۷	۱-۲-۱ ۱-۲-۱ نقش کتابخانه کلاس‌های پایه
۱۸	۱-۲-۲ ۱-۲-۱ ویژگی‌های زبان C#
۲۰	۱-۳ بررسی اسمبلی‌های .NET
۲۲	۱-۳-۱ ۱-۳-۱ اسمبلی‌های تک فایلی و چند فایلی
۲۲	۱-۳-۲ ۱-۳-۲ نقش زبان مشترک میانی
۲۵	۱-۳-۳ ۱-۳-۲ دلایل کاربرد CIL
۲۵	۱-۳-۴ ۱-۳-۲ ترجمه CIL به زبان ماشین
۲۶	۱-۳-۵ ۱-۳-۲ نقش فراداده‌های نوع در .NET
۲۷	۱-۳-۶ ۱-۳-۲ نقش مانیفست اسمبلی
۲۷	۱-۴ بررسی سیستم مشترک انواع
۲۸	۱-۴-۱ ۱-۴-۱ نوع کلاس در CTS
۲۹	۱-۴-۲ ۱-۴-۲ نوع واسط در CTS
۲۹	۱-۴-۳ ۱-۴-۲ نوع ساختار در CTS
۳۰	۱-۴-۴ ۱-۴-۲ انواع شمارشی در CTS
۳۰	۱-۴-۵ ۱-۴-۲ نوع عامل در CTS
۳۱	۱-۴-۶ ۱-۴-۲ اعضای انواع در CTS
۳۱	۱-۴-۷ ۱-۴-۲ انواع داده در CTS
۳۲	۱-۵ آشنایی با مشخصه‌های مشترک زبان
۳۵	۱-۶ آشنایی با محیط مشترک زمان اجرای زبان‌ها
۳۶	۱-۷ تفاوت بین فضای نامی، اسمبلی و انواع
۳۸	۱-۷-۱ ۱-۷-۱ نقش فضای نامی ریشه
۳۸	۱-۷-۲ ۱-۷-۲ دسترسی به فضای نامی از طریق برنامه
۴۰	۱-۷-۳ ۱-۷-۲ ارجاع به اسمبلی‌های خارجی در یک برنامه
۴۱	۱-۸ بررسی اسمبلی‌ها توسط ILDASM.EXE
۴۲	۱-۸-۱ ۱-۸-۱ مشاهده کد CIL
۴۲	۱-۸-۲ ۱-۸-۱ مشاهده فراداده یک نوع
۴۳	۱-۸-۳ ۱-۸-۱ مشاهده فراداده اسمبلی (مانیفست)
۴۳	۱-۸-۴ ۱-۸-۱ بررسی اسمبلی با استفاده از برنامه Reflector
۴۴	۱-۹ انتشار محیط زمان اجرای .NET
۴۶	جمع‌بندی فصل اول
۴۷	۲-۱ نقش بسته توسعه نرم‌افزار .NET. نگارش ۴.۰
۴۸	۲-۱-۱ ۲-۱-۱ برنامه خط فرمان در ویژوال استدیو
۴۸	۲-۲ ایجاد برنامه‌های کاربردی C# با استفاده از CSC.EXE
۴۹	۲-۲-۱ ۲-۲-۱ مشخص کردن ورودی/خروجی‌های کامپایلر
۵۱	۲-۲-۲ ۲-۲-۲ ارجاع به اسمبلی‌های خروجی
۵۲	۲-۲-۳ ۲-۲-۲ کامپایل همزمان چندین فایل
۵۳	۲-۲-۴ ۲-۲-۲ کار با فایل‌های پاسخ در C#
۵۴	۲-۳ برنامه‌نویسی در محیط SHARPDEVELOP
۵۵	۲-۳-۱ ۲-۳-۱ ایجاد یک پروژه آزمایشی با #Develop
۵۷	۲-۴ برنامه‌نویسی در محیط VISUAL C# 2010 EXPRESS
۵۷	۲-۴-۱ ۲-۴-۱ ویژگی‌های منحصر بفرد Visual C# 2010 Express
۵۸	۲-۵ برنامه‌نویسی در محیط VISUAL STUDIO 2010
۵۹	۲-۵-۱ ۲-۵-۱ برخی ویژگی‌های منحصر بفرد ویژوال استدیو ۲۰۱۰
۵۹	۲-۵-۲ ۲-۵-۲ تعیین نگارش .NET Framework در پروژه
۶۰	۲-۵-۳ ۲-۵-۲ استفاده از پنجره Solution Explorer
۶۰	۲-۵-۴ ۲-۵-۲ تعیین اسمبلی‌های خارجی به عنوان ارجاع پروژه
۶۱	۲-۵-۵ ۲-۵-۲ مشاهده خصوصیات پروژه

۶۲	مشاهده کلاس‌ها	۲-۵-۶
۶۳	برنامه کمکی مرورگر اشیاء	۲-۵-۷
۶۳	امکان بازسازی کد	۲-۵-۸
۶۵	استفاده از روش‌های گسترش و ایجاد بلاک کد	۲-۵-۹
۶۶	محیط طراحی بصری کلاس‌ها	۲-۵-۱۰
۷۱	ساختار یک برنامه ساده به زبان <b>C#</b>	۳-۱
۷۳	اشکال مختلف متد <b>Main()</b>	۳-۱-۱
۷۴	تشخیص کد خطای برنامه	۳-۱-۲
۷۶	پردازش پارامترهای وارد شده از خط فرمان	۳-۱-۳
۷۸	تعیین پارامترهای خط فرمان در ویژوال استدیو ۲۰۱۰	۳-۱-۴
۷۸	اعضای دیگری از کلاس <b>System.Environment</b>	۳-۱-۵
۸۰	کلاس <b>SYSTEM.CONSOLE</b>	۳-۲
۸۱	عملیات پایه ورودی و خروجی توسط کلاس <b>Console</b>	۳-۲-۱
۸۲	قالب بندی خروجی کنسول	۳-۲-۲
۸۲	قالب بندی داده‌های عددی	۳-۲-۳
۸۴	قالب بندی داده‌های عددی در سایر انواع پروژه‌ها	۳-۲-۴
۸۴	انواع داده‌ها در <b>NET</b>	۳-۳
۸۶	تعریف و مقداردهی اولیه متغیرها	۳-۳-۱
۸۸	انواع داده پیش ساخته و عملگر <b>new</b>	۳-۳-۲
۸۸	ساختار سلسله مراتبی کلاس‌های انواع داده	۳-۳-۳
۹۱	اعضای انواع داده عددی	۳-۳-۴
۹۱	اعضای نوع داده <b>System.Boolean</b>	۳-۳-۵
۹۲	اعضای نوع داده <b>System.Char</b>	۳-۳-۶
۹۳	به دست آوردن مقادیر از داده رشته‌ای	۳-۳-۷
۹۳	انواع داده <b>System.DateTime</b> و <b>System.TimeSpan</b>	۳-۳-۸
۹۴	فضای نامی <b>System.Numerics</b> در <b>NET 4.0</b>	۳-۳-۹
۹۶	کار با رشته‌ها در <b>C#</b>	۳-۴
۹۷	اعمال اولیه روی متن	۳-۴-۱
۹۸	به هم پیوستن رشته‌ها	۳-۴-۲
۹۹	کاراکترهای <b>Escape</b>	۳-۴-۳
۱۰۰	ایجاد رشته‌های تحت‌اللفظی	۳-۴-۴
۱۰۰	تساوی رشته‌ها	۳-۴-۵
۱۰۱	تغییر ناپذیری رشته‌ها	۳-۴-۶
۱۰۳	کلاس <b>System.Text.StringBuilder</b>	۳-۴-۷
۱۰۴	باریک سازی و تعریض در تبدیل انواع داده‌ها	۳-۵
۱۰۹	تنظیم پروژه برای تست سرریز	۳-۵-۱
۱۱۰	کاربرد کلمه کلیدی <b>Unchecked</b>	۳-۵-۲
۱۱۰	نقش کلاس <b>System.Convert</b> در تبدیل داده‌ها	۳-۵-۳
۱۱۱	تعریف ضمنی متغیرهای محلی	۳-۶
۱۱۳	محدودیت‌های کاربرد متغیرهای ضمنی	۳-۶-۱
۱۱۵	فواید کاربرد متغیرهای ضمنی	۳-۶-۲
۱۱۶	ساختارهای تکرار در <b>C#</b>	۳-۷
۱۱۶	حلقه <b>for</b>	۳-۷-۱
۱۱۶	حلقه <b>foreach</b>	۳-۷-۲
۱۱۷	ساختارهای حلقه <b>while</b> و <b>do/while</b>	۳-۷-۳
۱۱۸	ساختارهای تصمیم گیری و عملگرهای منطقی و رابطه‌ای	۳-۸
۱۱۸	دستور <b>if/else</b>	۳-۸-۱
۱۲۰	دستور <b>switch</b>	۳-۸-۲
۱۲۲	جمع بندی فصل سوم	
۱۲۳	اصلاح کننده‌های متدها و پارامترها	۴-۱
۱۲۴	رفتار پیش فرض ارسال پارامترها	۴-۱-۱
۱۲۶	اصلاح کننده <b>out</b>	۴-۱-۲
۱۲۸	اصلاح کننده <b>ref</b>	۴-۱-۳
۱۲۹	بررسی اصلاح کننده <b>params</b>	۴-۱-۴
۱۳۰	تعریف پارامترهای اختیاری	۴-۱-۵

