معرفی چهارچوب NET. به همراه زبان #C:

چهارچوب NET. یک سکوی[1] جامع است که توصیف آن کمی دشوار می باشد. من شنیده ام که آن را یک سکوی توسعه، محیط اجرایی و سیستم عامل توصیف می کنند. در حقیقت هر یک از این تعاریف در مکان هایی صحیح هستند اما دقیق نمی باشند.

صنعت نرم افزار پس از معرفی اینترنت پیچیده تر شده است. افرادی که تا پنج سال پیش حتی کامپیوتر را لمس نکرده بودند، در حال حاضر به راحتی اینترنت را به عنوان بخشی از زندگی خود پذیرفته اند. در این میان کاربرهای حرفه ای نیز ـ مانند انتظاراتشان از نرم افزارـ پیشرفت بسیاری کرده اند.

در واقع این انتظارات از صنعت نرم افزار است که حرکت دهنده صنعت ما است. هر گاه که یک توسعه دهنده نرم افزار[2] به دستاورد تازه ای می رسد، انتظار کاربران را برای کشف جدید بعدی افزایش می دهد. در واقع در تمامی این سال ها این حقیقت پا بر جا بوده است. اما در حال حاضر چالش جدیدی برای توسعه دهندگان نرم افزار به وجود آمده است و آن هم رفتن به سوی اینترنت و کاربران اینترنتی است که در گذشته با برنامه هایی کار می کردند که قابلیت اتصال به اینترنت را نداشته است. این چالش جدید چیزی است که چهارچوب NET. ما را به سمت آن می برد.

کدنویسی در یک دنیای توزیع شده[3]:

نرم افزاری که اینترنت را هدف گرفته است باید بتواند به اینترنت متصل شود و امکان بهره برداری از قابلیت های آن را داشته باشد. البته اینترنت فقط برای برقراری ارتباط و مخابره نیست. بلکه ارتباط ساده ترین نیازی است که یک نرم افزار در دنیای شبکه دارد.

به علاوه ارتباط خصوصیات دیگری نیز باید داشته باشد از قبیل: امنیت، پیمانه ای بودن[4]، کارایی و انعطاف پذیری.

ممکن است در آینده نزدیک کاربران انتظاراتی مانند اجرای یک کد اجرایی قرار گرفته روی یک سرور را بدون محدودیت داشته باشند. به طوری که کد اجرایی محدود به اجرا در پنجره مرورگر آنها نباشد. ممکن است انتظار داشته باشند داده ها و اطلاعاتشان بتوانند به راحتی و با امنیت کامل جا به جا شوند؛ به طوری که بتوانند آن را روی شبکه به راحتی انتقال دهند تا مجبور نباشند هر بار آنها را تایپ

---

[1] Platform : در دنیای امروز معمولا به نوع سیستم عامل و یا کامپیوتر مورد استفاده گفته می شود. به عنوان مثال سکوی ۸۰۸۶.

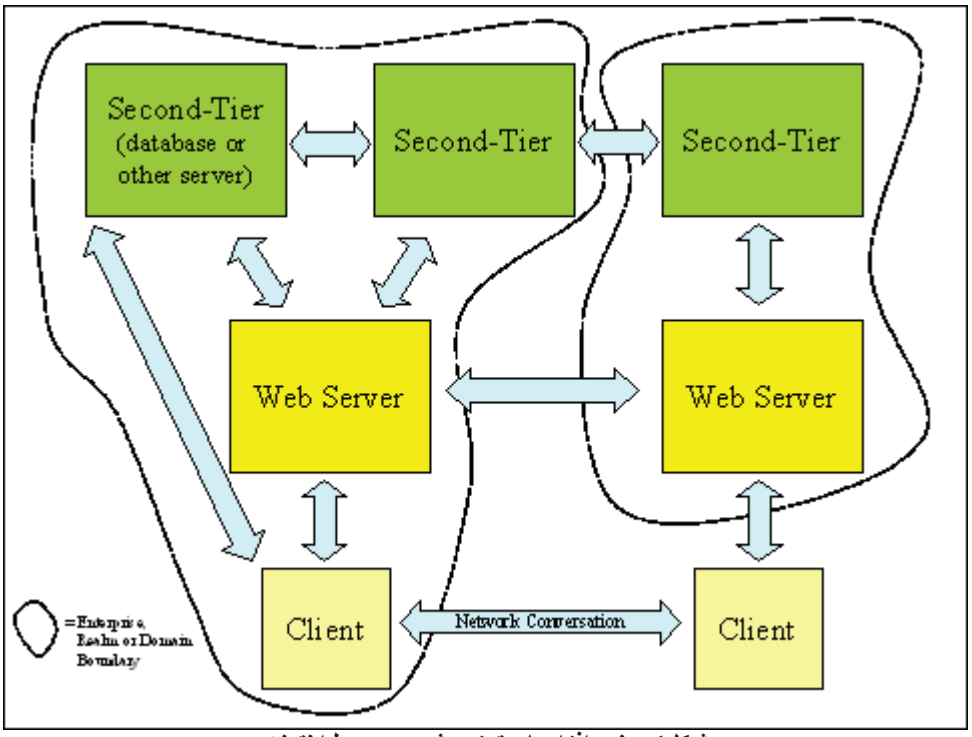
[2] Software Developer

[3] Distributed

[4] Modularity : در تعریف یک برنامه پیمانه ای باید گفت که نرم افزاری است که از بخش هایی ساخته شده است و هر بخش مخصوص انجام یک کار است و با به هم پیوستن تمامی اجزا نرم افزار کار خود را انجام می دهد.

نمایند. چنین انتظاراتی واقعا زیاد هستند و تمامی آنها باعث ایجاد چهارچوب .NET شده اند. چنین نیازهایی با ایجاد یک زبان برنامه نویسی جدید برطرف نمی شوند و نمی توان به افراد گفت که سیستم عامل جدیدی را برای برآورده کردن این نیازها تهیه کنند. به همین دلایل است که چهارچوب .NET دارای تعاریف متفاوتی است؛ چراکه در هر جنبه ای نیاز خاصی را برآورده می سازد.

یکی از چالش های مهمی که نرم افزار در یک محیط توزیع شده (مثل اینترنت) با آن مواجه است، وجود اجزای متفاوت با تکنولوژی های گوناگون است. بنابراین توسعه دهندگانی که سیستم های بزرگ را طراحی و ایجاد می نمایند مجبورند کار با زبانها و محیط های متفاوت را برای ایجاد یک محصول فرا بگیرند.



