

C# .NET 2005

من البداية حتى الاحتراف
لتتأهل لشهادة MCP من مايكروسوفت



إعداد وتنسيق: y85.fox

تأليف: محمد جمال

الفهرس

الموضوع	الصفحة
مقدمة	5-
.Net introduction (Platform And Frame Work).....	6-
Basic C# Programs.....	9-
Value Types in .net.....	16-
Statements.....	28-
Exception Handling.....	41-
Methods.....	43-
Arrays	48-
Collections.....	52-
Object Oriented Essentials.....	55-
Inheritance in C#.....	65-
Indexes and properties.....	67-
Overloading Operator.....	76-

Polymorphism

Delegates and events

Attributes

Modules and assemblies

I/O

Serialization

Graphics

Threading

Application domain and Services

Security , User and data security

Reflection

Instrumentation

Globalization

Interoperation

Mailing

Installing and configuring application

﴿بسم الله الرحمن الرحيم﴾

مقدمة:

أحببت إن أنشئ في منتدانا الحبيب الفريق العربي للبرمجة دورة للغة

C#.NET 2005

يتم وضع دروسها بشكل مستمر نبدأ من البداية لنصل إلى مرحلة الاحتراف في هذا المجال .حيث من المفترض إن تكون بعد إكمال الدورة قادرا على التقدم لامتحان MCP من شركة مايكروسوفت للغة

C#.NET 2005

ونجاحك به وسأقوم بوضع نماذج لاسئله شركة مايكروسوفت بإذن الله إن استطعت هنا في كل موضوع سوف يتم تناول موضوعين أو ثلاثة أو ربما أربعة حسب حجم الموضوع ليتم شرحه هنا فأرجو من الله إن يوفقنا ويفقكم لما يحب ويرضى وإن يقدم لي العون الكافي لأقوم بإكمال الدورة كاملة معكم المطلوب منك عزيزي العضو القيام بمتابعه الدورة بشكل مستمر والقيام بالاستفسار عن الأمور التي لا تفهمها وإن شاء الله ستجدني بالخدمة

1- .Net Introduction (Platform And Frame Work)

سوف نقوم بداية بالقيام بإعطاء مقدمة بسيطة عن بيئة الدوت نت ومحتوياتها

The Programmer in The Programming Platform : Operating System That Help

أي أنها عمليات في النظام تساعد المبرمج في عملية البرمجة

Framework 2.0 Components

محتويات الفريم وورك الجديدة من مايكروسوفت

System Compiler : Tools That Convert the Code to the Source in

أي إن الكومبايلر هو المترجم لأكواد السي شارب وهو الذي يقوم بإرجاعها لمصدرها

وعملنا كله سوف يترجم على هذا الجزء من الفريم وورك

Base Class Library -2

وهي ال DLL التي توفر موارد النظام عند عملية تشغيل البرنامج سيتم شرحها بشكل مفصل في المراحل المتقدمة

Runtime CLR : Common language -3

وهو الذي ينظم العمليات في النظام
وسوف نشرحه بشكل مفصل وإلى ماذا ينقسم

a - Garbage Collection

b - multimedia Threading support

c - debugger and exception Handler

a - وهو لعمليات الحفظ في الذاكرة والذاكرة مقسمة إلى قسمين

Stack

Heap

ال **Stack** يقوم بحفظ أسماء المتغيرات ويقوم بربطها بقيمتها التي في ال **Heap**

مثال:

كود

```
String Str = "Mr.Asp"
```

هنا سيتم حفظ المتغير **a** في ال **Stack**
وقيمته **mr.asp** في ال **Heap**

طبعا ال **Heap** ما بعرف مكانها غير الله سبحانه وتعالى

يعني ما في رابط خاص للدخول إليها

طيب مثلا ألان صار في عندنا معلومات بال **heap** وما في **stack** باشر عليها
رح يصير هون عندنا حجز من الرامات والميموري في الكمبيوتر

ورح يثقل بدون داعي

طيب كيف بدنا نحذف ال **heap** بدون ما نعرف مكانها ???

هنا يأتي دور **Collection Garbage**

حيث يقوم بفحص كافة **heap** ويرى إذا كان يوجد متغير بال **stack** يؤشر عليها يتركها إما
إذا كانت مقطوعة ولا يوجد إي شيء يؤشر عليها فانه يقوم بحذفها بشكل فوري

b- دائما واحنا بنلعب ألعاب بنشوف مثلا في رجل يمشي
أو سيارة جايه
أو مجموعة سيارات تمشي أو طيارات بنفس اللحظة

طيب كيف ال **Process** قادر على تشغل هذا المجموعة مع بعضها مع انه لا يستطيع تشغل تعليمه واحد بنفس الوقت

هنا يأتي دور ال **Multithreading**

thread معناها بالأصل خيط

وهنا تستخدم لوصل شيء بشي مثلا

يمكن تقسيم المعالج **process** لأكثر من **thread** كل واحد منها يقوم بعمل معين

مثلا يقوم هذا ال **thread** بعمل هذا الشيء و الثاني يقوم بعمل شيء آخر

وهكذا يتم توزيع موارد النظام بشكل منظم جدا

لن نتطرق ألان في هذه العمليات ولكن سنقوم بها بجلسة مخصصة فيما بعد

c- Debugger And Exception

الأخطاء في برامجنا

دائماً لما نشغل البرنامج يحدث معنا مشاكل
كل مشكلة بوقت معين

مثلاً إحنا مرات بتفحص الكود ما بطلع في مشكلة لما نشغل البرنامج يعطيك مشكلة

شو السبب !!!!!

رح نعرفه ألان

تقسم الأخطاء إلى الأنواع التالية

Error : in Compiler Syntax

إي بيكون عندك خطأ بالكود وبييصر بالكمبايلر

Error : CLR Runtime

مثلاً أنت طلبت تشغل ملف وما كان موجود

Debugger : Logical Error

تحدث في Debugger لأسباب معينة في البرنامج

بحمد الله وبفضلة أكملنا أول موضوع من مواضيع الدورة وهو

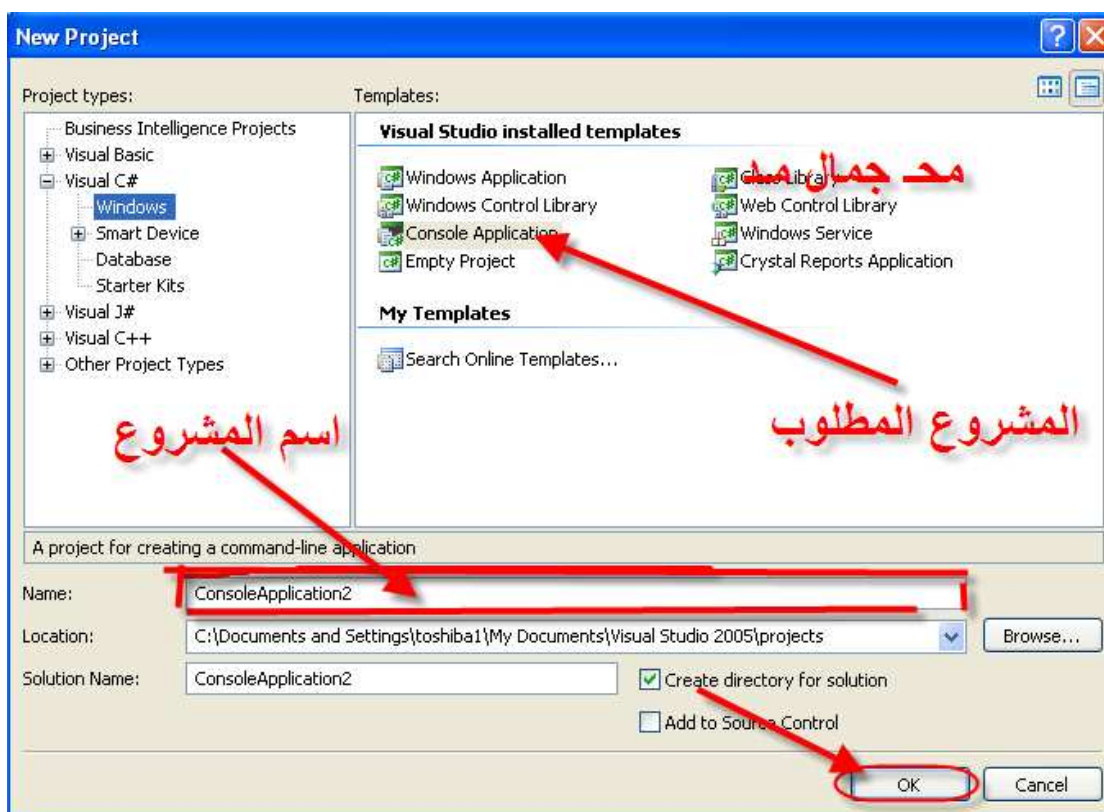
Net Introduction (Platform And Frame Work)

وسأبدأ بشكل مباشر في الموضوع الثاني
لأنني مش حاس الموضوع بصراحة ثقيل بالعكس هو معلومات عامه حول البرمجة
تحياتي

Programs Basic c#

الآن عملية البرمجة في السي شارب باستخدام تطبيقات إلى Console Application

بعد فتح السي شارب قم باختيار تطبيق Console Application من الشاشة كما هو موضح بالصورة التالية



إذا ما كنت بتعرف كيف بدك تعمل New Project في اضغط

N + Ctrl + Shift

الآن إمامنا الشاشة مكتوب عليها الكود التالي

كود

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

طبعا الوضع الافتراضي للجدار على البرمجة أنهم ما يفهمون أشي بس مش مشكلة

إن شاء الله رح يطلعوا فهمانيين كل شي.

