**C# nədir**

C#(C Sharp) Microsoft tərəfindən .Net texnologiyası üçün inkşaf etdirilən modern, tam obyekt yönümlü(OOP) və type safety bir proqramlaşdırma dilidir. C#, C ailəsinə daxildir və Java kimi məşhur dillərə yaxındır. Ilk dəfə 2000-ci ildə .Net framework’ün bir parçası olaraq çıxarılmışdır. Bu dilin C# adlandırılmasının səbəbi c++’ın bir üst versiyası olmasıdır. ++ operatorunu istifadə etdikləri üçün C++++ əvəzinə C# olaraq adlandırılmışdır.

**Type safety nədir?**

Bir proqramlaşdırma dilində type safety, dilin type error’lardan qaçmağına kömək edən abstract bir konstruksiyadır. Hər proqramlaşdırma dili gizli type safety səviyyəsinə malikdir. Beləliklə proqram compile olunan zaman compiler type’ları təsdiqləmək üçün type safety konstruksiyasını tətbiq edəcək və dəyişənə yanlış type təyin etməyə çalışsaq, xəta verəcək. Type safety yalnız compile zamanı yox, həm də run time işə salınır.

**Compiler nədir?**

Compiler biz yazdığımız .cs uzantılı C# kodlarını lower-level dilə (Assembly) compile edir. Bu da komputerin bizim yazdığımız insan dilinə daha yaxın olan kodları anlaya bilməsi üçün baş verən prosesdir.

**Solution nədir?**

Solution daxilində bir və ya bir-birləri ilə əlaqəli olan birdən çox project’in saxlandığı workspace, yəni iş sahəsidir.

**Top-Level Statements**

C# 9.0 ilə gəlmiş Top-Level Statements anlayışı bizə əvvəllər Program.cs daxilində yazdığımız main method’u yazmadan kodlarımızı yazıb compile edilə bilməsi üçün şərait yaradır.Yalnız Program.cs daxilində Top-Level Statements’dən istifadə edilə bilər. Compile olduqdan sonra özü Assembly daxilində main method yaradıb, bizim yazdığımız kodları işlədir.Compile zamanı class adından Program.cs olduğu başa düşüləcək və main method arxa tərəfdə yaranacaq. Top-Level Statements microservices arxitekturasında kodun inkşaf etdirilməsində sürət qazanmaq üçün istifadə edilir.

**Comment line**

Kritik nöqtələrdə yazılan kodları, başqa developerlərin və ya bir müddət sonra özümüzün daha rahat oxuya bilməsi üçün, comment sətrlərdən istifadə edilir.Comment sətrlər vasitəsi ilə biz yazılan kodların məğzini, yazılma səbəbini açıqlayırıq.Compile zamanı yazılan sətrin comment sətr olduğu anlaşılır və həmin sətr maşın dilinə compile olunmur.

//comment - 1 sətri comment’ə almaq üçün istifadə edilir.

/\*

Çox sətrli comment

\*/ - 1’dən çox sətri comment sətri kimi göstərmək üçün istifadə edilir.

**Region**

Yazılan kodları bir .cs file daxilində kateqorik bir hala gətirmək üçün, yəni qruplaşdırmaq üçün Region operatorundan istifadə edilir.Region istifadə edərək kodlarımızı daha oxunaqlı hala gətiririk.

#region RegionName

Code lines

#endregion

**TODO**

Bir Visual Studio funksiyası vasitəsi ilə biz .cs file daxilində comment sətri yazan zaman hər hansısa bir tapşırıq vermək istəsək və ya xatırladıcı yazmaq istəsək todo keyword’dən istifadə edirik.Yazılışı da bu şəkildədir //todo comment . View pəncərəsində Task List’ə daxil olaraq file daxilindəki todo’ları görə bilərik.

**Variable**

Applikasiya daxilində hər hansısa bir datanı ramda saxlamaq və onu lazım olan zaman ramdan çağırmaq üçün variable’dan istifadə edilir.

**Datatype**

C#’da dataları ramda saxlaya bilmək üçün bu datalara müəyyən type’lar verilməlidir.

