**C#**

by Reza Pouya

سی شارپ یک زبان همه منظوره ، type safe و شی گراست

**شی گرایی :**

* ارث بری
* کپسوله سازی
* چند ریختی
* انتزاع

سی شارپ از انواع ایستا پشتیبانی می کند ، این بدان معناست که این زبان در هنگام کامپایل شدن قوانین درستی انواع را اعمال می کند .انواع قوی و ایستا همچنین به سی شارپ کمک می کنند که قابلیت اجرا شدن در Sand Box را داشته باشند .

Sand Box

محیطی است که تمامی زمینه های امنیتی آن توسط میزبان کنترل می گردد ، در سندباکس این مهم است که شما نمی توانید به صورت دلخواه ، وضعیت شی را به وسیله ی دور زدن قوانین نوع آن ، خراب کنید .

مدیریت حافظه :

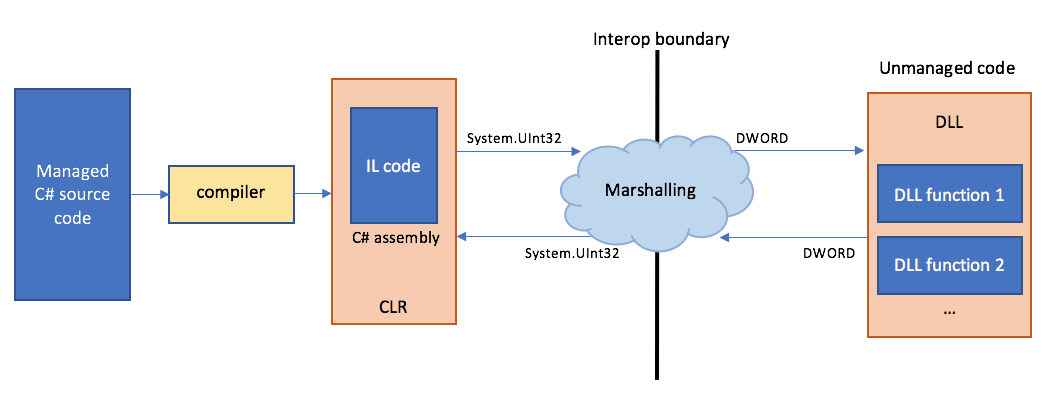
سی شارپ به runtime برای مدیریت حافظه وابسته است . ماژول Common Language Runtime (CLR) یک جمع کننده ی آشغال یا اصلاحا Garbage Collector دارد که به عنوان بخشی از برنامه اجرا می شود و حافظه را برای اشیایی که دیگر به آن ها ارجاعی داده نشده ، پس می گیرد .

سی شارپ برای اجرا به Runtime و پلتفرمی وابسته است که ویژگی هایی مانند « ***مدیریت خودکار حافظه*** » و « ***مدیریت استثناء ها*** » را انجام دهد . هسته ی اصلی فریمورک Net. ماژول Common Language Runtime (CLR) است که این ویژگی های زمان اجرا را مشخص می کند .

Common Language Runtime (CLR) :

* Memory Management
* Exception Handling

سی شارپ یکی از زبان مدیریت شده یا Managed Language است که برای اجرای کد آن ابتدا باید به Managed Code تبدیل شود و سپس این Managed Code توسط CLR به کد ماشین مقصد تبدیل می شوند .

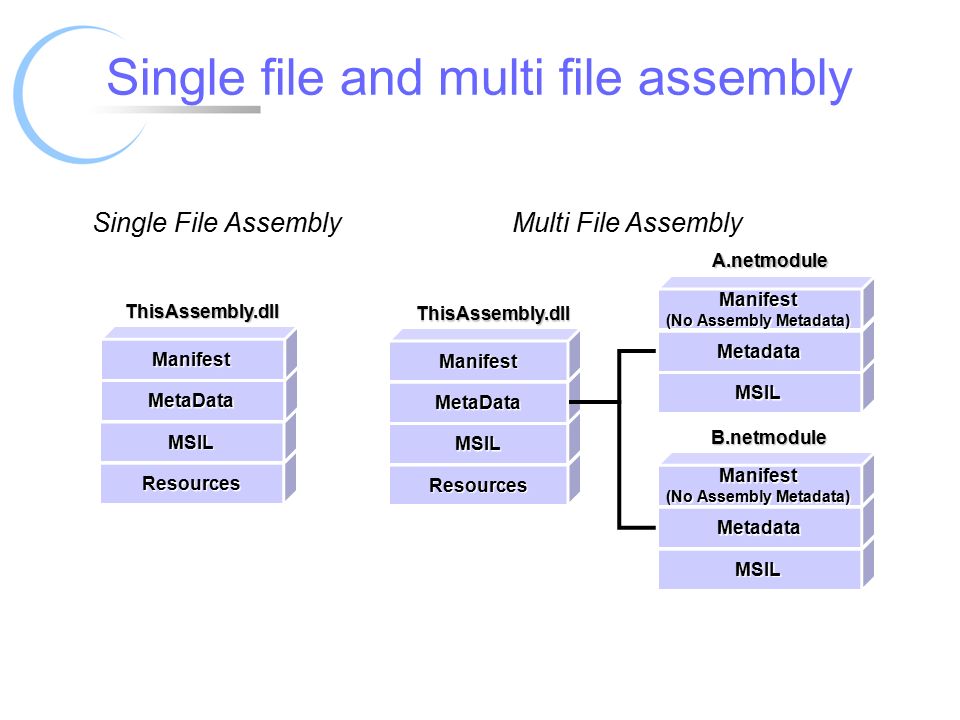


کدهای سی شارپ به Managed Code تبدیل می شوند سپس توسط کامپایلر به Intermediate Language تبدیل می شود و بعد در هنگام اجرا توسط Just In Time ( JIT ) به کد ماشین مقصد تبدیل و اجرا می شود . البته کامپایل مستقیم به کد ماشین مقصد یا اصطلاحا Native Code نیز موجود است که تحت نام کامپایل Ahead Of Time شناخته می شود .

ظرفی که Intermediate Language در آن قرار می گیرد را Assembly یا Portable Executable می نامند . یک اسمبلی می تواند یک فایل اجرایی ( .exe ) یا یک کتابخانه ( .dll ) باشد اسمبلی تنها حاوی IL نیست بلکه حاوی اطلاعات نوع Type Information می باشد . وجود متادیتا به اسمبلی ها اجازه می دهد که بتواند به انواع دیگر در سایر اسمبلی ها ، ارجاع/ Refrence دهند بدون اینکه به فایل های دیگر نیاز داشته باشند .