أولا:

Using

تستخدم هذه الكلمة بمثابة استدعاء لبعض الأوامر الموجودة من ال **namespace** في البرنامج يعني مثلا

احنا بدنا نكتب أمر بال Console Application

انو يروح يطبع سطر مكتوب في المحيط العربي

طيب كيف بدنا نعملها ??

إذا ما كان في **namespcae** اسمها **system**

رح نضطر نكتب التالي كل ما بدنا نطبع سطر

كود

```
System.Console.WriteLine("ArabMoheet")
```

طبعا للإعمال الضخمة والكبيرة هذه العملية ما بتوفي معنا

بدنا أشي مختصر

عشان هيك بنحط فوق

كود

Using System;

وبالتالي بنبطل كل مرة نكتب كلمة **system** وبنتريح منها بالكلاسة كلها

ويلي مكتوب تحتها نفس المهمة ورح نشرحهن بالتفصيل بالمستقبل

الان نيجي لتشرح القسم يلي بعده

يلي هو **namespace ConsoleApplication1**

هذا فقط يعني انو اسم المساحة يلي بتشتغل فيها **ConsoleApplication1**

وطبعا عشان نبين أولها من آخرها لازم نحط الأقواس التالية

هذا للبداية للنيم سبيس

{

وهذا للنهاية

}

وبعدة بتقدر تبدأ نيم سبيس جديدة لها أقواس

الان يلي بعده هو

Class Program

هذا يعني اسم الكلاس يلي بتشتغل فيه **Program**

نفس فكرة النيم سبيس يمكن إنشاء أكثر من كلاس بنفس النيم سبيس

وبداخلة

كود

```
Static void Main (string [] args)
```

```
{  
//Code here  
}
```

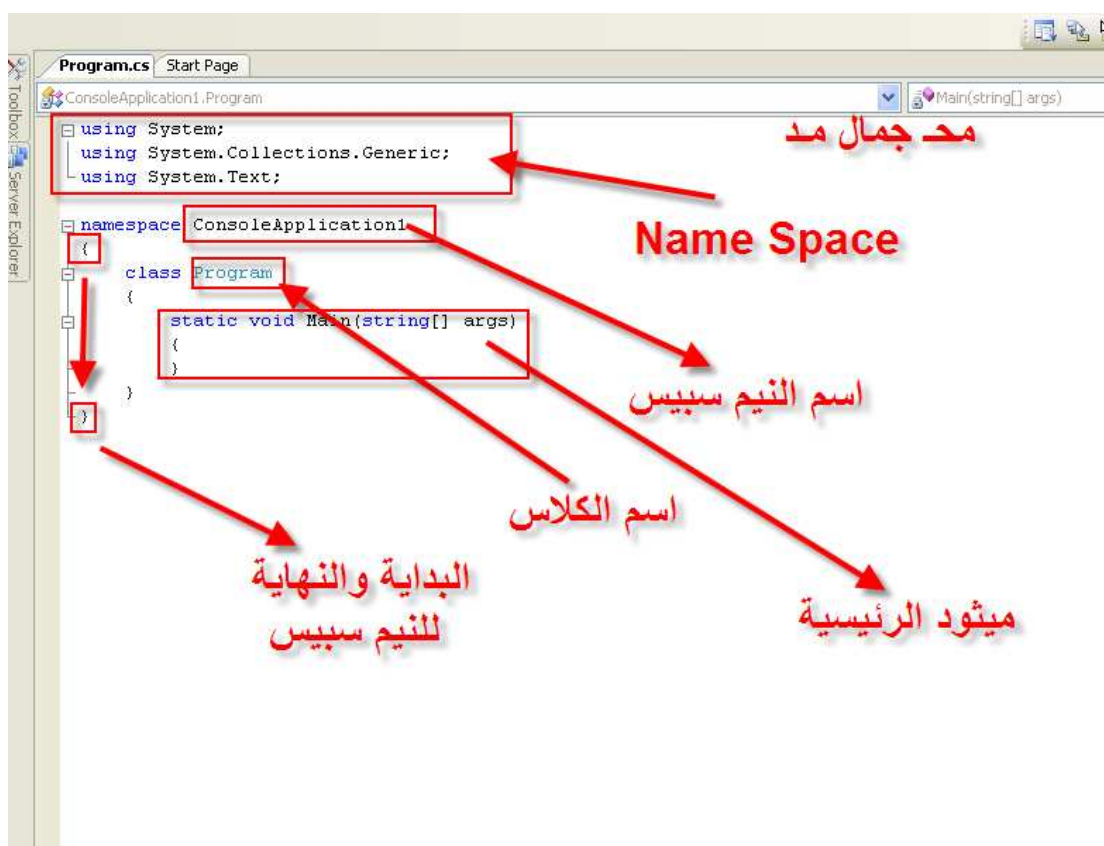
هذا الكود هو عبارة عن ميثود وهو الأمر الأول الذي يتم تنفيذه
إي يتم تنفيذ الكود المكتوب بين الأقواس هذه أول شيء عند بداية البرنامج

وتوضع الميثود الأخرى في هذا الميثود لتنفيذها

مفهوم ميثود سوف تعرفونه قريباً بشكل أوسع ولكن لحظة بسيطة عنه سأعطيها لكم

هو عبارة عن مجموعة من الأوامر المنظمة تحت اسم معين يمكن استدعاءها بأي جزء من البرنامج

وسأرفق لكم أيضاً صورة تشرح لشاشة الأولى للبرنامج بشكل مصور



الآن مثلاً كتبنا **نيم سبيس** اسمها **moheet** بعد النيم السبيس الافتراضية **ConsoleApplication1**

ونريد إن نحصل على معلومة داخل الكلاس **program**

فكيف نأتي بها ؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟

بكل بساطة أقول انه يمكنك ذلك عن طريق القيام بعملية التسلسل

إي

كتابة الكود التالي

كود

```
Namespace.nameclass.namething
```

إي لو قمنا بتعريف متغير داخل كلاس **program** بهذه الطريقة

كود

```
public int x = 200;
```

ونريد استدعاه من نيم سبيس أخرى فسنكتب التالي

كود

```
Console.WriteLine(ConsoleApplication1.Program.x);
```

بكل بساطة سوف يطبع الرقم **200** الذي أعطيناه للمتغير **x**

وكما قلت إن الميثود للبرنامج هي **Main**

وهي كالتالي

كود

```
Public Static Void Main()
```

```
{
```

```
// the code
```

```
}
```

