فهرست

مقدمه

زبان C# یک زبان برنامه نویسی جدید است که در ژوئن ۲۰۰۰ به عموم مردم عرضه شد . طراحی این زبان توسط تیمی متخصص از مایکروسافت به مدیریت Andres Hejlsberg انجام شده است.این شخص از مهندسینی است که در تولید سایر زبان ها چون Borland C++ و Borland Delphi نقش داشته است.تمرکز وی در C# حفظ جنبه های مثبت و افزودن امکاناتی برای ایجاد زبانی بهتر بوده است.

این زبان از نوادگان C به شمار می رود و امکانات بسیاری را از زبان های C++، Java و Visual Basic به ارث برده است. مایکروسافت ادعا می کند که C#، قدرت و کمال C++ را همراه زیبایی و سادگی Visual Basic یکجا ارایه می کند. در C# توان شما به قوه تخیل شما محدود می شود و خود زبان هیچ قیدی را به شما تحمیل نمی کند.

C# بر روی CLR[1] که ماشین مجازی برای اجرای برنامه های NET. است، اجرا میشود. کامپایلر این زبان توسط مایکروسافت به نام NET Framework SDK. ارایه شده و به طور مجانی قابل دریافت است. البته تلاش هایی در جهت پیاده سازی NET Framework. در سیستم عامل لینوکس شده است که از آن جمله می توان به پروژه dot GNU و پروژه mono اشاره کرد.
C# در اصل برای برنامه سازی در محیط سیستم عامل ویندوز طراحی و بهینه شده است. C# به عنوان زبان منتخب برای برنامه نویسی در محیط NET Framework. شناخته شده است. برنامه های نوشته شده برای محیط NET. این قابلیت را دارند که به هر زبان دلخواهی، از C++ و Visual Basic گرفته تا Delphi و PHP و ASP و حتی جاوا و J# و JavaScript برنامه نوشت و از امکانات و مزایای هر یک از این زبان ها استفاده کرد.

به عقیده بسیاری از مردم با وجود زبان هایی نظیر C++، جاوا ، پرل و ... نیازی به یک زبان برنامه نویسی جدید وجود ندارد زیرا زبان های موجود کلیه امکانات مورد نیاز را فراهم می کنند. حال چرا C# ؟

دلایل زیر قابل ذکر است:

- **C# ساده است:**

[1] Common Language Runtime

- در #C بسیاری از پیچیدگی ها و مشکلات سایر زبان ها حذف شده اند. برای نمونه می توان به حذف ماکروها، قالب ها، وراثت چندگانه و کلاس های مجازی اشاره کرد. حتی در #C بخش عمده ای از مشکلات ناشی از کاربرد اشاره گرها حذف شده است.

- **#C امروزیست:**

- چه چیزی یک زبان را امروزی می سازد؟ ویژگی هایی نظیر Exception Handling و Garbage Collection ،انواع داده ای قابل بسط و ایمنی کد از ویژگی های قابل انتظار زبان های امروزی هستند و #C تمامی آنها را داراست. شاید زیباترین ایده در #C مربوط به نحوه تعریف و استفاده از Delegate (شبیه pointer to function در C++) و event هاست.

- **#C شی گراست:**

- کلید زبان های شی گرا در مفاهیمی چون کپسوله کردن، وراثت و چندریختی نهفته است .#C به خوبی از این مفاهیم پشتیبانی می کند.

- **#C قدرتمند و انعطاف پذیر است:**

- این زبان قابلیت کاربرد گسترده ای برای ایجاد واژه پردازها، برنامه های گرافیکی، صفحات گسترده و حتی کامپایلرها دارد. همچنین کتابخانه این زبان، مربوط به .NET Framework بوده و بسیار غنی است. این کتابخانه شامل کلاس هایی برای گرافیک، وب، XML، تاریخ و کار با پایگاه داده است. انواع داده ای در #C بیشتر از آنکه عضوی از زبان باشند، عضو .NET Framework هستند؛ به عنوان مثال نوع داده ای string و String تفاوتی ندارند.

- **#C زبانی با حداقل کلمات کلیدی است:**

- تعداد کلمات کلیدی زبان #C انگشت شمار است و این خود قابلیت مهمی برای زبانی با این درجه از قدرت به شمار می آید.

- **#C ماجولار است:**

- کد #C در بخش هایی با عنوان کلاس نوشته می شود . کلاس ها و روال های آنها در سایر برنامه ها قابل استفاده مجدد هستند.