۱۳۲	۴-۱-۶	فراخوانی متدها با پارامترهای نامدار
۱۳۴	۴-۱-۷	سربارگذاری متدها
۱۳۶	۴-۲	آرایه‌ها در <b>C#</b>
۱۳۷	۴-۲-۱	تعریف آرایه با مقادیر اولیه
۱۳۸	۴-۲-۲	آرایه‌های محلی ایجاد شده به صورت ضمنی
۱۳۹	۴-۲-۳	آرایه‌ای از اشیاء
۱۴۰	۴-۲-۴	کار با آرایه‌های چند بعدی
۱۴۱	۴-۲-۵	آرایه‌ها به عنوان آرگومان و برگشتی متد
۱۴۲	۴-۲-۶	کلاس پایه <code>System.Array</code>
۱۴۴	۴-۳	کار با انواع شمارشی
۱۴۵	۴-۳-۱	تعیین چگونگی ذخیره سازی انواع شمارشی
۱۴۵	۴-۳-۲	تعریف متغیرهای شمارشی
۱۴۷	۴-۳-۳	بررسی کلاس <code>System.Enum</code>
۱۴۸	۴-۳-۴	دریافت پویای مقدار/سمبل مقادیر شمارشی
۱۴۹	۴-۴	ساختارها در <b>C#</b>
۱۵۱	۴-۴-۱	ایجاد متغیر از ساختار
۱۵۳	۴-۵	درک انواع ارجاعی و ارزشی
۱۵۴	۴-۵-۱	انواع ارزشی و ارجاعی و عملگر نسبت‌دهی
۱۵۶	۴-۵-۲	انواع ارزشی شامل انواع ارجاعی
۱۵۸	۴-۵-۳	ارسال انواع ارجاعی به صورت ارزشی به متدها
۱۵۹	۴-۵-۴	ارسال انواع ارجاعی به صورت ارجاعی
۱۶۰	۴-۶	انواع تهی‌پذیر در <b>C#</b>
۱۶۲	۴-۶-۱	کاربرد انواع تهی‌پذیر
۱۶۳	۴-۶-۲	مفهوم عملگر <code>??</code>
۱۶۴		جمع بندی فصل چهارم
۱۶۵	۵-۱	نوع کلاس در <b>C#</b>
۱۶۸	۵-۱-۱	جایگذاری اشیاء در حافظه با دستور <code>new</code>
۱۶۹	۵-۲	آشنایی با متدهای سازنده
۱۶۹	۵-۲-۱	نقش سازنده پیش فرض
۱۷۰	۵-۲-۲	ایجاد سازنده‌های با مقدار
۱۷۲	۵-۳	نقش کلمه کلیدی <b>THIS</b>
۱۷۴	۵-۳-۱	فراخوانی زنجیره‌ای سازنده‌ها توسط <code>this</code>
۱۷۷	۵-۳-۲	مشاهده مراحل اجرای سازنده‌ها
۱۷۹	۵-۳-۳	بررسی مجدد آرگومان‌های اختیاری
۱۸۰	۵-۴	بررسی کلمه کلیدی <b>STATIC</b>
۱۸۰	۵-۴-۱	تعریف متدهای استاتیک
۱۸۱	۵-۴-۲	تعریف فیلدهای استاتیک
۱۸۴	۵-۴-۳	سازنده‌های استاتیک
۱۸۶	۵-۴-۴	تعریف کلاس‌های استاتیک
۱۸۸	۵-۵	سه اصل پایه شیء‌گرایی
۱۸۸	۵-۵-۱	بررسی نقش کپسوله‌سازی
۱۸۹	۵-۵-۲	بررسی نقش ارث‌بری
۱۹۰	۵-۵-۳	بررسی نقش چندریختی
۱۹۲	۵-۶	اصلاح‌کننده‌های سطح دسترسی در <b>C#</b>
۱۹۳	۵-۶-۱	اصلاح‌کننده‌های سطح دسترسی پیش فرض
۱۹۳	۵-۶-۲	اصلاح‌کننده‌های دسترسی و انواع تو در تو
۱۹۴	۵-۷	کپسوله‌سازی در <b>C#</b>
۱۹۵	۵-۷-۱	کپسوله‌سازی با استفاده از متدهای دسترسی و تغییردهنده
۱۹۸	۵-۷-۲	کپسوله‌سازی با استفاده از خصوصیات <code>NET</code>
۲۰۱	۵-۷-۳	کاربرد خصوصیات در تعریف کلاس
۲۰۲	۵-۷-۴	نمایش داخلی خصوصیات
۲۰۵	۵-۷-۵	تعیین سطح دسترسی بلاک‌های <code>set</code> و <code>get</code>
۲۰۵	۵-۷-۶	خصوصیات فقط خواندنی و فقط نوشتنی
۲۰۶	۵-۷-۷	خصوصیات استاتیک
۲۰۷	۵-۸	خصوصیات خودکار

۲۰۹	۵-۸-۱	کار با خصوصیات خودکار
۲۰۹	۵-۸-۲	خصوصیات خودکار و مقادیر پیش فرض
۲۱۱	۵-۹	روش مقداردهی اولیه اشیاء
۲۱۳	۵-۹-۱	فراخوانی سازنده‌های با مقدار در مقداردهی اولیه شیء
۲۱۴	۵-۹-۲	مقداردهی انواع داخلی یک نوع
۲۱۶	۵-۱۰	فیلدهای ثابت یک کلاس
۲۱۷	۵-۱۰-۱	فیلدهای فقط خواندنی
۲۱۸	۵-۱۰-۲	فیلدهای فقط خواندنی استاتیک
۲۱۸	۵-۱۱	آشنایی با کلاس‌های پاره‌ای
۲۲۰		جمع بندی فصل پنجم
۲۲۱	۶-۱	زیرساخت‌های اساسی ارث‌بری
۲۲۲	۶-۱-۱	تعیین کلاس والد برای کلاس فرزند
۲۲۵	۶-۲-۲	کلاس‌های sealed
۲۲۶	۶-۲-۲	استفاده از کلاس دیاگرام‌های ویژوال استدیو
۲۲۷	۶-۳	دومین اصل شیء‌نگرایی، بررسی جزئیات ارث‌بری
۲۲۹	۶-۳-۱	دستیابی به کلاس پایه با کلمه کلیدی base
۲۳۱	۶-۳-۲	کاربرد سطح دسترسی protected
۲۳۳	۶-۳-۳	اضافه کردن کلاس‌های sealed
۲۳۴	۶-۴	برنامه‌نویسی ارتباط شمول
۲۳۵	۶-۴-۱	آشنایی با انواع تودرتو
۲۳۷	۶-۵	آشنایی با مفهوم چندریختی در C#
۲۳۸	۶-۵-۱	دستورات virtual و override
۲۴۱	۶-۵-۲	جلوگیری از بازنویسی متدهای مجازی
۲۴۲	۶-۵-۳	کلاس‌های مجرد
۲۴۳	۶-۵-۴	آشنایی با واسط چندریختی
۲۴۸	۶-۵-۵	مخفی‌سازی اعضا
۲۵۱	۶-۶	قوانین تبدیل کلاس‌های والد/فرزند
۲۵۳	۶-۶-۱	کاربرد کلمات کلیدی is و as
۲۵۴	۶-۷	اولین کلاس والد: SYSTEM.OBJECT
۲۵۸	۶-۷-۱	بازنویسی متد ToString()
۲۵۸	۶-۷-۲	بازنویسی متد Equals()
۲۶۰	۶-۷-۳	بازنویسی متد GetHashCode()
۲۶۱	۶-۷-۴	اعضای استاتیک کلاس Object
۲۶۲		جمع بندی فصل ششم
۲۶۳	۷-۱	آشنایی با نوع واسط
۲۶۵	۷-۱-۱	واسط‌ها در مقایسه با کلاس‌های مجرد
۲۶۷	۷-۱-۲	طریقه ایجاد واسط‌ها
۲۶۹	۷-۱-۳	پیاده‌سازی یک واسط
۲۷۱	۷-۱-۴	فراخوانی اعضای واسط در سطح شیء
۲۷۲	۷-۱-۵	کاربرد as, is در دستیابی به ارجاع واسط‌ها
۲۷۴	۷-۱-۶	واسط‌ها به عنوان پارامتر و خروجی متدها
۲۷۷	۷-۱-۷	آرایه‌ای از واسط‌ها
۲۷۸	۷-۱-۸	پیاده‌سازی واسط‌ها در ویژوال استدیو ۲۰۱۰
۲۷۹	۷-۱-۹	پیاده‌سازی صریح واسط‌ها
۲۸۲	۷-۲	کاربرد سلسله مراتبی از واسط‌ها
۲۸۳	۷-۲-۱	ارث‌بری چندگانه با واسط‌ها
۲۸۵	۷-۳	ایجاد انواع قابل پیمایش
۲۸۹	۷-۳-۱	ایجاد متدهای پیمایشی با استفاده از yield
۲۹۱	۷-۳-۲	ایجاد یک پیمایش‌گر نام‌دار
۲۹۳	۷-۴	ایجاد اشیاء قابل کپی
۲۹۸	۷-۵	ایجاد اشیاء قابل مقایسه
۳۰۱	۷-۵-۱	مرتب‌سازی بر اساس چندین پایه
۳۰۴		جمع بندی فصل هفتم
۳۰۵	۸-۱	مشکلات کار با کلکسیون‌های غیر ژنریک
۳۰۸	۸-۱-۱	کاهش کارایی برنامه



۳۱۲	مشکلات مربوط به امنیت نوع	۸-۱-۲
۳۱۶	نقش پارامترهای تعیین نوع در ژنریک‌ها	۸-۲
۳۱۷	تعیین پارامترهای نوع برای کلاس‌ها/ساختارهای ژنریک	۸-۲-۱
۳۱۹	تعیین پارامترهای نوع برای اعضا و واسط‌های ژنریک	۸-۲-۲
۳۲۰	<b>SYSTEM.COLLECTIONS.GENERIC</b> فضای نامی	۸-۳
۳۲۲	مقداردهی اولیه کلکسیون‌ها	۸-۳-۱
۳۲۳	کلکسیون لیست ژنریک <T>List	۸-۳-۲
۳۲۵	کلکسیون پشته ژنریک <T>Stack	۸-۳-۴
۳۲۶	کلکسیون صف ژنریک <T>Queue	۸-۳-۵
۳۲۸	کلاس کلکسیون مجموعه مرتب <T>SortedSet	۸-۳-۶
۳۲۹	ایجاد متدهای ژنریک	۸-۴
۳۳۲	تشخیص پارامترهای نوع توسط کامپایلر	۸-۴-۱
۳۳۳	ایجاد کلاس‌ها و ساختارهای ژنریک	۸-۵
۳۳۵	کلاس‌های ژنریک پایه	۸-۵-۱
۳۳۷	مقید سازی پارامترهای نوع	۸-۶
۳۳۹	فقدان محدود کننده‌های عملگرها	۸-۶-۱
۳۴۰	جمع بندی فصل هشتم	
۳۴۱	آشنایی با نوع <b>DELEGATE</b>	۹-۱
۳۴۳	تعریف یک <b>DELEGATE</b> در <b>C#</b>	۹-۲
۳۴۵	کلاس‌های پایه <b>DELEGATE</b> و <b>MULTICASTDELEGATE</b>	۹-۳
۳۴۷	ساده‌ترین مثال برای یک <b>DELEGATE</b>	۹-۴
۳۴۹	موشکافی یک delegate	۹-۴-۱
۳۵۰	ارسال پیام‌های وضعیت شیء توسط <b>DELEGATE</b>	۹-۵
۳۵۲	استفاده از چندنقشی	۹-۵-۱
۳۵۵	حذف متدها از لیست delegate	۹-۵-۲
۳۵۷	روش میان‌بر استفاده از نام متد	۹-۶
۳۵۸	کواریانس عامل‌ها ( <b>DELEGATES COVARIANCE</b> )	۹-۷
۳۶۰	<b>DELEGATE</b> های ژنریک	۹-۸
۳۶۲	شبه سازی delegateهای ژنریک	۹-۸-۱
۳۶۳	رویدادها در <b>C#</b>	۹-۹
۳۶۵	کلمه کلیدی event	۹-۹-۱
۳۶۷	گوش دادن به رویدادهای	۹-۹-۲
۳۷۰	ایجاد آرگومان‌های یک رویداد	۹-۹-۳
۳۷۱	عامل ژنریک <T>EventHandler	۹-۹-۴
۳۷۲	آشنایی با متدهای بی‌نام	۹-۱۰
۳۷۴	دسترسی به متغیرهای محلی	۹-۱۰-۱
۳۷۵	آشنایی با عبارات لاندا	۹-۱۱
۳۷۸	تشریح عبارات لاندا	۹-۱۱-۱
۳۷۹	پردازش چند خطی آرگومان‌های عبارت لاندا	۹-۱۱-۲
۳۸۰	عبارات لاندا همراه آرگومان و بدون آرگومان	۹-۱۱-۳
۳۸۳	جمع بندی فصل نهم	
۳۸۵	متدهای اندیس‌گذار	۱۰-۱
۳۸۸	اندیس گذاری داده‌ها با استفاده از مقادیر رشته‌ای	۱۰-۱-۱
۳۸۹	سربارگذاری متدهای اندیس‌گذار	۱۰-۱-۲
۳۸۹	اندیس‌گذارهای چند بعدی	۱۰-۱-۳
۳۹۰	تعریف اندیس‌گذار در یک واسط	۱۰-۱-۴
۳۹۱	سربارگذاری عملگرها	۱۰-۲
۳۹۲	سربارگذاری عملگرهای باینری	۱۰-۲-۱
۳۹۵	سربارگذاری عملگرهای یکانی	۱۰-۲-۲
۳۹۵	سربارگذاری عملگرهای تساوی	۱۰-۲-۳
۳۹۶	سربارگذاری عملگرهای مقایسه‌ای	۱۰-۲-۴
۳۹۷	نمایش داخلی عملگرهای سربارگذاری شده	۱۰-۲-۵
۳۹۹	آخرین نکات در مورد سربارگذاری عملگرها	۱۰-۲-۶
۳۹۹	تبدیل شخصی انواع	۱۰-۳
۴۰۱	ایجاد روتین‌های شخصی تبدیل نوع	۱۰-۳-۱