وسوف يتم تنفيذ الكود المكتوب

الآن نفرض إن لدينا الأمور التالية

ns = Name Space

ns < IO

ns < text

ns < ns , Class < ns , namespace < thread

ونريد الان ان نصل الى ال Class الذي بداخل ال Thread فكيف نصله ؟

نكتب الكود التالي

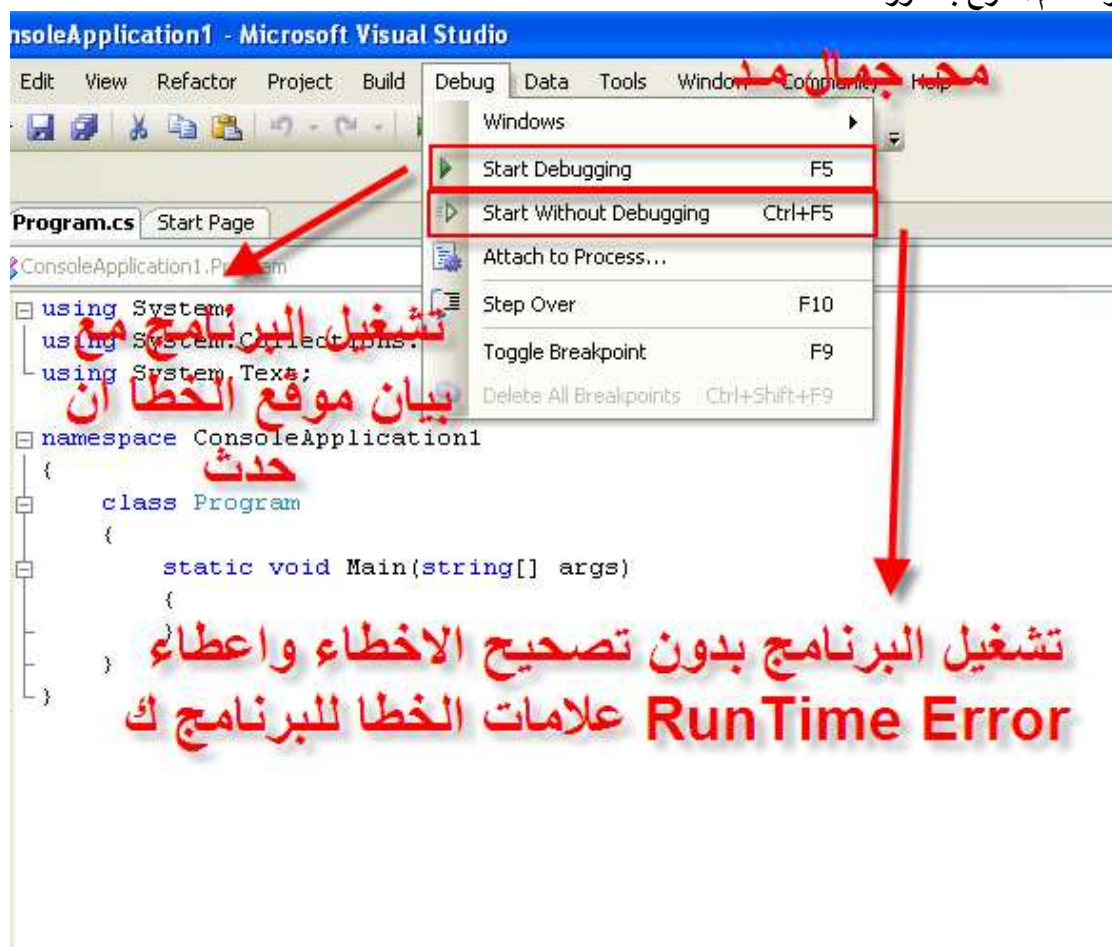
كود

Thread.Namespace.Class

طبعا مع مراعاة تغيير اسم النيم سبيس إلى النيم سبيس الخاصة ببرنامج وكذلك اسم الكلاس

طبعا لتشغيل برنامجك يوجد اكثر من طريقة

وسأدعم الشرح بالصور



تصحيح أخطاء إي يعطيك موقع الخطأ ونوع الخطأ وشرح مفصل عنه

وبدون تصحيح أخطاء إي يعطيك انه يوجد خطأ ويخرج من البرنامج إي انه ينفذ البرنامج كما لو إن المستخدم يقوم بالعمل عليه

إلى هنا نكون قد أنجزنا موضوع



Programs Basic c#

Value Type In .Net

بسم الله الرحمن الرحيم

اليوم موضوعنا أخواني الأفاضل عن القيم وأنواعها في الدوت نت بشكل مفصل

وسوف أبدا حديثي بالمثال التالي

كود

```
Int x = 10; // in C#
```

```
Dim x as Integer = 10' in vb.net
```

في المثال السابق قمنا فقط بعملية تعريف متغير x من نوع **Integer** وأعطيناه القيمة 10 في كلا المثالين

ولكن تختلف الطريقة بالتعريف

في المثال السابق لو تعمقنا في الموضوع قليلا لوجدنا إن طريقة تعريف المتغير من نوع رقمي في السي شارب يلي هو **Int** والتعريف في **VB.NET** يلي هو **Integer** في ال **Compiler** كله هذا يرجع لأصله

وكله يرجع ل **System.Int32**

يعني لو قمنا بالتعريف بالشكل الآتي

كود

```
Int32 x = 10;
```

```
dim x as int32 = 10
```

فلن يعطينا إي خطأ يذكر على العملية لأنها عملية سليمة جدا

الآن نترك VB.NET ونبدأ C#.NET

ألان سأضع جدول به اسم المتغير في ال Common Type System

CTS

وطريقة تعريفها في C#

alias in C#

والحد الأقصى لما تحمله من قيم

Rang

كود

CTS	Alias	Rang
Int32	Int	-231 to 231 -1
Int16	Short	-215 to 2 15 -1
Int64	Long	-263 to 2 63 -1
Boolean	Bool	True , False
Char	Char	2 byte
Single	float	4 byte
Double	Double	8 byte

وهنا صورة توضيحية للأنواع

CTS	Alias	Rang
Int32	<u>Int</u>	-2^{31} to $2^{31} - 1$
Int16	Short	-2^{15} to $2^{15} - 1$
Int64	Long	-2^{63} to $2^{63} - 1$
<u>Boolean</u>	<u>Bool</u>	True , False
Char	Char	2 byte
Single	float	4 byte
Double	Double	8 byte

الأنواع في Framework هي كالتالي

Type in Framework

Value type

مثال عليها

كود

```
int x = 10;
```

Reference type

مثال عليها

كود

```
String x = "Mohammad Jamal";
```

وهنا صورة توضيحية للعملية

```
Int x = 10;
```

```
String str = "Mohammad Jamal";
```

Stack		Heap
X [10] Value Type		
Str	Reference Type	Mohammad Jamal

شكل المتغيرات في الذاكرة

تعريف قيم المتغيرات
يمكننا تعريف المتغيرات كالتالي
متغير من نوع رقمي

كود

```
Int x;
```

هنا يحجز مساحة معينة لهذا المتغير في الذاكرة ولكننا لم نقم بوضع قيمة للمتغير
فكيف يمكننا إعطاءه قيمة

يمكننا إعطاءه كالتالي

كود

```
int x = 10;
```

حيث أعطيناها القيمة 10

المتغيرات من نوع الرقمي **int** لا يمكننا ان نضع فيها سوى الأرقام فقط

الآن نأتي لنوع جديد من المتغيرات
وهو المتغير النصي
String

كود

```
string str;
```

هنا عرفنا متغير من نوع نصي اسمه **str** وهو جاهز لاستقبال إي قيمة يتم وضعها بداخله من نصوص
ولوضع القيم بداخله يجب وضعها بين علامتي تنصيص
""

كالتالي

كود

```
string str = "ArabMoheet";
```

هكذا يتم التعريف عنها
حسنًا الآن كل شيء تمام

ماذا لو أردنا إن نعرف متغير يرجع لنا قيمة True OR False

في هذه الحالة تستخدم متغيرات من نوع **Boolean**
ويمكننا تعريفها كالتالي

كود

```
Bool b = False;
```

في المثال السابق لو طبعنا قيمة المتغير **b** لطبع كلمة **false** لأنها هي القيمة التي أعطيناه إياها

السؤال الآن هل يمكن إعطاء المتغيرات من نوع **Boolean** القيمة **null**

في **.NET**. الاجابه أصبحت نعم

فكيف ذلك

بساطة يمكن ذلك بطريقتين
الأولى :

كود

```
Bool?b=null;
```

والثانية:

كود

```
Nullable<bool>b=null;
```

بعض الأمثلة للمتغيرات التي يمكن تعريفها

كود

```
int x = 10;
```

```
string y = "Mohammad jamal";
```

```
bool n = True;
```

```
long b = logn.MaxValue();
```

```
float f = 22;
```

لاحظنا في المثال السابق انه وضعنا قيمة رقمية في أكثر من منطقة

وضعناها في ال Long وفي ال Float

فبماذا تختلف هذه الدوال

ببساطة تختلف في سعة الأرقام

ف long هي الأكبر في السعة

وبعدها ال int

وبعدها ال float

يعني لو عندنا رقم كبير يلزمه تعريف من نوع long

والأكبر بينهم جميعا هو Double

إذا أردت إن ترى القيم يمكنك ذلك عن طريق كتابة الكود التالي:

كود

```
long b = logn.MaxValue();
```

كود

```
double x = double.MaxValue();
```

كود

```
Int x = Int.MaxValue();
```

وعلى هذا المنوال

الآن مثلاً أردنا إن نقوم بتحويل نوع متغير من نوع **Int** إلى نوع **Long**

فيمكننا ذلك بكل سهولة كالتالي

كود

```
Int x = 500;  
long z = x;
```

فتتم العملية بدون أي مشاكل تذكر لأن ال **long** اكبر من ال **int** لذلك يحوي قيمته
ولكن ماذا لو أردنا إن نقوم بتحويل متغير من نوع **long** إلى نوع **int**

كود

```
long x = 500;  
int z = x;
```

هل سينجح هذا الكود !!!!!

