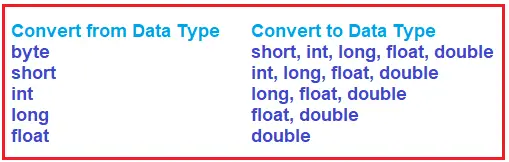
# Type Casting in C#

# تبدیل یک نوع داده به نوع داده دیگر تنها در صورتی که با یک دیگر سازگار باشند.

# Implicit Type Casting

# تبدیل یک نوع داده کوچکتر به نوع داده بزرگتر که به صورت خودکار توسط کامپایلر انجام می شود. نوع داده ها باهم سازگار باشند.



**Explicit Type Casting**

در این حالت تبدیل نوع داده امکان از بین رفتن بخشی از داده وجود دارد و ما باید به طور مستقیم تبدیل داده خود را اعلام کنید مثلا اگر نوع داده double را به int تبدیل کنیم قسمت اعشار از بین می رود یا ممکن است نوع داده ی رشته ی نامربوط را به نوع داده int بخواهیم تبدیل کنیم که سبب بروز خطا می شود.

روش های تبدیل نوع داده صریح

(int)

Convert.ToInt32()

Int.Parse()

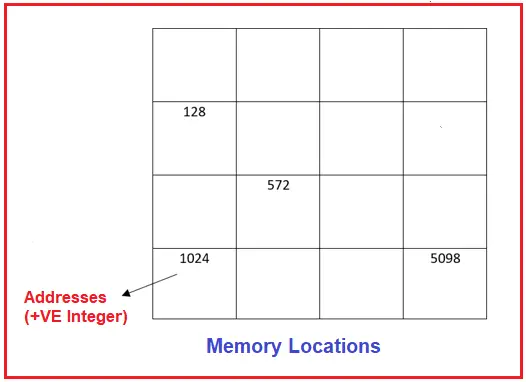
Int.tryParse()

امن ترین روش گزینه اخر است جون در صورت بروز خطا false بر میگرداند و در صورتی که موفق باشد مقدار خروجی را در متغییری به ما تحویل می دهد.

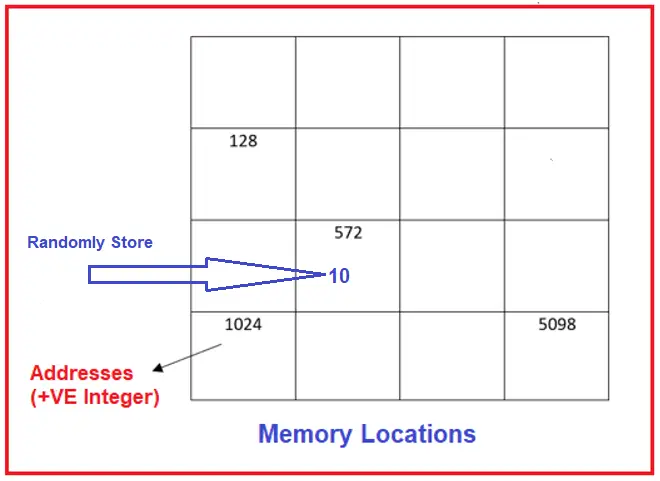
## **Variables in C# with Examples**

هر عملیاتی که در سیستم های نرم افزاری در حال انجام است به طور خلاصه بیانگر این است که بر روی داده های مثل شماره حساب بانکی مقدار موجودی و ... پردازش صورت می گیرد. پس بنابر این تمامی مقایر که روی آن ها پردازش صورت می گیرد باید در مکانی از حافظه وجود داشته باشند.

تمامی مکان های حافظه مانند صندلی های یک سالن تئاتر است که هر کدام دارای آدرس شناسایی منحصر به فرد خود است.



وقتی می خواهیم یک مقدار مثل عدد 10 را در حافظه ذخیره کنیم به صورت رندوم در یکی از خانه های حافظه که با شماره خانه محل آدرس آن مشخص است مقدار عدد ما ذخیره م شود



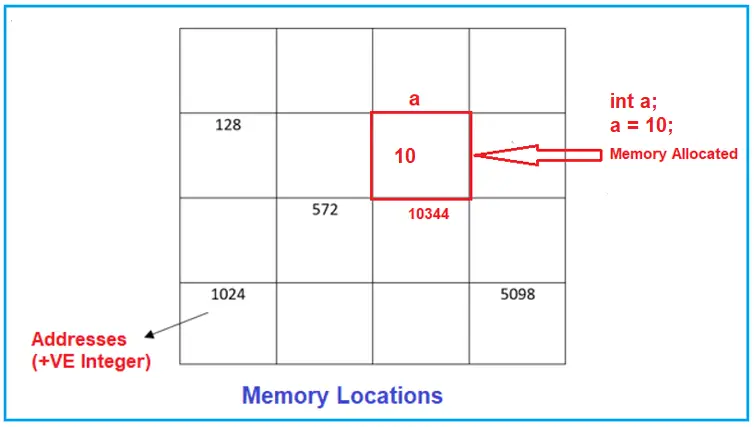
How To Access To Data

قبل از ذخیره کردن داده برای بازیابی آن باید آدرس محل حافظه که داده در آن ذخیره می شود را نگه داری کنیم.

How We Can Set Identify To Memory Location

می توانیم آدرس حافظه را در یک متغییر ذخیره کنیم

به طور مثل وقتی int a را داریم نوع داده را بیان کردیم و هویتش را با هر اسمی مشخص می کنیم مثلا a سپس به آن حافظه تخصیص داده می شود و با شناسه a محل حافظه آن مشحص می شود الان a نام محل حافظه به شماره ی 10344 است.



##### **What is a Variable in C# Language?**

نامی است که به یک خانه از حافظه داده شده است. هدف متعییر تخصیص نام به مکانی از حافظه است. کاربر متغییر را با نامش می شناسند و کامپایلر با داده آن را با آدرس حافظه.

نام متغییر باید

با letter یا underscore شروع شود

به صورت case sensitive است

می تواند با نوع داده خود constract شود

علائم خاص نمی پذیرد

##### **Types of Variables in a Class in C#**

Instance

Static

Constant

ReadOnly

نکته : اگر یک متغییر را در یک بلاک کد از نوع static تعریف کنیم نوع آن متغییر نیز از نوع static در نظر گرفته می شود.

using System;

namespace TypesOfVariables

{

internal class Program

{

static int x = 100; //Static Variable

int y = 200; //Non-Static or Instance Variable

static void Main(string[] args)

{

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

Console.WriteLine($"x value: {y}");

Console.Read();

}

}

}

قطعه کد بالا دارای خطا است به دلیل این که یک متغییر غیر استاتیک به نام y داریم که در یک متد از نوع static استفاد شده است باید این نکته را در نظر گرفت که تمامی متغییر های که از نوع non static هستند برای تخصیص حافظه به خود نیازمند ایجاد instance هستند به این معنی که با توجه به هر instance ساخته شده از خود به آن ها حافظه تعلق می گیرد در این مثال چون متد ما از نوع ststic است و نیازی به ساخت instance برای اجرا ندارد در آن نمی توان از متغییر non static استفاده کرد.

تنها در حالتی می شود که از کلاس program در این متد instance ساخته شود و به y مقداری تخصیص داده شود و سپس آن را چاپ کرد.

using System;

namespace TypesOfVariables

{

internal class Program

{

static int x = 100; //Static Variable

int y = 200; //Non-Static or Instance Variable

static void Main(string[] args)

{

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

Program obj = new Program();

Console.WriteLine($"y value: {obj.y}");

Console.Read();

}

}

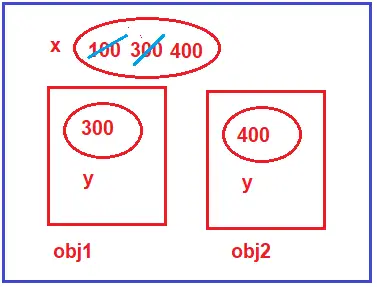
}

##### **When Static and Non-Static Variables are Initialized in C#?**

متغییر های استاتیک یک بر بلافاصله پس از شروع اجرای کد کلاس مقدار دهی می شوند و تا پایان عمر کلاس دارای همان مقدار می باشند اما متغییر های که از نوع غیر استاتیک هستند با هر ساخت نمونه از کلاس مقدار دهی می شوند

Can We Assign Value To Static Member With Constractor?

بله در هر با ساخت نمونه از کلاس می توانیم به متغییر استاتیک خود در سازنده مقداری را assign کنیم اما باید به این نکته توجه کرد که هر با مقداری دهی متغییر استاتیک باعث می شود که مقدار قبلی override شود به مقدار جدید.



###### **Instance/Non-Static Variables in C#**

1. **Scope of Instance Variable:** Throughout the class except in static methods.
2. **The lifetime of Instance Variable:** Until the object is available in the memory.

###### **Static Variables in C#**

1. **Scope of the Static Variable**: Throughout the class.
2. **The Lifetime of Static Variable**: Until the end of the program.

Constant Variable In C#

یک بار در موقعه ی ایجاد مقدار دهی می شود و از آن به بعد غیر قابل تغییر است اگر در موقع تعریف مقدار دهی نشود با خطای کامپایلر مواجه می شوید. به این دلیل که فقط یک بار ایجاد می شوند.

Const نیز نیازی به ساخت instance برای تخصیص مقدار به خود ندارد.

Const int a = 12;

ReadOnly

تشابه زیادی با متغییر های از نوع const دارد فقط با یک تفاوت که نیازی نیست در لحظه ی تعریف به آن مقدار دهید بلکه می توانید با constractor به آن مقدار دهید.

تقاوت با Const

Const یک مقدار ثابت برای کل کلاس ها است اما ReadOnly یک مقدار ثابت برای هر instance از کلاس است.

**Local Variables in C#**

1. **Scope of the Local Variables:** Within the block in which it is declared.
2. **The lifetime of the Local Variable:** Until the control leaves the block in which it is declared.

# ****Operators in C# with Examples****

# **Operator => + = ++ ,…**

# **Operand => 1 , 2, a ,v ,…**

**Types of Operators in C#**

**Arithmetic**

**Relational Or Comparison**

**Logical**

**Assignment**

**Bitwise**

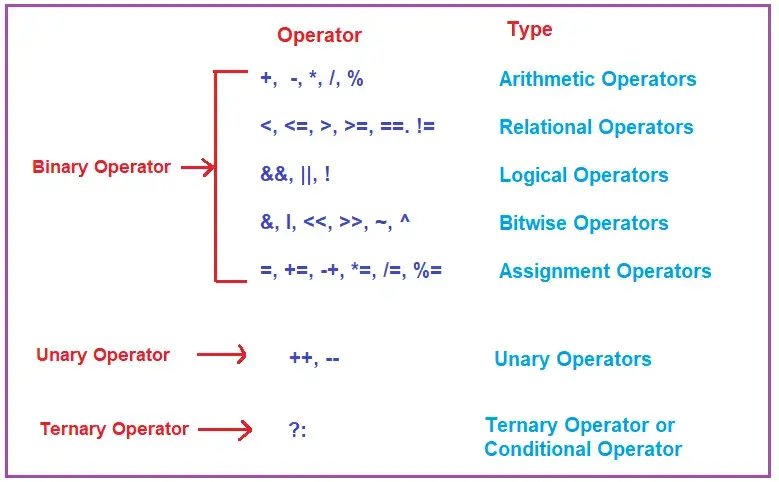
**Unary**

**Ternary**

**Conditional**

**با توجه به تعداد operand ها نیز تقسیم می شوند**

**Unary – binary – ternary**



##### **Arithmetic Operators in C#**

* **Addition Operator (+):**
* **Subtraction Operator (-):**
* **Multiplication Operator (\*):**
* **Division Operator (/):**
* **Modulus Operator (%):**

**Assignment Operators in C#**

* **Simple Assignment (=):**
* **Add Assignment (+=):**
* **Subtract Assignment (-=):**
* **Multiply Assignment (\*=):**
* **Division Assignment (/=):**
* **Modulus Assignment (%=):**

##### 

##### **Relational Operators in C#**

رابطه ای یا مقایسه ای نیز گفته می شود به این ها که معمولا به صورت boolen نتیجه را برمیگرداند.

* **Equal to (==):**
* **Not Equal to (!=):**
* **Less than (<):**
* **Less than or equal to (<=):**
* **Greater than (>):**
* **Greater than or Equal to (>=):**

**Logical Operators in C#**

* Logical OR (||)
* Logical AND (&&)
* Logical Not (!)

##### **Bitwise Operators in C#**

**Bitwise OR (|)**

عملگر بیتی | یا OR بر روی بیت های متناظر بین دو عملوند عملیات انجام می دهد اگر هر یک از بیت ها 1 باشد 1 می دهد و اگر نه 0 می دهد.