۴۰۴	تعریف روتین‌های تبدیل ضمنی
۴۰۶	نمایش داخلی روتین‌های تبدیل
۴۰۶	آشنایی با متدهای تعمیم یافته
۴۰۷	روش تعریف متدهای تعمیم یافته
۴۰۹	فراخوانی متدهای تعمیم یافته در سطح نمونه
۴۰۹	فراخوانی متدهای تعمیم یافته در سطح کلاس
۴۱۱	استفاده از انواع حاوی متدهای تعمیم
۴۱۳	ایجاد کتابخانه‌های تعمیم یافته
۴۱۵	توسعه واسط‌ها بر اساس متدهای تعمیم یافته
۴۱۶	متدهای جزئی
۴۱۹	کاربرد متدهای جزئی
۴۱۹	آشنایی با انواع بی‌نام
۴۲۱	نمایش داخلی انواع بی‌نام
۴۲۲	پایاده‌سازی متدهای ToString() و GetHashCode()
۴۲۴	انواع بی‌نامی که شامل انواع بی‌نام دیگر هستند
۴۲۵	کار با اشاره‌گرها
۴۲۷	کلمه کلیدی unsafe
۴۲۹	کار با عملگرهای * و &
۴۳۰	دسترسی به فیلدها توسط اشاره‌گرها
۴۳۱	کلمه کلیدی stackalloc
۴۳۱	قفل کردن یک نوع با کلمه کلیدی fixed
۴۳۳	کلمه کلیدی sizeof
۴۳۴	جمع بندی فصل دهم

## ۳۳۵ کاربرد LINQ بر اشیاء

۴۳۵	۱۱-۱ ساختارهای پایه برنامه‌نویسی LINQ
۴۳۶	۱۱-۱-۱ تعیین نوع متغیرهای محلی به صورت ضمنی
۴۳۷	۱۱-۱-۲ مقداردهی اولیه اشیاء و کلکسیون‌ها
۴۳۷	۱۱-۱-۳ عبارات لاتندا
۴۳۸	۱۱-۱-۴ متدهای تعمیم یافته
۴۳۹	۱۱-۱-۵ انواع بی‌نام
۴۴۰	۱۱-۲ آشنایی با نقش LINQ
۴۴۱	۱۱-۲-۱ اسمبلی‌های مرکزی LINQ
۴۴۲	۱۱-۳ اعمال کوئری‌های LINQ بر آرایه‌ها
۴۴۴	۱۱-۳-۱ کاربرد انعکاس در خروجی کوئری LINQ
۴۴۵	۱۱-۳-۲ LINQ و متغیرهای ضمنی محلی
۴۴۶	۱۱-۳-۳ LINQ و متدهای تعمیم یافته
۴۴۷	۱۱-۳-۴ نقش اجرای تاخیری
۴۴۸	۱۱-۳-۵ نقش اجرای بلافاصله
۴۴۹	۱۱-۴ برگرداندن نتیجه یک کوئری LINQ
۴۵۱	۱۱-۴-۱ برگرداندن خروجی کوئری LINQ توسط اجرای بلافاصله
۴۵۲	۱۱-۵ اعمال کوئری‌های LINQ به کلکسیون‌ها
۴۵۲	۱۱-۵-۱ اعمال شرط و فیلتر کردن خروجی کوئری
۴۵۳	۱۱-۵-۲ اعمال کوئری‌های LINQ به کلکسیون‌های غیر ژنریک
۴۵۴	۱۱-۵-۳ فیلتر کردن داده‌ها توسط OfType<T>()
۴۵۵	۱۱-۶ بررسی عملگرهای کوئری‌های LINQ
۴۵۷	۱۱-۶-۱ فرم کلی انتخاب، from...select
۴۵۸	۱۱-۶-۲ استخراج زیر مجموعه‌ای از منبع، where
۴۵۸	۱۱-۶-۳ ایجاد انواع داده جدید
۴۶۰	۱۱-۶-۴ شمارش در کوئری، count
۴۶۱	۱۱-۶-۵ معکوس کردن نتیجه خروجی، reverse
۴۶۱	۱۱-۶-۶ عبارات مرتب‌سازی، orderby
۴۶۲	۱۱-۶-۷ LINQ به عنوان یک دیاگرام ون
۴۶۳	۱۱-۶-۸ حذف عناصر تکراری، distinct
۴۶۴	۱۱-۶-۹ عملگرهای جمعی LINQ
۴۶۴	۱۱-۷ نمایش داخلی جملات LINQ

۴۶۶	ایجاد عبارات کوئری با استفاده از عملگرهای کوئری
۴۶۶	عبارات کوئری با استفاده از عملگرهای لاند و کلاس Enumerable
۴۶۸	عبارات کوئری با استفاده از متدهای بی نام
۴۶۸	عبارات کوئری با استفاده از عاملها
۴۷۰	جمع بندی فصل یازدهم
۴۷۱	<b>آشنایی با چند نخ و برنامه نویسی موازی</b>
۴۷۱	۱۲-۱ ارتباط بین <b>THREAD/CONTEX/APPDOMAIN/PROCESS</b>
۴۷۳	۱۲-۱-۱ مشکل همزمانی
۴۷۳	۱۲-۱-۲ نقش همگام سازی thread
۴۷۶	۱۲-۲ طبیعت غیر همگام عاملها
۴۷۷	۱۲-۲-۱ متدهای BeginInvoke() و EndInvoke()
۴۷۷	۱۲-۲-۲ واسط System.IAsyncResult
۴۷۸	۱۲-۳ اجرای ناهمگام متدها
۴۷۹	۱۲-۳-۱ همگام سازی thread
۴۸۰	۱۲-۳-۲ نقش عامل AsyncCallback
۴۸۲	۱۲-۳-۳ نقش کلاس AsyncResult
۴۸۳	۱۲-۳-۴ ارسال و دریافت داده های شخصی
۴۸۴	۱۲-۴ فضای نامی و کلاس مربوط به <b>THREAD</b> ها
۴۸۷	۱۲-۴-۱ به دست آوردن وضعیت thread جاری
۴۸۸	۱۲-۴-۲ خاصیت Name
۴۸۸	۱۲-۴-۳ خاصیت Priority
۴۸۹	۱۲-۵ ایجاد <b>THREAD</b> های ثانوی با کدنویسی
۴۹۰	۱۲-۵-۱ استفاده از عامل ThreadStart
۴۹۲	۱۲-۵-۲ کار با عامل ParameterizedThreadStart
۴۹۳	۱۲-۵-۳ کلاس AutoResetEvent
۴۹۴	۱۲-۵-۴ threadهای پس زمینه و پیش زمینه
۴۹۶	۱۲-۶ مشکل همزمانی
۴۹۹	۱۲-۶-۱ همگام سازی با استفاده از دستور lock
۵۰۱	۱۲-۶-۲ همگام سازی با استفاده از کلاس Monitor
۵۰۲	۱۲-۶-۳ همگام سازی با استفاده از کلاس InterLocked
۵۰۳	۱۲-۶-۴ همگام سازی با استفاده از صفت [Synchronization]
۵۰۴	۱۲-۷ برنامه نویسی بازخوانی <b>TIMER</b>
۵۰۶	۱۲-۸ آشنایی با کلاس <b>THREADPOOL</b>
۵۰۸	۱۲-۹ برنامه نویسی موازی تحت <b>.NET</b>
۵۰۸	۱۲-۹-۱ کتابخانه وظائف موازی
۵۰۹	۱۲-۹-۲ نقش کلاس Parallel
۵۱۰	۱۲-۹-۳ آشنایی با پردازش موازی داده ها
۵۱۲	۱۲-۹-۴ کلاس Task
۵۱۳	۱۲-۹-۵ مدیریت تقاضای لغو اجرا
۵۱۵	۱۲-۹-۶ آشنایی با موازی سازی وظائف
۵۱۸	۱۲-۱۰ کوئری های موازی <b>LINQ</b>
۵۲۲	جمع بندی فصل دوازدهم
۵۲۳	<b>کار با فایل ها و سریال سازی اشیاء</b>
۵۲۳	۱۳-۱ فضای نامی <b>SYSTEM.IO</b>
۵۲۵	۱۳-۲ انواع مرتبط با فهرست ها و فایل ها
۵۲۵	۱۳-۲-۱ کلاس پایه FileSystemInfo
۵۲۶	۱۳-۳ کاربرد کلاس <b>DIRECTORYINFO</b>
۵۲۸	۱۳-۳-۱ پردازش فایل ها با کلاس DirectoryInfo
۵۲۹	۱۳-۳-۲ ایجاد فهرست های فرعی با DirectoryInfo
۵۳۰	۱۳-۴ کاربرد کلاس <b>DIRECTORY</b>
۵۳۱	۱۳-۵ کاربرد کلاس <b>DRIVEINFO</b>
۵۳۲	۱۳-۶ کاربرد کلاس <b>FILEINFO</b>
۵۳۳	۱۳-۶-۱ متد FileInfo.Create()
۵۳۴	۱۳-۶-۲ متد FileInfo.Open()
۵۳۶	۱۳-۶-۳ متدهای OpenRead() و OpenWrite() از کلاس FileInfo