بالتأكيد لا لأن ال **long** اكبر من ال **int** لذلك لا تستطيع إن تحويها

فماذا نفعل بهذه الحالة

يمكننا استخدام طريقة ال **Explicit**

وهي كالتالي

كود

```
long x = 1000;  
int z = (int)x;
```

او

كود

```
long x = 2323;  
int z = convert.ToInt32(x);
```

هذه هي العملية بشكل كامل ولن يحدث أي مشاكل ما دامت قيمة المتغير في **long** اقل من القيمة القصوى ل **int**

Read Number From Keyboard

الآن سنقوم بإنشاء برنامج يقوم بطباعة ما تكتبه له مرة أخرى

ولتقوم بذلك قم بإنشاء مشروع Console Application

واكتب الكود التالي داخل ميثود ال Main

اي بين القوسين

كود

```
String Str = Console.ReadLine();  
Console.WriteLine(Str);
```

الكود الأول

كود

```
String Str = Console.ReadLine();
```

يقوم هذا الكود بفتح المجال للمستخدم على الشاشة السوداء بكتابه ما يريد
وبعد الضغط على زر **ENTER** يقوم بحفظ كل ما كتب في المتغير **Str**

كود

```
Console.WriteLine(Str);
```

يقوم هذا الأمر بطباعة القيمة المدخلة في المتغير **Str**
برنامج سهل للغاية

الآن نريد إن ندخل رقم وبعد الدخالة سوف نقوم بضمه بالعدد 10 ونظهر الناتج

الكثير منا سوف يعتقد إن الكود الصحيح هو كالتالي

كود

```
Int x = Console.ReadLine();  
int y = x * 10;  
Console.WriteLine = y;
```

ولكن في الحقيقة هذا الكود خاطئ تماما وسوف يعطيك عليه النظام علامة خطأ

ألان ماذا سنفعل في هذه الحالة !!!!!!!!!!!!!!!

سأخبركم.....

أولا: سوف نقوم بكتابة الكود التالي

كود

```
Int x = Int.Pares(Console.ReadLine());  
int y = x * 10;  
Console.WriteLine(y);
```

النقطة الغير مفهومة للجميع هي الكود الأول

كود

```
Int x = Int.Pares(Console.ReadLine());
```

فماذا يعني هذا الكود

يعني إن البرنامج سوف يحول إي قيمة مدخلة في الشاشة إلى قيمة رقمية ليتعامل معها وبالتالي يستطيع إن يقوم بعمليات الجمع والطرح والضرب والى آخر هذا الكلام

يمكننا إن نقوم ذلك على ال **Double** وليس فقط على ال **Int**

كود

```
double x = double.pares(Console.ReadLine());
```

Enum

وكيفية التعامل معه

سوف أضع لكم مثال كامل لتفهموه بسهولة

كود

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    enum Vare {emp1,emp2,emp3};

    class Program
    {
        static void Main(string[] args)
        {
            Vare v;
            v = Vare.emp1;
            Console.WriteLine(v);
        }
    }
}
```

الآن ال **Enum** قامت بعملية إضافة عناصر
إي جعلت المتغير **vare** يحوي العناصر ما بين القوسين

ويمكننا ان نقوم بتعريف أي متغير من نوع

VAer

الذي وضعناه في ال

Enum

وبعدها نضع للمتغير أي قيمه من

vaer

هكذا انتهى موضوع اليوم
الذي كان بعنوان

Value Type In. Net

متمنيا إن تكونوا قد استوعبتم ما كتبته

Statement In C#.Net

بعد الانتهاء من المواضيع السابقة التي كانت كوجبة خفيفة لنا لننتعود على طبيعة C#

سنبدأ اليوم بموضوع مهم جدا في C#

وهو ال Statements

تنقسم ال Statements إلى ثلاثة أقسام

Selection - 1

Interaction -2

Jump -3

الآن سنفصل بكل واحد منها

Selection-1

If -1

Switch 2

Interaction-2

While -1

Do while -2

For -3

Foreach -4

Jump-3

Break -1

Continue -2

Goto -3

وسنشرح كل واحدة منهن بالتفصيل الممل مع طرح بعض الأمثلة عليهن

أولا :

1-If Statements

وهي تستخدم للتحقق من أمر ما وإذا كان الأمر صحيحا فإنها ستبدأ بتنفيذ الأوامر الموكلة إليها
والصيغة العامة لها

كود

```
If (something exist)
{
// do some thing
}
else if (something exist)
{
// do some thing
}
else
{
// do some thing
}
```

أي أنه يمكنك وضع شروط بشكل غير محدود وإن لم تنفذ الشروط يمكنك وضع الأمر في كلمة **else** التي

الآن سنأخذ مثال بسيط عليها من تطبيقات ال Console Application

أولا :

سنفتح مشروع Console Application

ثم سنقوم بكتابة الكود التالي داخلة

كود

```
string str = console.readline();  
if (str == "Mr.Asp")  
{  
    console.writeline("Welocme Mr.Asp")  
}  
  
else if (str == "arabicprogrammer");  
{  
    console.writeline("welcome arabic programmer");  
}  
  
else  
{  
    Console.Writeline("I Do Not Know U");  
}
```

في الكود السابق قمنا بالتحقق مما أدخله المستخدم
فإذا كان ادخل كلمة mr.asp فانه يقوم بالتحقق منها والقيام بطباعه رسالة ترحيب بهذا المستخدم

إما إذا كانت Arabic programmer
فانه أيضا يقوم بالترحيب بهذا المستخدم

إما إذا كانت غير ذلك فانه يخبر المستخدم انه ليس على علم به

كأنه هذا مثال بسيط لتطبيقات هائلة يمكننا ان نقوم بها من خلال هذه الجملة البسيطة

2- Switch

تقوم هذه الجملة بأخذ قيمة متغير ومقارنته بالنتائج المخزنة له
وسنعطيك مثال على هذه الجملة

كود

```
string str = console.readline();

switch (str);
{
case "mr.asp" :
console.writeline("welcome mr.asp");
break;

case "arabic programmer" :
console.writeline("welcome arabic programmer");
break;

default :
console.writeline("i do not know you");
break;

}
```

في هذا المثال قمنا بإدخال قيمة معينة للمتغير **str** من خلال المستخدم وقمنا بمقارنتها

فإذا كانت قيمة المتغير **str = mr.asp**

فانه يقوم بطباعة كلمة الترحيب

welcome mr.asp

إما إذا كانت **Arabic programmer**

فانه يقوم بطباعه كلمة الترحيب

welcome Arabic programmer

إما إذا كان غير ذلك إي default

فانه يطبع كلمة

i do not know you

وطبعا كلمة break بعد كل واحدة هي ضرورية جدا

لأنه بعد تحقيق الشرط يتوقف البرنامج

وبهذه نكون انهينا ال Selection من ال Statements

2- interaction

1- while

تقوم جملة **while** بتنفيذ أمر معين استناد إلى شرط إذا كان قائما

إي مثلا

افعل هذا الأمر ما دام هذا الشرط غير محقق

فيقوم البرنامج بإعادة الشرط بشكل مستمر حتى يتحقق الشرط وتتوقف

وسنأخذ مثال عليه :

كود

```
int x = int.prase(console.readline());
```

```
while ( x > 0)
{
console.writeline(x);
x --;
}
```

في المثال السابق قام البرنامج بما يلي

جعل المستخدم يدخل رقم ويقوم بتحويله إلى نوع **Integer** ليتعامل معه على هذا الأساس كما قلنا في الدرس الماضي

ولنفرض إن المستخدم ادخل الرقم 10

فسيقوم البرنامج بالتحقق إذا كان $x < 0$ فسينفذ الأمر

وبما إن $x = 10$ إي إن x اكبر من 0 إذن سيقوم البرنامج بتنفيذ الأمر المكتوب وطباعه إي قيمة تحددها وهنا نحن حددنا

إن يطبع قيمة x لنقوم بتتبع البرنامج

وبعد إن ينفذ الحلقة يصل إلى أمر

كود

X--;

إي قم بتنقيص واحد من ال X

ليصبح 9

ويعود من جديد إلى الأعلى والتحقق والتنفيذ وتنقيص واجد فيصبح 8 وهكذا إلى إن يصبح 0 ويتوقف الشرط

ملاحظة هامة جدا :: يجب إن تقوم بتنقيص القيمة لأنه إذا قمت بتشغيل البرنامج دون إن تقوم بعمل X--; رح يعلق البرنامج

ويظله يطبع بشكل غير محدود لان الشرط ما زال قائما

لذلك احترس

هكذا انتهينا من ال While

2-Do While

ماذا تختلف هذه عن جملة **while**

سوف اشرح لكم صيغتها وانتم ستفهمونها

كود