**Value Data Types**

Value type’lara Integral numeric types, Floating-point numeric types, Boolean və Character daxildir.

**Value Data type ranges (Verilənlərin tiplərinin aralıqları)**

**Unsigned Integral Numeric Types**

Unsigned yəni işarətsiz value type’lar, işarəli bitlərdən istifadə etmirlər bu səbəbdən işarəli value type’lardan fərqli olaraq daha geniş pozitif dəyər aralığına sahibdirlər.

byte (Yaddaşda tutduğu yer 1 byte) 0 to 256

ushort (Yaddaşda tutduğu yer 2 bytes) 0 to 65,535

uint (Yaddaşda tutduğu yer 4 bytes) 0 to 4,294,967,295

ulong (Yaddaşda tutduğu yer 8 bytes) 0 to 18,446,744,073,709,551,615

**Signed Integral Numeric Types**

Signed yəni işarəli value type’lar, dəyərin işarətini bildirmək üçün işarəli bitlərdən istifadə edirlər.

sbyte (Yaddaşda tutduğu yer 1 byte) -128 to 127

short (Yaddaşda tutduğu yer 2 bytes) -32,768 to 32,767

int (Yaddaşda tutduğu yer 4 bytes) -2,147,483,648 to 2,147,483,647z

long (Yaddaşda tutduğu yer 8 bytes) -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

**Floating Point Numeric Types**

float(Yaddaşda tutduğu yer 4 bytes) vergüldən sonra 6 və ya 7 rəqəm üçün nəzərdə tutulub

double(Yaddaşda tutduğu yer 8 bytes) vergüldən sonra 16 rəqəm üçün nəzərdə tutulub

**High-precision decimal Floating-point Numeric Type**

Decimal typeByte’lardan əmələ gəlmişdir və value type’dır.

Float type variable yaradan zaman dəyərin sonuna F,Double type variable yaradan zaman D və Decimal type variable yaradan zaman isə M yazılmalıdır.

**Character Type**

char (Yaddaşda tutduğu yer 2 bytes) Simvollar və ya onlara uyğun olaraq ASCII kodları

**Boolean Type**

bool (Yaddaşda tutduğu yer 1 byte) true və ya false

**Object**

Object reference type’dır və bütün type’lar object type’dan törəyiblər. Bu səbəbdən object, bütün type dəyərləri özündə saxlaya bilir. Object dəyişənlər dataları RAM’da object type olaraq saxlayır lakin data’ların öz type’larını da özündə saxlayır. Bu da o deməkdir ki object’in daxilində saxladığı variable’ı öz type’ı ilə təkrar əldə edə bilərik.

**Boxing**

Boxing, hər hansısa type dəyərin object type’a assign olunmasına deyilir. Object type’a çevrilən dəyər object daxilində öz type’ı ilə saxlanılır. Lakin variable çağırılan zaman object type olaraq gəlir. Bu da o deməkdir ki variable’a öz type’ı ilə bağlı heç bir əməliyyat tətbiq edilə bilməz. Object type dəyişənin daxilindəki dəyərə öz type’ı ilə əldə etmək üçün Unboxing etmək lazımdır.

**Unboxing**

Unboxing, boxing olunmuş hər hansısa datanı öz type’na uyğun şəkildə əldə edilməsinə deyilir. Unboxing edə bilmək üçün explicit casting etmək lazımdır. Bunun vasitəsi ilə biz type uyğunluğunu təmin edirik. Yəni casting zamanı saxlanılan datanın orjinal dəyəri nədirsə ona casting edilməlidir. Bunun üçün is və as operatorları ilə müəyyən yoxlanılmalar aparılaraq qayıdan bool dəyərə uyğun olaraq yol izlənilə bilər.

**Casting**

Casting, type’lar arasında dəyişikliklər etmək üçün istifadə edilir. Cast operatoru (T) olaraq işarə olunur. Variable’ın əvvəlində yazılır və mötərizə daxilində hansı type’a cast edəcəyiksə həmin type yazılır. Casting, Implicit casting(Widening) və Explicit casting(Narrowing) olaraq 2 əsasə yerə bölünür.