۵۳۶	.....FileInfo.OpenText() متد	۱۳-۶-۴
۵۳۷	.....FileInfo.AppendText() و CreateText() متدهای	۱۳-۶-۵
۵۳۷	.....FILE کاربرد کلاس	۱۳-۷
۵۳۸	.....File اعضای دیگر کلاس	۱۳-۷-۱
۵۳۹	.....STREAM	۱۳-۸
۵۴۱	.....FileStream کار با	۱۳-۸-۱
۵۴۲	.....STREAMREADER و STREAMWRITER کاربرد کلاس‌های	۱۳-۹
۵۴۳	.....نوشتن در فایل متنی	۱۳-۹-۱
۵۴۴	.....خواندن محتویات یک فایل متنی	۱۳-۹-۲
۵۴۵	.....StreamWriter و StreamReader کاربرد مستقیم	۱۳-۹-۳
۵۴۶	.....STRINGWRITER و STRINGREADER کاربرد کلاس‌های	۱۳-۱۰
۵۴۷	.....BINARYWRITER و BINARYREADER کاربرد کلاس‌های	۱۳-۱۱
۵۴۹	.....بیده‌بانی فایل‌ها	۱۳-۱۲
۵۵۲	.....آشنایی با سریال‌سازی اشیاء	۱۳-۱۳
۵۵۳	.....نقش گراف شیء	۱۳-۱۳-۱
۵۵۵	.....پیکربندی اشیاء برای تثبیت	۱۳-۱۴
۵۵۵	.....تعریف انواع قابل تثبیت	۱۳-۱۴-۱
۵۵۶	.....سطح دسترسی اعضای کلاس در سریال‌سازی	۱۳-۱۴-۲
۵۵۷	.....انتخاب قالب‌دهنده سریال‌سازی	۱۳-۱۵
۵۵۸	.....IFormatter و IRemotingFormatter واسط‌های	۱۳-۱۵-۱
۵۵۹	.....وفاداری به نوع در سریال‌کننده‌ها	۱۳-۱۵-۲
۵۶۰	.....BINARYFORMATTER	۱۳-۱۶
۵۶۱	.....بازخوانی فایل سریال شده	۱۳-۱۶-۱
۵۶۲	.....SOAPFORMATTER	۱۳-۱۷
۵۶۳	.....XMLSERIALIZER	۱۳-۱۸
۵۶۴	.....کنترل داده‌های سند XML تولید شده	۱۳-۱۸-۱
۵۶۶	.....سریال‌سازی کلکسیون‌ها	۱۳-۱۹
۵۶۷	.....BINARY/SOAP	۱۳-۲۰
۵۶۸	.....نگاهی عمیق‌تر به سریال‌سازی اشیاء	۱۳-۲۰-۱
۵۶۹	.....ISerializable	۱۳-۲۰-۲
۵۷۲	.....شخصی کردن سریال‌سازی توسط صفات	۱۳-۲۰-۳
۵۷۴	.....جمع بندی فصل سیزدهم	
۵۷۵	.....ADO.NET قسمت اول، لایه متصل	
۵۷۵	.....ADO.NET	۱۴-۱
۵۷۷	.....ADO.NET	۱۴-۱-۱
۵۷۸	.....ADO.NET	۱۴-۲
۵۸۰	.....ارائه دهنده‌های داده منتشر شده توسط شرکت مایکروسافت	۱۴-۲-۱
۵۸۱	.....OracleClient	۱۴-۲-۲
۵۸۱	.....ADO.NET	۱۴-۳
۵۸۲	.....System.Data	۱۴-۳-۱
۵۸۴	.....IDbConnection	۱۴-۳-۲
۵۸۴	.....IDbTransaction	۱۴-۳-۳
۵۸۵	.....IDbCommand	۱۴-۳-۴
۵۸۵	.....IDataParameter و IDbDataParameter	۱۴-۳-۵
۵۸۶	.....IDataAdapter و IDbDataAdapter	۱۴-۳-۶
۵۸۷	.....IDataRecord و IDataReader	۱۴-۳-۷
۵۸۸	.....کار با ارائه دهنده‌های داده توسط واسط‌ها	۱۴-۴
۵۹۰	.....افزایش انعطاف پذیری با استفاده از فایل‌های پیکربندی	۱۴-۴-۱
۵۹۱	.....AUTOLOT	۱۴-۵
۵۹۲	.....Inventory	۱۴-۵-۱
۵۹۴	.....GetPetName()	۱۴-۵-۲
۵۹۵	.....Orders و Customers	۱۴-۵-۳
۵۹۷	.....ایجاد ارتباطات بین جداول	۱۴-۵-۴
۵۹۸	.....ADO.NET	۱۴-۶
۶۰۲	.....کار با کلاس ایجاد ارائه کننده‌های داده در	۱۴-۶-۱

۶۰۳	.....	عنصر <connectionStrings> در فایل پیکربندی	۱۴-۶-۲
۶۰۴	.....	ADO.NET در لایه متصل	۱۴-۷
۶۰۵	.....	کار با شیء اتصال	۱۴-۷-۱
۶۰۸	.....	استفاده از شیء ConnectionStringBuilder	۱۴-۷-۲
۶۰۹	.....	استفاده از شیء command	۱۴-۷-۳
۶۱۱	.....	DATA READER استفاده از اشیاء	۱۴-۸
۶۱۳	.....	به دست آوردن چندین مجموعه رکورد با یک data reader	۱۴-۸-۱
۶۱۳	.....	ایجاد کتابخانه دسترسی به داده‌ها با قابلیت استفاده مجدد	۱۴-۹
۶۱۵	.....	افزودن روتین‌های اتصال	۱۴-۹-۱
۶۱۶	.....	افزودن روتین‌های وارد کردن رکورد جدید	۱۴-۹-۲
۶۱۷	.....	افزودن روتین‌های حذف رکورد	۱۴-۹-۳
۶۱۸	.....	افزودن روتین ویرایش رکورد	۱۴-۹-۴
۶۱۸	.....	افزودن روتین انتخاب رکوردها	۱۴-۹-۵
۶۱۹	.....	استفاده از کوثری‌های پارامتری	۱۴-۹-۶
۶۲۲	.....	اجرای روال‌های نخی‌شده	۱۴-۹-۷
۶۲۴	.....	AUTOLOTDAL واسط کاربری آزمایش کتابخانه	۱۴-۱۰
۶۲۴	.....	پیاپی‌سازی متد Main()	۱۴-۱۰-۱
۶۲۶	.....	پیاپی‌سازی متد ShowInstructions()	۱۴-۱۰-۲
۶۲۶	.....	پیاپی‌سازی متد ListInventory()	۱۴-۱۰-۳
۶۲۷	.....	پیاپی‌سازی متد DeleteCar()	۱۴-۱۰-۴
۶۲۸	.....	پیاپی‌سازی متد InsertNewCar()	۱۴-۱۰-۵
۶۲۸	.....	پیاپی‌سازی متد UpdateCarPetName()	۱۴-۱۰-۶
۶۲۹	.....	پیاپی‌سازی متد LookUpPetName()	۱۴-۱۰-۷
۶۳۰	.....	آشنایی با تراکنش‌های پایگاه داده	۱۴-۱۱
۶۳۰	.....	اعضای کلیدی شیء تراکنش در ADO.NET	۱۴-۱۱-۱
۶۳۱	.....	کاربرد عملی تراکنش‌ها در پروژه AutoLot	۱۴-۱۱-۲
۶۳۶	.....	جمع بندی فصل چهاردهم	



# فصل اول

## آشنایی با مبانی .NET.

به روز کردن دانش خود، علی‌رغم نگرانی و تردیدی که احتمالاً همواره به هنگام انجام آن احساس می‌کنید، امری اجتناب ناپذیر است. با در نظر گرفتن این نکته، در این کتاب به بررسی آخرین محصول ارائه شده توسط شرکت مایکروسافت در رابطه با مهندسی نرم افزار، یعنی زبان برنامه‌نویسی #C در محیط ویژوال استدیو ۲۰۱۰ بر اساس .NET Framework 4.0 خواهیم پرداخت.

هدف اصلی این فصل پایه‌ریزی شالوده‌ای است که تمامی فصل‌های آینده کتاب بر روی آن استوار خواهد بود. در ادامه، به بررسی سطح بالایی از مباحث کلیدی در .NET مانند اسمبلی‌ها<sup>۱</sup>، زبان مشترک میانی<sup>۲</sup> (CIL) و روش کامپایل به موقع<sup>۳</sup> (JIT) خواهیم پرداخت. در این راستا ضمن بررسی برخی کلمات کلیدی زبان #C، با ارتباط بین بسیاری از جنبه‌های مختلف .NET Framework، مانند زبان مشترک زمان اجرا<sup>۴</sup> (CLR)، سیستم مشترک انواع<sup>۵</sup> (CTS) و مشخصه‌های مشترک زبان<sup>۶</sup> (CLS)، آشنا خواهید شد.

در این فصل همچنین با کارآیی‌های ارائه شده توسط برخی از کلاس‌های کتابخانه پایه .NET 4.0 (که گاهی اوقات به صورت مخفف BCL و یا FCL نامیده می‌شود)<sup>۷</sup>، آشنا خواهید شد. در این راستا با طبیعت مستقل از زبان و مستقل از سیستم عامل .NET. برخورد خواهید داشت. معنی جمله فوق این است که .NET محدود به سیستم عامل ویندوز نیست. همان طور که حدس می‌زنید، بسیاری از این مفاهیم، در طی فصول آینده این کتاب، با جزئیات دقیق، مورد بررسی و کاوش قرار خواهند گرفت.

### ۱-۱ نگاهی به تاریخ قبل از .NET.

قبل از ورود به دنیای .NET، بررسی مشکلاتی که به پیدایش و تکوین محیط برنامه‌نویسی فعلی مایکروسافت انجامید، سازنده خواهد بود. برای به دست آوردن نگرش درستی در این مورد و به منظور یادآوری ریشه‌ها و درک محدودیت‌های محیط‌های پیشین برنامه‌نویسی، اجازه دهید این فصل را با بررسی مسائل تاریخی (البته نه درس تاریخ به آن صورت کسل کننده که می‌شناسیم!) شروع کنیم. بعد از آن، توجه خود را بر مزایای استفاده از #C 2010 و .NET 4.0 معطوف خواهیم ساخت.

---

<sup>1</sup> assemblies

<sup>2</sup> common intermediate language

<sup>3</sup> just in-time compilation

<sup>4</sup> common language runtime

<sup>5</sup> common type system

<sup>6</sup> common language specification

<sup>7</sup> Base class libraries/Framework class libraries

## ۱-۱-۲ کدنویسی واسط برنامه‌نویسی C/Windows

برنامه‌نویسی سنتی، برای توسعه نرم افزاری خانواده سیستم عامل‌های ویندوز، بر پایه زبان C و استفاده از واسط برنامه‌نویسی ویندوز<sup>۱</sup>، مبتنی بود. گرچه نرم افزارهای کاربردی زیادی با استفاده از این روش، به صورت موفقیت آمیز تولید شدند، ولی افراد کمی ممکن است با این حقیقت که به کارگیری روش فوق سخت، وقت‌گیر و پیچیده است، مخالفت کنند.