**int a=12, b=25;**  
**int result = a|b; //29**  
**How?**  
12 Binary Number: **00001100**  
25 Binary Number: **00011001**  
Bitwise OR operation between 12 and 25:  
**00001100**  
**00011001**  
**========**  
**00011101 (it is 29 in decimal)**

**Bitwise AND (&):**

اگر هر دو 1 باشد 1 می دهد و اگر هر دو 0 باشد 0 می دهد.

**int a=12, b=25;**  
**int result = a&b; //8**  
How?  
12 Binary Number: 00001100  
25 Binary Number: 00011001  
Bitwise AND operation between 12 and 25:  
**00001100**  
**00011001**  
**========**  
**00001000 (it is 8 in decimal)**

**Bitwise XOR (^):**

**اگر بیت ها یکی باشند مقدار 0 می دهد و اگر بیت ها یکی نباشند مقدار 1 می دهد.**

**int a=12, b=25;**  
**int result = a^b; //21**  
How?  
12 Binary Number: 00001100  
25 Binary Number: 00011001  
Bitwise AND operation between 12 and 25:  
**00001100**  
**00011001**  
**========**  
**00010101 (it is 21 in decimal)**

**Unary Operators in C#**

1. **Increment operators (++): Example: (++x, x++)**
2. **Decrement operators (–): Example: (–x, x–)**
3. **Post-Increment Operator**
4. **Pre-Increment Operator**

**Syntax: Variable++;**  
**Example: x++;**

**Syntax: ++Variable;**  
**Example: ++x;**

##### **Post Decrement Operators**

**Syntax: Variable–;**  
**Example: x–;**

**Syntax: –Variable;**  
**Example: —x;**

##### **Ternary Operator in C#**

**Syntax: Condition? first\_expression : second\_expression;**

# Goto Statement in C#

# برای انتقال کنترل برنامه به label کاربرد دارد

# Label در واقع یک شناسه مشخص است. توصیه می شود که استفاده نشود چون خوانایی را کاهش داده و پیچیدگی را افزایش می دهد.

# 

using System;

namespace JumpStatementDemo

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("1st Statement: ");

goto label1; //goto label label1:

Console.WriteLine("2nd Statement: ");

label1: //label:

Console.WriteLine("3rd Statement: ");

Console.WriteLine("End of Main Method Statement: ");

Console.ReadKey();

}

}

}

# اگر لیبل مورد نظر وجود نداشته باشد خطای کامپایلری مواجه می شوید.

# از goto برای خروج از حلقه ها نیز می توان استفاده کرد.

از goto برای انتقال کنترل برنامه به یک case خاص در switch نیز می توان استفاده کرد

نوشتن حلقه با goto

**int** count = 1;

label\_Loop:

Console.WriteLine**(**count**)**;

count++;

**if** **(**count **<**= 10**)**

**{**

**goto** label\_Loop;

**}**

Console.Write**(**"C#Tutorials "**)**;

Console.Write**(**"Welcome "**)**;

XYZ:

Console.Write**(**"X "**)**;

Console.Write**(**"Y "**)**;

**goto** ABC;

Console.Write**(**"Programming "**)**;

ABC:

Console.Write**(**"Hello1 "**)**;

Console.Write**(**"Hello2"**)**;

Console.ReadKey**()**;

**Output: C#Tutorials Welcome X Y Hello1 Hello2**

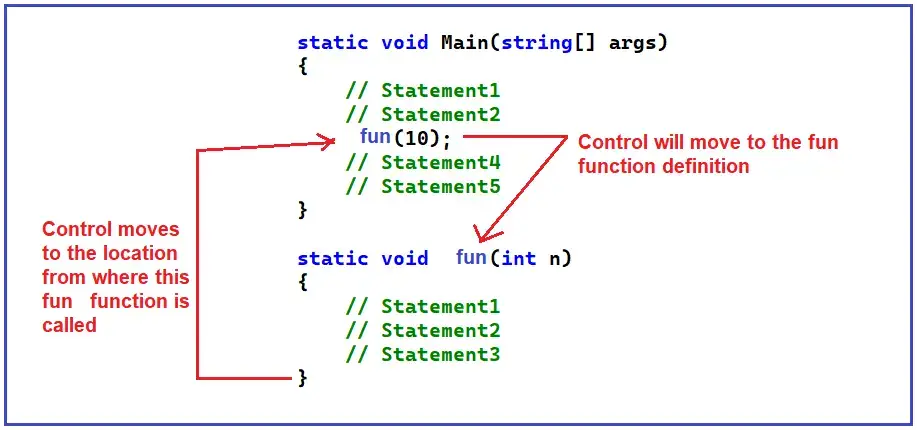
**اگر به یک برچسب برسیم که صدا نزیدم آن را کد های بعد از آن اجرا می شود و ارتباطی به عدم صدا کردن آن ندارد**

**در حلقه های goto عبارت های continue , break بی معنی است**

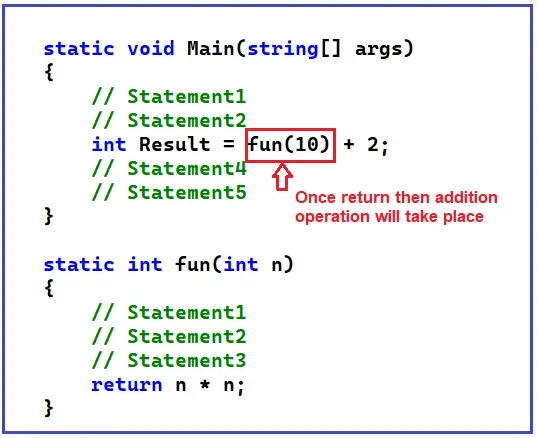
**برچسب ها به صورت caseSensetive هستند.**

### Recursion in C#

زمانی که متدی صدا زده می شود کنترل برنامه به آن متد رفته و پس از پایان عملیات مجدد کنترل برنامه به خط صدا زننده متد بازمی گردد.



اگر با مقدار بازگشتی از متد fun بخواهید کاری انجام دهید تنها زمانی قادر خواهید بود که متد مقداری را بازگردانده باشد.



نکته : اگر متدی a صدا کند متد b را و متد b صدا کند متد c را ترتیب انجام شدن کار ها اول اتمام متد c سپس b و در آخر a است.

##### **What does it mean by Recursive Function in C#?**

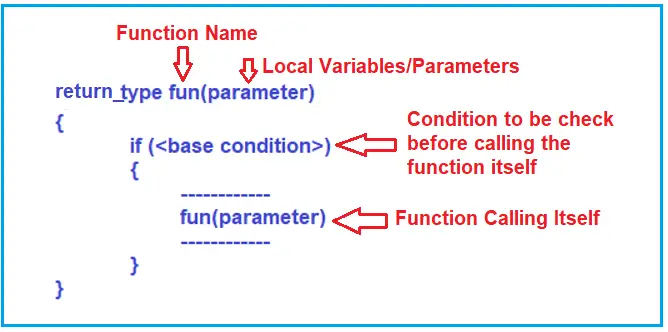
متدی که خودش را صدا می زند تا زمانی که یک شرط به نتیجه برسد و return کند.

برای حل مسائل recursive نیاز است که 2 شرط داشته باشیم.

اول مشکل باید به صورت recursive نوشته شود و متد خودش را صدا بزند

دوم باید شرطی برای پایان کار نیز داشته باشیم

زمانی که در متد recursive دارای متغییر محلی هستیم به اضای هر بار فراخوانی تابع متغییر ها مقدار های جدید را می گیرند و در stack مقدار نگه داشته می شود



متد های بازگشتی به صورت درختی trace می شوند.

نکته کار متد های بازگشتی تا پایان ریسدن ادامه دارد یعنی اولین صدا تا آخرین صدا زدن life cycle است.

از مزایای متد های بازگشتی حفط اطلاعات مربوط به فراخوانی تابع با بازگشت آن است.

معایب

کندی به دلیل stack overlapping

ممکن است سبب stack overflow شود

مکنن است سبب infinite loop شود

##### **How do you find the time complexity of a recursive function in C#?**

فرض کنید هر دستور در برنامه ما یک واحد زمانی برای اجرا نیاز دارد. حالا باید تکرار دفعاتی که متد بازگشتی با دستوری را اجرا می کند بشماریم در این مثال متد ما فقط مقداری را چاپ می کند بنابر این به ازای هر باز اجرا یک کار انجام میدهد یعنی رابطه ی دفعات تکرار برابر است با تعداد ورودی n اگر 3 باشد 3 بار اجرا دارد پس داریم O(n) یعنی بسته به ورودی ما پیچیدگی زمانی تغییر می کند.

##### **How do the variables work in a Recursive Function?**

مثال مهم مقدار n در زمان return متد صدا زده شده به مقدار ما افزوده می شود

**using** *System;*

**namespace** *RecursionDemo*

**{**

**class** Program

**{**

**static** **void** Main**(string[]** args**)**

**{**

**int** number = 5;

**int** Result = fun**(**number**)**;

Console.WriteLine**(**Result**)**;

Console.ReadKey**()**;

**}**

**static** **int** fun**(int** n**)**

**{**

**if(**n **>** 0**)**

**{**

**return** fun**(**n - 1**)** + n;

**}**

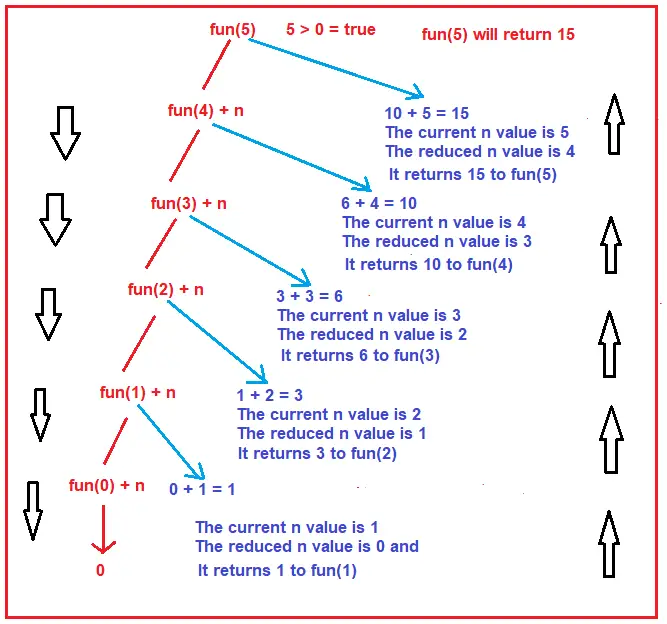
**return** 0;

**}**

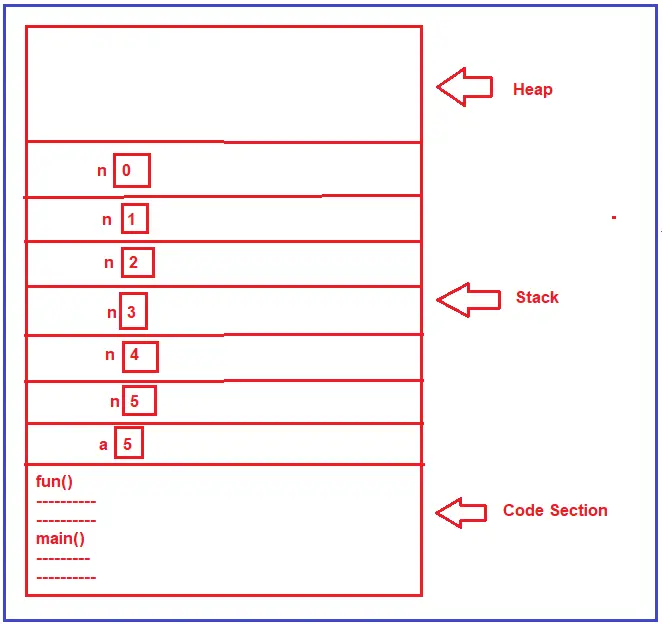
**}**

**}**

Trace در تصویر بعدی



دلیل stackOverflow شدن در حالت های recursive این است که متغییر در هر بار اجرا مقدار خود را در Stack نگه می دارد ولی در حلقه ها متغییر یک بار بیشتر در Stack مقدار دهی نمی شود.



نوع های داده در bcl در فضای نام system قرار دارند.

# **Command Line Arguments in C#**

راهکاری است برای ارسال parameter به متد main

نکته مهم این است که متد main از هیج متد دیگری paramer نمی گیرد. و فقط از command-line مقدار می گیرد.

از منوی properties و سپس تب Debug می توان command line arguemtns را مقدار دهی کرد که با استفاده از space مقادیر را از هم جدا می کند

# String in C#

کلاس است که دنباله ای از کاراکتر ها را ارائه می دهد و می توانیم عملیات های مختلفی بر روی آن انجام دهیم.