شکل1 : نرم افزارهای توزیع شده در محیط اینترنت

به تصویر ۱ نگاه کنید. همان طور که مشاهده می نمایید تصویر فوق، نمایی ساده از کامپیوترها و نرم افزار را در یک محیط توزیع شده به نمایش می گذارد. که شامل ارتباطات سرویس دهنده / سرویس گیرنده در چندین لایه به علاوه ارتباطات نظیر به نظیر است. در گذشته ابزار آلاتی که برای کد نویسی در هر بخش از نمودار استفاده می شد متفاوت بودکه شامل کتابخانه ها و زبان های برنامه نویسی متفاوتی می شد.

چهارچوب .NET می تواند برای توسعه نرم افزار در هر قسمتی از شکل فوق مورد استفاده قرار بگیرد. با چنین روشی شما می توانید از هر زبانی که مایل هستید برای توسعه نرم افزار استفاده نمایید. به

علاوه چهارچوب .NET از استانداردها استفاده می نماید. بنابراین لازم نیست که هر بخش را حتما با چهارچوب .NET پیاده سازی نمایید.

C# اولین طعم استفاده از کد مدیریت شده[۵]:

به نرم افزارهایی که با استفاده از چهارچوب .NET نوشته می شوند اصطلاحا نرم افزارهای مدیریت شده می گویند (نرم افزارهای عادی را که با چهارچوب .NET نوشته نمی شوند مدیریت نشده میگویند). در قسمت های آینده کد مدیریت شده را تعریف خواهیم کرد؛ اما تا آن زمان بهتر است چنین تصور کنید که کد مدیریت شده، کدی است که زیر نظر یک موتور اجرایی[۶] وظیفه خود را به انجام می رساند تا بتواند اهداف نرم افزارهای نوشته شده برای کار با اینترنت را فراهم آورد. این اهداف شامل: امنیت، قدرت، طراحی شیء گرا و... است.

قبل از هر چیز لازم دیدم نمونه ای از کد نوشته شده به زبان C# را به شما نشان دهم. زبان C# تنها یکی از زبانهایی است که می تواند برای نوشتن کد های مدیریت شده مورد استفاده قرار بگیرد. کد زیر نمونه ای از کدهای زبان C# را نشان می دهد که برنامه Hello World با آن نوشته شده است.

```
class App{
   public static void Main(){
      System.Console.WriteLine("Hello World!");
   }
}
```

کد فوق رشته Hello World را در یک پنجره متنی به نمایش می گذارد. کد زیر همان رشته را در یک پنجره گرافیکی نمایش می دهد:

```
using System.Windows.Forms;
using System.Drawing;

class MyForm:Form{
   public static void Main(){
      Application.Run(new MyForm());
   }

   protected override void OnPaint(PaintEventArgs e){
      e.Graphics.DrawString("Hello World!", new Font("Arial", 35),
         Brushes.Blue, 10, 100);
   }
}
```

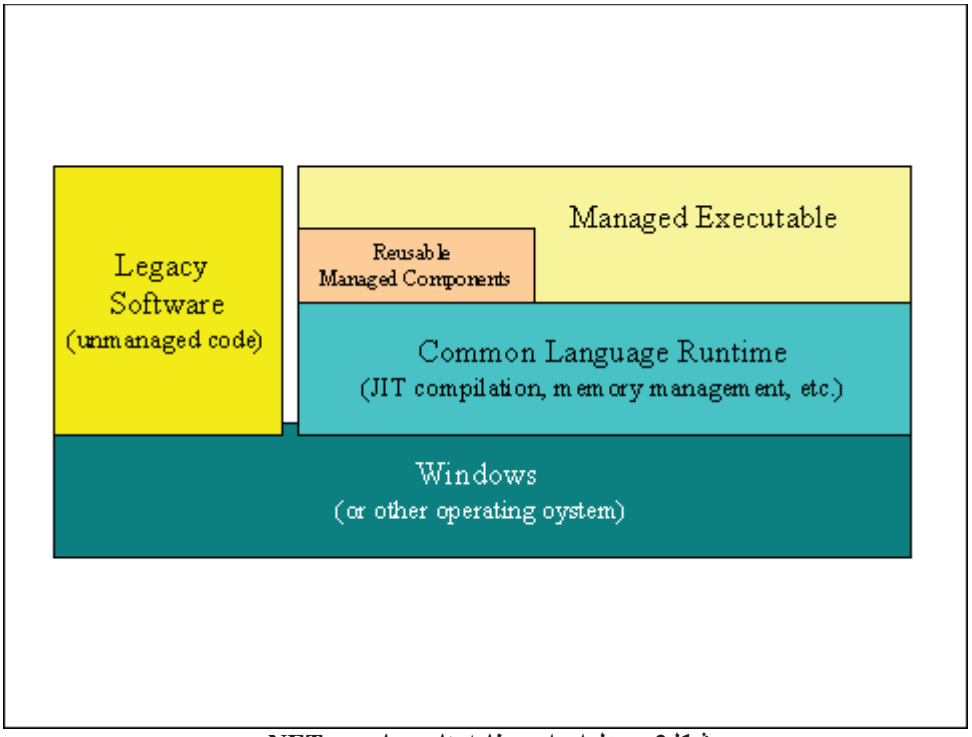
---

[۵] Managed Code
[۶] Execution Engine

هر دو مثال نمونه هایی از برنامه های کامل C# هستند. یکی از اهداف چهارچوب .NET افزایش میزان بهره وری و انعطاف پذیری برای توسعه دهنده نرم افزار است. یکی از راه های دستیابی به چنین هدفی ساده تر کردن ایجاد و توسعه نرم افزار است.

کد مدیریت شده و CLR[۷]:

به خاطر دارید که در تعریف کد مدیریت شده گفتیم که این کد، تحت نظر یک موتور اجرایی اجرا می شود و به وسیله چهارچوب .NET نوشته می شود. موتور اجرایی که کد مدیریت شده تحت آن اجرا می شود، CLR نام دارد.



شکل2 : مدل اجرایی برنامه های چهارچوب .NET.

در واقع CLR نیز چیزی مانند دیگر موتورهای اجرایی مثل JVM[۸] و Visual Basic است. CLR وظیفه مدیریت حافظه، تامین امنیت نوع داده[۹]، مدیریت نخ ها[۱۰] و امنیت را به عهده دارد. البته CLR چند مزیت مهم نیز بر موتورهای اجرایی دیگر دارد.

اولین و مهمترین ویژگی این است که کد CLR هیچ گاه تفسیر[۱۱] نمی شود. این جمله بسیار مهم است و ارزش تکرار شدن را دارد. بر خلاف دیگر موتورهای اجرایی کدی که تحت نظر CLR اجرا می

---

۷ Common Language Runtime
۸ Java Virtual Machine
۹ Type Safety
۱۰ Thread Management

شود کاملا به زبان CPU سیستم میزبان اجرا می گردد. در آینده نزدیک توضیح بیشتری درباره این ویژگی خواهم داد. برتری دوم چهارچوب NET. در این است که مدل امنیتی که توسط CLR روی کدهای مدیریت شده اعمال می شود با مدل های دیگر متفاوت است. در این مدل کد مدیریت شده روی یک ماشین مجازی اجرا نمی شود. البته با این حال CLR محدودیت هایی را بر روی کد اجرایی اعمال می نماید. امنیت کد مدیریت شده، بالا و قابل انعطاف است.