اولین مشکل آشکار این است که، برنامه‌نویسان C مجبور به مدیریت حافظه به صورت دستی، کار با اشاره‌گرها و درگیری با ساختارهایی هستند که از نقطه نظر کدنویسی نه تنها زیبا و خوش‌فرم نبوده بلکه زشت به نظر می‌آیند. علاوه بر این، با اینکه C زبانی ساخت یافته است، ولی از مزایای یک متدولوژی شیء‌گرا بی‌بهره است. وقتی به هزاران تابع و انواع داده متعددی که در واسط کاربری ویندوز، توسط چنین زبانی ایجاد شده است نظر می‌کنید، جای تعجب نخواهد بود که اینک این همه برنامه‌های کاربردی توأم با خطاهای ریز و درشت همچنان در دنیای نرم افزار سرگردانند.

## ۱-۱-۳ کدنویسی C++/MFC

زبان برنامه‌نویسی C++ گام بزرگی در جهت تکامل زبان C محسوب می‌شد. از جهات بسیاری، C++ را می‌توان به عنوان لایه‌ای شیء‌گرا بر روی زبان C به حساب آورد. بنابراین اگرچه برنامه‌نویسان C++ از مزایای ویژگی‌های اصلی شیء‌گرایی (کپسوله سازی، ارث بری و چندریختی) بهره‌مند می‌شوند، ولی باز هم درگیر جنبه‌های سخت زبان C خواهند بود (یعنی مدیریت حافظه دستی، کار با اشاره‌گرها و ساختارهای بدفرم زبان).

علی‌رغم پیچیدگی زیاد این زبان، برنامه‌های کاربردی زیادی با C++ نوشته شده و اکنون نیز موجود هستند. به عنوان مثال، کتابخانه کلاس‌های پایه میکروسافت (MFC)، به منظور تسهیل ایجاد برنامه‌های کاربردی ویندوز، مجموعه‌ای از کلاس‌های C++ را در اختیار برنامه‌نویس قرار می‌دهد. وظیفه اصلی MFC گردآوری مجموعه‌ای از روتین‌های مفید و کاربردی واسط برنامه‌نویسی ویندوز و ارائه آنها به صورت کلاس‌ها، ماکروها و انواع ابزارهای تولید کد (که ویزارد wizard نامیده می‌شوند) است. صرف نظر از کمک‌هایی که MFC به برنامه‌نویسان می‌کند، باز هم این حقیقت را نمی‌توان کتمان کرد که C++ ریشه در C داشته و برنامه‌نویسی با آن سخت و دشوار است.

## ۱-۱-۴ کدنویسی ویژوال بیسیک 6.0

بسیاری از برنامه‌نویسان بر اساس احساسی قلبی که در آنها تمایل به یک زندگی راحت‌تر را ایجاد می‌کرد، از دنیای C++ روی‌گردان شده و کم‌کم به سمت زبان‌های ساده‌تر و شکیل‌تر نظیر ویژوال بیسیک 6.0 (VB 6.0) متمایل شدند. این زبان به دلیل توانایی ایجاد واسط‌های کاربری پیچیده، کتابخانه‌های کد (به عنوان مثال سرویس دهنده‌های COM) و دسترسی به پایگاه‌های داده با کمترین زحمت و به سهولت، محبوبیت زیادی یافت. VB6 حتی

<sup>1</sup> windows application programming interface (API)



خیلی بیشتر از MFC، پیچیدگی‌های واسط ویندوز (windows API) را توسط ویزارد‌های تولید خودکار کد، انوع داده درونی VB، کلاس‌ها و توابع بیسیک، پنهان می‌کرد.

نقص اصلی VB 6.0 (که در NET رفع شده است)، این است که نمی‌توان آن را به طور کامل یک زبان شیء‌گرا<sup>۱</sup> محسوب کرد. در واقع حداکثر می‌توان آن را یک زبان مبتنی بر اشیاء (object based) دانست. به عنوان مثال توسط VB 6.0 نمی‌توان یک ارتباط از نوع "هست یک"<sup>۲</sup> بین کلاس‌ها ایجاد کرد (به معنی عدم وجود ارث بری کلاسیک) و علاوه بر آن این زبان، دارای توانایی استفاده از ساخت اشیاء به صورت پارامتری<sup>۳</sup> نیست؛ و بالاخره بدتر از همه این‌ها، VB 6.0 امکانی برای ایجاد برنامه‌های چند وظیفه‌ای فراهم نمی‌کند و در صورت اصرار بر انجام چنین کاری، برنامه‌نویس باید سختی کار با پایین‌ترین سطوح فراخوانی‌های (توابع) API را قبول کند (که در بهترین حالت پیچیده و در بدترین حالات، خطرناک هستند).

### ۵-۱-۱ کدنویسی به عنوان برنامه‌نویس Java

جاوا یک زبان برنامه‌نویسی شیء‌گراست (OOP) که ریشه‌های عمیقی در ++C دارد. همان طور که بسیاری از شما می‌دانید، قدرت جاوا به مراتب فراتر از ویژگی استقلال آن از سیستم عامل است. به عنوان یک زبان برنامه‌نویسی، جاوا بسیاری از جنبه‌های ناخوش‌آیند ++C را تصحیح کرده و کار برنامه‌نویسان را با ارائه بسته‌های<sup>۴</sup> از پیش آماده شامل تعاریف انواع مختلف و پرکاربرد، تسهیل کرده است. برنامه‌نویسان جاوا، با استفاده از انواع آماده فوق، امکان ایجاد برنامه‌های خالص جاوا با امکانات دسترسی به پایگاه‌های داده، توانایی استفاده از سرویس‌های پیام<sup>۵</sup>، واسط‌های منتنی بر وب و رابط‌های کاربری غنی ویندوز را دارند.

اگرچه جاوا زبانی قدرتمند است، ولی یک مشکل بالقوه در برنامه‌نویسی به زبان جاوا این است که از ابتدا تا انتهای پروژه و در تمامی فازهای توسعه نرم‌افزار، همواره باید از واسط‌های مبتنی بر جاوا استفاده کنید. در حقیقت، با جاوا، امید کمی برای اختلاط زبان‌ها باقی می‌ماند؛ اگرچه این نکته با اهداف اولیه این زبان مطابقت ندارد. هم اکنون در دنیا میلیون‌ها خط کد وجود دارد که به صورت ایده‌آل تمایل به ترکیب با جاوا دارند (که همان طور که گفته شد، غیر ممکن یا مشکل است //). متأسفانه جاوا انجام این کار را مشکل می‌کند.

### ۶-۱-۱ راه حل NET.

البته شاید مطالب بالا برای یک درس خلاصه شده تاریخی کمی زیاد بود. آخرین سخن این است که برنامه‌نویسی ویندوز همیشه کار مشکلی بوده است. همان طور که در ادامه این کتاب خواهید دید، NET Framework مجموعه‌ای نرم افزاری برای ایجاد برنامه‌های کاربردی در خانواده سیستم عامل‌های ویندوز و همچنین بر روی

<sup>۱</sup> object oriented programming language

<sup>۲</sup> is-a relationship

<sup>۳</sup> parameterized object construction

<sup>۴</sup> یک بسته یا package در جاوا چیزی شبیه به یک فضای نامی در NET، شامل انواع (کلاس‌ها) و متدهاست.

<sup>۵</sup> messaging support

بسیاری از سیستم عامل‌های دیگر از جمله Mac OS X و انواع سیستم‌های Unix/Linux است. در اینجا نگاهی سریع به برخی از ویژگی‌های کلیدی NET. خواهیم داشت:

- قابلیت اختلاط<sup>۱</sup> با کدهای موجود. این البته ویژگی خوبی است. کدهای باینری موجود COM می‌توانند با باینری‌های جدید NET. ترکیب شوند و بر عکس. از ارائه نگارش NET 4.0. به بعد، این ویژگی با معرفی کلمه کلیدی dynamic از گذشته نیز ساده‌تر شده است.
- حمایت از چندین زبان برنامه‌نویسی. برنامه‌های کاربردی NET. می‌توانند توسط زبان‌های C#, Visual Basic, F# و غیره نوشته شوند.
- یک موتور زمان اجرای مشترک<sup>۲</sup> برای همه زبان‌های NET.. جنبه‌ای مهم از این موتور زمان اجرا، مجموعه‌ای از انواع است که همه زبان‌های NET. قادر به فهم و استفاده از آن هستند.
- قابلیت کامل ترکیب با زبان‌های دیگر؛ از جمله قابلیت ارث‌بری بین زبان‌ها، قابلیت مدیریت استثناءها بین زبان‌ها و هینطور قابلیت دیباگ کد بین زبان‌ها.
- وجود کتابخانه پایه کلاس‌ها. این کتابخانه وسیع، مانند سپری حفاظتی بر پیچیدگی‌های فراخوانی‌های API عمل کرده و مدلی شیء‌گرا، قابل استفاده برای همه زبان‌های NET.، ارائه می‌دهد.
- مدلی ساده برای انتشار و نصب برنامه. در NET. نیازی به ثبت واحدهای باینری در رجیستری سیستم نیست. علاوه بر این، NET. اجازه می‌دهد که نگارش‌های متفاوتی از یک dll. روی یک کامپیوتر، به صورت هم زمان، موجود باشند.

همان طور که از نکات بالا دریافته‌اید، NET. کوچکترین ارتباطی با COM ندارد (غیر از اینکه منشاء هر دو میکروسافت است!). در واقع تنها یک راه برای کار مشترک NET. و COM وجود دارد و آن استفاده از همان امکان اختلاط بین زبان‌هاست (که در بالا به آن اشاره شد).

## ۲-۱ معرفی پایه‌های اصلی NET.

اکنون که با برخی از مزایای ارائه شده توسط NET. آشنا شده‌اید، اجازه دهید نگاهی داشته باشیم به سه واحد کلیدی که همه اینها را ممکن می‌سازند، یعنی CLR، CTS و CLS. از نقطه نظر یک برنامه‌نویس، NET. می‌تواند به صورت یک محیط زمان اجرا به همراه کتابخانه‌ای وسیع از کلاس‌ها تصور شود. لایه زمان اجرا به صورت مناسبی، زبان مشترک زمان اجرا<sup>۳</sup> یا به اختصار، CLR نامیده می‌شود. وظیفه اصلی CLR یافتن، بارگذاری و

<sup>۱</sup> interoperability

<sup>۲</sup> common runtime engine

<sup>۳</sup> common language runtime

مدیریت انواع موجود در NET، برای برنامه‌نویس است. علاوه بر این، CLR مسئول برخی اعمال سطح پایین مانند مدیریت حافظه، میزبانی برنامه<sup>۱</sup>، مدیریت واحدهای اجرایی داخل حافظه<sup>۲</sup> و اعتبار سنجی‌های امنیتی است.

یکی دیگر از پایه‌های تشکیل دهنده NET، سیستم انواع مشترک<sup>۳</sup> یا CTS است. CTS، تمامی انواع و ساختارهایی که ممکن است در زمان اجرا مورد نیاز باشند به علاوه ارتباطات بین این موجودیت‌ها<sup>۴</sup> و اینکه چگونه باید در قالب‌های فراداده NET<sup>۵</sup> نمایش داده شوند را، تعریف می‌کند.

