مرجع آموزشی

C# 2010

(جلد ۱)

مهندس نادر نبوی

انتشارات پندار پارس

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره ١٤، واحد ١٦

تلفن: ٦٦٥٧٢٣٣٥ - تلفکس: ٦٦٩٢٦٥٧٨همراه: ٠٩١٢٢٤٥٢٣٤٨

www.PendarePars.com info@PendarePars.com

نام کتاب : **مرجع آموزشی C# 2010 (جلد١)**

ناشر : **انتشارات پندار پارس ناشر همکار: مانلی**

تالیف : **نادر نبوی**

چاپ اول : **زمستان ٩٠**

شمارگان : **١٠٠٠ نسخه**

طرح جلد : **محمد اسماعیلی هدی**

لیتوگرافی : **ترام‌سنج**

چاپ، صحافی : **صالحان، خیام**

قیمت : **١٧٥٠٠ تومان با CD شابک : ٩-٠٥-٦٥٢٩-٦٠٠-٩٧٨ دوره: ٣-٠٧-٦٥٢٩-٦٠٠-٩٧٨**

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 "بلدند" 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[1] صوری آنهاست، به مرجعی نیاز داریم که نه تنها فرامین و دستورات زبان را مطرح کرده باشد (که در بسیاری موارد با جستجویی ساده در اینترنت نیز قابل دستیابی هستند)، بلکه به‌جای 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، می‌توانند آموزنده باشد. با این حال دارا بودن زمینه‌ای در C (و نه حتی C++) به درک بهتر مطالب آن کمک خواهد کرد.

---

[1] 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 نظمی 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 ١٤ 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 اول

آشنایی با مبانی .NET

به روز کردن دانش خود، علی‌رغم نگرانی و تردیدی که احتمالا 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[1]، زبان مشترک میانی[2] (CIL) و روش کامپایل به موقع[3] (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[4] (CLR)، سیستم مشترک انواع[5] (CTS) و مشخصه‌های مشترک زبان[6] (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)[7]، 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.

---

[1] assemblies
[2] common intermediate language
[3] just in-time compilation
[4] common language runtime
[5] common type system
[6] common language specification
[7] Base class libraries/Framework class libraries

۱-۱-۲ کدنویسی واسط برنامه‌نویسیC/Windows

برنامه‌نویسی سنتی، برای توسعه نرم افزاری خانواده سیستم عامل‌های ویندوز، بر پایه زبان C و استفاده از واسط برنامه‌نویسی ویندوز[1]، 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 توام با خطاهای ریز ودرشت همچنان در دنیای نرم افزار سرگردانند.

۱-۱-۳ کدنویسیC++/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. بنابرین 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 مایکروسافت (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 کم کم 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

---

[1] windows application programming interface (API)

خیلی بیشتر از MFC، پیچیدگی‌های واسط ویندوز (windows API) را توسط ویزاردهای تولید خودکار کد، انواع 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[1] 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[2]" 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[3] نیست؛ و بالاخره بد تر 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).

۱-۱-۵ 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[4] 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[5]، 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 م/). متاسفانه جاوا انجام این کار را مشکل می‌کند.

۱-۱-۶ 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

---

[1] object oriented programming language
[2] is-a relationship
[3] parameterized object construction
[4] یک بسته یا package در جاوا چیزی شبیه به یک فضای نامی در .NET، شامل انواع (کلاس‌ها) و متدهاست.
[5] 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[1] 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[2] 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 مایکروسافت 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).

۲-۱معرفی 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[3] یا به اختصار، CLR نامیده می‌شود. وظیفه اصلی CLR یافتن، بارگذاری و

---

[1] interoperability
[2] common runtime engine
[3] 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[1]، مدیریت واحدهای اجرایی داخل حافظه[2] و اعتبار سنجی‌های امنیتی است.

یکی دیگر از پایه‌های تشکیل دهنده .NET.، سیستم انواع مشترک[3] یا CTS است. CTS، تمامی انواع و ساختارهایی که ممکن است در زمان اجرا مورد نیاز باشند به علاوه ارتباطات بین این موجودیت‌ها[4] و اینکه چگونه باید در قالب‌های فراداده[5].NET 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 همان طور 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.

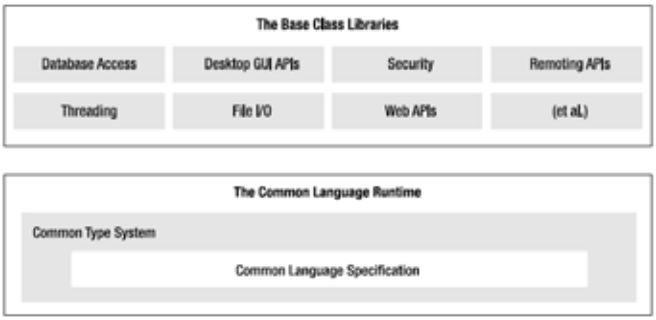
۱-۲-۱ نقش کتابخانه کلاس‌های پایه

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 واسط‌های 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.