قبل از اینکه به ادامه بحث درباره چگونگی کار CLR بپردازیم مایلم به پاسخ این سوال که چرا به کد مدیریت شده نیازمندیم بپردازم.

زبان واسط[۱۲] ،داده های توصیفی[۱۳] و کامپایلر JIT[۱۴] :

قبلا گفتیم کد مدیریت شده توسط CLR تفسیر نمی شود. اما چطور ممکن است که کد محلی یک ماشین دارای امنیت نوع داده، بدون خطا و امن باشد؟ پاسخ این سوال سه بخش دارد : ۱- زبان واسط مشترک ۲-داده های توصیفی ۳ -کامپایلر JIT.

زبان واسط مشترک که اغلب با تعبیر زبان واسط یا به اختصار IL از آن یاد می شود، یک نوع زبان اسمبلی ویژه است. طراحان چهارچوب NET. در مدت زمانی حدود ۵ سال با بسیاری از اساتید و نهادهای علمی کامپیوتر مشاوره کردند تا بتوانند یک زبان اسمبلی طراحی و ایجاد کنند که متعلق به هیچ نوع پردازنده خاصی نیست. یکی از اهداف مهم IL مجزا بودن از پردازنده است به شکلی که بتواند به طور کامل برای هر پردازنده ترجمه شود.

IL در واقع نوعی زبان اسمبلی سطح بالا است که شامل دستوراتی برای مفاهیم پیشرفته برنامه نویسی مثل ایجاد اشیای نمونه ای از یک کلاس و یا فراخوانی توابع مجازی است. زمانی که در پاراگراف قبلی کلمه "ترجمه" را ذکر کردم، منظورم دقیقا ترجمه بود. چرا که CLR زبان IL و دیگر دستورات را به صورت کامل در زمان اجرا به کد ماشین پردازنده میزبان ترجمه می نماید و سپس کد ترجمه شده را اجرا می کند. به چنین عملی در اصطلاح ترجمه در زمان اجرا (Just in Time Compiling) می گویند.

---

[۱۱] Interpret : به ترجمه و اجرای دستور العمل های برنامه به صورت تک تک می گویند. چنین عملی قابلیت اشکال زدایی را افزایش می دهد اما از سرعت اجرای برنامه می کاهد. مفسرها در مقابل کامپایلرها قرار دارند که تمامی کد را یکباره به کد زبان ماشین و قابل اجرا تبدیل می نماید.

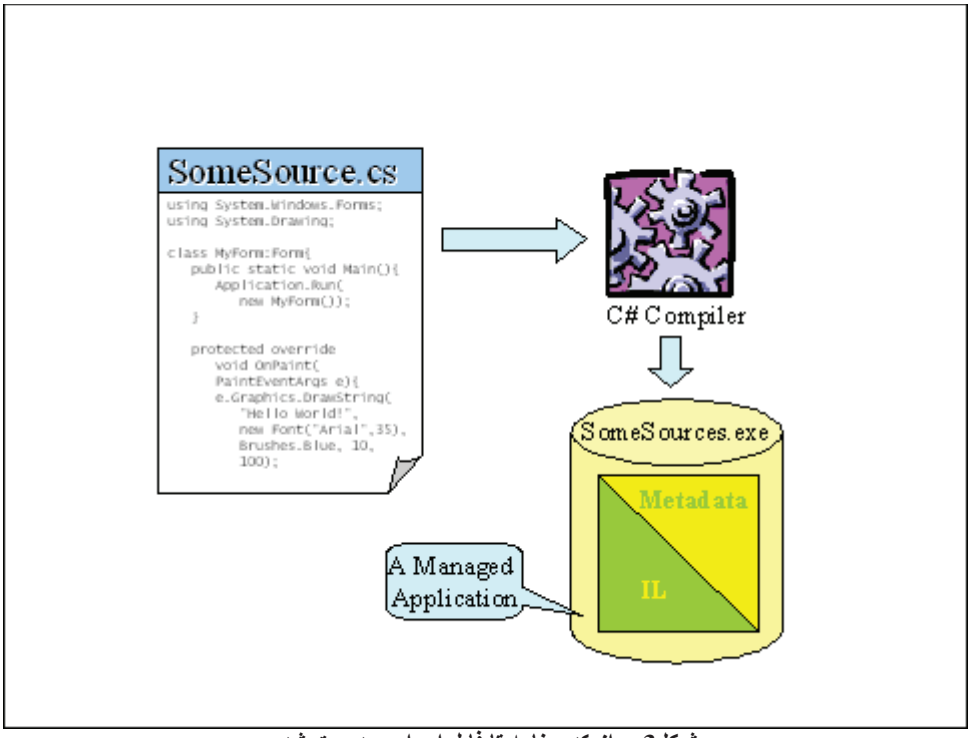
[۱۲] Intermediate Language (IL)

[۱۳] Metadata : در واقع دادهای توصیف داده هایی هستند درباره داده ها. یعنی اطلاعاتی که داده هایی را توصیف می نمایند به عنوان مثال : موضوع و مولف ایجاد کننده یک فایل اطلاعات توصیف درباره آن هستند. البته در چهار چوب NET. داده های توصیفی به داده هایی اطلاق می شود که هر المان قابل تعریفی در برنامه را توصیف می نمایند.