- **#C محبوبیت عام را کسب خواهد کرد:**

- یکی از دلایل کلیدی مایکروسافت و تکنولوژی .NET آن است. دلایل دیر محبوبیت ویژگی هایی است که پیشتر ذکر کردیم: سادگی، شی گرا بودن، انعطاف پذیری و ...

بررسی قابلیت های زبان

تعریف متغیرها

#C عناصری را که بکار می گیرد همانند اعداد و کاراکترها ، به صورت نوع ها (Types) طبقه بندی می کند. این انواع شامل موارد زیر می شوند :

نوع های پایه ایی از پیش تعریف شده مانند اعداد و غیره.

نوع های تعریف شده توسط کاربر که شامل STRUCT ها و ENUM ها می شوند.

نکته ی مهمی که در اینجا حائز اهمیت است، مقدار دهی اولیه ی متغیرها می باشد. در غیر اینصورت کامپایلر #C برنامه را بایک خطا متوقف می کند. دلیل این امر هم این است که از استفاده از متغیرهای بدون مقدار در طول برنامه جلوگیری شود تا میزان خطاهای در حین اجرا کاهش یابد.

نوع های داده ای پایه ی زیر در در #C به صورت پیش فرض مهیا هستند:

Object: نوعی است نامحدود که می تواند تمام انواع دیگر را نیز شامل شود.

String:رشته؛ در اینجا یک رشته توالی کاراکترهای یونیکد می باشد.

Sbyte و byte: نوع داده ایی صحیح ۸ بیتی.

Short و Ushort: نوع داده ایی صحیح ۱۶ بیتی.

int و uint: نوع داده ایی صحیح ۳۲ بیتی.

Long و ulong: نوع داده ایی صحیح ۶۴ بیتی.

در اینجا u به معنای unsigned و s به معنای signed است.

Float و double: نوع اعشاری

bool: نوع داده ایی Boolean که می تواند true و یا false باشد.

char: کاراکتر، دراینجا char یک کاراکتر یونیکد است.

decimal: نوع داده ای دسیمال با ۲۸ رقم معنی دار.

تمام نوع های پیش فرض تعریف شده در سی شارپ شیء هستند.

بارگذاری عملگرها[1]

[1] Operator Overloading

در C# نیز همانند C++، کاربر این امکان را دارد که یک عملگر را برای نوع خاصی از داده بارگذاری کند. در نتیجه کد نوشته شده بسیار ساده تر میشود. در C#، تمام عملگرها به صورت ایستا تعریف میشوند. البته محدود بوده و به عنوان مثال constant نمی توانند باشند.

```
class Complex {static Complex operator + (Complex c1, Complex c2) {}};
```

تنها عملگر های زیر را می توان overload کرد :

Unary Operators
+ - ! ~ ++ -- true false

Binary Operators
+ - * / % & | ^ << >> == != > < >= <=

ساختارهای تصمیم گیری

ساختارهای تصمیم گیری در C# شامل if و switch-case می باشد. هر دو عبارت if و switch توسط عبارتهایی Boolean کنترل می شوند. در switch همه انواع را می توان به کار برد.

شی گرایی

❋Encapsulation

قابلیت کپسوله سازی با کلمات کلیدی public، private، protected و internal پیاده سازی میشود. کلمه internal برای دسترسی کلاس های همان بسته[1] به متغیرها و کلاس های مربوطه استفاده میشود.

❋وراثت[2]

C# چون زبانی شی گراست وراثت را پشتیبانی می کند، اما وراثت چندگانه را در پشتیبانی ندارد. نداشتن وراثت چندگانه ضعف بزرگی است و برای جبران آن، در این زبان ها نوعی از کلاس که متغیری همراه خود ندارد تعریف شده و به آن واسط یا interface می گویند. هر واسط می تواند از یک یا چند واسط به ارث ببرد و هر کلاس می تواند تنها از یک کلاس و چند واسط به ارث ببرد.
با استفاده از کلمه کلیدی sealed در C# می توان جلوی به ارث بردن از یک کلاس را گرفت:

```
public sealed class A : Class1, Interface1, Interface2 {}
```

[1] package
[2] Inheritance

***چند ریختی[1] و متدهای مجازی**

این قابلیت به کاربر اجازه می دهد که تابعی را در کلاس مجردی تعریف کرده و بدنه آن را در کلاس های به ارث برنده به انواع مختلف بنویسد.