به این نکته توجه کنید که لازم نیست هر یک از زبان‌های قابل کاربرد در NET. از همه جنبه‌های تعریف شده در CTS حمایت کنند. مشخصه‌های مشترک زبان یا CLS، جزء دیگری است که مجموعه‌ای از انواع و ساختارهای مشترکی که همه زبان‌های فوق باید از آنها استفاده کنند، در آن تعریف شده است. معنی جملات بالا این است که اگر انواعی ایجاد کنید که تنها از جنبه‌های تعریف شده در CLS استفاده کرده باشند، می‌توانید مطمئن باشید که همه زبان‌های مختلف NET. توانایی کاربرد آنها را خواهند داشت. همین‌طور برعکس، اگر از انواع داده‌ای استفاده کنید که خارج از مرزهای CLS قرار دارند، هرگز نمی‌توانید تضمین کنید که هر زبان برنامه‌نویسی قابل استفاده در NET، امکان استفاده از کتابخانه کد شما را داشته باشد. خوشبختانه همان‌طور که در ادامه همین فصل خواهید دید، به سادگی می‌توان از کامپایلر تقاضا کرد که کل برنامه را برای سازگاری یا عدم سازگاری آن با مشخصه‌های CLS آزمایش کند.

### ۱-۲-۱ نقش کتابخانه کلاس‌های پایه

علاوه بر مشخصه‌های تعریف شده در CLR و CTS/CLS، کتابخانه پایه‌ای از کلاس‌ها در NET. تعریف شده است که در دسترس همه زبانهاست. کتابخانه فوق نه تنها کلاس‌های پایه‌ای در رابطه با ابتدایی‌ترین نیازهای برنامه‌نویسی همچون واحدهای اجرایی داخل حافظه، فایل‌های ورودی/خروجی (IO)، سیستم‌های پردازش گرافیکی و روش‌های تعامل با بسیاری از ابزارهای سخت افزاری را در خود گردآوری کرده است، بلکه سرویس‌های فراوانی را در رابطه با بیشترین نیازهای برنامه‌های کاربردی نیز ارائه می‌دهد.

به عنوان مثال، در کتابخانه کلاس‌های پایه، انواعی در رابطه با دسترسی به پایگاه‌های داده، پردازش و دستکاری اسناد XML، برنامه‌نویسی ملاحظات امنیتی و همین‌طور ایجاد واسطه‌های کاربری مبتنی بر وب، فرم‌های ویندوز و پروژه‌های کنسول، تعریف شده‌اند. اگر بخواهیم از دید برنامه‌نویسی سطح بالا به ارتباط بین CLR و CTS و CLS نگاه کنیم، شکل ۱-۱ را خواهیم داشت.

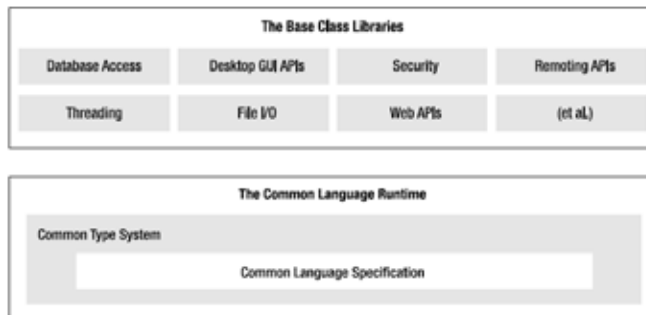
<sup>1</sup> application hosting

<sup>2</sup> thread management

<sup>3</sup> common type system

<sup>4</sup> entities

<sup>5</sup> metadata formats



شکل ۱-۱

## ۱-۲-۲ ویژگی‌های زبان C#

با توجه به فاصله زیادی که NET از تکنولوژی‌های قبل از خود گرفته بود، مایکروسافت زبان C# را منحصرآ برای آن ایجاد کرد. C# زبانی است که از نظر شکل ظاهری دستورات (syntax) شباهت زیادی با جاوا دارد؛ با این حال در نظر گرفتن C# به عنوان انشعاب یا نسخه‌ای از جاوا، نادرست است. هم جاوا و هم C# عضوی از خانواده زبان‌های مبتنی بر C محسوب می‌شوند و به همین دلیل دارای دستوراتی با اشکال مشابه هستند.

حقیقت این است که بسیاری از ساختارهای C# با پیروی از جنبه‌های مختلف VB 6.0 و ++C شکل گرفته‌اند. به عنوان مثال، درست مانند VB، برنامه‌نویس می‌تواند در کلاس‌ها خصوصیت تعریف کرده (بر خلاف متدهای getter و setter متداول در جاوا) و یا متدهایی با تعداد نامحدود آرگومان ایجاد کند. در C#، مانند ++C می‌توان عملگرها را سربارگذاری کرده و یا انواع شمارشی تعریف کرد.

علاوه بر آن، در مسیر مطالعه این کتاب، در خواهید یافت که C# از ویژگی‌هایی مانند عبارات لاند<sup>۱</sup> و انواع بی‌نام<sup>۲</sup> که معمولاً در زبان‌های تابع مدار (مثل LISP و Haskell)، یافت می‌شوند نیز حمایت می‌کند. باز هم فراتر از همه اینها، با ابداع LINQ<sup>۳</sup>، زبان C# دارای ویژگی‌هایی منحصر بفرد است که در سایر زبان‌های برنامه‌نویسی یافت نمی‌شوند. با تمام اینها نباید این حقیقت را کتمان کرد که قسمت اعظم C# از زبان‌های مبتنی بر C اقتباس شده است.

با توجه به این حقیقت که C# ترکیبی از بهترین ویژگی‌های چند زبان برنامه‌نویسی را در خود دارد، نتیجه نهایی، کدی تمیز و خوانا مانند برنامه‌های جاوا است که سادگی VB6 را داشته و همان قدرت و انعطاف پذیری ++C را ارائه می‌دهد. در زیر، لیستی از ویژگی‌های C# را ملاحظه می‌کنید که در همه نگارش‌های آن موجودند.

- نیازی به برنامه‌نویسی با اشاره‌گرها نیست.

<sup>1</sup> lambda expressions

<sup>2</sup> anonymous types

<sup>3</sup> language integrated query

- جمع‌آوری خودکار اشیاء به جا مانده در حافظه<sup>۱</sup>؛ به این ترتیب C# دستوری به صورت delete ندارد.
  - روش‌های ساده‌ای برای ایجاد کلاس‌ها، واسط‌ها<sup>۲</sup>، ساختارها، انواع شمارشی<sup>۳</sup> و عامل‌ها<sup>۴</sup>.
  - امکان سربارگذاری عملگرها، به روشی ساده.
  - امکان اضافه کردن صفت<sup>۵</sup> به تعریف یک نوع (مثلا یک کلاس)، به صورتی که بعدا بتوان از تاثیر این صفت در رفتار نوع مورد نظر استفاده کرد.
- با انتشار NET 2.0 (در سال ۲۰۰۵) ویژگی‌های جدیدی به C# اضافه شد که برخی از مهم‌ترین آنها عبارتند از:
- امکان ایجاد انواع و اعضای عام<sup>۶</sup>.
  - امکان ایجاد متدهای بی‌نام. می‌توان در هر محلی که احتیاج به یک عامل باشد، به سادگی از یک تابع بی‌نام استفاده کرد.
  - تغییرات زیادی در کاربرد مدل رویداد/عامل.
  - امکان تقسیم کد مربوط به تعریف یک نوع، توسط کلمه کلیدی partial، در چند فایل.
- NET 3.5. ویژگی‌های بیشتری به زبان C# اضافه کرد، از جمله:
- امکان استفاده از کوئری‌های LINQ که می‌توانند بر روی هر نوع داده‌ای به کار روند.
  - حمایت از انواع بی‌نام که امکان مدل‌سازی شکل یک نوع را، به جای رفتار آن، فراهم می‌کنند.
  - امکان گسترش کارآیی‌های یک نوع از طریق متدهای توسعه یافته.
  - امکان استفاده از عملگر لاندا (>=) که کار با عامل‌ها را ساده‌تر می‌کند.
  - روش جدیدی برای مقداردهی آغازین یک شیء به صورتی که در هنگام ایجاد شیء جدید بتوان خصوصیات آن را نیز مشخص کرد.
- نگارش فعلی NET 4.0. باز هم C# را گسترش داده و ویژگی‌های جدیدی به آن افزوده است. گرچه لیست عناوین زیر ممکن است محدود به نظر برسد، در طی مطالعه فصول این کتاب با اهمیت و فوائد آنها آشنا خواهید شد.

---

<sup>1</sup> automatic garbage collection

<sup>2</sup> interfaces

<sup>3</sup> enumerations

<sup>4</sup> delegates

<sup>5</sup> attribute

<sup>6</sup> generic type and members

- حمایت از کاربرد پارامترهای دلخواه برای متدها به همراه آرگومان‌های نام‌دار.
- حمایت از روش جستجوی اعضای یک نوع در زمان اجرا، با استفاده از کلمه کلیدی `dynamic`.
- در رابطه با نکته فوق، اکنون C# نگارش 4.0 کار با سرویس دهنده‌های COM را به مقدار زیادی ساده کرده است.
- کار با انواع عام در این نگارش باز هم ساده‌تر شده است.

مهم‌ترین نکته‌ای که در رابطه با زبان C# باید مد نظر داشته باشید این است که تنها می‌توان از آن برای ایجاد کدی استفاده کنید که در محیط زمان اجرای NET<sup>۱</sup>، به کار رود (هرگز نمی‌توانید از C# برای ایجاد سرویس دهنده‌های COM یا واسط‌های برنامه‌نویسی (API) C/C++ استفاده کنید). از نظر فنی، کدی که برای محیط زمان اجرای NET، به کار رود را کد مدیریت شده<sup>۲</sup> می‌گویند. فایل باینری در بر گیرنده کد مدیریت شده فوق را اسمبلی (assembly) می‌گویند. همین‌طور کدی را که نتوان در محیط زمان اجرای NET، به کار برد، کد مدیریت نشده<sup>۳</sup> می‌گویند.

### ۳-۱ بررسی اسمبلی‌های NET.

صرف نظر از زبانی که به کار می‌برید، به این نکته توجه داشته باشید که اگر چه فایل‌های باینری NET، دارای پسوندی مشابه سرویس دهنده‌های COM و باینری‌های مدیریت نشده ویندوز هستند (\*.\*.exe یا \*.dll)، با این حال این دو دارای هیچ گونه مشابهت داخلی نیستند. به عنوان مثال باینری‌های dll در NET، متدها را برای تسهیل تبادل اطلاعات با محیط زمان اجرای COM صادر نمی‌کنند (با توجه به اینکه NET، اصلاً COM نیست).

علاوه بر این، باینری‌های NET، با کتابخانه انواع COM تعریف نشده و در رجیستری هم ثبت نمی‌گردند. شاید مهم‌تر از همه اینها این باشد که باینری‌های فوق شامل دستورات وابسته به سیستم عامل بخصوصی نبوده و بلکه بر عکس دارای دستورات غیر مرتبط با سیستم عامل (زبان میانی<sup>۴</sup>) می‌باشند. شکل ۲-۱ نشان دهنده این موارد است.