```
int x = 1;
do
{
    console.WriteLine("mr.asp");
    x --;
}

while ( x > 0);
```

هل لاحظتم الفرق ???

حيث إن جملة **do while** تقوم بتنفيذ الكود مرة واحدة على الأقل قبل التأكد من صحة الشرط

إي تنفذ الكود بداية ثم تقوم بالتحقق وهكذا

وهذه أيضا انتهينا منها

3-For

جملة التكرار **For**

تستخدم للقيام بتكرير شيء معين عدد معين من المرات

مثال

كود

```
For ( int x = 0; x = 10; x ++);
{
    console.WriteLine(x);
}
```

تشبه قليلا جملة ال **while** في عملها ولكنها أسهل قليلا

أولا: قمنا بتحديد قيمة المتغير **x = 0**

وقمنا بتحديد نقطة النهاية التي هي **x = 10**

وقمنا بتحديد مقدار الزيادة في كل مرة إي كل مرة سوف يزيد واحد

كود

```
x ++;
```

تعني

كود

```
x = x + 1;
```

او

كود

```
x +=1;
```

وفي كل مرة يقوم بتنفيذ الأمر بين الأقواس حتى يتحقق الشرط يلي هو $10 = x$
ملاحظة وقعت فيها في بداياتي مع C# ولا أود إن يقع بها احد من طلابنا الأعزاء وهي ال **for** المتداخلة
وسأعطيكم مثال عليه

كود

```
for ( int x = 0; x = 10; x ++)  
{  
for ( int y = 0; y = 10; y ++)  
{  
Console.WriteLine(y);  
}  
}
```

عند تنفيذ هذا الكود سوف تلاحظون إن البرنامج طبع الأرقام من 1 إلى 10 10 مرات
فماذا يعني هذا

سوف اشرح لكم

*

*

79/36

إصدار تجريبي <2>

أولاً: **for** الأولى صاحبة المتغير **x** عند تنفيذ الأمر يلي بداخلها يلي هو **for** صاحبة **y** سوف يقوم البرنامج بتنفيذ **y** حتى

تصل إلى 10 ومن ثم يزيد قيمة 1 إلى **x**

إي مع تنفيذ حلقة **y** لفة كاملة يتم زيادة 1 لحقة **x**

وهي تستخدم عادة في المصفوفات الثنائية والثلاثية وغيرها

انتهينا من ال **for**

4- Foreach

وهي تستخدم عادة لطباعه عناصر المصفوفات وسأخذ مثال عليها

كود

```
int [] ar = {1,2,3,4,5,6,7,8};
```

```
foreach ( int x in ar)
```

```
{
```

```
console.WriteLine(x);
```

```
}
```

تعريف وإدخال المصفوفة في الوقت الحالي غير مطلوب منك ولكن انظر ماذا فعلت **foreach**

أولاً: قمنا بتعريف نوع البيانات التي سنسحب منها البيانات التي هي من نوع **int**

وقمنا بوضعها في المتغير **x** من **ar**

وقمنا بطباعه **x**

سيقوم البرنامج بطباعه كافة عناصر المصفوفة المدخلة

مثال بسيط لفهمها

الآن سننتقل إلى جمل ال **Jump**

3-Jump Statements

and continue break

كود

```
int x = 0;
while (true)
{
    console.WriteLine(x);

    x ++;

    if ( x < 10)
    {
        continue;
    }
    else
    {
        break;
    }
}
```

اعتقد إن المثال واضح حيث تقوم جملة ال **Continue** بجعل الكود يتابع نفسه من نفسه
وجملة ال **break** تقوم بإيقاف الكود عن التنفيذ

1-Goto statements

كود

```
int x = int.parse(console.readline());
```

```
if ( x % 2 ==0) goto Eevent;
```

```
console.writeline("odd");
```

```
goto end;
```

```
Eevent:
```

```
console.writeline("Eevent");
```

```
end;;
```

عارف انو الوجبة دسمة شوي على المبتدئين ومش فاهمين أي شيء من هالجمله بس ما عليكم
إن شاء الله ما تطلعوا إلا فاهمين ومبسوطين

ألان إحنا قمنا بتعريف متغير x من أنواع **integer** وجعلنا المستخدم يدخل القيمة داخلة بشكل تلقائي

ألان نفرض انو المتغير x قيمته 10

فماذا سيفعل البرنامج

رح يشوف إذا باقي القسمة على 2 هو صفر إذن الرقم زوجي
إذن عليه إن يبدأ بالتنفيذ من مكان معين

كيف لنا إن نحدد المكان الذي سينتقل إليه البرنامج لتنفيذه

ببساطة نقوم بالتعريف التالي

Eevent:

نكتب اسم المكان الذي سينتقل إليه الكود لتنفيذه وتنفيذ ما بعده ومن ثم نقطتين فوق بعض :

إي سنكتب كود الانتقال للتنفيذ هكذا

كود

```
goto Eevent;
```

وسيتجاهل كل الكودات يلي قبل ال **Eevent**:

وهكذا إذا كان الرقم زوجي سيذهب إلى نقطة ال **Eevent** ويطبع كلمة **Eevent**

إما إذا كان فردي فلن ينتقل

وسيطبع كلمة **odd** إي فردي

ومن ثم يذهب إلى نقطة
end;

التي سبق وذكرنا كيف يتم تحديدها

لاحظتم إن نقطة الوصول إلى ال **End** كتبت هكذا

End : ;
وليس فقط
End:

فلماذا

بكل بساطة

End::;

تعني انه لا يوجد كودات بعدها لتنفيذها

لذلك تتوقف عندها

وهكذا انتهينا من موضوع ال Statements

Exception Handling

اليوم موضوعنا عن تتبع واصطياد الأخطاء في **C#**
وهذه العملية سوف تصطاد الأخطاء الناتجة عند تشغيل البرنامج

أولاً: سأشرح لكم عملية اصطياد الأخطاء بشكل مفصل

1- يقوم الكومبايلر بعملية **Run** للبرنامج في **CLR**

مثلاً يقوم بفتح ملف **Open File**

في ال **OS**

الآن عند حدوث خطأ سوف يقوم ال **OS** بإرجاع معلومات حول الخطأ ال **CLR**

و ال **CLR** بدورة يضعها في ال **Heap**

ونحن نقوم بتعريف متغير في ال **Stack** يقوم بالتأشير على المعلومات في ال **Heap**

ويقوم بإظهارها لنا على شكل نص

مثال :

كود

```
Try
{
// do Some Thing
}

Catch ( Exception ex)
{
Console.WriteLine(ex.Message);
}
```

في هذا الكود سوف يقوم البرنامج بتنفيذ الكود وأي خطأ سوف يقوم بوضعه في المتغير **ex** ونقوم نحن بإظهاره بالشكل الذي نريده

يمكنك توقع نوع الخطأ أيضاً وإظهار الرسالة التي أنت تحددها

79/41

إصدار تجريبي <2>

مثلا :

كود

```
Try
{
// do some thing
}
catch (nameexception ex)
{
console.writeline("about error");
}
```

حيث ان name exception هو اسم الخطأ المتوقع حدوثه

هكذا انتهينا من ال Exception Handling

Methods

أمثلة على الميثود

Main

ReadLine

WriteLine

·
·
·

والكثير منها التي نستخدمها في C#

كيف يمكننا تعريف **Methods**؟؟

هناك طرق كثيرة في التعريف

وعناصرها التي تبدأ بها كالتالي

Public

Private

Protected

Internal

Protected Internal

كل من هذه **الميثود** يملك عمل معين سوف نشرحه في دروس لاحقة إن شاء الله

الآن سوف نقوم بعملية تعريف **ميثود** في **C#**

كود

```
Public Static Void Write()  
{  
int x = int.pares(Console.ReadLine());  
x ++  
Console.WriteLine(x);  
}
```

ولاستخدام هذا **الميثود** نضع في **Main** كلمة

كود

```
Write();
```

لاحظ انه يمكننا كتابة هذا الكود في ال **Main**

ولكننا قمنا بكتابته **بميثود** وقمنا باستدعائه

حيث إن **الميثود** يقوم بمساعدتنا بشكل كبير في عمليات تنظيم البرمجة

مثال آخر:

كود

```
Public Static Double M(int x);  
{  
Return math.Pow(x,3);  
}
```

وفي ال **Main** نكتب التالي:

كود

```
Console.WriteLine(M(10));
```

أنا عارف انو الدماغ **بلش** يضرب عند البعض

ولكن ما علينا سوف اشرحه

الآن قمنا بتعريف المتغير **x** في **الميثود M** وعرفناه من نوع رقمي **int**

وعندما كتبنا جملة

كود

```
Return math.pow(x,3);
```

قمنا بإرجاع قيمة **للميثود M**

تعمل ضمن المعادلة التالية

math.pow تعني الاس في الرياضيات

أي الرقم الأول هو الرقم الذي سيتم تكراره على عدد الرقم الثاني

مثال فرعي:

كود