Implicit casting ilə biz balaca datanı böyük dataya çeviririk və bu əməliyyat avtomatik olaraq compiler vasitəsi ilə həyata keçir çünki burada heç bir data itkisi olmur. Cast operatorundan istifadə etmək məcburiyyətində deyilik.

Explicit casting isə böyük datadan kiçik dataya keçmək üçün və ya əlaqəli olmayan type’lar arasında keçid etmək istifadə edilir. Explicit casting zamanı casting operatorunun istifadə olunması məcburidir. Əgər dəyər target type’a cast oluna bilmirsə hər hansısa data itkisi və ya potensial runtime exception baş verə bilər. Unboxing zamanı explicit casting’dən istifadə edilir.

**Operators & Keywords**

**Break**

Break keyword’ü müəyyən bir şərt daxilində loop’ları sonlandırmaq üçün və ya switch’dən çıxmaq üçün istifadə olunan bir ifadədir.

**Continue**

Continue keyword’ü vasitəsi ilə biz müəyyən şərt daxilində işləndiyi sətrdən sonra ki əməliyyatlar yerinə yetirilmədən loop’un əvvəlinə qayıtmaq üçün istifadə edilir.

Continue keyword’dən fərqli olaraq break keyword’ü loop’la yanaşı switch daxilində də işlənilir

**Return**

Return keyword’ü daxilində yazıldığı method’u dəyandırır və özündən sonra gələn code sətrlərini işə salmır. Əgər method void method deyilsə yəni hər hansısa bir return type’ı varsa bu type’a uyğun olaraq dəyər return keyword’ü vasitəsi ilə qaytarılmalıdır.Əgər method void method’dursa müəyyən bir şərt daxilində bu method’u sonlandırmaq üçün return keyword’ü istifadə edilir.

**Is**

Is keyword’ü vasitəsi ilə hər hansısa bir referansın və ya variable’ın null olub olmadığını yoxlaya bilərik. “== null” və ya “!= null” ifadəsi ilə eyni güclüdür. Və ya hər hansısa type olub olmadığını yoxlaya bilərik. Məsələn is int. Is keyword’ün işlənmə hallarında geriyə true və ya false olaraq bool dəyər qaytarır. Is keyword’ü C# 7 ilə birlikdə gəldi.

**Is Not**

Is not keyword’ü is keyword’ün negative halıdır. C# 9 ilə birlikdə gəldi. is not null və ya is not anyType şəklində yoxlanış aparıla bilər. Geriyə true və ya false olaraq bool dəyər qaytarır.

**?? Null-Coalescing Operator**

?? olaraq bilinən Null-coalescing operator null dəyəri idarə etmək üçün qısa yol təqdim edir. Əgər null dəyərə sahib deyilsə sol operandını qaytarır əks təqdirdə sağ operandını qaytarır.

Məsələn nullable olan “nullableNum” int dəyişən və 2-ci bir “num” int dəyişən verilmişdir. əgər bizim null dəyər ala bilən “nullableNum” dəyişənimiz null’dan fərqlidirsə 2-ci dəyişənimiz “num”a assign olunacaq əks təqdirdə operatorumuzun sağ operandına assign olacaq.

int? nullableNum = null

int num = nullableNum ?? 5;

**??= Null-Coalescing Assignment Operator**

??= olaraq bilinən Null-coalescing assignment operator dəyişənə qiymət təyin etmək üçün yalnız hal-hazırda null olduqda istifadə olunur.

string name = null;

string defaultName = “cavanshir”;

name??=defaultName;

Bu da o deməkdir ki əgər name variable null olarsa ona defaultName’i mənimsət.

**Var**

Var keyword’ü ilə variable’a value mənimsədildiyi zaman compile zamanı datatype müəyyən edilir və biz həmin variable’a başqa bir yerdə fərqli datatype’dan bir value mənimsədə bilmərik. Bunu etsək compile time error verəcək. var keyword’ü ilə variable yaranan zaman dəyər assign edilməlidir ki compile zamanı hansı datatype alacağı bəlli olsun.