---

■ نکته‌ای در مورد IL، در نگارش‌های اولیه NET، از عبارت مخفف MSIL (Microsoft Intermediate Language) برای اشاره به زبان میانی استفاده می‌شد. ولی در نگارش‌های بعدی از عبارت CIL (Common Intermediate Language) استفاده شد. بنابراین در متن این کتاب هر دو مورد به یک موجودیت اشاره داشته و مترادف هستند.

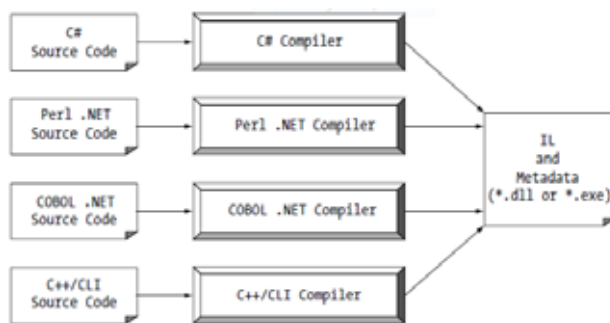
---

<sup>۱</sup> .NET runtime

<sup>۲</sup> managed code

<sup>۳</sup> unmanaged code

<sup>۴</sup> platform-agnostic intermediate language (IL)



شکل ۱-۲

وقتی یک فایل \*.dll یا \*.exe توسط کامپایلر .NET تولید می‌شود، آن را اسمبلی (assembly) می‌گویند. در فصل‌های آتی این کتاب با جزئیات فراوانی از اسمبلی‌های .NET آشنا خواهید شد. با این حال برای آشنایی با محیط زمان اجرای .NET، باید درک اولیه‌ای از برخی خصوصیات اصلی این گونه فایل‌ها (که دارای قالب جدیدی هستند) داشته باشید.

همان‌طور که اشاره شد، محتوای یک اسمبلی، کد CIL است. این کد را از این جهت که تا زمانی که مطلقاً ضروری نشده، به دستورات قابل فهم برای سیستم عامل ترجمه نمی‌گردد، می‌توانید شبیه bytecode در جاوا تصور کنید. تا زمانی که مطلقاً ضروری نشده، به معنی رسیدن به نقطه‌ای است که بلاکی از دستورات CIL داخل اسمبلی (مانند یک متد/تابع) برای به کارگیری در محیط زمان اجرای .NET، فراخوانی شود.

اسمبلی‌ها علاوه بر دستورات CIL حاوی فراداده‌هایی<sup>۱</sup> هستند که به صورت خیلی روشن مشخصات هر نوع به کار رفته در اسمبلی را تشریح کرده‌اند. به عنوان مثال اگر اسمبلی دارای کلاسی به نام SportsCar باشد، فراداده نوع، تمام جزئیات مربوط به کلاس پایه SportsCar (کلاسی که کلاس مذکور از آن مشتق شده است) به همراه تمام واسط‌های پیاده‌سازی شده توسط SportsCar و شرح کاملی از هر یک از اعضای کلاس فوق را در بر دارد. فراداده .NET همیشه درون هر اسمبلی تولید شده توسط کامپایلر .NET، موجود است.

و در نهایت، علاوه بر CIL و فراداده نوع، خود اسمبلی‌ها هم توسط فراداده‌ای تشریح می‌شوند که آن را به طور رسمی، مانیفست (manifest) می‌گویند. اصلی‌ترین اطلاعات این مانیفست عبارتند از داده‌هایی در مورد نگارش اسمبلی مورد نظر و لیستی از تمامی اسمبلی‌های ارجاع شده توسط آن (که برای اجرای مناسب اسمبلی، مورد نیاز هستند). در طی چند فصل آینده این کتاب با ابزار گوناگونی برای آزمایش نوع یک اسمبلی، فراداده آن و اطلاعات مانیفست مربوط به آن، آشنا خواهید شد.

<sup>۱</sup> metadata

### ۱-۳-۱ اسمبلی‌های تک فایلی و چند فایلی

در اکثر موارد اسمبلی، همان فایل باینری \*.dll یا \*.exe است. بنابراین اگر یک کتابخانه کلاس (class library) به صورت یک فایل \*.dll در .NET ایجاد می‌کنید، می‌توانید تصور کنید اسمبلی و فایل باینری یکی هستند. به همین صورت چنانچه یک برنامه ویندوز به صورت \*.exe ایجاد کنید، باز هم می‌توانید فایل اجرایی فوق را اسمبلی پروژه نیز محسوب کنید. با این حال همان طور که در فصول آینده ملاحظه خواهید کرد، این موضوع همیشه به همین صورت صادق نیست. اگر بخواهیم به صورت فنی بیان کنیم باید بگوییم، در صورتی که اسمبلی ایجاد شده مشتمل بر یک ماجول \*.dll یا \*.exe باشد، دارای یک اسمبلی تک فایلی هستیم. اسمبلی‌های تک فایلی تمامی CIL مورد نیاز، فراداده‌ها و مانیفست مربوطه را در یک بسته گرد آورده‌اند.

در نقطه مقابل، اسمبلی‌های چند فایلی از چندین فایل باینری تشکیل شده‌اند که هر کدام را یک ماجول<sup>۱</sup> می‌گویند. هنگامی که یک اسمبلی چند فایلی ایجاد می‌شود، یکی از باینری‌ها (که باینری اصلی یا main module نامیده می‌شود) باید دارای مانیفست اسمبلی و احتمالاً دستورات CIL و فراداده‌های مربوط به انواع مختلف به کار برده شده باشد. سایر ماجول‌های مربوطه دارای مانیفست، CIL و فراداده نوع در سطح ماجول خواهند بود. همان طور که ممکن است حدس بزنید، ماجول اصلی دارای مستندات در رابطه با مجموعه ماجول‌های فرعی در مانیفست اسمبلی خواهد بود.

---

■ توجه داشته باشید که در محیط ویژوال استدیو ۲۰۱۰ فقط اسمبلی‌های تک فایلی ایجاد می‌شوند. در موارد نادری که احتیاج به ساخت اسمبلی‌های چند فایلی داشته باشید، باید از ابزار مربوطه در خط فرمان (command line) استفاده کنید.

---

### ۱-۳-۲ نقش زبان مشترک میانی

در این مرحله اجازه دهید کد CIL، فراداده نوع و مانیفست اسمبلی را با جزئیات بیشتری مورد بررسی قرار دهیم. جایگاه CIL فراتر از مجموعه دستوراتی است که در هر زبان برنامه‌نویسی وجود داشته و معمولاً وابسته به سیستم عامل بخصوصی هستند. به عنوان مثال، تکه برنامه C# زیر، یک ماشین حساب را مدل سازی می‌کند. در حال حاضر زیاد نگران دستورات به کار رفته نباشید و بیشتر به چگونگی کاربرد متد Add() در کلاس Calc دقت کنید.

```
// Calc.cs
using System;
namespace CalculatorExample
{
    // This class contains the app's entry point.
    class Program
    {
```

---

<sup>1</sup> module



```

static void Main()
{
    Calc c = new Calc();
    int ans = c.Add(10, 84);
    Console.WriteLine("10 + 84 is {0}.", ans);

    // Wait for user to press the Enter key before shutting down.
    Console.ReadLine();
}

// The C# calculator.
class Calc
{
    public int Add(int x, int y)
    { return x + y; }
}

```

پس از کامپایل برنامه فوق توسط کامپایلر C# (csc.exe)، در نهایت یک اسمبلی \*.exe به دست می‌آورد که حاوی مانیفست، دستورات CIL و فراداده‌های تشریح‌کننده تمامی جنبه‌های مختلف کلاس‌های Calc و Programs است.

■ در فصل ۲ جزئیات مربوط به کامپایلر C# و همین‌طور چندین محیط مجتمع توسعه نرم‌افزار (IDE) مانند ویژوال استدیو ۲۰۱۰ و Microsoft Visual C# 2010 Express را بررسی خواهیم کرد.

به عنوان مثال اگر با کمک برنامه ildasm.exe (که به زودی در همین فصل معرفی خواهد شد)، اسمبلی ایجاد شده را باز کرده و مورد کاوش قرار دهید، خواهید دید که متد Add() به زبان CIL، به صورت زیر نشان داده شده است:

```

.method public hidebysig instance int32 Add(int32 x,
int32 y) cil managed
{
    // Code size 9 (0x9)
    .maxstack 2
    .locals init (int32 v_0)
    IL_0000: nop
    IL_0001: ldarg.1
    IL_0002: ldarg.2
    IL_0003: add
    IL_0004: stloc.0
    IL_0005: br.s IL_0007
    IL_0007: ldloc.0
    IL_0008: ret
} // end of method Calc::Add

```

در این مرحله نگران این نباشید که متوجه معنی دستورات کد فوق نمی‌شوید؛ نکته حائز اهمیت فقط این است که C# به جای تولید کد قابل فهم برای یک سیستم خاص (زبان ماشین)، کد میانی ایجاد می‌کند (CIL).

به علاوه در نظر داشته باشید که حقیقت فوق در مورد همه زبان‌های NET. صادق است. برای آشنایی بیشتر با موضوع، فرض کنید برنامه بالا را با زبان ویژوال بیسیک بنویسیم.

```
' Calc.vb
Imports System

Namespace CalculatorExample

    ' A VB "Module" is a class that contains only
    ' static members.
    Module Program

        Sub Main()
            Dim c As New Calc
            Dim ans As Integer = c.Add(10, 84)
            Console.WriteLine("10 + 84 is {0}.", ans)
            Console.ReadLine()
        End Sub

    End Module

    Class Calc
        Public Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
            Return x + y
        End Function
    End Class
```

End Namespace

اکنون اگر نگاهی به کد CIL و متد Add() داخل آن بیندازید، مشابهت زیادی بین این کد و آنچه که توسط کامپایلر C# ایجاد شده بود، خواهید یافت (کامپایلر VB یعنی vbc.exe کمی آن را تغییر داده است).

```
.method public instance int32 Add(int32 x,
int32 y) cil managed
{
    // Code size 8 (0x8)
    .maxstack 2
    .locals init (int32 v_0)
    IL_0000: ldarg.1
    IL_0001: ldarg.2
    IL_0002: add.ovf
    IL_0003: stloc.0
    IL_0004: br.s IL_0006
    IL_0006: ldloc.0
    IL_0007: ret
} // end of method Calc::Add
```

### ۳-۳-۱ دلایل کاربرد CIL

در این مرحله ممکن است این سوال برایتان مطرح شود که اصلا چرا کد اصلی برنامه به جای اینکه یکباره به زبان ماشین ترجمه شود باید ابتدا به زبان CIL تبدیل گردد؟ یکی از مزایای عمل کردن به این روش، قابلیت ترکیب زبانها است. همان طور که تا کنون دیده‌اید، همه کامپایلرهای NET، تقریبا کد مشابهی تولید می‌کنند؛ بنابراین تمامی این زبانها در یک محیط باینری (که به درستی تعریف شده باشد) می‌توانند با یکدیگر تعامل داشته باشند.