[۱۴] Just In Time Compiler

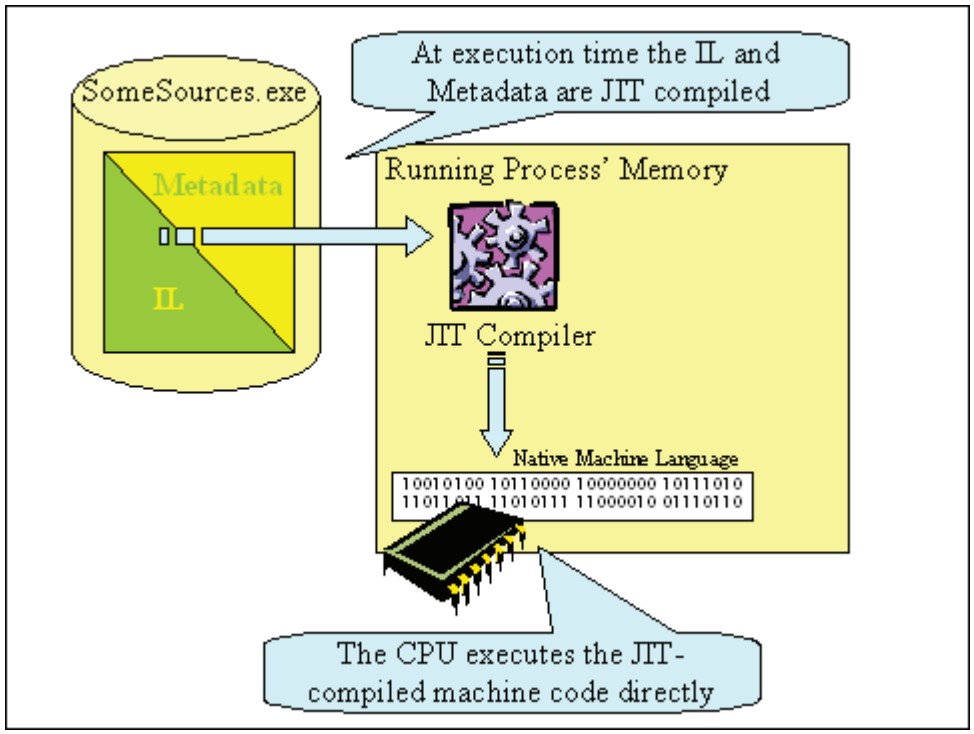
داده های توصیفی نیز تعاریف کلاس ها، پیش تعریف توابع ، نوع پارامترها و مقادیر بازگشتی از توابع و چنین اعمالی را توصیف می کنند. در واقع داده های توصیفی برای جنبه هایی از زبان برنامه نویسی کاربرد دارد که قابل اجرا نیستند.

کد مدیریت شده چیزی به جز IL و داده های توصیفی نیست. این نکته بسیار حائز اهمیت است. چراکه در فایل های اجرایی عادی فقط دستورات اجرایی برنامه وجود دارند و جنبه هایی مانند تعاریف کلاس ها و ... در زمان ترجمه از دست می روند؛ اما در فایل های اجرایی مدیریت شده داده های توصیفی و IL در کنار یکدیگر در یک فایل قرار می گیرند.



شکل3 : از کد برنامه تا فایل اجرایی مدیریت شده

کامپایلر JIT که خود بخشی از CLR است از داده های توصیفی و IL موجود در فایل اجرایی مدیریت شده برای ایجاد دستورات اجرایی پردازنده میزبان استفاده می نماید. سپس این دستورات ایجاد شده اجرا می گردند. البته درست است که مهمترین بخش این عمل کامپایل شدن است اما قبل از کامپایل برنامه عملیات دیگری برای بررسی ایمنی نوع داده و امنیت انجام می گیرد و سپس از صحیح بودن کد، اطمینان به عمل می آید (اعمالی از قبیل این که آیا داده ای وجود دارد که به چیزی ارجاع نکند) و سپس کد اجرا می شود.



شکل4 : نمایی از نحوه اجرای یک کد مدیریت شده.

اگر یک فایل اجرایی مدیریت شده طوری طراحی شده باشد که موجب نقض امنیت گردد، CLR این نقض امنیت را در زمان مطابقت کد و یا کامپایل آن متوجه می شود و مانع از اجرای کد مورد نظر می گردد. اگر در یک فایل مدیریت شده اجرایی، متغیری مقداردهی نشده باشد و یا یک نوع داده به نوعی ناسازگار تبدیل شده باشد، CLR از اجرای آن فایل جلوگیری می نماید.

مدیریت خودکار حافظه[۱۵] :

یکی دیگر از اعمال بسیار مهمی که CLR به عهده دارد مدیریت حافظه است. مدیریت حافظه یکی از اجزاء جدا نشدنی از توسعه نرم افزار است. در اکثر مواقع مدیریت حافظه در هر سطحی بر عهده خود نرم افزار است و برنامه نویس باید تمهیداتی را برای این امر در نظر بگیرد. البته اگر مدیریت حافظه یکی از بخش های خود سیستم در نظر گرفته شود، مزایای بسیاری را به همراه می آورد.

در این قسمت اشتباهات کوچکی را که ممکن است برنامه نویس باعث ایجاد آن در نرم افزار شود با هم بررسی می نماییم:

- کد ایجاد شده می تواند به یک بلاک داده ای ناموجود ارجاع نماید. چنین عملی می تواند موجب ناپایداری و رفتارهای پیش بینی نشده ای در نرم افزار شود.

[۱۵] Automatic Memory Management

- ممکن است نرم افزار نتواند بلاک حافظه ای را که در اختیار داشته است آزاد سازی نماید. در چنین حالاتی نشت حافظه به وجود می آید که می تواند باعث از کار افتادن نرم افزار و یا حتی کل سیستم شود.
- ممکن است بلاکی از حافظه آزاد شده باشد اما بخش هایی از نرم افزار هنوز ارجاعاتی به آن داشته باشند.

البته اشکالات بسیار دیگری نیز در اعمال مدیریت حافظه وجود دارند، اما هر یک از آنها به نحوی زیر مجموعه ای از همین اشکالات هستند.

در حالت کلی می توان گفت که مدیریت حافظه برای برنامه نویس عملی طاقت فرسا و وقت گیر است. به علاوه در حالاتی که شما کدی از یک منبع نامطمئن (مثل اینترنت) را در برنامه اصلی خود اجرا می نمایید، حتما مایل هستید که این کد نتواند به حافظه مربوط به داده های برنامه دسترسی داشته باشد. تمامی این دلایل نیاز به مدیریت حافظه خودکار را برای کدهای مدیریت شده اثبات می کنند.

تمام برنامه هایی که تحت چهارچوب .NET یا CLR اجرا می شوند حافظه مورد نیاز خود را از یک Heap[۱۶] مدیریت شده می گیرند. Heap مدیریت شده تحت نظر CLR قرار دارد و برای تمام انواع داده ای ارجاع[۱۷] اعم از: اشیاء ، بافرهای داده ای، رشته ها، مجموعه ها، پشته ها و... مورد استفاده قرار می گیرد.

Heap مدیریت شده می داند چه زمانی یک بلاک از حافظه توسط برنامه شما در حال استفاده است. در چنین مواقعی کاری به بلاک مورد استفاده ندارد، اما زمانی که یک بلاک از حافظه را بدون ارجاع به شیء خاصی می بیند آن را به دست جمع آوری کننده زباله[۱۸] می سپارد. عملیات جمع آوری زباله، یک عمل خودکار مربوط به Heap مدیریت شده است و هر زمان که مورد نیاز باشد انجام می شود.

کد شما هیچ گاه باعث آزادی بخشی از حافظه به صورت مستقیم نمی گردد و بنابراین نمی تواند موجب ایجاد نقص در حافظه و نشست آن شود. حافظه زمانی زباله تلقی می شود که ارجاعی به آن وجود نداشته باشد. بنابراین امکان ارجاع به یک شیء ناموجود در حافظه امکان ندارد. در نهایت به علت این که Heap مدیریت شده یک محیط فاقد اشاره گر است (حداقل در کدهای مدیریت شده) بازبینی کننده کد فقط اجازه خواندن بلاکی از حافظه را می دهد که قبلا چیزی در آن نوشته شده باشد.