```
math.pow(10,3);
```

حيث سيقوم بضرب العشرة بنفسها 3 مرات

وفي المثال السابق سوف يقوم بضرب ال x بنفسها 3 مرات

ولكن من أين سنحصل على قيمة ال x

بكل تأكيد من خلال هذه الجملة

كود

```
M(10);
```

حيث قمنا بإرجاع القيمة للميثود **M** وهو المتحكم بالقيم

وعند وضع

كود

```
M(10);
```

سوف يقوم بوضع قيمة ال 10 بدل المتغير x

وبإظهار النتيجة ضمن المعادلة الموضوعه

الآن كيفية طرق إرجاع المعلومة في الميثود

أولا : الطريقة العادية التي استخدمناها في المثال السابق

ثانيا : Reference

ثالثا : Output

مثال على Reference

كود

```
Public Static Void M1(ref int x);  
{  
    x +=10;  
}  
  
static void main()  
{  
    int input = 10;  
    M1(ref input);  
    Console.WriteLine(input);  
}
```

ال Output سنشرحها لاحقا وال reference ولكن الآن أخذنا مقدمة بسيطة عنها

هكذا انتهينا من ال Methods

Arrays

المصفوفات في C#

تعريف المصفوفة

كود

```
Type [] namearray
```

مثال

كود

```
Int [ ] ar;
```

تعريف المصفوفة وتحديد أبعادها

كود

```
int [ ] ar;
```

كود

```
ar[ ] = new int [5];
```

بعد واحد فقط

لتعريف المصفوفة بأكثر من بعد

كود

```
Int [ , ] ar2 = new int [ 2, 3];
```

أي إن بعد هذه المصفوفة 3×2

ولإدخال القيم للمصفوفة كالتالي

كود

```
int [ ] ar = {1,2,3,4,5,6,7,8};
```

أو

كود

```
int [ ] ar = new int [ ] {1,2,3,4,5,6,7,8,9};
```

أو

كود

```
char [ ] ch = new char [ ] {'n','r','j'};
```

أو

كود

```
string [ ] str = new string [ ] {"jamal","arab","programming","mohammad"};
```

ar.Rank : لمعرفة إبعاد المصفوفة

ar.lenght : لمعرفة عدد عناصر المصفوفة

ولتحديد عنصر في المصفوفة وطباعته نستخدم الكود التالي

كود

```
Console.WriteLine(ar[0]);
```

وهكذا

لإدخال العناصر لمصفوفة ثنائية الأبعاد كالتالي

كود

```
int [, ] ar = new int [,] {{1,2,3,4},{5,6,7,8}};
```

ولمعرفة العناصر كالتالي

كود

```
ar[1,3];
```

ولطباعة كافة عناصر المصفوفة نستخدم حلقة التكرار كالتالي

كود

```
For ( int x = 0; x <= ar.GetLength(x++);  
{  
console.WriteLine(ar[x]);  
}
```

وللقيام بعملية نسخ للمصفوفة نقوم بالتالي

كود

```
Int [] copy = (int[]) ar.Clone();
```

لترتيب العناصر في المصفوفة من الاكبر للصغر

كود

```
Array.Sort(ar);
```

ولعكسها

كود

```
Array.Reveres(ar);
```

هكذا انتهينا من Array

Collection

collection مجموعة من الميثود تستخدم Name Space

كود

```
Using System,Collection
```

وهي تقسم إلى أنواع سأذكر منها أهمها

اولا :

Array List

وهي مصفوفة ولكنها تختلف عن المصفوفة العادية بأنها تقوم باستقبال كافة أنواع البيانات على شكل Object

لنرى مثال عليها

كود

```
ArrayList ls = new ArrayList();  
ls.Add(122);  
ls.Add('c');  
ls.Add("Mohammad");  
ls.add();  
ls.Add(0.0323);
```

نرى في المثال السابق إن ال array list تستقبل كافة أنواع البيانات

الآن سنقوم بعملية عرض للبيانات من خلال حلقة التكرار **Foreach**
بسيطة

كود

```
Foreach(object el in ls)
{
    Console.WriteLine(el);
}
```

وكله رح يكون تمام ان شاء الله

Queue

نفس ال **array list** تقريبا في الو ضيفة تستخدم القاعدة التالية

FIFO: First in First Out

إي الذي يدخل أولا يخرج أولا
مثال

كود

```
Queue qw = new Queue();
qw.Enqueue(5);
qw.enqueue();

q.dequeue();
```

هنا سوف ترى إن ال **queue** عند القيام بادخال قيمة مثلا تم ادخال قيمة 5 أولا
وعند قيامنا بحذف قيمة فانه سيحذف ال 5 أولا

Stack

نفس فكرة ال **queue** ولكن تتبع الفكرة التالية

LIFO: Last In LIFE First Out

مثال

كود

```
Stack st = new Stack();  
st.Push(5);  
st.push(10);  
st.pop();  
foreach (object c in st)  
{  
    console.writeline(c);  
}
```

فسوف يقوم بحذف الرقم 10 لأنه آخر شيء تم ادخاله

بعكس ال **queue**

هذا بحمد الله نكون قد انتهينا من ال Collection

Object Oriented Essentials

نبذة تعريفية حول ال Object Oriented

البرمجة موجهة الهدف Programming object Oriented

في هذا الجزء نتناول مفاهيم البرمجة الموجهة الهدف OOP من خلال تناولنا للنقاط التالية :

1. مفاهيم البرمجة موجهة الأهداف Oriented Programming object

2. الأهداف objects والفصائل Classes

3. دالة البناء Constructor

4. التحميل الزائد لدوال البناء Constructor Overloading

مفاهيم البرمجة موجهة الأهداف Concepts OOP

وكما هو معروف فإن البرمجة موجهة الأهداف تعتمد أو تقوم فكرتها على 4 مبادئ رئيسية زهي :

1. الكبسلة Encapsulation

2. إخفاء البيانات Hiding Data

3. الوراثة Inheritance

4. تعدد الأشكال Polymorphism

لنبدأ بإلقاء نظره على ما تعنيه هذه المبادئ بشيء من الإيجاز ..

اولا : الكبسلة Encapsulation

لكي نفهم فكرة الكبسلة ممكن إن نأخذ مثال وليكن صندوق جهاز الحاسوب box والذي يتكون من اللوحة الأم motherboard .. والتي بدورها تتكون من دوائر وما الى ذلك .. ويتكون الجهاز أيضا من Power والذي هو وحده مكونه من أجزاء داخلية .. ومن ثم هناك معالج والذي عبارة عن جزء أو وحدة مكون من عدة أجزاء .. وبالنهيأة سنلاحظ إن جهاز الحاسوب مكون من تجمع أجزاء وهذه الأجزاء كل منها مكون من أجزاء ووحدات داخلية .. ومن المثال نستطيع إن نقول إن الكبسلة خاصية إن إي كائن يتكون من وحدات داخلية تتكامل مع بعضها لتكوين الكائن الأكبر

ولاحظ انك من مبدأ الكبسلة تستطيع إن تحقق مبدأ إخفاء البيانات

ثانيا : إخفاء البيانات Hiding Data

وهي خاصية كأن نقول مثلا انك تستطيع استخدام أي جهاز جوال دون الحاجة لمعرفة طريقة الصنع والتفاصيل الفنية لجهاز الجوال الذي تستخدمه .. فقط يكفيك انك تستطيع استخدامه والتعامل معه.

ثالثا : الوراثة Inheritance

تعتمد فكرتها على أمكانية إعادة استخدام كود البرمجة لبناء أجزاء جديدة من اكود سابقة دون الحاجة لان تعيد الكتابة من الصفر وبالطبع الكود الجديد سيحتوي تفاصيل الكود القديم ..
وكمثال لذلك .. إي إصدار لبرنامج حاسوب فانه يكون يحتوي على تفاصيل مشابهة للإصدار السابق لان مبرمج البرنامج لن يعيد برمجة البرنامج إنما سيطور عليه مع زيادة تفاصيل ..
ويمكننا ان نأخذ مثال أكثر قربا وهو السيارة. فالسيارة لموديل هذا العام لن تختلف عن موديل العام السابق بالخصائص إنما ستزيد الشركة على الموديل السنة الماضية المزيد من التفاصيل الجديدة والمميزات فبالطبع لن تعيد الشركة المصنعة التصميم وتنفيذ أجزاء السيارة بالكامل ولكنها ستقوم بالتعديل والتطوير على الموديلات السابقة وإخراجه بشكل جديد وبالتالي فان الوراثة ستوفر الكثير من الوقت والجهد ..

رابعا : تعدد الأشكال Polymorphism

تعتمد فكرتها على ان الكائن يتصرف بطريقة مختلفة على حسب الموقف الذي يستدعي الرد عليه مثلا لو أن شخصا قام بتحريك فسوف تحيه وتسره منه .. بينما لو قام نفس الشخص بإيذاءك وشتمك فانك بالطبع سيكون لك تصرف آخر غير السرور .. رغم انك لم تتغير والشخص هو نفسه إلا ان المواقف تجعلك تغير من تصرفك حسب الموقف الذي تتعرض له ..

الأهداف Objects والفئات Classes

لنفهم معنى objects دعنا نأخذ مثال..

بفرض انك تريد ان تبني بيتا بلعبه اللي جو (المكعبات) بالطبع بالبداية ستحضر (مكعبات كافيه للبناء ونماذج الشبائيك ونماذج الأبواب ...) ومن ثم تبدأ بتجميع هذه الأجزاء أو العناصر لبناء أجزاء البيت إلى ان تنتهي منه .

وهو يشبه تماما كيفية بناء البرامج باستخدام البرمجة موجهة الأهداف OOP فكل عنصر من عناصر البناء إنما هو عبارة عن هدف object والبرنامج (كالبنت) عبارة عن مجموعة من الأهداف تتفاعل مع بعضها البعض

ولكن عليك ان تلاحظ انك بلعبه المكعبات لا تملك تصنيع احد العناصر كان تقوم بصنع مكعبات البناء بنفسك إنما تستخدمها جاهزة .. ولو كان بإمكانك ان تصنعها فانه لكي تستطيع ذلك عليك ان يكون لديك قوالب التصنيع ليتسني لك ان تصب المادة الخام بداخله للحصول على العنصر .. وبالتالي لابد ان يكون لكل عنصر قالبه الخاص فالأبواب لها قالبها والشبائيك كذلك

والمكعبات أيضا كل له قالبه الخاص ..

وهذا القالب هو ما يمثل الفصيلة **Class** بالبرمجة موجهة الأهداف .. ويمكنك بناءها ببرنامجك لتنتج العناصر **Objects** التي تريدها

وبالطبع الفصيلة الواحدة تستطيع إن تنشئ منها أي عدد من الأهداف **objects**

كما انك عليك ملاحظة إن الفصيلة **class** في لغة **C#** تتكون من جزأين وهما

1. الخصائص **Attributes** وهي البيانات **data** أو المتغيرات **variables member**

2. السلوك أو بمعنى آخر الوظيفة **Behavior** وهي الدوال **member functions** أو **member method** وهو

الاسم الغالب في لغة **C#**

لنأخذ مثال على ذلك وهو إنسان .. فالإنسان له عدة خصائص منها الاسم والعمر ...

وله وظائف : يتحرك ويتعلم ويفكر و ...

دعونا نرى كيف يمكننا إن نكتب الكود للفصيلة الإنسان **Class Human**

قم بفتح مشروع جديد من نوع **Console**

*

*

*

واكتب به الكود التالي ..

*

*

*

*

*

كود