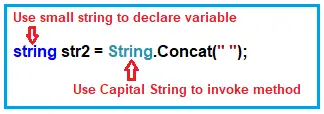
نکته » String از نوع reference type است

نکته » نوع های داده دیگر را اگر نگاه کنید می بینید که از نوع struct هستند به معنی value type اما string از نوع class است به معنی reference type بودن.

String vs string

می توان از هر دو استفاده کرد و تفاوتی ندارد در واقع string یک alias بر روی کلاس String است.

اما از String بهتر است برای method invoking استفاده کرد.



**Strings are Immutable in C#**

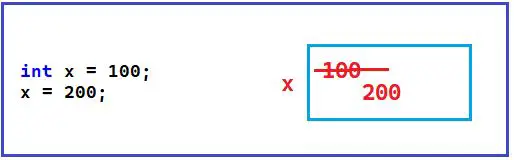
به معنی غیر قابل تغییر بودن string در سی شارپ است. به این معنی که وقتی رشته ای را به متغییری تخصیص می دهید که از نوع string است اگر مجدد مقدار جدیدی به آن تخصیص دهید مقدار قبلی pervious object به garbage collector رفته و یک object جدید ایجاد می شود و مقدار جدید را می گیرد. یعنی 2 مکان در حافظه ایجاد می شود. و هر بار که مقدار جدید به متغییر خود دهیم همین اتفاق رخ خواهد داد.



در واقع override اتقاق نمی اقتد برای مقدار قبلی بلکه حافظه جدید تخصیص داده می شود برعکس اتفاقی که در value type ها رخ می دهد. در این نوع override داده ی قبلی رخ می دهد.

Mutable به این معنی که داده ی آن با تغییر override می شود اما

Immutable داده ی آن با تغییر در حافظه ی جدید که ایجاد شده قرار می گیرد.



* مثال مدت زمان برای ایجاد حلقه ی بزرگ و تغییر آن در داده ی mutable , immutable
* **string** str = "";
* Console.WriteLine**(**"Loop Started"**)**;
* var stopwatch = new Stopwatch**()**;
* stopwatch.Start**()**;
* **for** **(int** i = 0; i **<** 30000000; i++**)**
* **{**
* str = Guid.NewGuid**()**.ToString**()**;
* **}**
* stopwatch.Stop**()**;
* Console.WriteLine**(**"Loop Ended"**)**;
* Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;

Time = 25991

Now with integer

**int** ctr =0;

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000000; i++**)**

**{**

ctr = ctr + 1;

**}**

stopwatch.Stop**()**;

Console.WriteLine**(**"Loop Ended"**)**;

Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;

Time = 84

Not with same integer

**string** str = "";

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000000; i++**)**

**{**

str ="DotNet Tutorials";

**}**

stopwatch.Stop**()**;

Console.WriteLine**(**"Loop Ended"**)**;

Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;

Time = 94

در این حالت به دلیل این که هر بار در حافظه مقدار جدید نمی سازد. به دلیل **String intern**

به این معنی که سی شارپ اگر مقدار رشته تغییر نکند از همان حافظه ی قبلی استفاده می کند. یعنی هر بار مقدار رشته را با مقدار رشته جاری بررسی می کند. برای افزایش performance

**StringBuilder for Concatenation in C#**

**نکته :** . The C# string immutability behavior can be very very dangerous when it comes to string concatenation

اگر یک رشته را مکررا با رشته ی دیگر ترکیب کنید هر بار مکانی جدید از حافظه اشغال می شود و مکان قبلی به g c تحویل داده می شود.

**string** str = "";

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000; i++**)**

**{**

str ="DotNet Tutorials" + str;

**}**

stopwatch.Stop**()**;

Console.WriteLine**(**"Loop Ended"**)**;

Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;



String builder حافظه ی جدیدی نمی سازد و از همان مکان حافظه ی قبلی استفاده می کند و سبب افزایش شدید کارایی می شود. Time 1ms vs 5473

StringBuilder stringBuilder = new StringBuilder**()**;

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000; i++**)**

**{**

stringBuilder.Append**(**"DotNet Tutorials"**)**;

**}**

اما چرا string به صورت immutable است ؟ به دلیل thread safety به موقعیتی فکر کنید که چندی thread بخواهند یک string را تغییر دهند.

# Static Keyword in C#

اگر از هر developer بپرسید که چرا به static نیاز داریم می گویید که در factory و singleton به آن نیاز داریم و همچنین در data sharing اما 3 هدف مهم وجود دارد که در ادامه توضیح خواهیم داد.

نکته abstraction به این معنی است که هر چیزی که نیاز است از دید دیگران مخفی باشد را باید به این صورت پیاده سازی کرد.

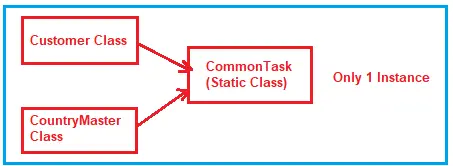
همچنین در oop باید در یک کلاس متد های که کاری مرتبط به هم انجام میدهند قرار داشته باشند مانند کی شی در دنیای واقعی.

نکته : زمانی که یک کلاس را به صورت static تعریف می کنیم یعنی کل اجزای کلاس نیز باید static باشد. در این حالت نیازی به new کردن کلاس برای دسترسی به اعضای آن نیست.

وقتی کلاس به static است clr یا common language runtime فقط یک نمونه از آن کلاس می سازد و به همه ارائه می دهد

نکته: ما هیج یک از قواعد oop را به کلاس static نمی توانیم پیاده سازی کنیم.

کلاس static یک بار instantiated می شود تا بتوانیم از اعضای آن استفاده کنیم.



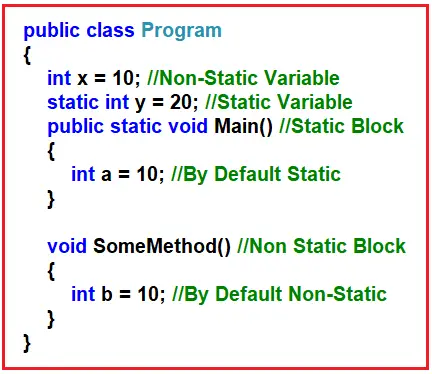
# Static vs Non-Static Members in C#

اعضای کلاس به 2 دسته بندی static و non static تقسیم می شوند.

Static : اعضای از کلاس که نیازی به ساخت instance. برای مقدار دهی یا اجرا ندارند را گویند.

اگر یک variable را به صورت staic تعریف کنیم یه در بلوک کدی که static است ایجاد کنیم نوع متغییر Static خواهد بود.

اعضای داخل متد static به صورت پیش فرض Static هستند.



مناسب برای نگه داری داده های از اپلیکیشن که در کل برنامه و برای همه ی object های یکسان است.

Static بلافاصله یک بار بعد از اجرای کلاس مقدار دهی میشود.

اما non static ها بعد از ایجاد object از کلاس مقدار دهی می شوند.

##### **What is the Scope of Static and Non-Static Variables in C#?**

Non staci با object ساخته می شود و با object ازبین می رود در واقع scope یک object برابر است با scope مربوط به reference variable

1. **Non-Static to Static:** Non-Static Members can be consumed only by using the object of that class inside a static block.
2. **Static to Static:** Static Members can be consumed directly (within the same class) or by using the class name (from outside the class) inside another static block.
3. **Static to Non-Static:** Static Members can be consumed directly or by using the class name inside a Non-Static block.
4. **Non-Static to Non-Static:** Non-Static Members can be consumed directly or by using the “this” keyword inside another non-static block.
5. **class** Example
6. **{**
7. **int** x = 100;
8. **static** **int** y = 200;
9. **static** **void** Add**()**
10. **{**
11. //we can access non static members X with the help of Example object
12. //We can access the static member directly or through class name
13. Example obj = new Example**()**;
14. //Console.WriteLine(obj.x + Example.y);
15. Console.WriteLine**(**"Sum of 100 and 200 is :" + **(**obj.x + y**))**;
16. Console.WriteLine**(**"Sum of 100 and 200 is :" + **(**obj.x + Example.y**))**;
17. **}**
18. **void** Mul**()**
19. **{**
20. //we can access static members directly or through class name
21. //we can access the non-static members directly or through this keyword
22. Console.WriteLine**(**"Multiplication of 100 and 200 is :" + **(**this.x \* Example.y**))**;
23. Console.WriteLine**(**"Multiplication of 100 and 200 is :" + **(**x \* y**))**;
24. **}**
25. **static** **void** Main**(string[]** args**)**

// Main method is a static method

1. // ADD() method is a static method
2. // Statid to Static
3. // we can call the add method directly or through class name
4. Example.Add**()**; //Calling Add Method using Class Name
5. Add**()**; //Calling Add Method Directly
6. // Mul() method is a Non-Static method
7. // We can call the non-static method using object only from a static method
8. // Static to Non-Static
9. Example obj = new Example**()**;
10. obj.Mul**()**; //Calling Mul Method using Instance
11. Console.ReadLine**()**;

##### **Example to Understand Static and Non-Static Constructors in C#**

نکته : سازنده غیر static اول از همه اجرا می شود.

**class** Example

**{**

//Static Constructor

//Executed only once

//First block of code to be executed inside a class

//Before Main Method body start executing, this constructor will execute

**static** Example**()**

**{**

Console.WriteLine**(**"Static Constructor is Called"**)**;

**}**

//Non-Static Constructor

//Executed once per object

//When we create an instance, this constructor will execute

**public** Example**()**

**{**

Console.WriteLine**(**"Non-Static Constructor is Called"**)**;

**}**

//Program Execution will start from the Main method

//But before executing the Main method body, it will

//execute the static constructor

**static** **void** Main**(string[]** args**)**

**{**

Console.WriteLine**(**"Main Method Execution Start"**)**;

Example obj1 = new Example**()**;

Example obj2 = new Example**()**;

Console.WriteLine**(**"Main Method Execution End"**)**;

Console.ReadLine**()**;

**}**

نکته » کلاس های static فقط اعضای static می توانند داشته باشند. و قابلیت ساخت instance ندارند

# Stack and Heap Memory in .NET

وقتی یک متغییر تعریف می کنیم چه اتفاقی رخ می دهد؟

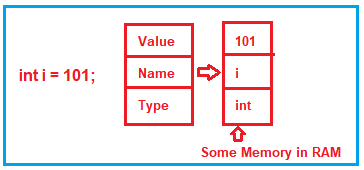
مقداری از حافظه ram را به متغییر تخصیص می دهد. این حافظه 3 چیز را بدنبال دارد

1 نام متغیرر

2 نوع داده متغییر

3 مقدار متغییر

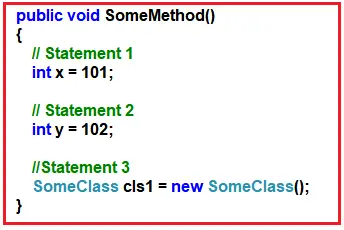
و وابسته به این که نوع داده value است یا reference حافظه stack یا heap اختصاص داده می شود.



##### **Understanding Stack and Heap Memory in C#**

2 نوع تخصیص حافظه در .net وجود دارد stack و heap

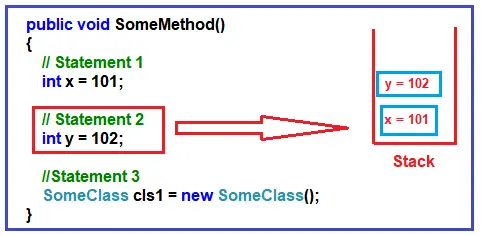
مثال



وقتی دستور اول اجرا می شود کامپایلر حافظه stack اختصاص می دهد.

وقتی دستور دوم اجرا می شود مجددا حافظه stack تخصیص داده می شود به این صورت که متغییر دوم در بالای متغییر اول در حافظه ی stack قرار می گیرد مانند ظرف های که روی هم قرار دارند.

تخصیص و عدم تخصیص حافظه stack به صورت last in – first out است یعنی از اول stack بر می دارد.



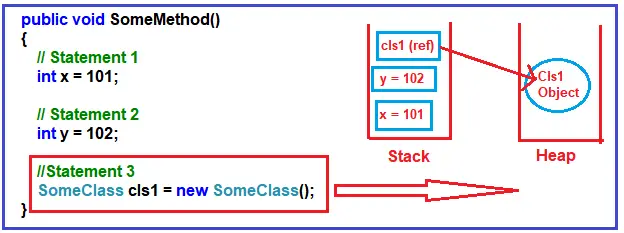
زمانی که دستور سوم اجرا می شود:

به صورت داخلی یک pointer در stack قرار می گیرد و object اصلی در مکان دیگری از حافظه ذخیره می شود به نام heap

Heap معمولا برای dynamic memory allocation کاربرد دارد.