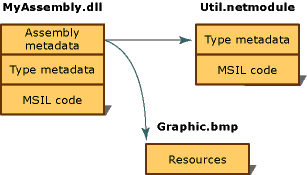
می توانید اسمبلی را به وسیله ی ابزارهای زیر مشاهده کنید ( به صورت Intermediate Language ) و یا آن را به کُد سی شارپ برگردانید :

* Microsoft Ildasm
* ILspay
* dotPeek ( Jet Brain )
* Reflector ( Red Gate )



**Reflection**

یک برنامه می توانید بر روی متا دیتا خود کوئری بزند ( پرس و جو ) کند و حتی در زمان اجرا یک Intermediate Language را بسازد . ( Reflection.emit )



علاوه بر این می توانید Managed Code را درون MS SQL Server اجرا کنید . با یکپارچه سازی MS SQL Server integration ، می توانید توابع سفارشی Store Procedure و aggregation ها را در C# بنویسند و آنها را از درون MS SQL Server فراخوانی کنید . این در هماهنگی با .Net Framework استاندارد کار می کند ، ولی یک CLR اختصاصی « میزبانی شده » با ایجاد یک Sandbox از یکپارچگی فرایندهای SQL سرور ( SQL Server Process ) محافظت می کنید .

Library (.dll) - - - - > .exe ( without entry point )

انواع توابع در سی شارپ

* Method
* Constructor
* Deconstructor
* Property
* Indexer
* Finalizer
* Operator

کلاس

گروهی از اعضای توابع و اعضای داده که یک بلاک شی – گرا را تشکیل می دهند را کلاس گویند . کلاس یک type است . در بیرونی ترین سطح یک برنامه ، کلاس ها درون فضاهای نام ( namespace ) ، سازماندهی می شوند .

Type : یک نقشه ، یا قالب برای یک مقدار است .

متغیر : یک مکان در حافظه است

در سی شارپ تمامی مقادیر نمونه هایی از یک type هستند . معنای مقدار و مجموعه ای از مقادیری که یک متغیر می تواند داشته باشد توسط نوع آن مشخص می گردد .

یک Type حاوی موارد زیر است :

1. اعضای داده / data members
2. اعضای توابع / function members

داده با نمونه سازی از یک « نوع » ایجاد می گردد .

فرق اعضای نمونه و اعضای ایستا / static members

اعضای تابع و اعضای داده ای را که بر روی یک نمونه از نوع ، عمل می کنند را instance member می گویند .

اعضای داده و اعضای توابعی که بر بروی یک نمونه از یک نوع عمل نمی کنند ، بلکه بر روی خوع نوع type عمل می کنند را با کلمه ی کلیدی static نشانه گذاری می کنیم .

کلاس static ، کلاسی است که تمامی اعضای آن ، static باشند .

انواع type ها درون سی شارپ :

* Value Types
* Reference Types
* Generic Type Parameters
* Pointer Types

فرق اساسی بین Value Type ها و Reference Type ها ، در نحوه ی چگونگی قرارگیری و مدیریت شدن آنها درون حافظه است .

Value Type ها

شامل اکثر انواع درونی سی شارپ می شوند ( به خصوص تمامی انواع عددی ، char و bool ) به همراه انواع سفارشی struct و enum است .

محتوای یک value type یک « مقدار » است .

می توانید یک نوع سفارشی از نوع value type را با استفاده از کلمه ی کلیدی sturct درون سی شارپ تعریف کنید .

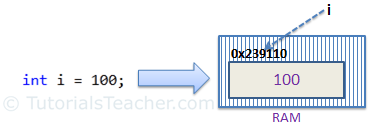
Public struct Point {

public int x ;

public int y ;

}

انتساب یک value type ، همیشه مقدار نمونه را کپی می کند .



Reference Type

شامل تمامی کلاس ها ، array ، delegate ، interface و string ها می باشد .

یک reference type شامل دو جزء می باشد :

1. یک شی
2. یک ارجاع به آن شی

محتوای یک متغیر ، ثابت یک reference type ، یک ارجاع به یک شی است که حاوی مقدار است .

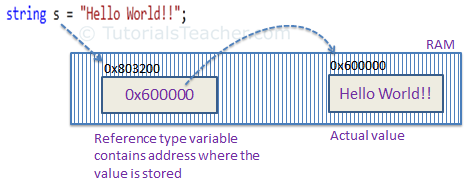
می توانید انواع reference type های سفارشی را با استفاده از کلمه ی کلیدی class تعریف کنید .

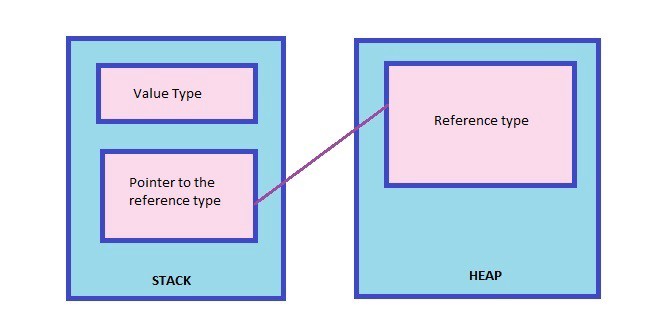
Public class Point {

public int x ;

public int y ;

}





انتساب یک reference type ، خود reference را کپی می کند نه نمونه ای از شی را ، این ویژگی اجازه می دهد که چندین متغیر به یک شی ارجاع دهند .

می توان به یک reference ، لیترال null را منصوب کرد که به این معناست که reference به شیئی اشاره نمی کند .

نکته : value type به صورت دقیق حافظه ای که برای ذخیره ی فیلد های آن نیاز است را اشغال می کند .

Public struct Point {

public int x ; // 4 bytes

public int y ; // 4 bytes

}

نکته ی 2 : به صورت تکنیکی CLR فیلدهای یک نوع را در آدرسی قرار می دهد که چند برابر سایز فیلد اصلی هستند .

نکته ی 3 : reference type نیاز به گرفتن جداگانه ی حافظه برای یک شی دارد . شی به اندازی فیلدهایش به علاوه ی سربار مدیریتی ، حافظه اشغال می کند .