در C# توابع به صورت پیش فرض در کلاس های به ارث برنده قابل بازنویسی نیستند و در صورت نیاز کلاس پدر باید از کلمه کلیدی virtual استفاده کند. کلاس فرزند نیز باید با بکار بردن کلمه کلیدی override به ارث بردن متد را اعلام کند. همچنین کلاس فرزند می تواند با استفاده از کلمه کلیدی sealed از به ارث رسیدن این متد به فرزندان خود جلوگیری کند. همچنین برای اینکه مشکل مخفی شدن متد پدر ایجاد نشود، زبان C# کلاس فرزند را ملزم می کند که مخفی شدن را به صورت غیر ضمنی با کار بردن کلمه کلیدی new اعلام کند. به این روش نسخه گذاری[2] گفته میشود.

همچنین می توان یک متد را به صورت مجرد با استفاده از کلمه کلید abstract، تعریف کرد تا در کلاس های به ارث برنده بدنه آن نوشته شود.

```
public abstract class A
{
        public virtual void f() {}
        public abstract void g() ;
        public void h() {}
}

public class B : A
{
        public sealed override void f( ) {}
        public override void g( ) {}
        public new void h( ) {} // explicitly hides A::h
}
```

***Boxing و UnBoxing**

این مفهوم اولین بار توسط زبان C# معرفی شد. C# زبان کاملا شی گراست و حتی با انواع داده ای اصلی خود نیز مانند object برخورد می کند.

```
Console.WriteLine(1.ToString());
```

به اینگونه که در صورت نیاز، متغیرهای از نوع داده اصلی (مانند int) به نوع کلاسی آن (یعنی Integer) تبدیل (Boxing) میشود. همچنین، در صورت نیاز از کلاسی از نوع Integer نیز قابل تبدیل (Unboxing) به نوع داده ای int است.

```
object o = 1; // boxing
```

[1] Ploymorphism
[2] versioning

int x = (int) o; // unboxing

ثابت ها

در #C از کلمه کلیدی readonly برای تعیین مقادیر ثابت استفاده میشود. البته کلمه const نیز جزو کلمات کلیدی #C است و تقریبا همان کار readonly را می کند.

Enum Data Types

در #C نیز همانند C++ می توان enum تعریف کرد که تنها مقادیر خاصی را بپذیرد. با این تفاوت که در #C هر متغیر حتما مقدار معتبری را دارد و همچنین می توان در زمان اجرا نام آنرا چاپ کرد. اگرچه enum در #C عملا کلاس است ولی می توان آن را در بلوک های تصمیم گیری switch-case نیز استفاده کرد و در کل می توان گفت پیاده سازی مناسبی دارد.

ذخیره سازی پیشرفته

❋ آرایه

در #C آرایه ها نوعی کلاس هستند و به کاربر اجازه می دهند اندازه آنها را در زمان اجرا محاسبه کند. آرایه های چند بعدی به صورت آرایه ای از آرایه ها تعریف می شوند که ممکن است گاه باعث از دست رفتن سرعت در زمان اجرا بشود.

در این زبان نوع دیگری از آرایه های چند بعدی وجود دارد که به جای آرایه ای از آرایه ها به صورت مستطیل (در حالت دو بعدی) تعریف میشود.

به طور کلی #C دو نوع آرایه ی چند بعدی را پشتیبانی می کند rectangular and jagged :

در یک آرایه ی rectangular هر ردیف ، طولش با ردیف بعدی یکی است. آرایه ی jagged در حقیقت آرایه ای از آرایه ها است ، بنابراین هر کدام از آنها می تواند طول مختلفی داشته باشد .

تعریف یک آرایه ی دوبعدی به صورت زیر است :

type [,] array-name = new type array-name[2,3];

Jagged arrays آرایه ای از آرایه ها است و همانطور که ذکر شد لزومی ندارد که هر ردیف آن با ردیف بعدی هم طول باشد . هنگام تعریف این نوع آرایه شما تعداد ردیف ها را مشخص می نمایید. هر ردیف یک آرایه را نگهداری می کند. در اینجا هر آرایه باید تعریف شود. روش تعریف Jagged array به صورت زیر است:

type [] []...

```
int[][] jaggedArray = new int[rows][];
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[2];
```

هنگام کار با آرایه های rectangular برای دسترسی به اعضا به صورت زیر عمل می شد :

rectangularArrayrectangularArray[i,j]

اما در اینجا بدین صورت است :

jaggedArray[3][i]

Structs❊

مساوی قرار دادن دو متغیر از نوع کلاس در زبان C# به کپی شدن اشاره گر این متغیرها منتهی میشود. در واقع از نوع کلاس به صورت اشاره گری به داده آن تعریف می شود و اصل کلاس بر روی heap قرار می گیرد، این عمل گاهی غیر ضروری است و گاهی نیز باعث کندی برنامه میشود.