نکته : reference pointer در Stack ذخیره می شود.

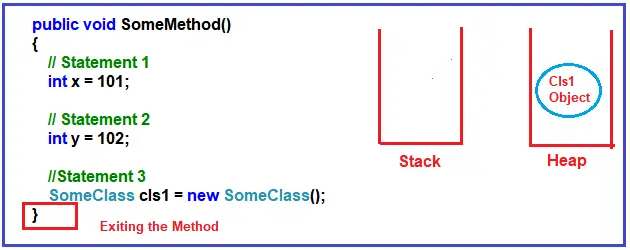
نکته : عبارت Person p1 هیچ حافظه ی را برای p1 اختصاص نمی دهد فقط یک variable با نام p1 در stack قرار می دهد و مقدار null را با آن تخصیص می دهد. و زمانی که کلمه کلیدی new برخورد می کند حافظه را در heap اختصاص می دهد.



##### **What Happens When the Method Completes Its Execution?**

وقتی اجرای متد به آخر می رسد و کنترل اجرا از متد خارج می شود یعنی به } می رسد. تمام variavble های ساخته شده در stack را پاک می کند به صورت lifo .

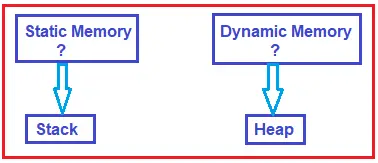
اما حافظه ی اختصاص داده شده به heap را از تخصیص خارج نمی کند. بعدا حافظه heap توسط garbage collector خالی می شود.



##### **Why do we have two types of memory?**

در سی شارپ primitive data type مثل int , double , bool تنها یک مقدار را نگه داری می کنند و در حالی که reference type ها یا object data type از نوع complex هستند یعنی یک نوع complex دارای reference به نوع های object دیگر و primitive های دیگر است.

بنابراین reference data type نگه دار ارجاع به چندین داده است و هر کدام از آن ها باید در حافظه ذخیره شود. نوع های object به dynamic memory نیاز دارند در حالی که primitive ها به static memory نیاز دارند.



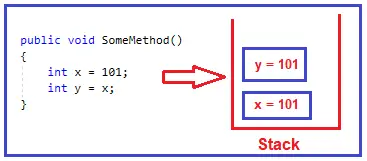
Value type vs reference type

Value ها نوع های هستند که هم Data و هم memory در یک مکان قرار دارند. اما Reference type نوع از داده هستند که دارای یک pointer که اشاره می کند به مکان اصلی حافظه.

##### **Understanding Value Type in C#:**

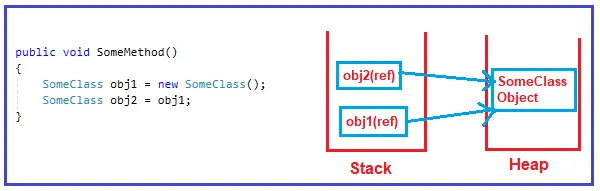
زمانی که به متغییری از نوع int مقداری را assign می کنیم و آن را به متغییر دیگری نیز تخصیص می دهیم یک copy از داده متغییر اول به متغییر دوم تخصیص داده می شود که کاملا از هم مستقل هستند. و تغییر یکی بر دیگری تاثیری ندارد مانند bool

Int double float و ...



##### **Understanding Reference Type in C#**

در این نوع اگر 2 متغییر از یک object ایجاد کنیم و سپس اولی را new کرده و مقدار آن را به دومی assign کنیم هر دو آن ها به یک مکان از حافظه اشاره خواهند کرد. و اگر یکی را تغییر دهید بر دیگری نیز تاثیر گذار خواهد بود مانند class interface object string , delegate



##### **How is the Heap Memory Freed Up?**

در حافظه ی Stack زمانی که کنترل اجرای برنامه در متد به اتمام می رسد و بعد از آخرین { شروع به deallocate کردن حافظه می کند اما در نوع heap این گونه نیست بلکه حذف حافظه با garbage است به این معنی که زمانی که object موجود در heap دیگر استفاده نمی شود به این معنی است که فاقد reference pointing شده و زمان حذف آن فرا رسیده.

Stack Key Points

* برای static memory allocation و local variable کاربرد دارد و توسط cpu مدیریت می شودکه سریع و کارآمد است.
* زمانی که یک متد فراخوانی می شود a block of memeory در stack برای local variable و parametrs اختصاص داده می شود و زمانی که به return می رسد بلوک غیر استفاده می شود و برای فراخوانی بعدی متد آماده می شود.
* متغییر های ذخیره شده در stack فقط در زمان فراخوانی متد در دسترس هستند.
* نگه داری value type or primitive و struct و reference object در آن انجام می شود

##### **Heap Memory Key Points:**

* برای dynamic memory که شامل object و complex data structures است و انعطاف پذیری بالا دارد و توسط garbage collector مدیریت می شود.
* Object ها در heap هستند و memory در runtime مدیریت می شود object جدید با کلمه کلیدی new ایجاد می شود و زمانی که استفاده نشوند gc آن را ازبین می برد.
* زمان آن ها از ایجاد با new تا زمانی که دیگر استفاده نشوند است.
* برای object array class , … استفاده می شود.

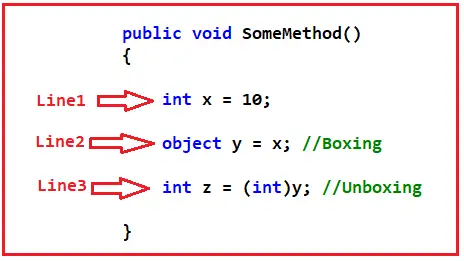
نکته : حافظه Stack فضای محدودی دارد ولی heap با توجه به نیاز رشد می کند. محدود آن حافظه کل است.

نکته : دسترسی به Stack سر راست تر است ولی دسترسی به heap دارای پیچیدگی بیشتر است.

# Boxing and Unboxing in C#

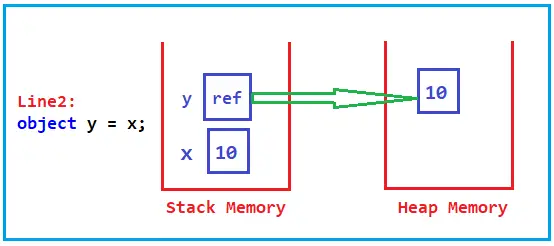
Boxing پروسه ی تبدیل value type به reference type است زمانی که Value type باکس می شود یک object جدید در heap تخصیص داده می شود و value در آن کپی می شود.

Unboxing برعکس boxing است زمانی که value از object استخراج می شود یعنی تبدیل reference به value و سبب کپی مقدار از heap به stack می شود

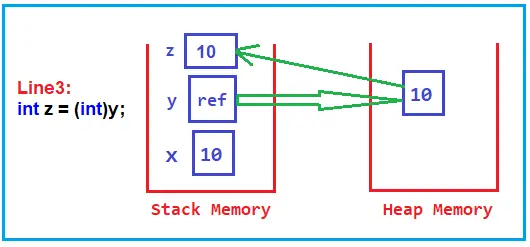


دستور اول متغییر x در حافظه stack ایجاد می شود و مقدار 10 را می گیرد

دستور دوم انتقال مقدار x که 10 است به نوع داده object انجام می شود و مقدار value type به reference type تخصیص داده می شود که عملیات boxing است



دستور سوم مقدار object که برابر 10 است را به مقدار z که از نوع value type است تخصیص می دهیم تبدیل reference به نوع value است عملیات unboxing



نکته عملیات boxing و unboxing که تبدیل مقدار value type به نوع reference type است می تواند روی عملکرد سیستم تاثیر گذار باشد و کاهش دهنده آن باشد.

به طور مثال 2 متد را 100000 بار اجرا کنید و در کی عملیات boxing را برای تخصیص مقدار یک متغییر به دیگری انجام دهید و در دیگری به طور ساده یک مقدار را از نوع value به Value دیگر تخصیص داده مدت زمان انجام عملیات متفاوت خواهد بود.

بهتر است انجام این نوع کار ها پرهیز شود. چون کاهش عملکرد دارد و تخصیص حافظه بالا چون در unbox حافظه heap و gc درگیر می شوند و همچنین unboxing نیازمند explicit case است که ممکن است exception دهد

نکته : در گذشته arrayList فقط object ذخیره می کردند که نیازمند عملیات boxing برای سایر نوع داده بوده اما اکنون generic وجود دارد از box پرهیز شده و نوع داده مستقیما در کالکشن ذخیره می شود.

# Object Oriented Programming (OOPs) in C#

یک متودولوژی تولید نرم افزار است که شامل تعداد قاعده است که باید رعایت شوند. امروزه بیشتر زبان ها از آن پیروی می کنند. بیشتر به طراحی نرم افزار مرتبط است طراحی داخلی نرم افزار و به توسعه دهنده ها مرتبط است. قابلیت سیستماتیک و مهندسی کردن فرآیند تولید نرم افزار را می دهد

برنامه نویسی شی گرا وابسته به دید ما نسبت به سیتم است.

Modular programing problems

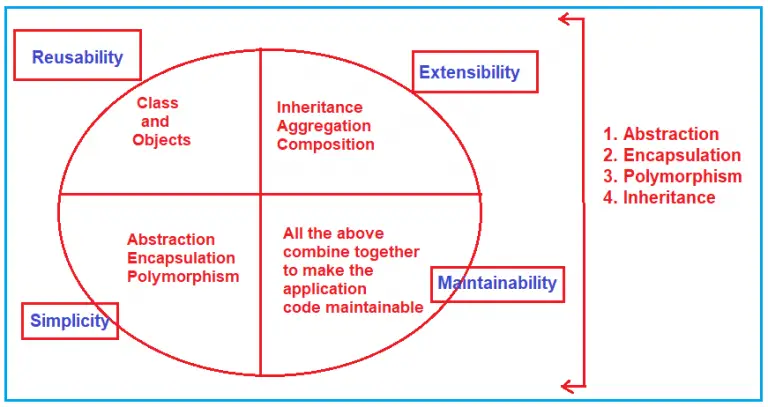
استفاده مجدد

نگه داری از کد

عدم سادگی

عدم گسترش ویژگی ها متد ها

Oop is design approach



Reusablility : با استفاده از class می توان از استفاده مجدد استفاده کرد.

**Extensibility : با استفاده از ingeritance aggregation , composition می توان ویژگی ها را گسترش داد**

**Simplicity : با استفاده از ویژگی های abstraction encapsulation polymorphism می توان کد را ساده کرد.**

**Maintanablility در نهایت با استفاده از ویژگی ها قبلی نگه داری ساده تر می شود.**

##### **What are the OOPs Principles or OOPs Concepts in C#?**

Abstraction , encapsulation , polymorphism , inheritance

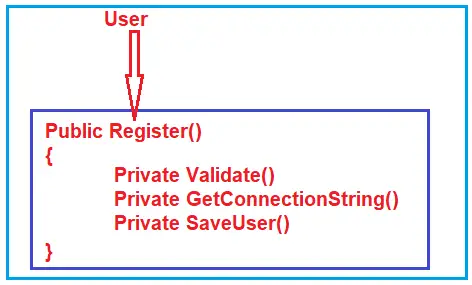
##### **What are Abstraction and Encapsulation?**

Abstraction : ارائه دهی ویژگی ها بدون این که از backgroundقضیه اطلاعی داشته باشیم . یعنی کلاسی داشته باشیم که فقط جزئیات ضروری را به بیرون می دهد و مابقی از مخفی می کند.

Encapsulation : قرار گرفتن داده ها و متد ها در یک unit را می گویند. به زبان ساده تعریف کلاس و مخفی کردن data member ها از بیرون از کلاس و دسترسی به آن ها فقط از داخل کلاس.

Abstraction , encapsulation به هم مرتبط هستند abstract در واقع فکر منطق است و encapsulation پیاده سازی فیزیکی است.

Example



##### **What is Inheritance?**

قابلیت استفاده از data member , function ,.. مربوط به یک کلاس base در کلاس sub را ارث بری گویند.

##### **What is Polymorphism?**

چند ریختی به این معنی که یک متد رفتاری متفاوت از خود نشان دهد با گرفتن نوع های داده مختلف

Compile time / early binding / overoading

Run time / late binding / ovderriding

# Class and Objects in C#

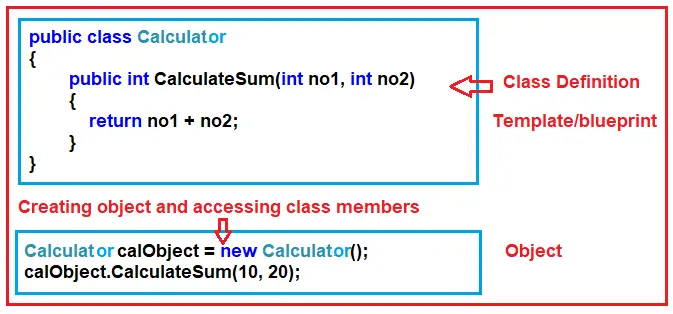
نکته : class در واقع تعریف است و object یک نمونه است.