انواع از پیش تعریف شده در سی شارپ

* Value Types :
  + Numeric :
    - Singed Integer ( sbyte , short , int , long )
    - Unsigned Integer ( byte , ushort , uint , ulong )
    - Real Number ( float , double , decimal )
  + Logical ( bool )
  + Character (char)
* Reference Type :
  + String ( string)
  + Object (object)

مجموعه ای از پیش تعریف شده ی value types ها به جزء decimal را به عنوان انواع ابتدایی در CLR می شناسند . انواع اولیه را به این جهت نام گذاری کرده اند زیرا که آن ها مستقیما توسط دستور العمل های پردازنده پشتیبانی می شوند و در زمان کامپایل مستقیما به دستورالعمل های پردازنده ، تبدیل می گردند .

نکته : در میان انواع عددی ، سی شارپ int و long را می پسندد ( runtime نیز به همچنین ) سایر انواع فقط برای استفاده بهینه از حافظه اند.

نکته : انواع صحیح 8 و 16 بیتی مانند byte ، sbyte ، short و ushort فاقد عملگرهای ریاضی هستند و سی شارپ به صورت غیر صریح برای محاسبات آنها را به int یا long تبدیل می کنند .

نکته : نمی توان برابری دو NaN را با == مقایسه کرد ، برای تست این موضوع می توانید از دستورات زیر استفاده کنید :

float.IsNaN;

double.IsNaN;

کلمه ی کلیدی checked

عملگر checked به runtime می گوید در صورتی که مقدار یک نوع مقداری از محدوده ی آن در یک دستور یا گزاره بالاتر یا پائین تر برود ، به جای overflow کردن آن در سکوت ، یک استثنای OverflowException را پرتاب کند .

int c = checked(c\*b);

// or

checked

{

C = a \* b ;

}

کلمه ی کلیدی unchecked

برای غیر فعال کردن چک کردن برای overflow

int c = unchecked(x+1);

// or

Unchecked { int z = x\*y ; }

نکته : به صورت پیش فرض کدها در حالت unchecked اجرا می گردند .

آرایه ها

آرایه نماینگر تعدادی مشخص و ثابت از متغیر ها ( که به آنها المان گفته می شود ) از یک نوع مشخص است . المان های آرایه همیشه در یک بلاک پیوسته از حافظه و به صورت ترتیبی ذخیره می گردند که دسترسی به آنها را بسیار سریع و موثر می سازد .

char[] vowels = new char[5]; // Declare an array of 5 characters

براکت ها به عنوان اندیس آرایه عمل می کنند . می توان به یک المان خاص به صورت زیر دسترسی پیدا کرد:

vowels[0] = 'a';

vowels[1] = 'e';

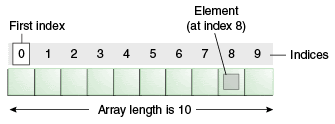
vowels[2] = 'i';

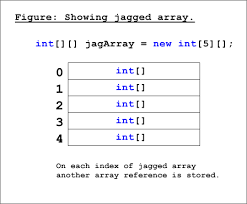
vowels[3] = 'o';

vowels[4] = 'u';

Console.WriteLine (vowels[1]); // e

زمانی که یک آرایه ایجاد شد ، نمی توان اندازه ی آن را تغییر داد . ویژگی Length در آرایه همیشه تعداد اعضای آرایه را مشخص می کند . مهم نیست نوع المان های یک آرایه چه هستند ، خود آرایه همیشه یک شی از نوعی reference type می باشد .





نوع مقداری یا نوع ارجاعی بودن المان های تابع بر روی عملکرد آرایه و رفتار آن تاثیر مستقیم می گذارد.

اگر نوع المان ها از نوع مقداری باشد ، به صورت پیش فرض با ساخت آرایه تمامی المان ها با نوع پیش فرض مقدار مورد نظر ، پر می شوند و به آنها حافظه تخصیص داده می شود.

اگر نوع المان های آرایه ی ساخته شده از نوع ارجاعی باشد ، به صورت پیش فرض ، مقدار پوچ null در المان ها قرار می گیرد که می تواند باعث خطاهای زمان اجرا شود . برای جلوگیری از این نوع خطاها می توانید تمامی اعضای آرایه را به صورت صریح با نوع مورد نظرتان ، نمونه سازی کنید .

متغیر های و پارامتر ها

متغیر نماینگر یک مکان در حافظه است که دارای مقدار قابل تغییر است . یک متغیر می تواند یکی از موارد زیر باشد :

* متغیر محلی
* پارامتر ( value ، ref یا out )
* فیلد ( instance یا static )
* المان آرایه

پشته و انباره ( The Stack and The Heap )

پشته و انباره مکان هایی هستند که متغیرها و ثابت ها درون آنها قرار می گیرند . هر کدام از این موارد از لحاظ معنایی و عملی ، طول حیات بسیار متفاوتی دارند .

پشته ( Stack ) :

پشته بلاکی از حافظه است که متغیرهای محلی و پارامتر های درون آن قرار می گیرند . پشته به صورت منطقی با ورود و خروج به متدها/توابع ، بزرگ و کوچک می گردد .

انباره ( heap ) :

انباره ( heap ) بلوکی از حافظه است که « اشیاء/ object » درون آن قرار می گرند . هر زمان یک شی ساخته می شود . بلاکی از حافظه درون heap به آن تخصیص داده می گردد و یک reference به آن شی ، برگشت داده می شود .

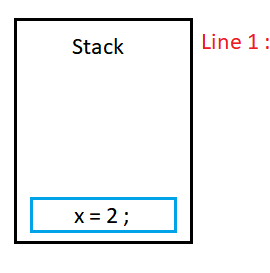
Runtime یک garbage collector دارد که به صورت دوره ای اشیا درون heap هستند را در صورتی که هیچ ارجاع زنده از آنها موجود نباشد را آزاد می کند تا برنامه با کمبود حافظه مواجهه نگردد.

یک شی به محض اینکه دیگر توسط شی زنده ای ارجاع داده نشود ، آماده ی آزاد سازی توسط garbage collector خواهد شد .

نمونه های value-types ( و ارجاع به اشیاء ) در همان جایی که اعلان شده اند ، قرار می گیرند . اگر یک نمونه به عنوان یک فیلد درون یک کلاس یا به عنوان یک آرایه تعریف شده باشند ، نمونه درون heap قرار خواهد گرفت .

علاوه بر این ، heap فیلدهای استاتیک را نگه می دارد . بر خلاف اشیائ که در heap به آنها فضا تخصیص داده شده ( که می توانند garbage collect گردند ) . این ها تا زمانی که Application Domain پابرجاست ، درون حافظه باقی می مانند .