```
using System;
namespace HumanBeing
{
    class human
    {
        public string Name;
        public int Age;
        public void Move()
        {
            System.Console.WriteLine("I move");
        }
        public void think()
        {
            System.Console.WriteLine("I think");
        }
        public void Learn()
        {
            System.Console.WriteLine("I learn");
        }
    }
}
class CMain
{
    static void Main1(string[] args)
    {
        human H = new human();
        H.Name="Ali";
        H.Age=15;
        System.Console.WriteLine("My name is "+H.Name);
        System.Console.WriteLine("I am "+H.Age+" years old");
        H.Move();
        H.Learn();
        H.think();
        System.Console.ReadLine();
    }
}
```

لو قمنا بتنفيذ المشروع سيكون لدينا الناتج التالي ..

```
My name is Ali
I am 15 years old
I move
I learn
I think
-
```

المحيط العربي

وكما لاحظتم فإنه لتعريف هدف **object** من الفصيلة **class** نتبع طريقة التعريف التالية:

كود

```
className objectName = new className();
```

وهي الصيغة التي عرفنا بها الهدف **H** من الفصيلة **human**

ولاستدعاء إي خاصية وإسناد قيمه لها فما علينا إلا إتباع الصيغة التالية :

كود

```
objectName.variableName=value;
```

ونفس الشيء عند استدعاء دالة من فصيلة

كود

```
objectName.methodName();
```

أحب إن الفت الانتباه إن **Console** إنما هي فصيلة من فصول المكتبة **namespace** التي تسمى **System**
إن كنت لم تكتب ببداية الكود

كود

```
Using System;
```

فيجب عليك استدعاء الدالة Console بالطريقة التالية:

كود

```
System.Console.WriteLine();
```

هنا نكون قد استدعينا الدالة write Line من الفصيلة Console التي توجد داخل المكتبة (النطاق) System

دالة البناء Constructor

في الدروس السابقة كنا نعرف متغيرات ونعطيها قيم ابتدائية ..
كما كان يمكننا إن نغير قيم المتغيرات متى نشاء في البرنامج

كود

```
int i=0;
```

```
.  
. .  
. .
```

```
i=10;
```

فقد كانت القيمة الابتدائية مجرد ضمان منا لوجود قيمه محدد بالمتغير إذ انه يمكننا إن نغيرها وقتما نشاء .

ولكن كيف يمكننا فعل ذلك بالفصائل **classes** ؟؟

بالواقع يوجد داله خاصة تتوفر بالفصائل **classes** تسمى دالة البناء **Constructor** وهي المسئولة عن إعطاء قيم

ابتدائية لمتغيرات الفصيلة حيث يتم استدعائها تلقائيا عندما نقوم بإنشاء هدف **Object** من هذه الفصيلة

من الجدير بالذكر إن دالة البناء يمكنها إن تستقبل أي عدد من المعاملات **Parameters** لتحديد القيم الابتدائية لمتغيرات

الفصيلة Class

الآن قد نتساءل عن أوجه الاختلاف بين الدوال العادية وبين داله البناء ..؟! فهناك عدة نقاط تختلف بها دالة البناء عن

الدوال العادية وهي :

1. لابد إن يكون اسم دالة البناء نفس اسم الفصيلة **class** المعرفة بداخلها .

7/9/00

إصدار تجريبي <2>

2. دالة البناء لا تقوم بإرجاع إي قيمه ولا حتى **void**

مثلا للفصيلة human يمكننا إن نكتب دوال البناء مثل :

كود

```
human()  
human(string name)  
human(int Age)  
human(string Name,int Age)
```

الآن أصبح بإمكانك التعديل على الكود السابق لكي تنشئ دالة البناء كالتالي

كود

```
using System;  
namespace constructor  
{  
class human  
{  
public string Name;  
public int Age;  
public human(string n, int a) // constructor method  
{  
Name=n;  
Age=a;  
}  
public void Move()  
{  
System.Console.WriteLine("I move");  
}  
public void think()  
{  
System.Console.WriteLine("I think");  
}  
public void Learn()  
{
```

```
System.Console.WriteLine("I learn");
}
}
class CMain
{
static void Main(string[] args)
{
human H = new human("Ali",15);
// H.Name="Ali";
// H.Age=15;
System.Console.WriteLine("My name is "+H.Name);
System.Console.WriteLine("I am "+H.Age+" years old");
H.Move();
H.Learn();
H.think();
System.Console.ReadLine();
}
}
}
```

بالتالي ستلاحظ أنك لم تعد بحاجة لإسناد قيم لمتغيرات الفصيلة عند إنشاء هدف بل تمرر القيم مباشرة عند تعريف الهدف

..

لو قمت بتنفيذ البرنامج ستحصل على نفس النتيجة التي حصلت عليها سابقا ..

التحميل الزائد لدوال البناء Overloading Constructor

المعني من ذلك إن نقوم بكتابه عدة صيغ لدالة البناء بعده إشكال وبنفس الاسم طبعا والذي سيكون له نفس اسم الفصيلة ..
ولكن لفعل ذلك علينا إن نفرق بين الواحدة و الاخرى من خلال احدي النقاط التالية :

1. عدد المعاملات Parameters

2. نوع المعاملات ولو كان اختلاف لنوع معامل واحد على الأقل أو حتى اختلاف ترتيب للمعاملات ..
تستخدم داله البناء بإشكالها المتعددة لكي توفر للمبرمج إن يشحن متغيرات الفصيلة لأحد الأهداف بقيم ولآخر بقيم اخرى

...

19/02

إصدار تجريبي <2>

لكتابة دوال بناء يمكننا إن نضف على الكود السابق هذه الدوال داخل الفسيطة human

كود

```
class human
{
    public string Name;
    public int Age;

    public human(string n, int a) // constructor method
    {
        Name=n;
        Age=a;
    }
    public human(string n)
    {
        Name=n;
        Age=0;
    }
    public human(int a)
    {
        Name="Unknown";
        Age=a;
    }
    public human()
    {
        Name="UnKnown";
        Age=0;
    }
    public void Move()
    {
        System.Console.WriteLine("I move");
    }
    public void think()
    {
        System.Console.WriteLine("I think");
    }
}
```