Bmw is car Toyota is car both of them are objects of class car

Class in c#

نوع داده ای تعریف شده توسط کاربر است که ارائه دهنده state , behavioral است با استفاده از properties و action ها.

Object در واقع یک instance از کلاس است.



Class is a template or blueprint and the object is instance of it.

Type of class in c#

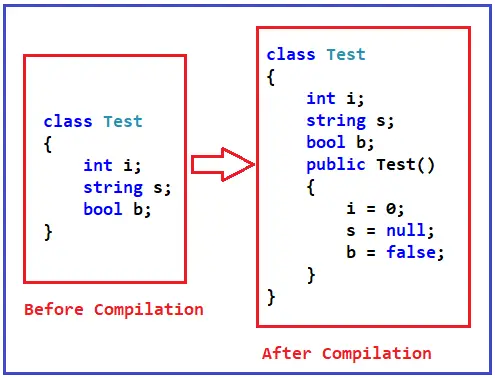
# Constructors in C#

متدی هم نام با اسم کلاس است که برای initialize کردن variable های موجود در سطح کلاس استفاده می شود.

متدی است که با هر بار instance ساختن از کلاس فراخوانی می شود

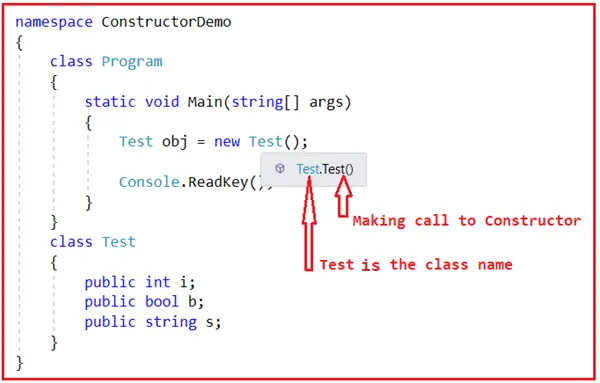
Constructor فاقد هرگونه return است. و متدی هم نام با ctor نمی توانیم داشته باشیم.

نکته : هر کلاسی که instance می سازیم باید دارای constructor باشد. اما در صورتی که constructor را برای کلاس قرار ندهیم نیز امکان ساخت instance وجود دارد چون compiler به صورت خودکار ctor را در صورتی که ما ننویسیم می سازد و متغییر ها را init می کند. Ctor که compiler می سازد implicit ctor یا default نام دارد و ما بسازیم explicit ctor است.



نکته :implicit ctor به صورت public است.

نکته : زمانی که **instance** می سازیم **ctor** **صدا زده می شود**. به ازای هر بار instance ساختن متد ctor فراخوانی می شود.



# Types of Constructors in C#

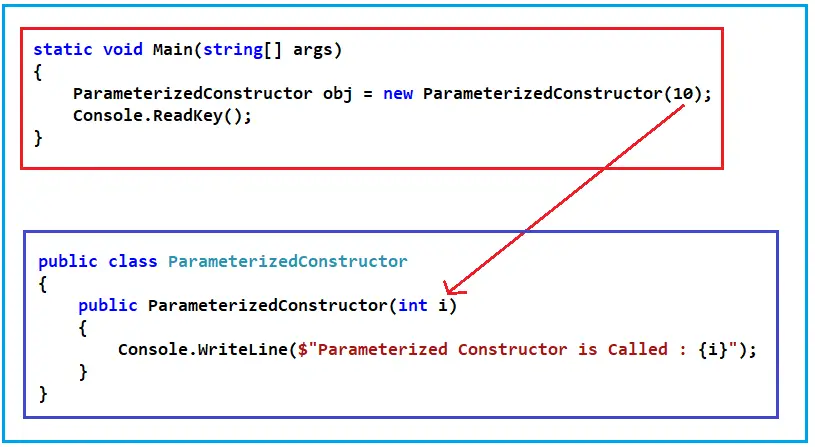
1. **Default or Parameter Less Constructor**
2. **Parameterized Constructor**
3. **Copy Constructor**
4. **Static Constructor**
5. **Private Constructor**

##### **Default or Parameterless Constructors in C#**

اگر ctor فاقد پارامتر باشد default است. 2 نوع است یا کاربر تعریف می کند یا سیستم.

##### **What is Parameterized Constructor in C#?**

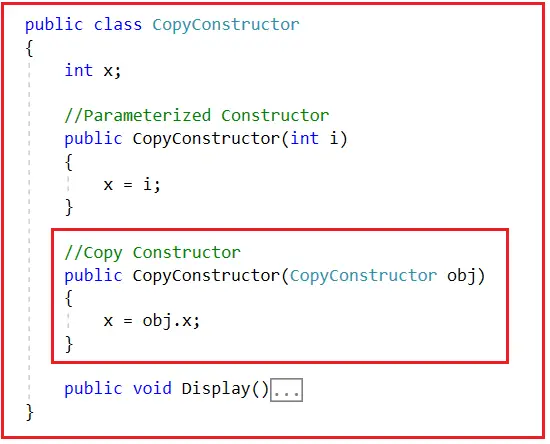
اگر ctor ما دارای parameter باشد ما آن را parameterized ctor می نامیم که توسط برنامه نویس تعریف می شوند

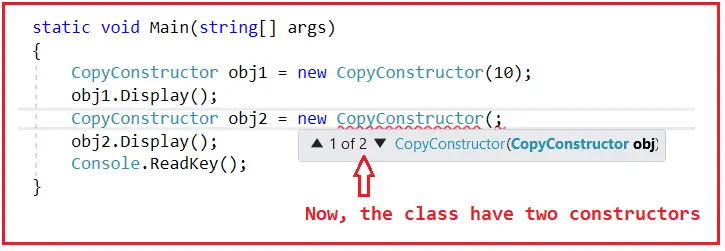


##### **Copy Constructor in C#**

اگر بخواهیم چندین instance با مقدار های یکسان ایجاد کنیم باید از copy ctor استفاده کنیم در این حالت ctor ما به عنوان پارامتر کلاسی مشابه خود را دریافت می کند.

به طور مثال شرایطی را فرض کنید که ctor شما 10 پارامتر داشته باشد.





##### **How many Constructors can be Defined in a Class in C#?**

هر تعداد با signature متقاوت می توان ایجاد کرد.

Static Constructor

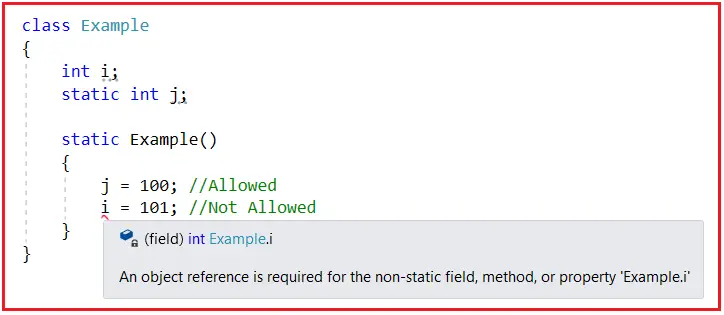
Ctor که دارای کلمه کلیدی static است و فاقد هر گونه access modifier است زمانی که از این نوع ctor داریم به صورت پیشفرض compler برای کلاس ما implicit ctor را هم قرار می دهد.

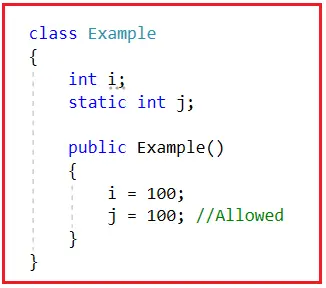
نکته : زمانی که در کلاس Staci field داریم در این حالت compiler به صورت implicitly سازنده static را قرار می دهد.

نکته : static ctor برای init کردن static variable ها کاربرد دارد و هیچ وقت explicitly فراخوانی نمی شوند بلکه implicitly فراخوانی می شوند این نوع ctor ها اول از همه صدا زده می شوند.

نکته : static ctor is not parameterized اما چرا ؟ چون این نوع به صورت implicitly فراخوانی می شوند و ما به آن دسترسی نداریم.

نکته : در هر کلاس فقط یک عدد داریم فاقد قابلیت overload است. و فاقد پارامتر و فقط امکان مقدار دهی static field را دارد. اما در ctor های non static قابلیت مقدار دهی static field وجود دارد.





Private Constructor in c#

در سی شارپ امکان تعریف ctor به صورت private نیز وجود دارد در این حالت امکان ساخت object از بیرون از کلاس وجود ندارد و فقط می توان در داخل خود کلاس از آن object ساخت

معمولا برای singleton design pattern استفاده می شود.

در حالتی که کلاس فقط شامل static member است کاربرد دارد. و اجازه ساخت object از بیرون ندارد.

# Why We Need Constructors in C#

هر کلاسی که میخواهیم instance از آن بسازیم به ctor نیاز دارد. زمانی که بخواهیم فیلد های کلاس را از بیرون مقدار دهی کنیم در هنگام instance ساختن.

# Static vs Non-Static Constructors in C#

اگر در تعریف ctor دارای کلمه کلیدی static باشد از نوع static است.

Static فقط برای init کردن field and variable of static type است. و non static برای غیر static ها کاربرد دارد.

Static constructors are implicitly called whereas non-static constructors are explicitly called.

Static ctor فقط یک بار بلافاصله بعد از اجرای کلاس فراخوانی می شود اما non static ها بعد از ساخت instance فراخوانی می شوند.

نکته : static ctor executed only once whras the non static ctor are executed 0 or n number of time depending on the number of the instance we created for the class.

وقتی کلاسی فیلدی از نوع Static دارد به صورت ضمنی سازنده آن فیلد static را مقدار دهی می کند بدون نیاز به ساخت instance از کلاس پس سازنده non static اصلا فراخوانی نمی شود در حالی که static یکبار فراخوانی شده.

**class** Program

**{**

//Main Method is the Entry Point for our Application Execution

**static** **void** Main**(string[]** args**)**

**{**

Console.WriteLine**(**"Main Method Started"**)**;

//As soon as it finds ConstructorsDemo.x,

//it will first execute the static constructor of the class

Console.WriteLine**(**ConstructorsDemo.x**)**;

Console.WriteLine**(**"Main Method Completed"**)**;

Console.ReadKey**()**;

**}**

**}**

**public** **class** ConstructorsDemo

**{**

**public** **static** **int** x; //It is going to be initialized by static constructor

**public** **int** y; //It is going to be initialized by non-static constructor

//Static Constructor

**static** ConstructorsDemo**()**

**{**

//This constructor initialized the static variable x with default value i.e. 0

Console.WriteLine**(**"Static Constructor is Called"**)**;

**}**

//Non-Static Constructor

**public** ConstructorsDemo**()**

**{**

//This constructor initialized the static variable y with default value i.e. 0

Console.WriteLine**(**"Non-Static Constructor is Called"**)**;

**}**

**}**

نکته : زمانی که از کلاسی که داری static ctor است چند instance بسازیم فقط یک بار static ctor آن فراخوانی می شود در حالی non static ها به تعداد instance فراخوانی می شوند

نکته : زمانی که یک متغییر static یا متد static را از یک class فراخوانی می کنیم ابتدا ctor static آن کلاس فراخوانی می شود

# Private Constructors in C#

اگر کلاسی دارای private ctor باشد امکان ساخت object از کلاس بیرون از کلاس وجود ندارد. مگر اینکه دارای ctor از نوع public هم باشد. در حالت اول فقط داخل کلاس میتوان object ساخت.

در حالتی که کلاس دارای ctor از نوع public , private باشد می توان از آن object ساخت اما باید این نکته را به خاطر سپرد که فقط ctor که از نوع public است فراخوانی می شود به هنگام ساخت instance

اگر فقط دارای private باشیم و بخواهیم خارج از کلاس instance بسازیم به خطای compile time بر می خوریم.

نکته : اگر کلاسی دارای private ctor باشد نمی تواند در inheritance شرکت کند فقط در صورتی می تواند که دارای یک public ctor هم باشد. را حل دیگر کلاس child در داخل کلاس parent باشد.

Sealed

اگر می خواهید به هیج وجه از یک کلاس ارث بری نشود باید آن را به صورت sealed تعریف کرد.