---

[1] application hosting
[2] thread management
[3] common type system
[4] entities
[5] metadata formts



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 منحصرا 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 setter و getter 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 لاندا[1] و انواع بی‌نام[2] که معمولا در زبان‌های تابع مدار (مثل LISP و Haskell)، یافت می‌شوند نیز حمایت می‌کند. باز هم فراتر از همه اینها، با ابداع LINQ[3]، 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.**

---

[1] lambda expressions
[2] anonymous types
[3] language integrated query

- **جمع‌آوری خودکار اشیاء به جا مانده در حافظه[1]؛ به این ترتیب #C دستوری به صورت delete ندارد.**
- **روش‌های ساده‌ای برای ایجاد کلاس‌ها، واسط‌ها[2]، ساختارها، انواع شمارشی[3] و عامل‌ها[4].**
- **امکان سربارگذاری عملگرها، به روشی ساده.**
- **امکان اضافه کردن صفت[5] 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[6].**
- **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.

---

[1] automatic garbage collection
[2] interfaces
[3] enumerations
[4] delegates
[5] attribute
[6] 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[1] 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[2] 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[3] می‌گویند.

۳-۱ بررسی اسمبلی‌های .NET

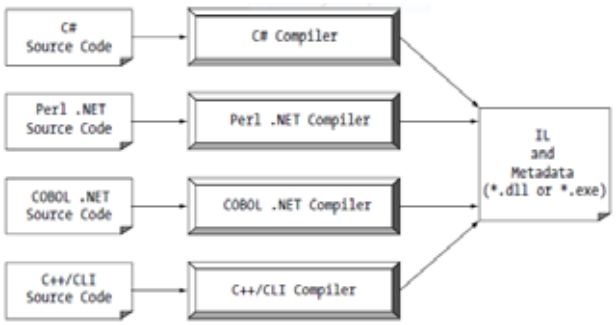
صرف نظر از زبانی که به کار می‌برید، به این نکته توجه داشته باشید که اگر چه فایل‌های باینری .NET دارای پسوندی مشابه سرویس دهنده‌های COM و باینری‌های مدیریت نشده ویندوز هستند (*.dll یا *.exe)، با این حال این دو دارای هیچ گونه مشابهت داخلی نیستند. به عنوان مثال باینری‌های dll در .NET. متدها را برای تسهیل تبادل اطلاعات با محیط زمان اجرای COM صادر نمی‌کنند (با توجه به اینکه .NET. اصلا 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[4]) 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. بنابرین در متن این کتاب هر دو مورد به یک موجودیت اشاره داشته و مترادف هستند.**

---

[1].NET runtime
[2]managed code
[3]unmanaged code
[4] platform-agnostic intermediate language (IL)



شکل ۲-۱

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.

همان طور که اشاره شد، محتوای یک اسمبلی، کد CIL است. این کد را از این جهت که تا زمانی که مطلقا ضروری نشده، به دستورات قابل فهم برای سیستم عامل ترجمه نمی‌گردد، می‌توانید شبیه bytecode در جاوا تصور کنید. تا زمانی که مطلقا 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[1] 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.

---

[1] metadata

۱-۳-۱ اسمبلی‌های تک فایلی و چند فایلی

در اکثر موارد اسمبلی، همان فایل باینری dll.* یا exe.* است. بنابرین 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[1] 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 احتمالا 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 مستنداتی 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 معمولا 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
   {
```

---

[1] 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 می‌آورید که حاوی مانیفست، دستورات CIL و فراداده‌های تشریح کننده تمامی جنبه‌های مختلف کلاس‌هایCalc 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
```

۳-۳-۱دلایل 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؛ بنابرین 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.

۴-۳-۱ 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 بهینه سازی 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 سرویس 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 کاملا 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)
CallCnvntn: [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[1] که به هنگام تایپ دستورات در محیط‌های کدنویسی ویژوال استدیو ظاهر می‌شود، از فراداده موجود در اسمبلی استفاده می‌کند. ابزارهای مرور اشیاء[2]، 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[3] (WCF)، اعمال انعکاسی[4] و تثبیت اشیاء به وسیله سریال‌سازی[5] است.

---

[1] IntelliSense
[2] object browsing utilities
[3] windows communication foundation
[4] reflection
[5] serialization

۱-۳-۶ 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 قبلا 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 (مثلا 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.

۱-۴ 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 می‌کنند. 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 اعضایی 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 کلاس‌هایCTS 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[1] 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 (برنامه‌ای 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 تعیین کننده این است که آیا کلاس توسط اسمبلی‌های خارجی هم قابل دسترسی باشد یا نه؟

جدول ۱-۱ مشخصه‌های کلاس‌های NET.

[1] sealed