---

[۱۶] Heap : در اصطلاح به مجموعه ای از خانه های حافظه می گویند که نرم افزار می تواند به صورت دینامیک (در زمان اجرا) از سیستم برای اهداف خود بگیرد و پس از اتمام کار خود دوباره آن را به سیستم باز می گرداند.

[۱۷] Reference Type : نوع داده ای که به صورت مستقیم مقداری را در خود ندارد بلکه به مکان نگه داری یک قلم داده در حافظه ارجاع دارد. (مانند اشاره گرها)

[۱۸] Garbage Collector

همان طور که مشاهده نمودید Heap مدیریت شده هر سه مورد از اشکالاتی را که ممکن است توسط برنامه نویس ایجاد شوند، رفع می نماید.

مفاهیم زبان ها و CLR:

کد مدیریت شده تحت نظارت دائم CLR اجرا می شود. CLR وظیفه مدیریت حافظه، مدیریت نوع، امنیت و چندنخی بودن را بر عهده دارد. از این حیث CLR یک محیط اجرایی است.

البته بر خلاف محیط های اجرایی دیگر کد مدیریت شده محدود به زبان برنامه نویسی خاصی نیست. خوب تا به اینجا حتما مطالبی را درباره زبان C# شنیده اید. در واقع C# یک زبان برنامه نویسی جدید است که مخصوص نوشتن برنامه های مدیریت شده توسط چهارچوب NET. به کار می رود. با این حال این بدین معنا نیست که فقط از زبان C# می توان برای نوشتن برنامه های مدیریت شده استفاده کرد. در واقع هر نوع کامپایلری می تواند به گونه ای طراحی شود که کدهای مدیریت شده تولید کند. فقط کافی است قابلیت تولید ابرداده و IL را به طور کامل داشته باشد.

در حال حاضر شرکت ماکروسافت پنج کامپایلر و اسمبلر مختلف را به همراه چهارچوب NET. عرضه می نماید که شامل : C# ، Visual Basic ، C++ ، Java Script و IL می شود(شما میتوانید کد مدیریت شده خود را به صورت مستقیم با IL بنویسید اما چنین کاری غیر معمول است). به علاوه پنج زبان برنامه نویسی که به همراه چهارچوب NET. عرضه می شوند شرکت ماکروسافت قصد دارد در آینده یک کامپایلر جاوا هم که قابلیت ایجاد کد مدیریت شده را داشته باشد به چهارچوب NET. اضافه کند.

علاوه بر کامپایلرهای شرکت ماکروسافت تولیدکنندگان دیگر نیز برای حدود بیست زبان دیگر که شامل : Eiffel، PERL، COBOL، و... می شود کامپایلرهای مخصوص NET. را ارائه داده اند.

در واقع CLR احتیاج به دانستن هیچ چیزی درباره هیچ زبان برنامه نویسی به جز IL ندارد و تمامی نرم افزارهای مدیریت شده به دستورات IL و داده های توصیفی تبدیل می شوند. این نکته بسیار حائز اهمیت است؛ چرا که باعث می شود تمامی زبان ها از نظر CLR یکسان باشند. نکته مهم دیگر این است که IL خود تقریبا یک زبان شیء گرا است (یا حداقل برای زبان های شی گرا طراحی شده است)؛ اما کامپایلر زبان ها اسکریپت نویسی و رویه ای[۱۹] نیز به راحتی می توانند کد IL مورد نیاز خود را تولید نمایند.

---

[۱۹] Procedural

مباحث پیشرفته:

در این قسمت قصد دارم توضیحات بیشتری را درباره کامپایلر JIT و جمع آوری زباله بدهم.

اولین باری که یک فایل اجرایی مدیریت شده به یک کلاس و یا نوع داده ای ارجاع می نماید سیستم باید ماژول کد[۲۰] یا ماژول مدیریت شده ای را که نوع را پیاده سازی کرده است بارگذاری نماید. در نقطه بارگذاری کامپایلر JIT ، متدهایی به نام متدهای Stub[۲۱] را به زبان محلی ماشین مقصد برای هر متد عضو کلاس مورد بارگذاری ایجاد می نماید. این متدهای Stub فقط شامل داده هایی هستند که باعث پرش به یک تابع خاص در کامپایلر JIT می شود.

پس از آن که توابع Stub ایجاد شدند سیستم هر فراخوانی متد را در کد ارجاعی مشخص می نماید. به طوری که به توابع Stub ایجاد شده اشاره نمایند. در این زمان هنوز هیچ عمل کامپایل JIT صورت نگرفته است. پس از این عمل زمانی که یکی از این توابع Stub فرا خوانی شوند کامپایلر JIT به کد منبع در فایل مورد ارجاع رفته و آن را به کد محلی ماشین میزبان تبدیل می نماید و سپس متد Stub را به نحوی تغییر می دهد تا به محلی که کد کامپیال شده جدید در آن قرار دارد پرش نماید. سپس دفعه بعدی که همین تابع فراخوانی شود برنامه با سرعت کد محلی ماشین میزبان اجرا می شود و نیازی به کامپایل JIT مجدد وجود ندارد.

مزیت استفاده از این روش این است که زمان سیستم صرف کامپایل JIT متدهایی که در این اجرا در برنامه شما مورد استفاده قرار نمی گیرند نمی شود.

در نهایت زمانی که یک متد با JIT کامپایل می شود همه نوع هایی که مورد ارجاع هستند توسط CLR مورد بررسی قرار می گیرند تا بررسی شود آیا آنها در این اجرای برنامه جدید هستند یا خیر. در این حالت اگر این متد جدید باشد و دفعه اولی باشد که مورد ارجاع قرار می گیرد، تمامی مراحل برای آن از ابتدا انجام می شود.

خوب یک نفس عمیق بکشید و به آرامی آن را بیرون دهید، چرا که می خواهم درباره جمع آوری زباله توضیحاتی را ارائه دهم.

جمع آوری زباله یک فرآیند وقت گیر است. CLR باید همه یا اکثر نخ های برنامه مدیریت شده شما را در حین خالی شدن بافرها و اشیاء از Heap متوقف کند و سپس به عمل جمع آوری زباله بپردازد.