نکته : اگر کلاسی دارید که تمامی data member های آن به صورت static هستند می توانید برای جلوگیری از ارث بری و instance ساختن از آن private ctor تعریف کنید اما راه حل بهتر این است که static class تعریف کنید که هم از ارث بری و هم از instance ساختن جلوگیری می کند.

**public** **static** **class** Test

**{**

**public** **static** **double** PI = 3.14;

**public** **static** **int** GetSquare**(int** x**)**

**{**

**return** x \* x;

**}**

**}**

//A class cannot Derive from a Static Class

**public** **class** Child: Test

**{**

**}**

##### **Is Private Constructor going to be Parameterless in C#?**

نه لزوما اینطور نیست چون که امکان ساخت instace در صورتی که کلاس فرزند باشد وجود دارد همچنین در سی شارپ امکان ctor overloading وجود دارد چون در سربارگذاری متد access modifier جزو signature متد نیست.

Private ctor معمولا در singleton کاربرد دارد. چون فقط یک instance می سازد.

# Destructors in C#

برای هر گونه پاک سازی نهایی زمانی که یک instance از کلاس توسط garbage collector جمع آوری می شود کاربرد دارد.

نام مشابه ای با نام کلاس دارد با ~ prefix tilde

Destructor به صورت implicitly صدا زده می شود زمانی که class object is destroyed

Destructor فاقد parameter و overload است.

Destructor به صورت پیشفرض call می شود توسط gc

##### **When will the Object of a Class get Destroyed in C#?**

* در پایان اجرای برنامه هر object مرتبط با برنامه توسط gc نابود می شود destroyed
* بعضی مواقع در حین اجرای برنامه رخ می دهد implicitlyمثلا حافظه پر شده باشد gc تمامی object های بدون استفاده را ازبین می برد.
* به صورت explicitly در حین اجرای برنامه فراخوانی می شود با استفاده از دستور Collect که اشیای بدون استفاده را ازبین می برد.

نکته مهم : نمی توان پیش بینی کرد که چه زمانی destructor صدا زده می شود چون به صورت implicitly توسط gc فراخوانی می شود.

مثال

**class** DestructorDemo

**{**

**public** DestructorDemo**()**

**{**

Console.WriteLine**(**"Constructor Object Created"**)**;

**}**

~DestructorDemo**()**

**{**

**string** type = GetType**()**.Name;

Console.WriteLine**(**$"Object {type} is Destroyed"**)**;

**}**

**}**

**class** Program

**{**

**static** **void** Main**(string[]** args**)**

**{**

DestructorDemo obj1 = new DestructorDemo**()**;

DestructorDemo obj2 = new DestructorDemo**()**;

//Making obj1 for Garbage Collection

obj1 = **null**;

Console.ReadKey**()**;

**}**

**}**

در مثال بالا کد موجود در destructor به دلیل بیان شده در کنسول چاپ نمی شود.راه حل فراخوانی دستی gc !

**static** **void** Main**(string[]** args**)**

DestructorDemo obj1 = new DestructorDemo**()**;

DestructorDemo obj2 = new DestructorDemo**()**;

//Making obj1 for Garbage Collection

obj1 = **null**;

GC.Collect**()**;

Console.ReadKey**()**;

نکته اگر در مثال بالا هر دو object را برابر null می کردیم gc 2 بار فراخوانی می شد برای هر object و 2 بار نیز destructor صدا زده می شد.

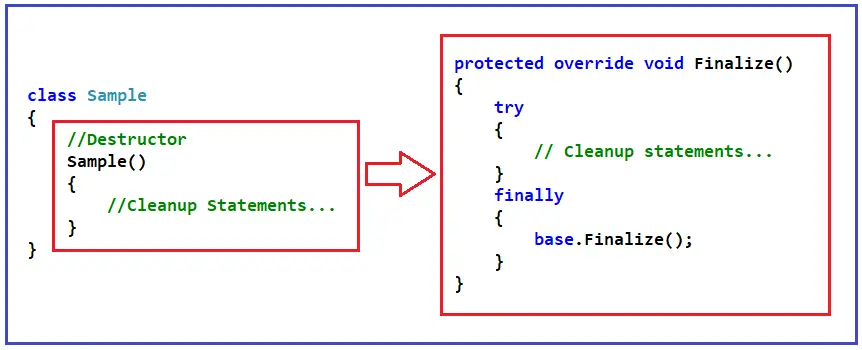
نکته : destructor فقط برای class کاربرد دارد و نوع های دیگر مانند struct

نکته : روش فراخوانی destructor استفاده از GC.Collect() است.

نکته : فاقد هرگونه modifires است

نام دیگر آن Finalizer است.

اتفاقی که در پشت پرده می افتد به شرح زیر است.



نکته : در حالت inheritance از آخرین کلاس شروع به destruct کردن و فراخونی آن می کند.

**نکته:** هیچ وقت بلوک خالی destructor را صدا صدا نزنید چون در صفی که gc برای finalize ایجاد کرده است قرار می گیرد و یک به یک آن ها gc فراخوانی می کند از آن جایی که کاری انجام نمی دهد باعث از دست رفتن منابع و کاهش عملکرد می شود.

نکته : بهتر است از Gc.Collect استفاده نشود چون سبب مشکل کارایی می شود. و خودش به صورت خودکار object های ار که بلااستفاده هستند را collect می کند.

##### **When to use Destructor in C#?**

زمانی که اپلیکیشن ما از منابع مدیریت نشده ماننده io , http connection , files استفاده می کند ما باید از destructor برای خالی کردن حافظه استفاده کنیم.

##### **Explicit Release of Resources using Dispose Pattern in C#**

اگر اپلیکیشن شما از منابع خارجی گرانبها استفاده می کند پیشنهاد می شود که به صورت explicitly حافطه را آزاد کنید یعنی قبل از gc را حل آن پیاده سازی IDisposable interface است. متد Dispose آن را implement کنید.

# Differences Between Finalize and Dispose in C#

Finilize و Dispose برای ازاد سازی منابع کاربرد دارد و هرکدام هدف متفاوتی دارد و در سناریو خود کاربرد دارد.

##### **Finalize Method in C#:**

هدف: برای پاک کردن عملیات ها قبل از این که یک object توسط gc جمع آوری شود. معمولا override می شود برای رهاسازی منابع مدیریت نشده که یک object استفاده می کند. Gc به طور خودکار متد finalize را صدا می زند.

کنترل: به صورت خودکار صدا زده می شود و نه توسط ما. زمان صدا زدن آن مشخص نیست و توسط gc

ارث بری: از کلاس object ارث بری کرده است باید هر موقع که override می شود از کلاس base هم صدا زده شود تا تمامی منابع را حذف کند.

##### **Dispose Method:**

هدف : از رابط IDisposable استفاده کرده است و برای رها سازی حافظه managed , unmanaged کاربرد دارد.

کنترل: برخلاف finilaze این متد به صورت explicitly فراخوانی می شود زمانی که کار ما با object تمام شده برای رها سازی حافظه.

قابل پیش بینی است یعنی شما می دانید چه زمانی فراخوانی کردید و منابع را آزاد می کند.

الگو : پیروی از الگوی dispose معمول است که شامل فراخوان finalizer (GC.SupppressFinalize(this)) تا اگر dispose قبلا فراخوانی شده باشد gc نتواند finalize را فراخوانی کند.

نکته : Finalize عموما برای آزاد کردن unmanaged resource استفاده می شود اما dispose برای هردو کاربرد دارد.

نکته : زمانی که از dispose استفاده می کنید از الگوی اطمینان از عدم فراخوانی gc finalize استفاده کنید.

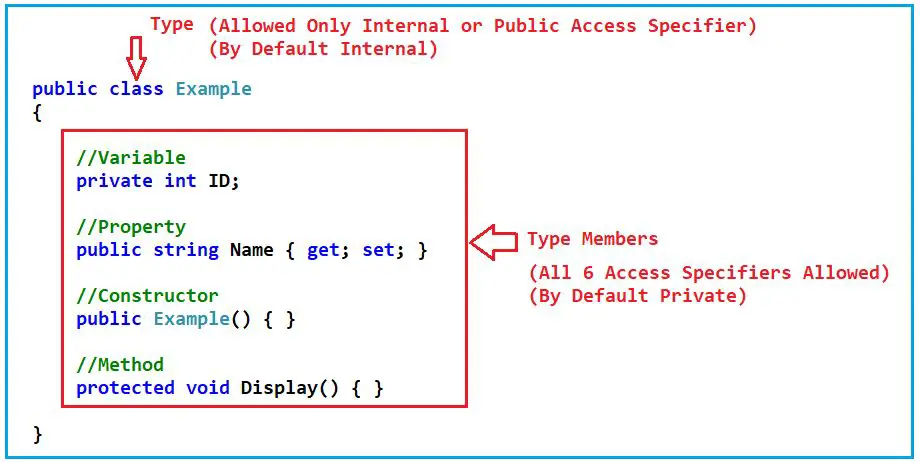
## **Access Specifiers in C#**

هر کلمه کلید مثل public private abstract partial sealed , … را به عنوان Modifires می شناسیم.

Access Specifiers  به عنوان بیان کننده سطح دسترسی یا چه کسانی آن را ببینند کاربرد دارد. مثلا برای کلاس و اعضای آن کاربرد دارد.

1. **Private**
2. **Public**
3. **Protected**
4. **Internal**
5. **Protected Internal**
6. **Private Protected**

Type vs TypeMember



Type member می تواند دارای 6 سطح دسترسی باشد در حالی که type میتواند دارای 2 سطح دسترسی internal , public باشد.

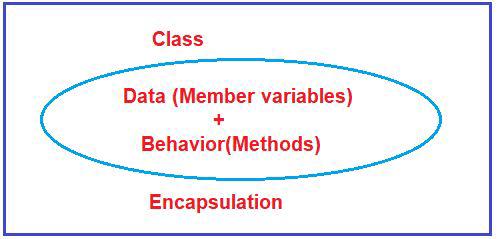
اگر برای type سطح دسترسی را set نکنیم به صورت پیش فرض type از نوع internal و member از نوع private خواهند بود.

Assembly : براساس مستندات ماکروسافت بلوک های از .net application هستند. در واقع precompiled از .net code است که توسط CLR اجرا می شود

# Encapsulation in C#

مخفی کردن state های داخلی و functionality یک object و فقط اجازه دسترسی به public functions ها.

یه زبان دیگر قرار دادن state , behavior مثل Data member , fucntions در یک unit را می گویند که تضمین کننده این است که state , behavior از unit دیگر قابل دسترسی نیست.



به طور مشابه یک کپسول که هر جیزی که لازم دارد را در خود دارد و از بیرون چیزی دریافت نمی کند.

مهم ترین مزیت encapsulation در واقع data hiding است. که هدف آن جلوگیری از سواستفاده است.

##### **What is Data Hiding in C#?**

به معنی مخفی کردن داده های داخلی از دنیای بیرون است. که data encapsulation نیز نام دارد.

How to implement in c#?

تعریف اعضای داده ای به صورت private و دسترسی دادن به آن ها از طریق setter , getter

Advantages of providing setter and getter?

از مزایای آن می توان به validation مقدار کاربر قبل از قرار گرفتن آن در در variable باشیم. که سبب افزایش امنیت داده نیز میشود.

نکته : بهترین راه حل برای پیاده سازی data hiding استفاده از Properties است.

**public** **class** Bank

**{**

**private** **double** \_Amount;

**public** **double** Amount

**{**

**get**

**{**

**return** \_Amount;

**}**

**set**

**{**

// Validate the value before storing it in the \_Amount variable

**if** **(value** **<** 0**)**

**{**

**throw** new Exception**(**"Please Pass a Positive Value"**)**;

**}**

**else**

**{**

\_Amount = **value**;

**}**

**}**

**}**

**}**

# Abstraction in C#

این اطمینان را می دهد که سایر قواعد oop در کنار هم کار میکنند و هدف نهایی را ارائه می دهند.

ارائه ویژگی ها بدون اطلاع از background و جزئیات آن ها را abstraction می گویند. به زبان دیگر تعریف کلاس و ارائه جزئیات ضروری به همراه مخفی کردن جزئیات پیاده سازی ار می گویند. یعنی هر چیزی که غیر ضروری است را مخفی کنید و فقط موارد ضروری را به دنیای بیرون ارئه دهید.

به طور مثال وقتی در سایتی لاگین می کند و سپس وارد می شود آیا اطلاعی از اتفاق های افتاده در پشت ماجرا دارید. یا عدم اطلاع از نحوه ارسال شدن پیام کوتاه.

پس به این معنی است که جزئیات استفاده از service , function را می دهیم و خبری از جزئیات پشت آن ندهیم.