شکل شماتیک نحوه ی پر شدن و خالی شدن ، پشته و انباره



public void SomeMethod()

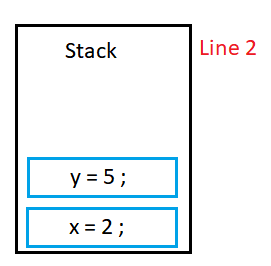
{

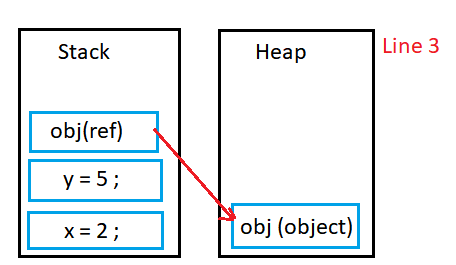
int x = 2 ;

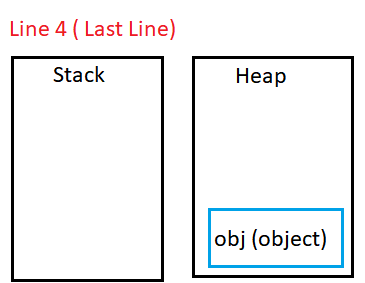
int y = 5 ;

MyClass obj = new MyClass();

}







Definitive Assignment

سی شارپ اجازه نمی دهد خارج از زمینه ی unsafe به صورت مستقیم به حافظه ی مقدار اولیه داده نشده ، دسترسی داشته باشیم . فیلد ها و المان های آرایه به صورت خودکار با مقادیر پیش فرض نوع مورد نظر ، مقدار دهی اولیه می شوند .

مقادیر پیش فرض :

تمامی reference type ها null

تمامی انواع عددی و enum 0

نوع char ‘\0’

نوع bool false

پارامترها :

پارامترها ، آرگومان هایی هستند که باید برای متد ، ارائه ( آماده ) شوند . می توانید نحوه ی ارسال پارامترها را با ref و out کنترل کنید .

ارسال پارامتر به وسیله ی مقدار/ value

به صورت پیش فرض ، ارسال آرگومان ها در سی شارپ به وسیله ی ارسال با مقدار ( passed by value ) انجام می شود ؛ این به این معناست که در هنگام ارسال ، یک کپی از مقدار اصلی ساخته شده و به متد مورد نظر ارسال می گردد .

ارسال پارامتر به وسیله ی ارجاع / reference

این کار یک کپی از reference را در هنگام ارسال به متد ، می سازد . برای ارسال به وسیله ی ارجاع باید از کلمه ی کلیدی ref استفاده کنید .

باید برای ارسال به وسیله ی ارجاع ، از کلمه ی کلیدی *ref* هم در هنگام تعریف متد و هم در هنگام فراخوانی متد استفاده کنید تا بتوانید از قابلیت ارسال به وسیله ی ارجاع در سی شارپ استفاده کنید .

static void increaseNumberBy2( ref int p )

{

p += 2 ;

}

static void Main()

{

int x = 8 ;

increaseNumberBy2(ref x);

Console.WritelLine( $" now x is equal to { x}"); // it should be 10

}

کلمه ی کلیدی out

مانند ref است ولی این تفاوت ها را دارد:

* نیازی به تخصیص داده شده قبل از ورود به تابع/متد ندارد
* باید پیش از خروج از تابع/متد ، تخصیص داده شود
* معمولا برای داشتن چند خروجی از تابع استفاده می گردد

static void split (string name , out string fname , out string lname )

{

int i = name.LastIndexOf('');

fname = name.SubString(0,i);

lname = name.SubString(i+1);

}

static void Main()

{

string a ;

split("Reza Pouya", out fname , out lname);

Console.WriteLine($"My Name is {fname} {lname}");

}

البته اگر علاقه ای نداشته باشید که مقداری بعضی از پارامترهای out برای شما برگشت داده شود ، می توانید از کارکتر \_ استفاده کنید .

split("Reza Pouya", out \_ , out lname);

کلمه ی کلیدی params

می توان آن را به عنوان آخرین پارامتر یک متد ستفاده کرد و به این معناست که می توان هر تعداد آرگومان از آن نوع خاص را گرفت . params باید از نوع آرایه باشد.

static int Sum(params int[] numbers){

int sum = 0 ;

for(int i = 0 ; i < numbers.Length ; i++)

sum += numbers[i];

return sum;

}

ref محلی :

از ref محلی برای بهینه سازی سطح پائین استفاده می شود و معمولا از آن به همراه ref returns استفاده می گردد . هدف ref محلی باید یک المان درون آرایه ، یک فیلد یا یک متغیر محلی باشد . نمی توان یک property را به عنوان ref در نظر گرفت .

int[] numbers = {0,1,2,3,4};

ref int numRef = ref numbers[2]; // 3

// now we can directly manipulate numbers[2] value

ref return

می توانید یک ref local ( ref محلی ) را با از یک متد برگشت دهید . به این عمل ref return می گویند . برای این کار باید نوع برگشتی متد با کلمه ی کلیدی ref مشخص شده باشد و همچنین متغییری که این مقدار درون آن قرار می گیرد نیز با این کلمه ی کلیدی مشخص شود .

علاوه بر این برای حفظ امنیت دو محدودیت زیر بر روی این قابلیت اعمال شده است :

* Ref های بازگشتی باید فقط به اعضای قابل تغییر ( mutable ) شی اشاره کنند ( نه اعضای readonly ) ، یا چیزهایی که به وسیله ی reference به درون متد فرستاده شده باشند .
* نمی توان ref local ها یک نقطه را در مکانی دیگر ، تغییر داد ( muted )

static string name = "old value";

static ref string GetName(){ret ref name;}

static void Main()

{

ref string = ref GetName();

}

مثال بعدی : Ref Return

public class Score

{

private int value = 5;

public ref int Get()

{

return ref this.value;

}

public void Print()

{

Console.WriteLine($"Score: {this.value}");

}

}

مثال بعدی re locals

public void test1()

{

var score = new Score();

ref int highscore = ref score.Get();

int anotherScore = score.Get();

score.Print();

Console.WriteLine($"Highscore: {highscore}");

Console.WriteLine($"Another Score: {anotherScore}");

highscore = 10;

anotherScore = 20;

this.change(highscore);

score.Print();

Console.WriteLine($"Highscore: {highscore}");

Console.WriteLine($"Another Score: {anotherScore}");

}

public void change(int value)

{

value = 30;

}

Output:

Score: 5

Highscore: 5

Another Score: 5

Score: 10

Highscore: 10

Another Score: 20

از خروجی مشاهده می کنیم که متغیر highscore دقیقا دارد به متغیر محلی score.value ارجاع می دهد ، چون مقدار آن تغییر کرده است . ولی با این حال متغیر anotherScore که فقط یک کپی از مقدار score.value را دارد ، بنابراین تغییر دادن آن هیچ تغییر در مقدار متغیر score.value نمی دهد . در انتها می بینیم که درون متد change() می بینیم که وقتی به یک ref local بدون استفاده از کلمه ی کلیدی ref دسترسی پیدا می کنیم ، مانند متغیر های محلی ساده عمل می کنند و مقدار آنها به وسیله ی مقدار به سایر متدها ارسال می شود .