در C# علاوه بر کلاس ها، نوع دیگری از داده وجود دارد که struct نام دارد و بر روی stack قرار می گیرد. اگر دو متغیر از نوع struct را مساوی هم قرار دهیم، مقدار اولی در دومی کپی میشود. یک کلاس مانند struct می تواند فرزند یک struct باشد و یا از چند واسط به ارث ببرد. struct ها میتوانند interface های مختلف را implement کنند ولی نمیتوانند از کلاس و یا structِ دیگری ارث ببرند. با متغیر از نوعِ struct مانند انواعِ داده ای مثلِ int رفتار میشود یعنی احتیاجی به new کردن ندارند.

```
class A { public int x; }
struct B { public int x; }
class Test
{
        static void Main( )
        {
                A a1;
                a1.x = 10;
                A a2 = a1; // copies the refrence
                a2.x = 12; // a1.x = 12, a2.x = 12

                B b1 = new B( );
                b1.x = 10;
                B b2 = b1; // copies the contents
                b2.x = 12; // b1.x = 10, b2.x = 12
        }
}
```

Enumeration and Iterators❊

بیشتر داده ساختارها به طریقی امکان حرکت و خواندن تمام مقادیر آن داده ساختار را می دهند. برای یک پارچه سازی این روند، مفاهیم enumeration و iteration به وجود آمده است (اگرچه تقریبا معادلند).
در C#، کلاس هایی که واسط Ienumerable را پیاده سازی کرده اند، این امکان را به کاربر می دهند که به روی آن داده ساختار حرکت کند. همچنین امکان استفاده از کلمه کلید foreach را می دهد. در نتیجه می توان بدون تعریف متعیر اضافی بر روی داده ساختار حرکت کرد.

```
collection< int > v = new Collection< int > ( );
foreach ( int x in v) System.Console.WriteLine ( x );
```

کلاس

❊Properties

در C# این امکان وجود دارد که متغیری مجازی تعریف نمود. این متغیر در واقع کار getter و setter برای متغیرها را انجام میدهد. فرم زیبای تعریف و استفاده از این متغیرهای مجازی استفاده از آن را راحت نموده است. متغیرهای مجازی یا Property در حقیقت چیزی جز تابع نیستند و همانند توابع قابلیت بارگذاری و وراثت را دارند.

```
private int x;
public int X
{
        get { return x; }
        set { this.x = value; }
}
```

لازم به ذکر است که value یک کلمه کلیدی است.

❊ سازنده[1] و خراب کننده[2]

سازنده برای مقدار دهی اولیه یک کلاس و خراب کننده به هنگام آزاد سازی آن استفاده میشود.
در C# این امکان به کاربر داده شده است که کدی را برای مقدار دهی اولیه به متغیرهای static بنویسد. چنین کدی در C#، به عنوان سازنده ی static معرفی شده است.

```
class A
{
        static readonly int x;
```

[1] Constructor
[2] Destructor

```
        static A( )
        {
                x = 10
        }
        public ~A( ) { }
}
```

فراخوانی با مقدار، ارجاع و output parameter

در #C فراخوانی پارامترها به طور پیش فرض به صورت فراخوانی با مقدار است و با استفاده از کلمه کلیدی ref می توان فراخوانی پارامتر را فراخوانی با ارجاع[1] تعریف کرد.

در #C امکان دیگری در نظر گرفته شده که با استفاده از کلمه کلیدی out که یک پارامتر تابع به عنوان خروجی در نظر گرفته شود.

```
namespace ex
{
        class Class1
        {
                public static int TestOut(out char i)
                {
                        i = ‘b’;
                        return -1;
                }

                static void Main(string[] args)
                {
                        char i; // variable need not be initialized
                        Console.WriteLine(TestOut(out i));
                        Console.WriteLine(i);
                        Console.ReadLine();
                }
        }
}
```

کلمه ی کلیدی ref نیز دقیقا همانند out عمل می کند و نحوه ی تعریف و استفاده از آن نیز مشابه است با این تفاوت که آرگومانی که به این نوع توابع فرستاده می شود باید مقدار دهی اولیه شده باشد.