The problem

اگر جزئیات را مخفی نکنیم کاربر ما به سادگی می تواند مستقیما به جزئیات پیاده سازی ما دسترسی داشته باشد.

##### **How to Implement Abstraction Principle in C#?**

در مباحث بعدی آن ها توضیح خواهیم داد اما کافی است بدانید که کاربرد آن ها این است که مکانیسمی ارئه می دهند برای مخفی سازی جزئیات پیاده سازی و سرویس را به بیرون ارائه می دهند. و کاربر فقط می داند که سرویس های به بیرون ارائه می شوند و جزئیاتی از پیاده سازی آن ها نمی داند.

نکته : abstract ها می تواند یک یا تمام متد ها در خود implement کنند. یعنی می تواند یک پیاده سازی base داشته باشد.

Abstraction => detail hiding

Encapsulation => data hiding

Advantages of abstraction

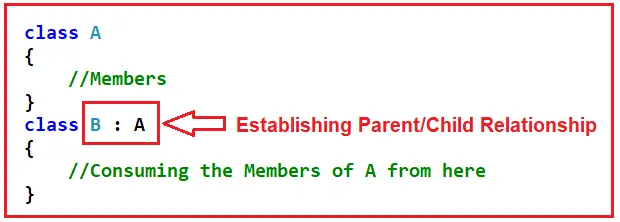
مخفی کردن جزئیات پیچیده از دید کاربر و فقط نمایش method signature

بالا بردن امنیت اپلیکیشن

بهبود آسان زیرا بدون اطلاع کاربر نهایی می توان هر تغییری در سیستم خود داشته باشیم.

# Inheritance in C#

مکانیسمی برای استفاده از اعضای تعریف شده یک کلاس در کلاس دیگر است. برقرار کننده رابطه parent , child است. یعنی تمامی ویژگی ها والد به فرزند می رسد.



اعضای private به فرزند ارث نمی رسند.

6 Rules of inheritance

Rule1

در سی شارپ ctor کلاس parent باید برای کلاس فرزند قابل دسترسی باشد به دلیل این که اول ctor کلاس پدر فراخوانی می شود. اگر سطح دسترسی کلاس و ctor را بیان نکنید به صورت private در نظر می گیرد. اما دلیل این که اول کلاس parent فراخوانی می شود این است که باید در ابتدا data member های کلاس پدر init شود تا در کلاس فرزند قابل استفاده باشند.

Rule2

در ارث بری کلاس پدر نمی تواند از اعضای کلاس فرزند خود استفاده کند.

Rule3

با ایجاد نمونه از کلاس فرزند و قرار دادن آن در متغییر از کلاس پدر می توان به اعضای پدر هم دسترسی داشت اگر instance نسازیم قابل دسترسی نیستند از طریق متغییر عادی

**static** **void** Main**()**

**{**

A p; //p is a variable of class A

B q = new B**()**; //q is an instance of Class B

//We can initialize a Parent class variable using child class instance as follows

p = q; //now, p is a reference of parent class created by using child class instance

//Now you can call members of A class as follows

p.Method1**()**;

p.Method2**()**;

یکی از دلایل این است که تا زمانی که new نکنید هیچ حافظه ای تخصیص داده نمی شود. بعد از instance ساختن.

Rule4

هر کلاسی که توسط ما یا توسط کتابخانه های .net ایجاد شده است دارای default parent است کلاس object .

نکته : در زمان compile بررسی می شود که آیا کلاس شما زا کلاسی دیگر ارث برده است یا نه اگر برده باشد که هیچ در غیر این صورت از کلاس object ارث می برد. دلیل آن این است که کلاس که از آن ارث برده خود از object ارث برده است.

Rule5

در سی شارپ فقط می توان از یک کلاس ارث بری کرد در آن واحد.

Rule6

در صورتی که کلاس پدر ctor parameterLess باشد به صورت implicitly فراخوانی می شود اما اگر parameterized باشد باید به صورت explicitly فراخوانی شود و مقادیر پارامتر به آن ارسال شوند.

**class** B : A

**{**

**public** B**(int** num**)** : **base(**num**)**

**{**

Console.WriteLine**(**"Class B Constructor is Called"**)**;

**}**

**public** **void** Method3**()**

**{**

Console.WriteLine**(**"Method 3"**)**;

**}**

**static** **void** Main**()**

**{**

B obj1 = new B**(**10**)**;

B obj2 = new B**(**20**)**;

B obj3 = new B**(**30**)**;

Console.ReadKey**()**;

**}**

**}**

مزیت مهم ارث بری استفاده مجدد از کد است.

# Types of Inheritance in C#

تعداد کلاس های پدر که یک کلاس فرزند دارد را می گویند.

In c ++

1. Single Inheritance
2. Multi-Level Inheritance
3. Hierarchical Inheritance
4. Hybrid Inheritance
5. Multiple Inheritance

##### **Single Inheritance**

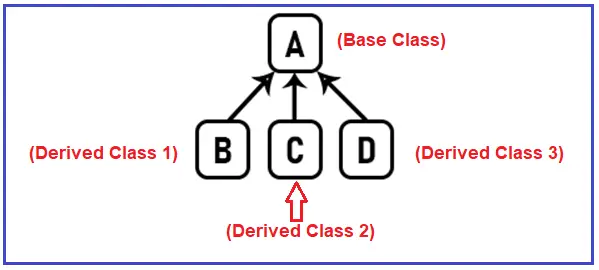
زمانی که یک کلاس به طور مستقیم تنها از یک کلاس دیگر ارث برده باشد.

##### **Multilevel Inheritance**

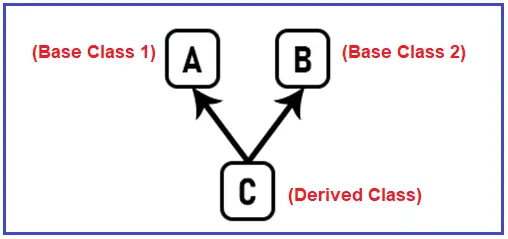
زمانی که کلاس a از کلاس b ارث برده و کلاس b از کلاس c ارث برده باشد ارث بری چندگانه اتفاق افتاده است.

##### **Hierarchical Inheritance**

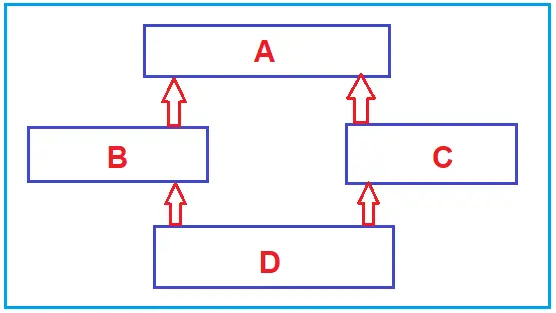
زمانی که بیش از یک کلاس از یک کلاس base ارث برده باشد.



##### **Multiple Inheritance**



##### **Hybrid Inheritance**



ارث بری های مجاز در سی شارپ

1. **Single Inheritance (Single, Multilevel, and Hierarchical)**

در سی شارپ پیاده سازی چندین interface توسط یک کلاس مجاز است اما یک کلاس نمی تواند از چند کلاس به طور همزمان ارث بری کند برعکس زبان c ++

# IsA and HasA Relationship in C#

یک مفهوم پایه ای در رابطه را مبحث ارثبری است.

##### **IsA Relationship (Inheritance):**

همان رابطه ای ارث بری است که به صورت hierarchical بین کلاس base, child وجود دارد در واقع

**The drive class is a specialized version of base class**

##### **HasA Relationship (Composition)**

نشان دهنده ترکیب است. بیان گر این است که یک object از یک کلاس شامل یا ترکیبی از object های کلاس دیگری است.

**Is a sample**

The class **Cuboid** is inherited from the **Rectangle** class. So, can we say that a **Cuboid IS A Rectangle?** **Yes**. So, the relationship between the Rectangle and Cuboid classes is the **IS-A** relationship. So, the point you need to remember is when we are making the Inheritance or Parent-Child relationship, we can say that relationship is an IS-A relationship in object-oriented programming languages.

**Has a sample**

Next, our Table class has a top variable that is of the Rectangular type. That means the **Table** class has an object of the Rectangle class. So, can we say that the **Table class HAS A Rectangle? Yes**, the Table class Has a Rectangle. So, the relationship between the Table and the Rectangle class is the **Has A** relationship.

به زبان ساده می توان گغت که ارث بری عادی بیان گر رابطه ی is a است به این معنی که اگر کلاس base با نام shape داشته باشیم و کلاس فرزندی با نام rectangle داشته باشیم و در صورتی رابطه parent , child بین این 2 برقرار باشد می توان گفت که rectangle is a shapr

اما حالت دوم بیان گر ترکیب استفاده از کلاس های مختلف در یک کلاس است در حالتی که می خواهیم encapsulation را رعایت کنیم و قرار است ویژگی های کلاس دیگری را استفاده کنیم نه این که کلاس خود را گسترش دهیم در این حالت اگر شی از کلاس مورد نیاز خود بسازیم در حال پیاده سازی رابطه ی has a هستیم.

# Abstract Class and Abstract Methods in C#

##### **What are Abstract Methods in C#?**

متد های هستند که در یک abstract class یا یک interface تعریف می شوند که فاقد پیاده سازی هستند و وظیفه ی پیاده سازی آن متد ها به عهده ی هر drived class از آن abstract class , interface است.

فاقد پیاده سازی یعنی صرفا signature و بدون curly brace و دارای semicolon

##### **What are Abstract Classes in C#?**

کلاس های ارائه دهنده blue print یا طرح آماده هستند که قابلیت instance ساختن ندارند و باید توسط کلاسی که از آن drived شده پیاده سازی شود. متد های abstract را در کلاس abstract تعریف می کنیم

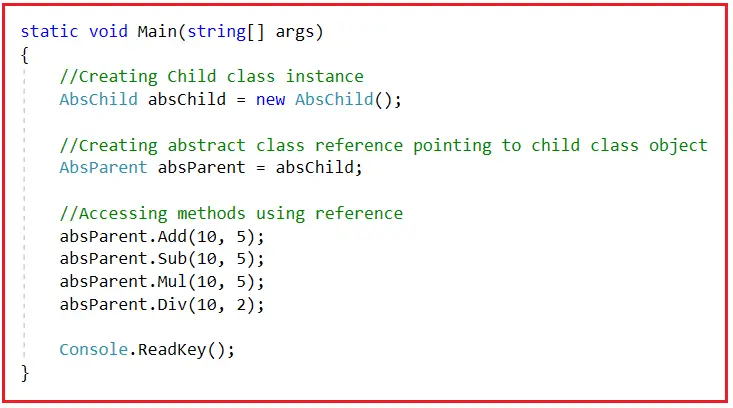
##### **What is the use of the Abstract Method in C#?**

زمانی که کلاس پدر abstract است ویژگی به کلاس child اضافه نمی کند برای مصرف بلکه محدودیت های را به کلاس فرزند می دهد که باید آن ها را پرکند.

**نکته:** کلاس abstract می تواند هم شامل متد های non abstract باشد و هم شامل متد های abstract باشد. و فقط میتوان از متد های non abstract پدر مستقیم استفاده کرد و کلاس فرزند هم می تواند abstract باشد. چون بدنه ندارد نمی توان از آن instance ساخت. می تواند اعضای static هم داشته باشد.

##### **Can we Create a Reference for the Abstract Class in C#?**

بله می توان برای کلاس abstract یک reference ایجاد کرد به کمک کلاس فرزند مانند interface که به کلاس پیاده سازی کننده خود می تواند تغییر شکل دهد. در این حالت کلاس پدر نمی تواند متد های کلاس فرزند را فراخوانی کند. بلک override شده ها را فقط می تواند استفاده کند



# Method Overloading in C#

از نوع compile time است static است. رویکردی برای تعریف چندین متد با نام یکسان در یک کلاس است که دارای param های متفاوت هستند.

قابلیت متد overloading در کلاس parent , child نیز وجود دارد. یعنی متدی هم نام در کلاس فرزند و پدر overload می شود.

مهم تفاوت signature متد است. که compiler تشخیص می دهد. امضای متد شامل ترتیب نوع تعداد پارامتر های است.

نکته مهم : signature متد شامل return type و param modifiers نمی شود. که اکر نام متد یکسان باشد اما return type متفاوت داشته باشد overloading محسوب نمی شود.

به طور مثال فرض کنید 2 متد با نام یکسان و param یکسان دارید اما return type آن ها متفاوت است موقع فراخوانی compiler نمی تواند که تشخیص دهد کدام را فراخوانی کرده اید.

پس در ازای ورودی های مختلف دارای خروجی های مختلفی نیز خواهیم بود.

نکته : متد ها را در همان کلاس خود می توان overload کرد اما نمی توان override انجام داد.