علاوه بر این، از آنجا که CIL مستقل از سیستم است، خود NET Framework هم به همین صورت مستقل از سیستم بوده و در نتیجه همان کارآیی‌هایی را ارائه می‌دهد که برنامه‌نویسان جاوا همیشه به آنها عادت داشته‌اند (یعنی اجرای یک کد در سیستم عامل‌های مختلف). در حقیقت یک استاندارد بین‌المللی برای #C وجود داشته و هم‌اکنون مجموعه بزرگی از پیاده‌سازی‌های NET، برای سیستم عامل‌های غیر ویندوزی موجود است؛ در حالیکه بر خلاف جاوا، NET امکان ایجاد پروژه را بر اساس زبان انتخابی برنامه‌نویس فراهم می‌آورد.

### ۳-۳-۴ ترجمه CIL به زبان ماشین

با توجه به این حقیقت که اسمبلی‌ها به جای کد زبان ماشین، حاوی کدی به زبان میانی هستند، محتوای آنها (همان کد میانی CIL) باید قبل از اجرای نهایی کامپایل شود. عنصری که مسئول ترجمه کد میانی به دستورات قابل فهم برای CPU است، کامپایلری به نام JIT است که برخی اوقات Jitter هم نامیده می‌شود. محیط زمان اجرای NET، برای هر پردازشگر خاصی، کامپایلر JIT مخصوص به آن را که به بهترین صورت برای همان سیستم بهینه‌سازی شده است، ارائه می‌دهد.

به عنوان مثال اگر قرار است برنامه‌ای را برای اجرا روی ابزار قابل حمل مانند موبایل‌های ویندوز بنویسید، Jitter به صورتی پیکربندی شده که در محیط‌های با حجم اندک حافظه بتواند کار کند. بر عکس، اگر برنامه‌ای برای اجرا روی یک سرورس دهنده (که معمولا حجم حافظه، آخرین مشکل آنهاست) نوشته شده باشد، Jitter مربوطه برای کار در محیط‌هایی با حافظه بالا بهینه‌سازی شده است. با این رویکرد، برنامه‌نویسان می‌توانند کدی بنویسند که قابلیت اجرا در محیط‌های متفاوت را داشته باشد.

علاوه بر این، وقتی کامپایلر Jitter دستورات درون CIL را به فرامین یک پردازشگر بخصوص ترجمه می‌کند، نتایج را در حافظه، به بهترین صورت برای سیستم عامل مورد نظر، ذخیره (cache) می‌کند. به این ترتیب اگر متدی به نام PrintDocument() یک بار فراخوانی شود، دستورات میانی آن به زبان ماشین در حال اجرا، ترجمه شده و برای استفاده‌های بعدی در حافظه باقی می‌مانند؛ نتیجه اینکه در فراخوانی‌های بعدی دیگر نیازی به ترجمه مجدد این متد نخواهد بود.

### ۵-۳-۱ نقش فراداده‌های نوع در .NET.

اسمبلی‌ها در .NET علاوه بر فرامین CIL دارای فراداده کامل و دقیقی هستند که نه تنها تمامی انواع موجود در فایل باینری (کلاس‌ها، ساختارها، انواع شمارشی و ...) بلکه همه اعضای انواع فوق را نیز تشریح می‌کند (به عنوان مثال خصوصیات، متدها و رویدادها را). خوشبختانه این همیشه وظیفه کامپایلر است (و نه برنامه‌نویس) که مناسب‌ترین فراداده را در اسمبلی نهایی قرار دهد.

به منظور نمایش قالب فراداده‌های .NET، نگاهی خواهیم داشت به فراداده تولید شده برای متد Add() که در کلاس Calc (به زبان C#) در بخش‌های گذشته ایجاد کردیم (فراداده تولید شده برای VB کاملاً مشابه است).

```
TypeDef #2 (02000003)
```

```
-----
TypDefName: CalculatorExample.Calc (02000003)
```

```
Flags : [NotPublic] [AutoLayout] [Class]
```

```
[AnsiClass] [BeforeFieldInit] (00100001)
```

```
Extends : 01000001 [TypeRef] System.Object
```

```
Method #1 (06000003)
```

```
-----
MethodName: Add (06000003)
```

```
Flags : [Public] [HideBySig] [ReuseSlot] (00000086)
```

```
RVA : 0x00002090
```

```
ImplFlags : [IL] [Managed] (00000000)
```

```
CallConvntn: [DEFAULT]
```

```
hasThis
```

```
ReturnType: I4
```

```
  2 Arguments
```

```
  Argument #1: I4
```

```
  Argument #2: I4
```

```
  2 Parameters
```

```
  (1) ParamToken : (08000001) Name : x flags: [none] (00000000)
```

```
  (2) ParamToken : (08000002) Name : y flags: [none] (00000000)
```

علاوه بر محیط زمان اجرای .NET، ابزار توسعه بسیاری هم هستند که به مقدار زیادی از فراداده‌ها استفاده می‌کنند. به عنوان مثال کادر کمکی<sup>۱</sup> که به هنگام تایپ دستورات در محیط‌های کدنویسی ویزوال استدیو ظاهر می‌شود، از فراداده موجود در اسمبلی استفاده می‌کند. ابزارهای مرور اشیاء<sup>۲</sup>، دیباگ برنامه و خود کامپایلر C# نیز از فراداده استفاده می‌کنند. در واقع، فراداده ستون فقرات بسیاری از تکنولوژی‌های .NET. از جمله بنیان تبادل اطلاعات ویندوز<sup>۳</sup> (WCF)، اعمال انعکاسی<sup>۴</sup> و تثبیت اشیاء به وسیله سریال‌سازی<sup>۵</sup> است.

---

<sup>۱</sup>IntelliSense

<sup>۲</sup>object browsing utilities

<sup>۳</sup>windows communication foundation

<sup>۴</sup>reflection

<sup>۵</sup>serialization

### ۶-۳-۱ نقش مانیفست اسمبلی

آخرین نکته و نه البته کم اهمیت‌ترین آنها، این است که یک اسمبلی دارای فراداده‌ای است که خود آن را تشریح می‌کند (این فراداده را مانیفست manifest می‌نامند). از نکاتی که در مانیفست درج شده‌اند، می‌توان از تمامی اسمبلی‌های خارجی ارجاع شده به وسیله اسمبلی جاری، شماره نگارش اسمبلی و اطلاعات مربوط به حق کپی نام برد. درست مانند فراداده‌های تشریح کننده انواع، در این مورد هم، تولید مانیفست وظیفه کامپایلر است. در ادامه، قسمتی از مانیفست تولید شده برای کد Calc.cs را، که قبلاً در همین فصل ایجاد کردیم، مشاهده می‌کنید (فرض کنید نام اسمبلی ایجاد شده توسط کامپایلر Calc.exe است).

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 4:0:0:0
}
.assembly Calc
{
  .hash algorithm 0x00008004
  .ver 1:0:0:0
}
.module Calc.exe
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
```

به صورت خلاصه، این مانیفست علاوه بر جنبه‌های مختلف خود اسمبلی (مثلاً شماره نگارش و نام ماجول)، شامل اسامی اسمبلی‌های خارجی مورد نیاز Calc.exe نیز هست. در فصول آینده این کتاب، جزئیات بیشتری از ویژگی‌های مانیفست را مطالعه خواهیم کرد.

### ۴-۱ بررسی سیستم مشترک انواع

یک اسمبلی ممکن است دارای هر تعداد انواع مختلف باشد. در دنیای .NET، نوع، نامی عمومی برای عنصری است از مجموعه {class و interface و structure و enumeration و delegate}، وقتی با یکی از زبان‌های .NET، پروژه‌ای ایجاد می‌کنید، به احتمال زیاد با بسیاری از این انواع برخورد خواهید کرد. پروژه‌ای، به عنوان مثال، ممکن است کلاسی را تعریف کند که خود چندین واسط (interface) را پیاده‌سازی کرده باشد. به همین صورت امکان دارد متدی در یکی از واسط‌های یاد شده، نوعی شمارشی (enumeration) را به عنوان پارامتر ورودی بپذیرد و باز محتمل است که همین متد، ساختاری (structure) را به عنوان خروجی باز گرداند.

به خاطر دارید که CTS بیان کننده مشخصه‌هایی (قوانینی) است که چگونگی تعریف انواع برای اجرا در CLR را تعیین می‌کند. به طور کلی تنها کسانی که باید خود را درگیر جزئیات CTS کنند، افرادی هستند که ابزار محیط‌های نرم‌افزاری را تولید کرده و یا برای زبان‌های .NET، کامپایلر می‌نویسند. با این حال برای همه

برنامه‌نویسان، آشنایی با طریقه به کار بردن پنج نوع اصلی تعریف شده توسط CTS، ضروری است. در زیر، خلاصه‌ای از این موارد را ملاحظه می‌کنید.

### ۱-۴-۱ نوع کلاس در CTS

هر یک از زبان‌های NET. باید حداقل از نوع کلاس، که سنگ زیربنای شیء‌گرایی است، حمایت کنند. یک کلاس ممکن است دارای اعضای به هر تعداد باشد (مانند سازنده‌ها، خصوصیات، متدها و رویدادها). همچنین یک کلاس ممکن است دارای عناصر داده (فیلد field) باشد. کلاس‌ها در C# توسط کلمه کلیدی class تعریف می‌شوند.

```
// A C# class type with 1 method.
class Calc
{
    public int Add(int x, int y)
    { return x + y; }
}
```

در فصل ۵ کار ایجاد کلاس‌های CTS را آغاز خواهید کرد، با این حال جدول ۱-۱ برخی مشخصه‌های کلاس‌ها را لیست کرده است.

مشخصه کلاس	معنی و مفهوم
آیا کلاس بسته شده <sup>۱</sup> است؟	کلاس‌های بسته شده نمی‌توانند به عنوان کلاس پایه برای سایر کلاس‌ها به کار روند.
آیا کلاس واسطی را پیاده‌سازی می‌کند؟	یک واسط مجموعه‌ای از اعضای مجرد است که قراردادی را بین شیء و کاربر (برنام‌ای که از شیء استفاده می‌کند) شیء، تعریف می‌کنند. CTS اجازه می‌دهد که یک کلاس از هر تعداد واسط پیروی کند.
آیا کلاس مجرد است؟	از کلاس‌های مجرد نمی‌توان نمونه‌سازی کرد؛ بلکه این گونه کلاس‌ها باید به عنوان پایه‌ای برای سایر کلاس‌ها به کار روند.
سطح دسترسی کلاس چگونه است؟	هر کلاس باید با یک سطح دسترسی مشخص، مانند public یا internal تعریف شود. این مشخصه، به طور کلی تعیین‌کننده این است که آیا کلاس توسط اسمبلی‌های خارجی هم قابل دسترسی باشد یا نه؟

جدول ۱-۱ مشخصه‌های کلاس‌های NET.

<sup>۱</sup> sealed