تعریف تابعی با تعداد آرگومانهای نامعلوم

[1] Call by refrence

گاهی از اوقات نیاز است تا تابعی تعریف کنیم که تعداد آرگومانهای آن متغیر باشند. برای این منظور از کلمه ی کلیدی params استفاده می شود.
دونکته در اینجا حائز اهمیت است:

۱. در هر تابعی تنها می توان یکبار از params استفاده کرد .
۲. پس از بکار بردن params دیگر نمی توان هیچ آرگومانی را تعریف کرد .

یکی از مثالهایی که در این زمینه می توان ارائه داد استفاده از آرایه ها به عنوان آرگومان ورودی است. در این حالت یا می توان یک آرایه را به صورت کامل به تابع معرفی کرد و یا تنها نام آنرا به تابع پاس کرد.

```
namespace ex
{
	class Class1
	{
		public static void UseParams(params int[] list)
		{
			for ( int i = 0 ; i < list.Length ; i++ )
			Console.WriteLine(list[i]);
			Console.WriteLine();
		}

		static void Main(string[] args)
		{
			UseParams(1, 2, 3);
			int[] myarray = new int[3] {10,11,12};
			UseParams(myarray);
			Console.ReadLine();
		}
	}
}
```

Pointer و Type aliasing

این دو قابلیت در C# وجود ندارد.
در C# برخلاف C++ اشاره گر وجود ندارد و عملیات تخصیص حافظه و مدیریت آن کاملا بر عهده خود زبان است. برای جلوگیری از به هدر رفتن حافظه و ایجاد حافظه هرز، C# از garbage collector استفاده می کند.

پیمانه بندی و کتابخانه ها

پیمانه بندی در C# با مفهوم فضای نام[1] و با کلمات کلیدی namespace و using مشخص میشود. using این امکان را می دهد که به صورت منطقه ای از یک کلاس خاص استفاده کنیم. فضاهای نام روشی برای مدیریت کد نویسی هستند. برای مثال آنها ایجاد شده اند تا تداخلی بین نام های توابع در برنامه شما رخ ندهد. برای ایجاد یک فضای نام به صورت زیر عمل می شود :

```
namespace anyName
{
……..
        Class anyClassName
        {
        …….
        }
…….
}
```

تمام فضاهای نام به صورت پیش فرض public می باشند و در خارج از کد شما قابل دسترسی هستند. روش استفاده از آنها به صورت زیر است :

ProjectName.NameSpace.ClassName.MemberName

جابجایی پذیری[2]

از آنجا که .NET Framework. تنها توسط مایکروسافت برای سیستم عامل خانواده ویندوز ارایه شده عملا مبحث قابل حمل بود کد منتفی است، اما به نظر میرسد که پیاده سازی موجود در سایر سیستم عامل ها، امکان اجرای چنین فایل هایی را نیز دارند و از این نظر C# قابل حمل است.

Indexer

این قابلیت حالت خاصی از بارگذاری عملگرهاست. به صورت عادی تعداد پارامترهای عملگرهای بارگذاری شده ثابت است (به جز عملگر پرانتز). یکی از عملگرهایی که قابلیت بارگذاری شدن را دارد، عملگر [ ] است. در C# فرم تعریف و استفاده از این عملگر زیباتر است.

```
private int [ , ] arr = new int [10, 10];
public int this [ int x, int y]
{
        get { return arr [x] [y] + 1; }
```

[1] Namespace
[2] Portability

```
        set { this.arr [x] [y] = value -1 ; }
}
```

با استفاده از Indexer ها می توان با یک کلاس همانند آرایه ها رفتار کرد.

```
class IntIndexer
{
        private string[] myData;

        public IntIndexer(int size)
        {
                myData = new string[size];

                for (int i=0; i < size; i++)
                {
                        myData[i] = “empty”;
                }
        }

        public string this[int pos]
        {
                get
                {
                        return myData[pos];
                }
                set
                {
                        myData[pos] = value;
                }
        }
}
```