جمع آوری زباله عملی فعال نیست، در واقع این نوع عملیات به نوعی انفعالی و غیر فعال است و به عواملی بستگی دارد و فقط زمانی انجام می شود که حافظه کافی برای رفع نیاز برنامه و دستورات آن

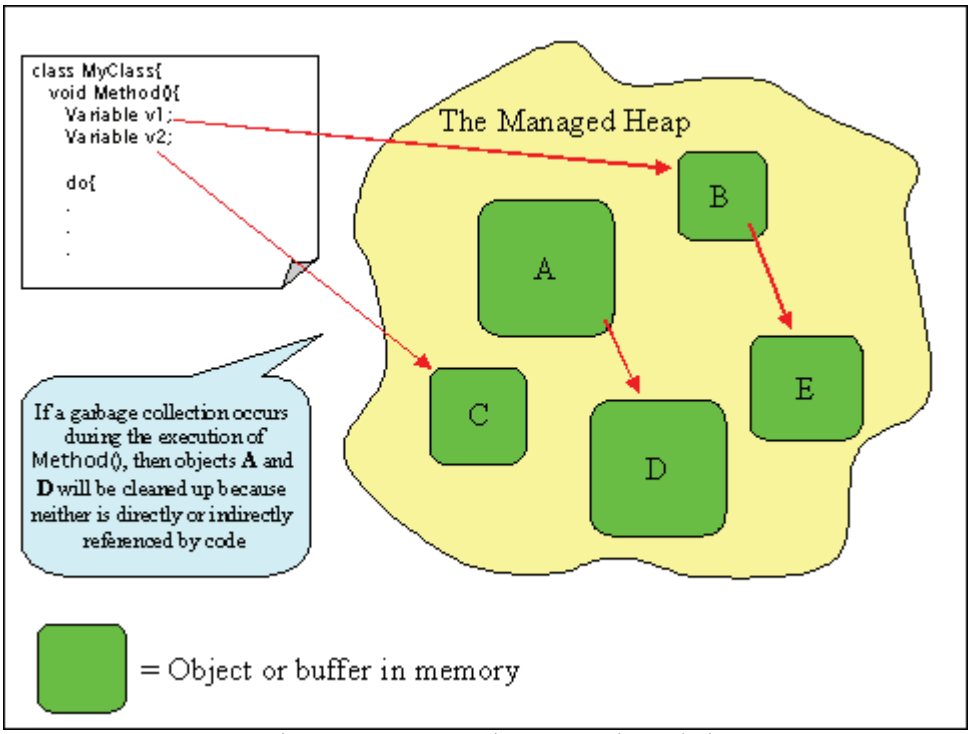
---

[۲۰] Code Module : فایلهایی هستند که در آنها توابع و ساختارهای از پیش تعریف شده ای برای استفاده در برنامه های مختلف وجود دارد.

[۲۱] متد Stub : به متدهایی گفته می شود که شامل کد اجرایی نیستند بلکه فقط شامل توضیحاتی هستند که کدی را که در آن قسمت درج خواهد شد شرح می دهند. این نوع متدها به عنوان یک نگه دارنده مکان (Place Holder) استفاده می شوند تا در آینده متد اصلی به جای آنها قرار گیرد. به این نوع متدها اصطلاحا Dummy Method هم گفته می شود.

وجود نداشته باشد. هنگامی که حافظه کافی وجود نداشته باشد، جمع آوری کننده زباله تلاش مینماید حافظه را به اندازه کافی آزاد کرده و در اختیار برنامه شما قرار دهد.

زمانی که عملیات جمع آوری زباله به وقوع می پیوندد سیستم تمامی اشیایی را که توسط متغیرهای محلی و متغیرهای عمومی مورد ارجاع قرار گرفته اند پیدا می کند. این اشیاء زباله نیستند، چراکه توسط نخ های اجرایی برنامه در حال استفاده هستند. سپس سیستم اشیای مورد ارجاع توسط اشیای دیگر را جستجو می نماید. این اشیاء نیز زباله نیستند چراکه مورد ارجاع قرار دارند. این عملیات تا زمان پیدا کردن آخرین ارجاع تکرار می شود. پس از آن تمامی اشیایی که دیگر کد ارجاعی به آنها وجود ندارد زباله در نظر گرفته شده و نابود می شوند. شکل زیر این وقایع را نمایش می دهد:



شکل5 : اشیای مدیریت شده در Heap مدیریت شده

در حین عملیات جمع آوری زباله، حافظه اشغال شده توسط اشیای زباله باز پس گرفته می شود و در اختیار دیگر اشیاء قرار می گیرد. به عنوان نتیجه، نرم افزارهای مدیریت شده از حافظه به صورت موثرتری استفاده می کنند؛ چرا که قطعه قطعه شدن حافظه در آنها غیر ممکن است، زیرا پس از باز پس گرفتن حافظه، حافظه مرتب می شود و در واقع نوعی عمل Defragment روی آن انجام میگیرد.[۲۲]

[۲۲] ضعف جمع آوری زباله، وقت گیر بودن آن است. البته زمان بسیار اندکی (در حد کسری از ثانیه) را تلف می نماید. اما در کل مصرف حافظه را بهینه می نماید.

: Visual Studio.NET

در واقع VS.NET قسمتی از چهارچوب NET. به شمار نمی رود. اما با این حال بهتر دیدم نام آن را به همراه معرفی چهارچوب NET. ذکر کنم. VS.NET یک محیط جامع تولید و توسعه نرم افزار[۲۳] است که توسط شرکت ماکروسافت برای نوشتن برنامه های تحت ویندوز ارائه شده است. VS.NET می تواند برای نوشتن کدهای مدیریت شده در زبان های C#، C++، Visual Basic و هر زبان دیگری مثل PERL مورد استفاده قرار بگیرد. البته امکانات لازم برای دیگر زبان ها باید توسط تولیدکنندگان دیگر ایجاد شده و در دسترس قرار بگیرد.

به طور کلی VS.NET خود یک نرم افزار مدیریت شده است و برای اجرا نیاز به چهارچوب NET. دارد. این محصول دارای یک رابط بسیار کاربر پسند است و امکانات مختلفی در محیط آن تعبیه شده، به شکلی که می توان از آن برای نوشتن کدهای مدیریت شده استفاده نمود. این محیط دارای جادوگر[۲۴]های سودمند بسیاری برای ایجاد کد می باشد و به علاوه خصوصیاتی مانند امکان تغییر رنگ کد[۲۵]، راهنمای آنلاین، قابلیت تکمیل خودکار کد، خطایاب زمان تایپ برنامه و بسیاری امکانات دیگر دارد. با تمام این اوصاف برای نوشتن برنامه های مدیریت شده حتما احتیاجی به داشتن VS.NET ندارید.

البته من به شما پیشنهاد می کنم که حتما از VS.NET استفاده نمایید چرا که حقیقتا محصول بسیار عالی است. در ضمن دانستن این نکته هم ضروری است که VS.NET و چهارچوب NET. محصولات کاملا جداگانه ای هستند.