کلمه ی کلیدی var

نوع متغیر را مشخص می کند ، کامپایر در زمان کامپایل آن را با نوعی که تشخیص می دهد ، جایگزین می کند . استفاده از کلمه ی کلیدی var برای anonymous method اجباری است .

گزاره ها و عملگرها ( Expression & Operators )

یک گزاره به به صورت بنیادین نشانگر یک مقدار است . ساده ترین نوع گزاره ها ، متغیرها و ثابت ها هستند . می توان با استفاده از عملگرها ، گزاره های را ترکیب و انتقال داد .

بعضی از گزاره های مورد استفاده در سی شارپ :

* نقطه ( دات ) . : گزاره ی جستجوی عضو

var obj = new myClass();

var value = obj.item1 ;

* پرانتر () : گزاره ی فراخوانی یک متد ( به همراه عملگر () )

عملگر های چپ انتساب ( left – associative operators ) :

عملگرهای دودوی ( به جزء عملگرهای انتساب ، لامبدا و null coalescing ) چپ – مقدار هستند . به عبارت دیگر ، این عملگرها از چپ به راست به مقادیر اعمال می شوند .

8/4/2

(8/4)/2 // 1

البته می توانید با استفاده از عملگر پرانتز ، اولیت بندی اجرا را تغییر دهید :

8 / (4/2) // 4

عملگرهای راست انتساب :

عملگرهای انتساب ، لامبدا ، null coalescing و null conditional راست انتساب هستند.یعنی ابتدا مقدار سمت راست آنها بررسی شده و سپس در مقدار سمت چپ قرار می گیرد .

عملگر null coalescing :

برای اینکار از کلمه ی کلیدی ?? استفاده می کنیم :

string s1 = null ;

string s2 = s1 ?? "nothing" ; // now s2 is equal to "nothing"

عملگر null-condtional

برای این کار از عملگر ?. استفاده می کنیم :

var obj = new MyClass();

string s = obj?.ToString();

اگر obj برابر null نبود ، متد ToString() فراخوانی می شود و مقدار آن درون s قرار می گیرد ، در غیر این صورت مقدار s برابر null قرار می گیرد .

می توان عملگرهای null coalescing و null conditional را با هم ترکیب کرد :

var v1 = x?.y?.z;

int f = m?.Math() ; // if m is null throw null refrence Exception

int? d = m?.Math() ; // ok

StringBuild sb = null ;

string s = sb?.ToString() ?? "nothing" ;

switch

static vod showCard(int cardNumber)

{

switch(cardNumber)

{

case 13 : Console.WriteLine("Shah"); break ;

case 12 : Console.WriteLine("Shah Banoo"); break ;

default : Console.WriteLine(cardNumber);

}

}

از سی شارپ نسخه ی 7 به بعد می توانید بر روی انواع نیز switch کنید :

static vod TellMeTheType(object x)

{

switch(x)

{

case int i :

{

Console.WriteLine("The Type is int");

Console.WriteLine($"The square of {i} is {i\*i}");

break ;

}

default : Console.WriteLine(“I don’t now the type !!”);

}

}

می توانید یک Case را با کلمه ی کلیدی When ، شرطی کنید ( predicate (:

switch (x)

{

case bool b when b == true :

Console.WriteLine("The value is true"); break;

case bool b when b == false :

Console.WriteLine("The value is false"); break;

default : break ;

}

کلمه ی کلیدی using

بنا به زمینه ی استفاده کاربردهای گوناگون دارد ، یکی برای استفاده ی پکیج ها و کتابخانه ها و فضاهای نام دیگر و دیگری برای استفاده بهینه از حافظه و کمک به garbage collector برای از بین بردن منابع بلااستفاده .

دستور using یک سینتکس ساده برای فراخوانی متد Dispose یک شی که اینترفیس IDisposable را پیاده سازی کرده است را می دهد .

در واقع using در پس زمینه به try {} finaly {} تبدیل می شود :

using (SqlConnection connection = new SqlConnection(sqlServeConnectionString))

{

// code which will use connection obj

}

در واقع به کد زیر کامپایل می شود

Try {

SqlConnection connection = new SqlConnection(sqlServeConnectionString);

// code which will use connection obj

}

finally{

If(connection != null)

Connection.Dispose();

}

Readonly

این کلمه ی کلیدی مانع از تغییر فیلد پس از به پایان رسیدن اجرای سازنده می شود . یک فیلد readonly پس از به پایان رسیدن اجرای سازنده ، فقط قابل خواندن است .

متد

امضای یک متد باید درون یک نوع ، یکتا باشد .

امضای یک متد شامل :

* نام متد
* نوع بازگشتی
* نوع پارامترهای ورودی
* ترتیب پارامترهای ورودی ( غیر هم نوع )

می باشد .

متد می تواند شامل تغییر دهنده های زیر ( modifier ) باشد :

|  |  |
| --- | --- |
| استاتیک | static |
| سطوح دسترسی | public , private , internal , protected , internal protected |
| سطوح ارث بری | new , virtual , abstract , override , sealed |
| متدهای چند بخشی | Partial |
| تغییر دهنده های کدهای مدیریت نشده | unsafe , extern |
| تغییردهنده های کدهای غیر همزمان | Async |

متدهای محلی :

متدهای محلی می توانند درون بدنه ی هر متدی ، درونی getter ها و setter ها ی property ها و سازنده ها قرار بگیرند . می توانید یک متد را درون متد دیگر تعریف کنید ، متد محلی فقط در متد والد آشکار و قابل دسترس است . چون متد های محلی در زمان اجرا به متد های استاتیک private تبدیل می شوند بنابراین دسترسی به آنها دارای سرباری نیست و علاوه بر این می توانید تمامی ویژگی های متدهای معمولی را داشته باشند ، یعنی می توانند Async باشند ، می توانند generic ، و یا dynamic باشند . البته تفاوت هایی وجود دارد ، متدهای محلی نمی توانند استاتیک باشند ، و همچنین متدهای محلی می توانند به متغیرهای محلی بلاک در برگیرنده ی این متد به دست بیاورند .