Bu keyword’ün yaradılma məqsədi fərqli dillər arasında dəstəklənməyən datatype dəyişənləri saxlamaq üçün ortaq keyword olsa da, günümüzdə istifadə məqsədi dəyişənin type’nı yazmağa tənbəllik etməkdir.Compiler’a əlavə iş gördürməmək üçün type’nı bildiyimiz dəyişənləri var keyword’ü ilə təyin etməməliyik.

Var ilə object arasındaki fərq odur ki, var keyword’dür, object isə type’dır. var ilə təyin etdiyimiz dəyişən stack yaddaşda saxlanılır, lakin object type dəyişən yaddaşda object şəklində saxlanacaq və daxilində həmin dəyəri saxlayacaq. Boxing etdiyimiz üçün unboxing edərək həmin dəyəri istifadə edə bilərik.

**Dynamic**

Dynamic keyword’ü ilə assign olunmuş variable’a hər dəfə müxtəlif datatype dəyər mənimsədilə bilər. Runtime olaraq datatype mənimsədildiyi üçün var keyword’dən fərqlənir və onun runtime versiyasıdır.

**Default Value**

Value type variable’ların default dəyəri 0’dır

Char type variable’ın default dəyəri ‘/0’ dır.

Boolean type variable’ın default dəyəri false’dur

Reference type variable’ın default dəyəri null’dır.

**Nullable**

Hər hansısa verilmiş variable’ın null dəyərə də sahib ola biləcəyini göstərmək üçün T? variable = value; istifadə edilir. Burada T bütün əsas type’ları işarə edir. Əsas type’ların qeyri-müəyyən dəyərini təmsil etmək üçün istifadə edilir.

Nullable<T> variable = value; Nullable type(yəni struct və ya enum type) variable’ların heç bir dəyərə sahib olmadığını bildirən əlavə null bir dəyər saxlaya bilərlər. Non-Nullable olan type’ları bu şəkildə istifadə edə bilmərik.

Nullable<T> və ya T? İstifadə olunma məqsədi ondan ibarətdir ki, məsələn bizim məlumat bazamızda hər hansısa sütun yalnız true və ya false dəyər qəbul edir. Əgər heç bir data verilməyibsə biz bunu bool dəyişənə mənimsədə bilməyimiz üçün Nullable<bool> və ya bool? Olaraq dəyişənin tipini göstərməliyik.

**Arithmetic operators**(Hesab operatorları)

**/** - div bölmə adlanır , tam hissə götürülür

**%** - mod bölmə adlanır , qalıq hissə götürülür

**Increment** 🡪 ++

**Declerement** 🡪 --

int num = 10;

num++ && num-- 🡪 Postfix increment & decrement adlanır. Burada gedən proses özündən sonraki sətrə aid olur. Console.WriteLine(num++); yazarsaq ekrana **10** çıxacaq lakin artıq yaddaşda num **11** olaraq qalacaq və daha sonra num dəyişənini ekrana verərsək **11** kimi çıxacaq

++num && --num 🡪 Prefix increment & decrement adlanır. Burada həmin sətr daxilində artıq proses getmiş olur. Console.WriteLine(++num); yazarsaq ekrana **11** çıxacaq çünki artıq proses həmin sətrdə getmiş olacaq

**Assigment Operators**(Mənimsətmə operatorları)

a += b 🡪 a = a + b

a -= b 🡪 a = a - b

a \*= b 🡪 a = a \* b

a /= b 🡪 a = a / b

a %= b 🡪 a = a % b

**Comparison Operators**(Müqayisə operatorları)

Müqayisə operatorları bizə true və ya false dəyər qaytarır

== 🡪 bərabərdirmi?

!= 🡪bərabərliyin əksi, bərabər deyilmi yəni not equals

> 🡪 böyükdürmü

< 🡪 kiçikdirmi

>= 🡪böyük bərabərdirmi

<= 🡪kiçik bərabərdirmi

Bu operatorlar bizə boolean dəyər qayatırlar. Və əsasən if şərti daxilində və ya loop’larda istifadə olunurlar.