چهارچوب NET. شالوده و بنیادی برای کدهای مدیریت شده است. چهارچوب NET. شامل CLR و اجزای دیگری است که در آینده درباره آنها توضیحات بیشتری خواهم داد. درضمن چهارچوب NET. شامل یک SDK[۲۶] نیز هست که کامپایلرهای خط فرمان برای C#، C++، VB و IL را در بر می گیرد.

در حقیقت تمام چیزی که برای ایجاد و توسعه برنامه های C# نیاز دارید چهارچوب NET. است، اما استفاده از VS.NET باعث سهولت بیشتر و صرفه جویی در وقت می شود.

در اشکال زیر نماهایی را از Visual Studio.NET 2008 مشاهده می نمایید:

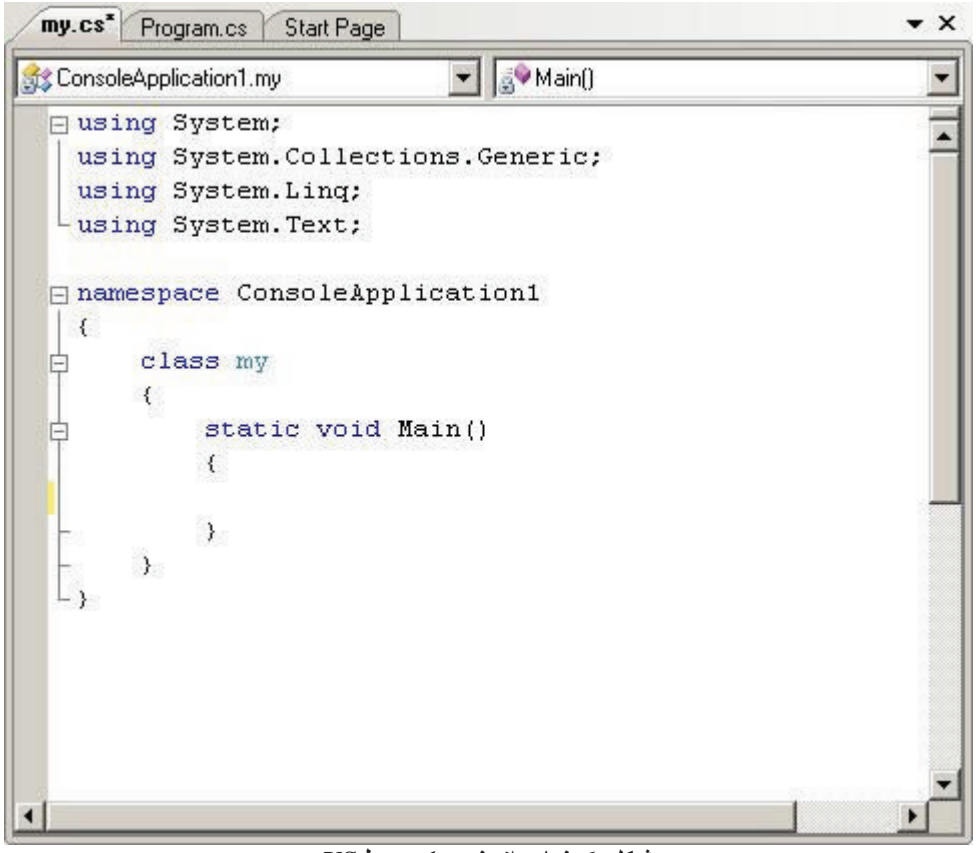
---

[۲۳] Integrated Development Environment : محیطی برای ایجاد و توسعه نرم افزار است و دارای امکاناتی از قبیل : کامپایلر، ویرایشگر کد، دیباگر و.... است.

[۲۴] Wizard : دراینجا به امکاناتی اطلاق می شود که در یک IDE تعبیه شده است و اعمال خاصی را برای برنامه نویس به صورت خودکار انجام می دهد و یا او را در انجام مراحل راهنمایی می نماید.

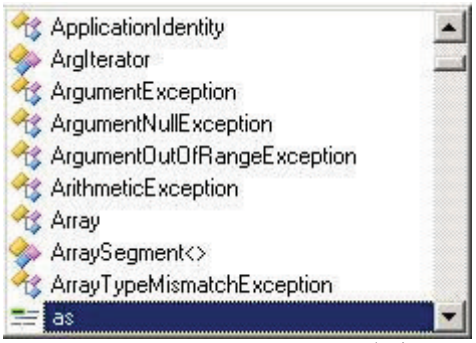
[۲۵] Syntax Highlighting

[۲۶] Software Development Kit : اصولا به مجموعه ای از ابزارهایی گفته می شود که به برنامه نویس امکان نوشتن برنامه ها را برای یک محیط و یا سیستم عامل خاص می دهد. این ابزارها شامل : کامپایلر، لینکر، ویرایشگر و... می شود.



شکل :6 نمایی از پنجره کد محیط VS

همان طور که مشاهده می نمایید، پنجره کد VS.NET دارای امکان تغییر رنگ کد و رعایت خودکار تورفتگی ها است که کمک بسیار زیادی را به برنامه نویس در هنگام نوشتن برنامه خود میکند. در شکل زیر یکی دیگر از امکانات بسیار عالی شرکت ماکروسافت که در این محصول جاسازی شده است را مشاهده می نمایید:



شکل7 : نمایی از منوی Intellisense

به حق یکی از بهترین تکنولوژی ها و امکاناتی که شرکت ماکروسافت در اختیار برنامه نویسان قرار داده است، منوی (تکنولوژی) IntelliSense است. با استفاده از این تکنولوژی برنامه نویسان نیازی به به خاطر سپردن اسامی طولانی و شکل کلی توابع ندارند و هرجایی که نیاز داشتند می توانند با فشردن

کلیدهای Ctrl+J این منو را فراخوانی نموده و از آن استفاده کنند. البته در نسخه های اخیر VS.NET منوی IntelliSense با نوشتن اولین حرف یک کلمه در یک خط ظاهر می شود و تا آخر دستور در نوشتن کد شما را یاری می دهد.

البته محیط VS.NET دارای امکانات بسیار دیگری نیز هست که فکر می کنم ذکر تمامی آنها حتی در یک کتاب هم ممکن نباشد. از ویژگی های بسیار مهم دیگر این محیط می توان به Debugger قدرتمند آن اشاره کرد که برنامه نویسان را در اشکال زدایی برنامه هایشان یاری می نماید.

کتابخانه FCL :

تا اینجا اندکی درباره اهداف چهارچوب NET. و کدهای مدیریت شده صحبت کردیم. CLR مبنا و بنیادی برای هر چیز مدیریت شده ای است و بخش بسیار مهمی از چهارچوب NET. است. اما در برنامه نویسی شما کاری به طریقه کار چهارچوب NET. ندارید؛ بلکه سعی می کنید از کتابخانه ها و اجزای موجود استفاده نمایید و طریقه کار با آنها را بیاموزید. تمامی این اجزا که به همراه چهارچوب NET. ارائه می شوند بخشی از کتابخانه کلاس چهارچوب NET. می باشند.