class Bar

{

public void Foo(int[] numbers)

{

foreach(var i in numbers)

{

Console.WriteLine(AsPrintable(i));

}

}

private string AsPrintable(int i) => $"I have {i} here";

}

متدهای محلی بهترین کاندیدا برای استفاده درون iterator ها و متدهای async در هنگام تائید اعتبار پارامترها هستند .

public IEnumerable<T> AsEnumerable<T>(params T[] items)

{

if (items == null)

throw new ArgumentException(nameof(items));

IEnumerable<T2> Enumerate<T2>(T2[] array)

{

foreach(var item in array)

{

yield return item;

}

}

return Enumerate(items);

}

علاوه بر این ، متدهای محلی می توانند [Caller Info Attributes](https://msdn.microsoft.com/en-us/library/hh534540.aspx) را فراخوانی کنند :

public static void SlimShady()

{

void Hi([CallerMemberName] string name = null)

{

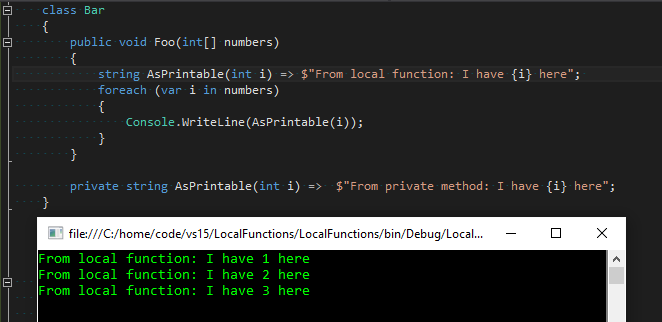
Console.WriteLine($"Hi! My name is {name}");

}

Hi();

}

علاوه بر این ، اگر یک متد private و یک متد محلی همنام داشته باشیم ، متد محلی ، متد private را مخفی می کند ، درست مثل لامبدا ها است .



کلمه ی کلیدی Deconstructor

از سی شارپ 7 به بعد ما می توانیم از الگوی Deconstructor استفاده کنیم .

class Rectangle

{

public readonly float width , height;

public Rectangle(float width , float height)

{

this.witdth = width ; this.height = height;

}

public void Deconstruct(out float width , out float height)

{

witdth = this.width ; height = this.height;

}

}

var rect = new Rectangle(3,4);

(float width , float height) = rect; // deconstruction

Indexer

ایندکسر های یک سینتکس طبیعی برای دسترسی به المان ها درون یک کلاس یا struct که یک لیست یا دیکشنری از مقادیر را کپسوله می کنند ، هستند . ایندکسر های مشابه property ها هستند ، ولی به جای دسترسی به وسیله ی نام به وسیله ی یک آرگومان ایندکس قابل دسترسی هستند .

class StringDataStore

{

private string[] strArr = new string[10]; // internal data storage

public string this[int index]

{

get

{

if (index < 0 && index >= strArr.Length)

throw new IndexOutOfRangeException("Cannot store more than 10 objects");

return strArr[index];

}

set

{

if (index < 0 || index >= strArr.Length)

throw new IndexOutOfRangeException("Cannot store more than 10 objects");

strArr[index] = value;

}

}

}

class Program

{

static void Main(string[] args)

{

StringDataStore strStore = new StringDataStore();

strStore[0] = "One";

strStore[1] = "Two";

strStore[2] = "Three";

strStore[3] = "Four";

for(int i = 0; i < 10 ; i++)

Console.WriteLine(strStore[i]);

}

}

نکاتی که در مورد ایندکسر ها باید به خاطر بسپارید :

* یک ایندکس درست مثل property است با این تفاوت که به وسیله ی کلمه ی کلیدی this و یک براکت که تعداد پارامترها را می گیرد ، تعریف می شود .
* می توان ایندکسر ها با گرفتن نوع جدیدی از پارامترها ، override کرد .
* امکان استفاده از پارامترهای ref و out برای ایندکسرها وجود ندارد .
* می توان ایندکسرها را به عنوان یک پارامتر interface ها استفاده کرد .

فرق constant و static readonly:

Constant در زمان کامپایل مقداردهی و جایگزین می شود ولی استاتیک readonly فقط در زمان اجرا و درون سازنده ی شی مقدار دهی می شود . یعنی static readonly به ازای هر برنامه می تواند یک مقدار جداگانه و خاص همان برنامه را بگیرد .

Static class

باید تمام اعضای آنها static باشد .

Finalizers

این ها تنها متدهایی هستند که فقط برای کلاس ها وجود دارند . این متدها پیش از آنکه garbage collector حافظه را از یک شی ارجاع داده نشده ، پس بگیرد ، فراخوانی می شوند .

class SomeClass

{

~SomeClass()

{

// code here

}

}

در واقع Finalizer ها یک سینتکس C# برای override کردن متد Finalize از object است .

protected override void Finalize()

{

// code here

base.Finalize();

}

Finalizer اجازه ی استفاده از unsafe را می دهد .

قالب گیری و تبدیل ارجاع ها ( Casting and reference Conversions ) :

ارجاع به یک شی را می توان :

* به صورت غیرصریح به ارجاع کلاس پایه ( والد ) آن تبدیل کرد . ( upcast )
* به صورت صریح به ارجاعی از کلاس های زیرین ( فرزند ) آن تبدیل کرد ( downcast )

کلمه ی کلیدی as

عملگر as یک تبدیل به کلاس زیرین ( downcast ) را انجام می دهد . اگر تبدیل با شکست مواجه شد ، به جای پرتاب کردن یک اسثناء ، ارجاع را برابر null قرار می دهد .

public class Asset

{

public string Name;

}

Public class Stock : Asset

{

}

Asset asset = new Asset();

Stock stock = asset as Stock ; // ok – downcast

as نمی تواند تبدیل های سفارشی را انجام دهد .

long x = 3 as long ; // compile time error

کلمه ی کلیدی is :

این عملگر چک می کند که آیا امکان تبدیل ارجاع وجود دارد یا خیر !؟ به عبارت دیگر ، چک می کند که آیا یک شی از یک کلاس خاص مشتق شده یا ارث بری کرده است !؟ ( یا یک اینترفیس خاص را پیاده سازی کرده است !؟ )

معمولا از این عملگر پیش از downcast ( عملگر as ) استفاده می شود .

if( asset is stock )

Console.WriteLine(((Stock)asset).SharesOwned);

عملگر is همچنین اگر unboxing conversation قابل انجام باشد ( تبدیل یک نوع به نوع اصلی را unboxing گویند ) ، به مقدار true تبدیل می شود . هر چند نمی توان از آن برای تبدیل سفارشی یا عددی استفاده کرد .