**Logical Operators**(Məntiqi operatorlar)

&& 🡪 and(və) deməkdir. Işləndiyi bütün şərtlər true olduğu halda true dəyər qaytarır

|| 🡪 or(və ya) deməkdir. Işləndiyi şərtlərin ən azı 1-I true olduğu halda true dəyər qaytarır

! 🡪 not(əks) deməkdir. Işləndiyi şərt true olarsa false , false olsarda true dəyər qaytarır.

**Ternary Operator**

var variable = (condition)? ValueIfConditionIsTrue: ValueIfConditionIsFalse

Ternary operatoru vasitəsi ilə biz if condition istifadə etmədən 1 sətr olaraq true və ya false case’ə görə fərqli value mənimsətək üçün istifadə edilir.

**Condition**

**If Statements**

If(condition) {

Statement;

}

Statement2;

Bu ifadəyə if-then ifadəsi deyilir. Əgər verilən şərt doğru olarsa if daxilindəki ifadə yerinə yetirilir və daha sonra 2-ci ifadə yerinə yetirilir. Əks təqdirdə sadəcə 2-ci ifadə yerinə yetirilir.

If(condition) {

Statement;

} else{

Statement2;

}

Bu ifadəyə if-then-else ifadəsi deyilir. Əgər verilən şərt doğru olarsa sadəcə if daxilindəki ifadə yerinə yetirilir əks təqdirdə sadəcə 2-ci ifadə yerinə yetirilir.

if(condition) {

Statement;

} else if(condition2){

Statement2;

} else if(condition3){

Statement3;

} else{

StatementN;

}

Else-if ifadəsi müəyyən bir şərt ödənmədikdə verilə biləcək digər şərtə uyğun prosesin işə salınması üçün istifadə edilir

**Switch case**

Bizim təyin etdiyimiz dəyişənin müxtəlif dəyərlərə bərabər olma halı yoxlanılırsa if statement yerinə switch case’dən istifadə edilir. Əgər bizim dəyişənimizə heç bir case halı uyğun olmazsa default daxilindəki əməliyyatlar işə düşür.

Switch case’də hər case daxilində müəyyən statement’dən sonra break || return || goto case VALUE keyword’lərindən biri yazılmalıdır. Əks təqdirdə compile time olan “fall-through” error meydana gələcək. "goto case VALUE” ifadəsi bizim case halımız işə düşdükdən sonra müəyyən bir case’ə yönləndirmə etmək üçün istifadə edilir.

Hər hansısa case’dən sonra heç bir şey yazmayıb bir başa olaraq başqa bir case’ə keçə bilərik. Bu da o deməkdir ki əgər bu case doğru olarsa ondan sonra gələn case daxilindəki proses həyata keçiriləcək.

Yuxarıda danışdıqlarımızın hamısına bir misal gətirək.

switch(expression) {

case value1:

case value2:

Statement2;

break;

case value3:

Statement3;

Goto case value4;

case value4:

Statement4;

//yuxarıda sadalanan keyword’lərdən heç biri işlənmədiyi üçün compile zamanı fall-through error’u verəcək.

case value5:

Statement5;

return;

...

case valueN:

StatementN;

break;

default:

StatementDefault;

break;

}

**Switch case expression**

var month = “October”;

var monthCode = month switch {

“January” => 1,

...

“December” => 12,

\_ => -1

};

Switch case expression c# 8.0’dən sonra gəlmişdir. Switch case expression hər hansısa variable’a müxtəlif case’lərdə dəyər mənimsətmək üçün istifadə edilir. Switch case və ya müxtəlif else-if condition’lar yazmaqdansa daha rahat və qısa yazılış ilə dəyər mənimsətmək üçün istifadə edilir. Default ‘\_’ simvolu ilə işarələnir.

Biz switch case expression vasitəsi ilə həmçinin müəyyən şərtləri aşağıdaki şəkildə yoxlaya bilərik.

int num = 6;

string result = number switch{

<= 0 => “0-dan kiçikdir və ya 0-a bərabərdir.”,

> 0 and <5 => “1 və 4 aralığındadır 1 və 4-ün özü də daxildir.”,

>= 5 and <=10 when number %2 == 0 => “5 və 10 aralığında cüt ədəddir”,

>= 5 and <=10 => “5 və 10 aralığında tək ədəddir”,

\_ => “10-dan böyükdür.”