FCL[۲۷] در حقیقت یک مجموعه بزرگ از کلاس ها، ساختمان ها، انواع داده شمارشی و رابط هایی است که تعریف و پیاده سازی شده اند تا برنامه نویسان بتوانند از آنها در کدهای مدیریت شده خود استفاده نمایند. کلاس های موجود در FCL شامل محدوده وسیعی از انواع کلاس ها برای اعمالی نظیر کار با ورودی و خروجی تا ایجاد صفحات وب، پنجره های گرافیکی و ... است. اجزای موجود در کتابخانه FCL دارای قابلیت استفاده مجدد هستند. بدین معنا که می توان از آنها در برنامه های مختلفی استفاده کرد. قابلیت استفاده مجدد یکی از اهداف دانشمندان نرم افزار برای دهه های متمادی بوده و هست و تا به امروز روش های متفاوت برای نیل به این هدف ایجاد شده اند. یکی از این روشها برنامه نویسی شیء گرا است که امروزه یکی از پیشرفته ترین متدهای مورد استفاده در برنامه نویسی به شمار می آید. برنامه نویسی شیء گرا بر پایه اشیاء بنا نهاده شده است. در این متد، برنامه نویس سعی می کند تا هر ماهیتی را به صورت یک شیء در نظر بگیرد که دارای خصوصیات و رفتارهایی است که می تواند از طریق آنها با اشیای دیگر و یا با کاربر ارتباط برقرار نماید. در این متد برنامه نویسی ایده ـ ها و مفاهیم بسیاری وجود دارند که از حوصله این مقاله خارج هستند[۲۸]، در اینجا فقط قصد بحث درباره قابلیت استفاده مجدد را داریم، چرا که اصلا یکی از اهداف بسیار مهمی که برنامه نویسی شیءگرا برآورده می سازد قابلیت استفاده مجدد است.

---

[۲۷] Framework Class Library

[۲۸] برای اطلاعات بیشتر درباره برنامه نویسی شیء گرا می توانید کتب مربوطه را در مورد زبانهایی مطالعه نمایید که از این قابلیت پشتیبانی می نمایند. به عنوان مثال یکی از کتبی که خواندن آن را شدیدا به شما پیشنهاد می کنم کتاب Professional C++ انتشارات Wiley است.

CLR سکویی است که از ابتدا به صورت شیء گرا طراحی شده است و بنابراین تمامی ایده های برنامه نویسی شیء گرا را ترویج می نماید.

امروزه اکثر نرم افزارها از ابتدا نوشته می شوند. معمولا منطق اصلی هر برنامه ای می تواند با چندین دستور کوتاه شرح داده شود؛ اما هنوز هم اکثر برنامه ها شامل هزاران و یا میلیون ها خط کد هستند تا بتوانند به اهدافشان دست پیدا کنند و این امر نمی تواند تا ابد ادامه پیدا کند.

در واقع تعداد نرم افزارهایی که در صنعت نرم افزار ایجاد می شوند آنقدر زیاد هست که نتوان همه آنها را از ابتدا تولید نمود و بنابراین نیاز به یک روش سیستماتیک در استفاده مجدد از کد وجود دارد.

در اینجا نمی خواهیم به توضیحات طولانی در مورد نیاز به برنامه نویسی شیء گرا و قابلیت استفاده مجدد بپردازیم. به همین دلیل به توضیحاتی تا همین حد اکتفا می کنیم و تحقیق بیشتر درباره این مباحث را به شما خواننده عزیز وا گذار می نماییم. در ادامه فقط برخی از ویژگی های CLR که با برنامه نویسی شی ءگرا در ارتباط هستند را مورد بررسی قرار می دهیم:

- CLR به طوری کلی خود یک سکوی شیءگرا است.
- CLR دارای یک دید همگن نسبت به تمامی انواع داده ها است. به طوری که هر نوع داده ـ ای اعم از انواع ابتدایی[۲۹] و یا انواع ایجاد شده توسط کاربر همگی از یک کلاس پایه به نام Object مشتق می شوند. از این نظر تمامی انواع داده ای موجود در برنامه از یک کلاس پایه هستند و فقط خصوصیات متفاوتی دارند.
- کد مدیریت شده پشتیبانی کاملی از ساختارهای شیءگرا مانند رابط ها[۳۰]، خصوصیات[۳۱]، انواع شمارشی[۳۲] و کلاس ها دارد.
- کد مدیریت شده الگوهای جدیدی را مانند : خواص، توابع سازنده Static و .... را پشتیبانی می نماید و جای خالی چنین ویژگی هایی را در محیط های مشابه پر می کند.
- با استفاده از کد مدیریت شده می توان از کتابخانه ای پیش ساخته با قابلیت استفاده مجدد استفاده کرد. به این کتابخانه های اجزا اصطلاحا اسمبلی های مدیریت شده می گویند.
- CLR دارای یک مکانیسم اختصاص نسخه بسیار قدرتمند است که باعث می شود CLR کاملا در مورد نسخه ای از یک کتابخانه که مورد استفاده برنامه شما است اطلاعات کامل داشته باشد.
-

---

[۲۹] Primitive Data types : به این انواع اصطلاحا انواع داخلی Build-in نیز اطلاق می شود.

[۳۰] Interfaces

[۳۱] Properties

[۳۲] Enumerated

کتابخانه کلاس چهارچوب .NET :

کتابخانه کلاس چهارچوب .NET. اولین قدم در ایجاد برنامه هایی بر پایه اجزا است. بنابراین شما می ـ توانید از اشیای موجود در کتابخانه .NET. برای اعمالی نظیر خواندن فایل ها، نمایش پنجره ها و اعمال دیگر استفاده نمایید. اما برای استفاده و بهره وری بهتر از امکانات می توانید بخش هایی از کتابخانه FCL را که مورد نیاز برنامه شما است در راستای توسعه نرم افزارتان توسعه دهید.

FCL یک کتابخانه کلاس بوده که دارای امکانات متفاوت برای برنامه های گوناگون است. به عنوان مثال دارای فضاهای نام گوناگون برای انجام اعمال مختلف است که هر یک از این فضاهای نام برای عمل خاصی در نظر گرفته شده اند.

در ضمن چهارچوب .NET. دارای یک مجموعه اسناد بسیار قدرتمند به نام MSDN[۳۳] می باشد که طریقه کار اجزای آن را شرح می دهد.

---

[۳۳] Microsoft Developer Network

مآخذ:

1. http://www.devhood.com/training _modules/dist-a/Intro.NET/intro.net.htm

2. Microsoft Computer Dictionary Fifth Edition (2002)