نکته : می توان از is برای تعریف یک متغیر از نوع مورد نظر در لحظه استفاده کرد .

if( asset is Stock stock )

Console.WriteLine(stock.SharesOwned);

if( asset is Stock)

Stock stock = asset as Stock // or // (Stock)asset ;

Abstract

انتزاع کردن یا Abstraction در سی شارپ به فرایندی گفته می شود که در آن جزئیات داخلی مخفی می شوند و فقط رفتار و عملیات نشان داده می شوند . تغییر دهنده و کلمه ی کلیدی abstract تنها نشان دهنده ی پیاده ی سازی کامل نشده است . از کلمه ی کلیدی abstract پیش از نام کلاس یا متد می توان استفاده کرد تا اعلان نمود کلاس یا متد abstract هستند و پیاده سازی آنها کامل نشده است . علاوه بر این می توان از تغییر دهنده ی abstract به همراه indexer ها ، property ها و event ها استفاده کرد .

متد های abstract :

متدهایی هستند که دارای پیاده سازی نیستند ( بدنه ندارند ) و باید حتما در کلاس فرزند پیاده سازی شوند . ( آنها حتما باید درون کلاس فرزند با استفاده از کلمه ی کلیدی override ، پیاده سازی شوند . )

فقط امکان تعریف این متدها درون کلاس های abstract وجود دارد .

public abstract class AbstractClass

{

public abstract void SomeMethod();

}

کلاس abstract :

این راهی است که می توان abstraction را درون سی شارپ به دست آورد . هیچ وقت نمی توان از یک کلاس Abstract را به صورت مستقیم نمونه سازی کرد . این کلاس حداقل باید یک متد abstract داشته باشد . معمولا از کلاس های abstract برای تعریف یک کلاس پایه در سلسله مراتب شی گرایی و ارث بری استفاده می شود .

نکات مهم :

* معمولا از کلاس abstract زمانی استفاده می کنیم که بخواهیم از ارث بری استفاده کنیم .
* باید قبل از پیاده سازی متدهای abstract درون کلاس های فرزند ، از کلمه ی کلیدی override استفاده کرد .
* کلاس abstract امکان ارث بری توسط struct ها را ندارد .
* می تواند سازنده و نابودگرها را درون کلاس های abstract تعریف کرد . ( constructor و Deconstructor )
* کلاس های abstract می توانند متدهای معمولی و غیر abstract داشته باشند .
* نمی تواند ارث بری چندگانه داشته باشند .
* نمی تواند static باشد .
* می توان از abstract برای get و set نیز استفاده کرد .

// C# program to show the working

// of abstract class with the

// get and set accessors

using System;

abstract class absClass {

protected int myNumber;

public abstract int numbers{ get; set; }

}

class absDerived : absClass {

// Implementing abstract properties

public override int numbers

{

get{ return myNumber; }

set{ myNumber = value; }

}

}

// Driver Class

class geek {

// Main Method

public static void Main()

{

absDerived d = new absDerived();

d.numbers = 5;

Console.WriteLine(d.numbers);

}

}

کلاسی که به عنوان abstract تعریف شده را نمی توان نمونه سازی کرد ، به جای آن ، فقط subclass های معمولی آن را می شود نمونه سازی کرد . می توان درون کلاس های abstract ، اعضای abstract تعریف کرد . اعضای abstract مانند اعضای virtual هستند ، با این تفاوت که نمی توان پیاده سازی پیش فرض داشته باشند . پیاده سازی باید توسط subclass ایجاد گردد . ( مگر اینکه خود subclass هم abstract باشد . )

کلمه ی کلیدی new

این کلمه ی کلیدی ، در زمینه های مختلف ، معانی مختلفی دارد .

* برای تخصیص یک ارجاع و نمونه سازی از یک شی
* برای مخفی کردن یک عضو از کلاس والد به صورت صریح

در حالت دوم ، استفاده از این کلمه ی کلیدی فقط پیام اخطار compiler را مخفی می کند .

نکته : کلمه ی کلیدی new به کامپایلر می گوید که از پیاده سازی کلاس فرزند به جای پیاده سازی کلاس والد استفاده کند . هر کدی که به کلاس فرزند ارجاعی نمی دهد و به جای آن به کلاس والد ارجاع می دهد ، از پیاده سازی کلاس والد استفاده می کند .

public class Base

{

public virtual void DoIt()

{

}

}

public class Derived : Base

{

public new void DoIt()

{

}

}

Base b = new Derived();

Derived d = new Derived();

b.DoIt(); // Calls Base.DoIt

d.DoIt(); // Calls Derived.DoIt

فرق new و override

new مانند آنچه در بالا گفته شد ، عمل می کند .

کلمه ی کلیدی override ، ممکن است بر روی متدهای virtual استفاده شود ، در عین حال باید برای متدهای abstract حتما استفاده شود . این کلمه ی کلیدی به کامپایلر می گوید که همیشه از آخرین پیاده سازی تعریف شده ی متد استفاده کند . حتی اگر متد در یک ارجاع از کلاس پایه ( base ) فراخوانی شده باشد ، با این حال از پیاده سازی آن برای override کردنش استفاده می کند .

public class Base

{

public virtual void DoIt()

{

}

}

public class Derived : Base

{

public override void DoIt()

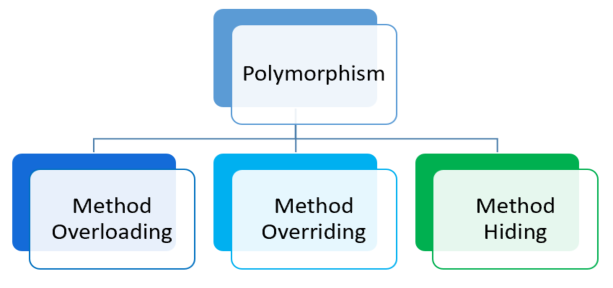
{

}

}

Base b = new Derived();

b.DoIt(); // Calls Derived.DoIt



کلمه ی کلیدی Sealed

مانع از override شدن اعضای کلاس و کلاس می شود . می توان آن را در دو سطح کلاس و اعضای کلاس استفاده کرد .البته ادعا می شود که استفاده از کلمه ی کلیدی sealed باعث بهبود سرعت اجرای برنامه می شود و دلیل این ادعا این است که درون Intermediate Language دو نوع متد برای فراخوانی توابع وجود دارد ، یکی call و دیگری callvirt . معمولا call سریعتر از callvirt می باشد چون شما از callvirt زمانی استفاده می کنید که نمی دانید آیا کلاس شما دارای فرزند است یا خیر !! پس تعدادی از افراد فرض را بر این گذاشته اند که با نشانه گذاری کلاس ها به عنوان sealed ، به کامپایلر می فهمانیم که این کلاس دارای فرزند نیست و باید از call برای فرخوانی آن در IL استفاده شود .