};

when statement qaydası qalıb

valueCase => expression əvəzinə, datatype variable2 when statement => value yazaraq əgər statement bizə true dəyərini qaytararsa variable’a value’nu mənimsədir.Burada variable2 bizim expression kimi göndərdiyimiz dəyərə bərabər olur və statement daxilində istifadə etmək üçündür.

**Loop Statements**

Loop(dövr) qarşıya qoyulmuş məsələni müəyyən şərt daxilində müəyyən say qədər yerinə yetirmək üçün istifadə edilir.

**While**

while(condition) {

Statement;

}

Şərtimiz doğru olduğu müddətcə blok daxilindəki ifadə yerinə yetiriləcək.Bu da o deməkdir ki ilk olaraq hər dəfə gedib şərt yoxlanılacaq və true dəyər qayıdarsa prosess yerinə yetiriləcək.Bu səbəbdən While’a həm də ön şərtli dövr operatoru deyilir.

**Do While**

do{

Statement;

}while(condition)

Do while ilə while’ın fərqi odur ki, do while zamanı şərt yoxlanılmadan ilk başda proses 1 dəfə yerinə yetirilir daha sonra şərt yoxlanılır. Bu da o deməkdir ki verilən condition false olsa, while loop’da proses heç vaxt yerinə yetirilməyəcək amma do while’da isə proses 1 dəfə yerinə yetiriləcək. Bu səbəbdən Do While’a həm də son şərtli dövr operatoru deyilir.

**For**

for(dataType variable; condition; process1){

Process2;

}

For loop’u 3 hissəyə bölünür 1-ci hissə dəyişən təyin etmək üçün, 2-ci hissə şərt vermək üçün və sonuncu hissə müəyyən bir əməliyyatı yerinə yetirmək üçün istifadə olunur.Şərt ödəndiyi bütün hallarda For loop’un bloku daxilindəki proses işə düşür.

for(;;){

Process;

}

Bu yazılış infinite for loop adlanır. Müəyyən şərt daxilində loop break olunmazsa sonsuzadək davam edəcək.

**Foreach**

foreach(var item in collectionName) {}

Foreach loop’u hər hansısa array, collection bir sözlə desək iterasiya oluna bilən hər bir object’ə tətbiq edilə bilər. Foreach vasitəsi ilə object’in hər bir elementinə 1-1 müraciət edəbilərik. Bir sıra situasiyalarda foreach bizim köməyimizə çatır. Nəzər yetirmək lazımdır ki for loop’undan daha sürətlidir. C# 8’dən sonra gəlib.

**Function & Method**

Function bir sıra ifadələri daxilində saxlayan,müəyyən bir tapşırığı yerinə yetirən kod blokudur. Function’lar geriyə hər hansısa type dəyər qaytara bilər(return edə bilər) və ya void(heç bir dəyər qaytarmayan) ola bilər. 2’dən artıq yerdə təkrarlayacağımız kod sətrlərini DRY(Don’t Repeat Yourself) prinsipini qorumaq üçün function halına çıxarmaq lazımdır. Function’ların istifadə məqsədi budur, təkrarçılığın qarşısını almaq və kodu optimallaşdırmaq.

Method isə function’ların hər hansısa bir class’a(type’a) aid edilmiş formasıdır.

Method yazılışı bu şəkildə olacaq. İlk olaraq Access modifiers, daha sonra optional olaraq optional modifiers , return type və ya void , method name və sonda method parametr gəlir.

Method təyin olunan zaman hər hansısa access modifier yazılmazsa class daxilində digər bütün class member’lər kimi(constructor xaric) private,struct daxilində isə internal olur.

**Method Signature**

Həmçinin method’lar müəyyən bir imzaya sahib olurlar, bu da Method signature adlanır. Method signature’də 3 əsas məqama diqqət edilir. Method’un adı, qəbul etdiyi parametrlərin sayı, parametrlərin type’ı. Yadda saxlamaq lazımdır ki,method’un return type’ı method signature’ə aid deyil.