نکته : متد overloading را می توان در سطح ctor هم انجام داد.

# Operator Overloading in C#

# در سی شارپ این امکان پذیر است که کاربران برای user define data type هار operator ها را overload کنند

# هدف این است که برای operator مثل + برای یک نوع داده خاصی معنی جدیدی ایجاد کنیم.

# در کلاس یا struct مدنظر خود oprator را باید overload کنیم.

# 

# Op برابر است با مقداری مثل + یا –

# 

# فرض کنید که می خواهیم مقدار فیلد های 2 کلاس را با هم جمع کنیم 2 راه داریم یا متدی به نام add بنویسیم یا این که عملگر + را برای آن نوع داده overload کنیم.

# 

# با توجه به مثال بالا می توان دریافت که عملیات انجام شده در operator overloading مشابه همان add function است فقط با تفاوت جزئی در syntax متد.

# Method Overriding in C#

# روندی است که در آن یک متد کلاس parent را که نه private ,sealed , static است را در کلاس child بازنویسی می کنیم با signature یکسان در کلاس parent آن.

##### **When do we need to override a method in C#?**

# زمانی که در کلاس child نیاز به منطق خاصی برای یک متد در کلاس parent داریم که پیاده سازی که در کلاس پدر انجام شده نیازمندی ما را رفع نمی کند بنابراین ما در کلاس فرزند با override کردن متد نیازمندی را طبق بیزنس خود رفع کرده و مجددا پیاده سازی می کنیم.

# نکته : برای override کردن متد باید متد کلاس پدر به صورت virtual تعریف شده باشد. فقط کلاس های فرزند می توانند متد virtual را override کنند.

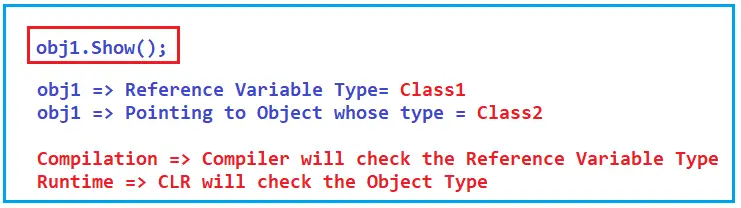
# نکته مهم : کلاس parent می تواند reference variable از کلاس child را در خود نگه داری کند. اما برعکسش امکان پذیر نیست.

# 

# به تصویر زیر بادقت نگاه کنید.

# 

در رابطه با نحوه compile و runtime اجرای متد overrideing در ادامه مباحثی مطرح می شود با توجه به مثال بالا.



در زمان compile متد فراخونی شده به محل تعریف خود bind می شود یعنی compiler ارتباطی بین محل invoke و defition متد برقرار می کند

در زمان compile بررسی می شود که در type متغییر ساخته شده متد فراخوانی شده وجود دارد یا نه اگر وجود نداشته باشد خطای زمان compile می دهد.

در زمان اجرای برنامه runtime زبان CLR بررسی می کند که متد را از اجرا کند از reference type object اگر در آن وجود نداشت سعی می کند که از طریق object type متد را اجرا کند. بنابراین به این فرآیند می گوییم dynamic or runtime polymorphism

##### **What is Dynamic Polymorphism or Runtime Polymorphism?**

نکته: اگر متدی در موقع اجرا در کلاسی متفاوت از کلاسی که به آن در زمان کامپایل bound شده اجرا شود به این فرایند dynamic polymorphism می گویند.

##### **What is Static Polymorphism or Compile Time Polymorphism?**

اگر متد در زمان اجرا به همان کلاسی که در زمان کامپایل bound شده اجرا شود را می گوییم.

##### **How can we execute the superclass method if it is overridden in the sub-class in C#?**

2 راهکار برای انجام این مورد وجود دارد اول استفاده از کلمه کلیدی base برای فراخوانی متد در کلاس پدر و راهکار دوم ایجاد یک object از کلاس پدر و فراخوانی متد از این طریق.

نکته : امکان overloading متد های کلاس parent در کلاس child وجود دارد.

# Method Hiding in C#

به معنی پیاده سازی مجدد متد های کلاس پدر با همان signature که در کلاس پدر وجود دارد حتی آن های که virtual نیستند.

How To implement method hiding in c#?

در کلاس فرزند با همان امضای کلاس پدر متد را بازنویسی کرده حتی آن های را که virtual نیستند و از کلمه کلیدی new بعد از access modifier استفاده کرده برای جلوگیری از اخطار compiler

می توان از کلمه کلیدی new نیز استفاده نکرد.

تفاوت override و hiding در این است که با override فقط متد های virtual را می توان بازنویسی کرد ولی با hiding می توان تمامی متد ها را بازنویسی کرد. With permission and without permission

به هنگام فراخوانی نیز متد hide شده استفاده می شود.

نکته به کد زیر توجه کنید.

**namespace** *MethodHiding*

**{**

**public** **class** Parent

**{**

**public** **virtual** **void** Show**()**

**{**

Console.WriteLine**(**"Parent Class Show Method"**)**;

**}**

**public** **void** Display**()**

**{**

Console.WriteLine**(**"Parent Class Display Method"**)**;

**}**

**}**

**public** **class** Child : Parent

**{**

//Method Overriding

**public** **override** **void** Show**()**

**{**

Console.WriteLine**(**"Child Class Show Method"**)**;

**}**

//Method Hiding/Shadowing

**public** new **void** Display**()**

**{**

Console.WriteLine**(**"Child Class Display Method"**)**;

**}**

**}**

**class** Program

**{**

**static** **void** Main**(string[]** args**)**

**{**

Parent obj = new Child**()**;

obj.Show**()**;

obj.Display**()**;

Console.ReadKey**()**;

**}**

**}**

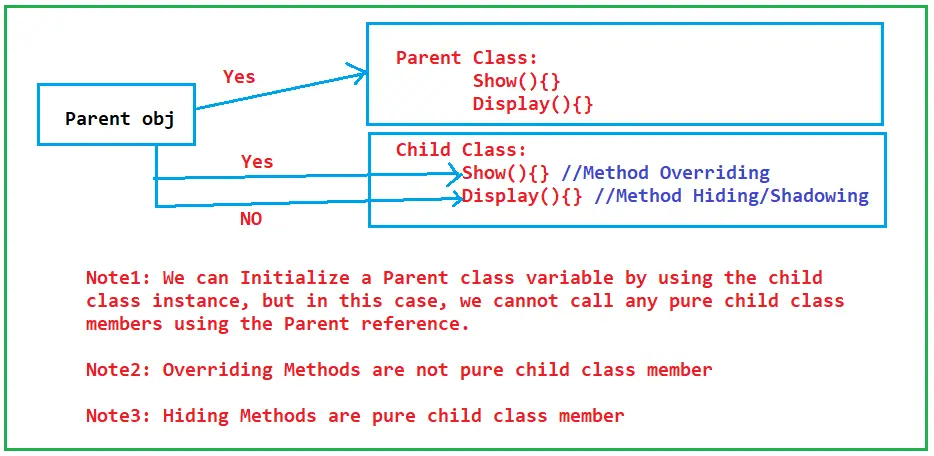
Output

Child Class Show Method

Parent Class Display Methdo

چرا در حالی که کلاس فرزند متد Display را hide کرده اما کماکان متد Display کلاس پدر فراخوانی می شود؟

به دلیل این که زمانی که از کلاس پدر با استفاده از کلاس فرزند reference variable می سازیم چون نوع Variable ما از نوع کلاس پدر است و reference از نوع کلاس child پس بنابراین موضوع می توانیم مستقیم به متد های که در کلاس پدر هستند و در کلاس فرزند نیز override شده اند یا نه دسترسی داشته باشیم اما به متد های purly در کلاس فرزند هستند دسترسی نداریم method hiding نیز از این نوع است pure کلاس فرزند بوده و کلاس پدر نمی تواند مستقیما به آن دسترسی داشته باشد یعنی به تمامی متد های pure فرزند خود دسترسی ندارد.



چون در حالت hiding ما از کلاس پدر permission ندارم این اتفاق می افتد اما در virtual دارای permission هستیم.

# Partial Class and Partial Methods in C#

##### **What is Partial Class in C#?**

ویژگی است که یک کلاس را در چندین فایل به صورت فیزیکی تعریف کنیم اما به صورت منطقی همگی آن ها در یک فایل در نظر گرفته می شوند. با استفاده از کلمه کلیدی partial

می توان interface , struct را نیز به همین ترتیب split کرد و در زمان compiler همگی آن تکه ها یکی در نظر گرفته می شوند.

اگر با ILDASM کد برنامه را نگاه کنید می بینید که همگی در یک فایل قرار داردند کلاس های پارشیال یکی شده اند.

می تواند در مواقعی که کلاس بزرگی داریم کاربردی باشد.

قوانین:

در تمامی کلاس های partial نام کلاس یکسان باشد و کلمه کلیدی partial فراموش نشود.

همگی باید دارای سطح دسترسی یکسان باشند و به هم دسترسی داشته باشند.

اگر یکی از کلاس ها abstract یا sealed باشد تمامی partial ها از آن تبعیت می کنند.

تمامی تکه ها باید از یک کلاس ارث بری کنند و اگر یکی متفاوت باشد خطا کامپایلری می دهد.

کلاس partial می تواند interface های متفاوتی را پیاده سازی کند و به شرطی که تمامی متد ها پیاده سازی شده باشد.

هر عضوی که پیاده سازی شده باشد توسط سایرین نیز قابل استفاده است.

##### **What are Partial Methods in C#?**

می توان امضای یک متد partial را در یک کلاس partial نوشت و پیاده سازی آن را در همان کلاس انجام داد یا در سایز کلاس های پارشیال انجام داد.

در صورتی که متد partial فاقد پیاده سازی باشد و جایی در کد فراخوانی شود compiler به صورت خودکار فراخوانی متد را حذف می کند چون فاقد پیاده سازی است.

**partial** **class** PartialClass

**{**

// Declaration of the partial method.

**partial** **void** PartialMethod**()**;

// A public method calling the partial method

**public** **void** PublicMethod**()**

**{**

Console.WriteLine**(**"Public Method Invoked"**)**;

PartialMethod**()**;

**}**

**}**

Rules

یک متد partial به صورت پیش فرض private است و نمی تواند دارای سطح دسترسی explicit از هر نوعی باشد در غیر این صورت خطا می دهد.

یک متد partial را نمی تواند به صورت هم زمان در declare و implement کرد که سبب بروز خطا می شود اول تعریف و بعدا پیاده سازی شود.

متد partial باید به صورت void پیاده سازی شود و نمی تواند return type داشته باشد.

باید حتما در کلاسی که از نوع partial است تعریف شود.

Signature متد در تعریف و پیاده سازی باید یکسان باشد وگرنه خطا می دهد.

متد partial فقط یک بار می تواند پیاده سازی شود. اگر بیش از یک بار پیاده سازی شود خطا می دهد.

# Sealed Class and Sealed Methods in C#

##### **Sealed Class in C#**

کلاسی که از این نوع باشد هیج کلاس دیگری نمی تواند از آن ارث بری کند.

نمی تواند شامل هر گونه متد abstract باشد. این کلمه کلیدی در class method properties کاربرد دارد.

با استفاده از ایجاد object از کلاس متد ها آن قابل استفاده است.

##### **Sealed Methods in C#**

اگر در کلاس پدر دارای متد های باشیم از نوع sealed فرزندان نمی توانند آن را override کنند به عبارت دیگر در کلاس پدر هر متدی که virtual نباید sealed است.

نکته: اگر می خواهید متدی در سلسلمه مراتب ارث بری از یک جایی به بعد دیگر override نشود می توان در فرزندی که آن متد را override کرده است از کلمه کلیدی sealed استفاده کرد پس از آن کلاس های که از آن فرزند ارث بری کنند نمی توانند آن را override کنند.

**class** Parent

**{**

**public** **virtual** **void** Show**()** **{** **}**

**}**

**class** Child : Parent

**{**

**public** **sealed** **override** **void** Show**()** **{** **}**

**}**

**class** GrandChild : Child

**{**

//'GrandChild.Show()': cannot override inherited member 'Child.Show()' because it is sealed

**public** **override** **void** Show**()** **{** **}**

**}**

زمانی که اطمیان دارید که تمامی فزرند ها باید از منطق یک متد استفاده کنند در کلاس پدر آن را salad کنید.

##### **What is the difference between the private and sealed methods in C#?**

متد که private است در کلاس فرزند قابل فراخوانی و استفاده نیست اما متدی که sealed است در کلاس فرزند قابل فراخوانی و استفاده است اما نمی توان آن را override کرد. اگر کلاس پدر دارای متدی private باشد کلاس فرزند هم با همان نام و امضا می تواند متدی داشته باشد بدون خطا.