در بسیاری از زبانها استفاده از اعداد صحیح روشی است متداول برای دسترسی به اعضای آرایه ها اما Indexer ها در C# فراتر از این می رود. Indexer ها را می توان با پارامترهای متعددی تعریف کرد و هر پارامتر با نوعی مختلف (دقیقا همانند پارامترهای ورودی متدها). البته محدودیتی که اینجا وجود دارد در مورد نوع پارامتر ها است که تنها می تواندinteger ، enum و string باشد . بعلاوه قابلیت Overloading ایندکسرها نیز وجود دارد. به همین جهت به آنها آرایه های هوشمند هم گفته می شود(smart arrays)

Delegates and Events

یکی از مفاهیم زیبا در C# مفهوم نماینده یا Delegate است. به این صورت که دسته ای از توابع یک شکل را به کمک کلمه کلیدی delegate تعریف کرده و سپس متغیرهای متفاوتی که توابع هستند را تعریف می کنیم.

دیگر قابلیت جالب #C، نحوه تعریف کردن و استفاده و صدا کردن رخداد هاست که باعث شده، کد مربوط به واسط گرافیکی کاربر و سایر کلاس ها بسیار دلپذیر باشد. هر رخداد با کلمه کلید event مشخص می شود و از نوع یک نماینده تابع خواهد بود. هر رخداد مجموعه ای نماینده هاست که به ترتیب اضافه شدنشان اجرا میشوند.

```
class A{
        public delegate void MyFunc( int x );
        public event MyFucn CallMe;

        public void caller( )
        {
                if ( callMe != null )
                CallMe( 10 );
        }

        static void f( int x ) { System.Console.Write( “f “); }
        void g( int x ) { System.Console.Write( “g “); }

        public static void Main( )
        {
                A mm = new A( );
                mm.CallMe += new MyFunc( f );
                mm.CallMe += new MyFunc( mm.g );
                mm.caller( );
        }
}
```

Multi-Threading

#C پشتیبانی کاملی از برنامه سازی به صورت Multi-Threading را به کاربر می دهد. همچنین برای حل مشکل همزمانی از کلمه کلیدی lock استفاده میشود.

Attributes

مشخصه ها اجزایی هستند که توسط طراحان #C در این زبان گنجانده شده اند و هدف از طراحی آنها ایجاد قابلیت توسعه برای این زبان بوده است.

از ایده های جالب در #C توصیف خصوصیات یک کلاس، متغیر و یا تابع از طریق مشخصه هاست. یک attribute قسمتی از کد اجرایی نیست اما می تواند بر روی خصوصیات و نحوه استفاده از آن تاثیر بگذارد. به

عنوان مثال می تواند یک property را قابل تغییر اعلام کرد و همچنین، پنجره ه ای را نیز به آن تخصیص داد تا از کاربر از طریق واسط گرافیکی بتواند مقادیر جدید را بپرسد.
به طور کلی استفاده از مشخصه ها در سه مرحله انجام می شود:

- تعریف مشخصه
- مرتبط ساختن مشخصه ها با عناصر کد برنامه
- پرس و جوی مشخصه ها در زمان اجرای برنامه

Reflection and RTTI[1]

برخی اوقات تکیه زدن و مشاهده بازتاب زندگی کار بسیار خوبی است. با مشاهده بازتاب زندگی خود، قطعا به مواردی برخورد می کنید که تا به حال متوجه آن نشده اید. بازتاب نه تنها برای شما بلکه برای برنامه های C# نیز امکان پذیر است. این نوع بازتاب می تواند در یادگیری هر چه بیشتر برنامه ها بکار رود. برای مثال می توانید کلاسی داشته باشید که با بازتاب، شما را از متدها و خواص خود آگاه کند. به بیان دیگر بازتاب امکان دسترسی به صورت پویا به متغیرها و توابع در زمان اجرا را می دهد.
قابلیت دیگر که به موازات بازتاب وجود دارد، امکان تشخیص نوع یک متغیر در زمان اجراست.

مقابله با خطاها

EXCEPTION یک خطای زمان اجر است که بدلیل شرایطی غیرنرمال در برنامه ایجاد می شود. در این زبان exeption کلاسی است در فضای نام سیستم. شیء ایی از نوع exception بیانگر شرایطی است که سبب رخ دادن خطا در کد شده است. C# از exception ها به صورتی بسیار شبیه به جاوا و C++ استفاده می نماید، بدین صورت که کلمات کلیدی try و catch اساس مدیریت خطا را تشکیل می دهند.

[1] Run-Time Type Identifying