**Parameter & Argument**

Method təyin edilən zaman onun qəbul etdiyi hər hansısa variable parametr adlanır. Method çağırılan zaman method daxilində göndərilən variable isə arqument adlanır.

**Optional Parameters**

**Default Value**

Method’a hər hansısa parametr təyin edən zaman biz bu parametr’ə default bir dəyər təyin edə bilərik.Yəni biz method’u call edən zaman həmin parametrə uyğun olaraq bir arqument göndərməsək default olaraq default value mənimsədiləcək. Əks təqdirdə yeni dəyər set olunacaq və default value override ediləcək.

**Params**

Method yaranan zaman eyni type’dan neçə parametr təyin etməli olduğumuzu bilməyə bilərik. Bu problemin qarışısını almaq üçün params’dan istifadə etməliyik. Sadəcə göndərməli olduğumuz type array yaratmalı və qarşısına param yazmalıyıq. Beləliklə eyni type’dan istənilən qədər arqument göndərə bilərik.Avtomatik olaraq göndərilən arqumentlər bir array halına salınacaq.

Optional parametr olaraq verilən parametrlər tələb olunan parametrlərdən sonra yazılmalıdır. Yəni optional parametr’in istifadə ardıcıllığı sonuncudan başlayaraq sağdan sola doğrudur.

**Ref & Out**

Method daxilində value type’da edilən dəyişiklik ümumi olaraq da(stack yaddaşda) dəyişilməsi üçün ref və out keyword’dan istifadə edilir. Hər zaman value type üçün istifadə edilmiş həmçinin reference type’lar üçün də istifadə edilə bilər.

Method parametr olaraq qəbul etdiyi dəyişənin əvvəlinə ref və ya out yazılır və arqument olaraq göndəriləndə də ref və ya out keyword’ü ilə göndərilir.

**Ref & Out fərqi**

Out keyword’ü istifadə etdikdə variable’a əvvəldə dəyər mənimsədilməsə də olar amma method daxilində mütləq variable’a dəyər mənimsədilməlidir. Ref keyword’dən istifadə zamanı isə mütləq variable’a əvvəldə dəyər mənimsədilməlidir yəni unassign olan dəyər göndərmək olmaz.

**Extension method**

C#’da hal-hazırda var olan hər hansısa class və ya interface’in orjinal mənbə kodlarını dəyişmədən mövcud class və ya interface’ə yeni method’lar əlavə etmək üçün Extension method’lardan istifadə edilir. Extension method’lar vasitəsi ilə biz, tərəfimizdən əlçatan olmayan, müdaxilə edilə bilməyən class və ya interface’lərə də yeni funksionallıqlar əlavə etməyimiz üçün imkan verir. Extension method’lar static method şəklində, static class’lar daxilində təyin olunurlar. Method’un ilk parametri extension method’u tətbiq edəcəyimiz type olmalı və qarşısında this keyword’ü ilə yazılmalıdır. Extension method’lar göstərilən type’ın reference’na tətbiq edilir. MyType myType = new(); myType.MyExtensionMethod(); yazaraq biz MyType tipindən olan object ilə custom olaraq yaratdığımız Extension method’u call edirik. Dediyimiz kimi həmçinin interface’lər üçün də Extension method yazmaq olur. Bu zaman dəyişən tək şey extension method’u call edən zaman, extension method tətbiq olunan interface’dən miras alan hər hansısa class’ın reference’ı istifadə edilir.

**Indexer**

Array’lərdə və s qarşılaşdığımız index(yəni arr[i]) məntiqini biz custom olaraq özümüz də lazım olarsa yarada bilərik. Bunun üçün bizim class daxilində property məntiqində return type’ı R olan

public R this[int index]{

get{return collectionObj[index];}

set{collectionObj[index] = value; }

}

indexer’i yazmağımız kifayət edir. Bununla biz bildirmiş oluruq ki həmin class’ın referece’ı qarşısında [index] yazsaq, uyğun olaraq class daxilindəki collection’ın əgər varsa həmin indexdəki dəyərini almış olarıq.