متاسفانه callvirt کارهای دیگری نیز می کند ، مثلا چک کردن پوچ بودن ارجاع ( null reference checking ) که این بدین معناست که حتی با نشانه گذاری کلاس به عنوان sealed امکان null بودن ارجاع آن وجود دارد و کامپایلر باید از callvirt استفاده کند .

Struct ها همیشه از call استفاده می کنند چون struct ها هیچگاه null نمی شوند. بنابراین استفاده از struct ها بسیار سریع تر است .

کلمه ی کلیدی base

این کلمه ی کلیدی در زمینه های گوناگون ، عملکردهای مختلفی دارد .

* برای دسترسی به یک متد override شده درون زیرکلاس ( کلاس فرزند )
* برای فراخوانی سازنده ی کلاس والد ( پایه )

سازنده و ارث بری

یک زیر کلاس باید سازنده ی خود را تعریف کند . سازنده های کلاس پایه قابل دسترس برای کلاس مشتق شد هستند ولی هیچ وقت به صورت خودکار ، به ارث برده نمی شوند .

سازنده های پایه همیشه پیش از سازنده های زیر کلاس ، فراخوانی و اجرا می شوند . این باعث می شود که همیشه نمونه سازی از کلاس پایه ، پیش از نمونه سازی از کلاس های فرزند انجام می شود .

اگر یک سازنده در زیر کلاس کلمه ی کلیدی base را استفاده نکنند ، سازنده ی بدون پارامتر کلاس پایه به صورت غیر صریح فراخوانی می گردد . اگر کلاس پایه هیچ سازنده پیش فرضی نداشته باشد ، زیر کلاس ها مجبور هستند که از کلمه ی کلیدی base درون سازنده های خود استفاده کنند.

ترتیب اجرای سازنده و مقداردهی های اولیه به فیلد ها :

1. از زیر کلاس به کلاس پایه :
   1. ابتدا فیلدها مقدار دهی اولیه می شودند
   2. آرگومان ها به سازنده ی کلاس چایه ابتدا اجرا می شوند
2. از کلاس پایه به زیر کلاس
   1. ابتدا بدنه ی سازنده ها اجرا می گردد .

public class Valed

{

int x = 1 ; // execute order : 3

public Valed ( int x )

{

// exeute order : 4

}

}

public class Farzand : Valed

{

int y = 1 ; // execute order : 1

public Farzand(int m) : base(m) // execute order : 2

{

int f = 2 ; // execute order : 5

}

}

نوع Object

object (System.Object) کلاس پایه برای تمامی انواع است . هر نوعی را می توان به این نوع upcastکرد.

Boxing and UnBoxing

به عمل تبدیل یک نمونه از value-type به reference-type ، عمل Boxing می گویند . reference-type می تواند class یا interface باشد.

int x = 9 ;

object obj = x ; // boxing

عمل unboxing درست عکس عمل boxing است ، در این حالت ، شی به مقدار اولیه برگشت داده می شود .

int y = (int)obj ; // unboxing

long m = (long)obj; // InvalidCastException

عمل unboxing نیاز به تبدیل صریح دارد . runtime چک می کند مقدار مورد نظر با نوع اصلی object یکسان است یا نه ، در غیر این صورت InvalidCastException را پرتاب می کند .

نکته : عمل boxing ، نمونه ی value-type را به درون شی جدید کپی می کند .

نکته : عمل unboxing ، محتوای object را به درون value-type کپی می کند .

چک کردن انواع در زمان اجرا و به صورت ایستا Runtime Checking And Static Checking

برنامه های سی شارپ ، « انواع » را به صورت ایستا ( در زمان کامپایل ) و در زمان اجرا ( به وسیله ی CLR ) چک می کنند .

چک کردن زمان اجرا ، در زمانی که یک downcast به وسیله ی unboxing یا reference conversion در حال انجام است ، انجام می شود .

این کار – چک کردن دز زمان اجرا – به این دلیل ممکن است چون اشیاء درون heap یک توکن کوچک نگه می دارند که « نوع » شی را نگه می دارد. این توکن را می توان با فراخوانی متد GetType() از object ، دریافت کرد .

عملگرهای GetType() و typeof

در سی شارپ تمامی انواع در زمان اجرا به وسیله ی نمونه ای از کلاس System.Type نمایش داده می شوند . دو راه ساده برای به دست آوردن شی System.Type وجود دارد :

* استفاده از متد GetType() بر روی یک نمونه ی کلاس .
* استفاده از کلمه ی کلیدی / عملگر typeof بر روی یک نام نوع ( مثلا اینترفیس یا کلاس )

GetType() در زمان اجرا و بر روی شی قابل استفاده است ولی typeof به صورت ایستا و در زمان کامپایل اجرا می گردد . ( وقتی انواع جنریک مورد استفاده باشند ، به وسیله ی کامپایلر Just In Time ، قابل دستیابی می شود .

System.Type دارای ویژگی هایی برای نام ، اسمبلی ، نوع پایه و ... می باشد . به مثال زیر توجه کنید .

using System;

public class Point { public int X, Y; }

class Test

{

static void Main()

{

Point p = new Point();

Console.WriteLine (p.GetType().Name); // Point

Console.WriteLine (typeof (Point).Name); // Point

Console.WriteLine (p.GetType() == typeof(Point)); // True

Console.WriteLine (p.X.GetType().Name); // Int32

Console.WriteLine (p.Y.GetType().FullName); // System.Int32

}

}

علاوه بر این ها ، System.Type دارای متدهایی است که به عنوان دروازه ی ورودی قابلیت Reflection ( بازتاب) در سی شارپ مورد استفاده قرار می گیرند .