```
}  
public void Learn()  
{  
    System.Console.WriteLine("I learn");  
}  
  
}
```

تم الانتهاء من **Object Oriented Essentials**

Inheritance In C# 2005

الوراثة

إخواننا الأفاضل اليوم سنتكلم عن الوراثة

في

C#.NET

فماذا نقصد بالوراثة

في C# قدمت لنا مايكروسوفت تسهيلات رائعة

حيث يمكننا إنشاء كلاس

وان نقوم بعمل وراثة منه من الكلاس بدلا من إعادة الكود المكتوب

مثال على ذلك

كود

```
Public Class Person
{
Public String Name;
Public int age;
public string id;

public void M1()
{
Console.WriteLine("Person");
}
}

Public Class Employee : Person
{
Public double Salary
public void M1()
```

```
{
Console.WriteLine("Employee");
}
}

public class manager : employee
{
public double com;
public void M1()
{
Console.WriteLine("Manager");
}
}

Static Void Main()
{
Person p = new Person();
Person e = new Employee();
Person m = new Manager();
}
Person [ ] ar = new Person [ ] {p,e,m};
foreach ( person el in ar)
{
el.M1();
}
}
```

إخواني الأفاضل في المثال السابق اعتقد إنني شرحت الوراثة بشكل كامل

إي إننا نلاحظ إننا عرفنا متغير من نوع **person** على ال **employee**

وعرفنا أيضا **person** في ال **manager**

بدون إن يعطينا إي خلل

وذلك لان ال **person** جزء من كل منهما ويمكن تعريفها من خلالهم

تتم عملية الوراثة بوضع نقطتين راسيتين بجانب اسم الكلاس ووضع الكلاس المراد اخذ الوراثة منه

وتنقل كافة المعلومات من الكلاس إلام إلى الكلاس الابن

هذه هي كل الوراثة.....

Index And Properties

سنقوم الآن بشرح مفصل عنها

أولاً: ما فائدتها

Properties & Index ستسهل علينا عمليات كثيرة في النظام يمكننا إن نقوم بها بكل سهولة

حيث يمكننا إدخال قيمة معينة بشكل مستمر وإرجاعها قيمة أخرى حسب ما نريد

كل هذه الأمور تمكنا إياها **الدوت نت** بشكل سهل جداً

وأكثر من ذلك

سنأخذ مبدئياً مثال بسيط للـ **Properties & Index**

أولاً: نضيف كلاس بشكل مبدئي اسميه **Bank Account**

ونكتب فيه الكود التالي

كود

```
Class BankAccount
{
private double balance;
set balance;
get balance;
}
public void setbalance ( double v)
{
}
public void getbalance
{
Return balance;
}
```

في المثال السابق قمنا بعملية تعريف للمتغير **Balance** داخل الكلاسة

وقمنا بوضع عملية **Set** و **Get**

إي يأخذ قيمة معينه

ويرجعها قيمة معينه

ونلاحظ انه قام بإرجاع القيمة كما هي يعني فقط عملنا **Return** ورجع القيمة بشكل تلقائي نفسها

إما رح تلاحظه الاختلاف لو قلنا **2 * Return Balance**;

على كل حال

الآن سنأخذ مثال آخر لتسهيل العملية عليكم

أولا نقوم بعملية إضافة كلاس

كود

```
Private Double Balance;  
Public Double Bal  
{  
Set  
{  
Balance = Math.Abs(Value);  
}  
Get  
{  
Return Balance;  
}  
}
```

طبعا المثال واضح مثل المثال السابق ولكن بطريقة أخرى

قمنا هنا بتعريف **Set** و **Get**

وجعلنا ال **Set** تأخذ القيمة وتحولها لعدد صحيح
وال **Get** ترجعها مثل ما تم التعديل عليها بال **Get**

نلاحظ إن المسألة بسيطة جدا
بس بدها شوية تركيز

كود

```
Private Double Balance;  
Public String ClientName;  
Public String BankName;  
Public String id;
```

```
Public String This [int index]
```

```
{  
Set  
{  
Switch (Index)  
{  
Case 0 :  
ClientName = Value;  
break;
```

```
case 1 :  
bankName = Value;  
break;
```

```
Case 2 :  
id = value;  
break;  
}  
}
```

```
Get  
{  
Switch (Index)  
{  
Case 0 :  
return ClientName;  
break;
```

```
Case 1 :
```

```
Return BankName;
```

```
Break;
```

```
Case 2 :
```

```
Return id;
```

```
Break;
```

```
Default :
```

```
Return "";
```

```
break;
```

```
}
```

```
}
```

```
Public double bal;
```

```
{
```

```
Set
```

```
{
```

```
Balance = math.abs(value);
```

```
}
```

```
Get
```

```
{
```

```
Return balance;
```

```
}
```

```
}
```

```
}
```

في المثال السابق قمنا بعمل مصفوفة
حيث يمكننا ان نقوم باضافة الأشياء على شكل مصفوفة

ولنقم بذلك نصع في ال **Main**

الكود التالي

كود

```
Public static void Main();
{
Bankaccount b = new backaccount();
b.bal = 100;
console.writeline(b.bal);
b.ClientName= "Mohammad";
console.writeline(b.clname);
b[0]= "Ahmad";
console.writeline(b[0]);
}
```

حيث إننا نقوم أولاً بعملية تعريف للكلاس ومن ثم نحدد القيمة للعنصر **bal**

فيقوم بإعطائها القيمة الصحيحة وإرجاعها

ومن ثم قمنا بتحديد قيمة ال **ClientName**

ونطبعها

ومن ثم نقوم بتحديد قيمة من عناصر المصفوفة التي قمنا بإنشاءها **b**

ونطبع العنصر **0** الذي حددنا

هذا كان كل ما لدينا اليوم

البرنامج بشكل كامل

Class1

كود

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ddd
{
    class BankAccount
    {
        private double balance;
        public string clientname;
        public string bankname;
        public string id;

        public string this[int index]
        {
            set
            {
                switch (index)
                {
                    case 0:
                        clientname = value;
                        break;
                    case 1:
                        bankname = value;
                        break;
                    case 2:
                        id = value;
                        break;
                }
            }
        }
        get
        {
```

```
{
    switch (index)
    {
        case 0:
            return clientname;
            break;
        case 1:
            return bankname;
            break;
        case 2:
            return id;
            break;
        default :
            return "";
            break;
    }
}

public double bal
{
    set
    {
        balance = Math.Abs(value);
    }
    get
    {
        return balance;
    }
}
}
```

Program.cs في

كود

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ddd
{
    class Program
    {
        static void Main(string[] args)
        {
            BankAccount b = new BankAccount();
            b.bal = 100.11;
            Console.WriteLine(b.bal);

            b.bankname = "Mohammad";
            Console.WriteLine(b.bankname);

            b[0] = "Ahmad";
            Console.WriteLine(b[0]);
        }
    }
}
```

Operator Overloading

السلام عليكم ورحمة الله وبركاته

سأقوم بتوضيح الفكرة لكم فالكثير منا لا يعرفها

مثال:

كود

```
a = 1 + 3;
```

هذا الكود عادي ولن يعطي البرنامج عليه إي خطأ يذكر

مثال آخر :

كود

```
int x = 5;
```

```
int y = 10;
```

```
int a = x + y;
```

وهذا الكود لو قمنا بتجربته فلن يعطي عليه إي خطأ
الآن ماذا لو قمنا بكتابة هذا الكود:

كود

```
MyClass cl1 = new MyClass();
```

```
MyClass2 cl2 = new MyClass2
```

```
MyClass res = cl1 + cl2;
```

في الكود السابق سوف يقوم بإعطائنا خطأ أكد

فلماذا

لان عملية الجمع غير معرفة في الكلاسات

إما في غيرها من العمليات كتعريف متغير ووضع قيمة فيه وجمعها كل هذه الأمور توفره وجاهزة

إما عمليات جمع الكلاسات فانت من سيقوم بعملية تجهيز لها

سوف نأخذ المثال التالي:

أولاً:

قم بإضافة كلاس إلى المشروع وسمه **Time**

وقم بكتابة الكود التالي بداخله

كود

```
public int Hour, Minute, Second;

public static Time operator +(Time r, Time L)
{
    Time temp = new Time();

    temp.Hour = r.Hour + L.Hour;

    temp.Minute = r.Minute + L.Minute;

    temp.Second = r.Second + L.Second;

    return temp;
}
```

الآن سأشرح الكود التالي

أولا قمنا بعملية تعريف للمتغيرات في الأعلى

Hour

Minute

Second

الآن قمنا بكتابة كود الميثود الجديد وقمنا بإضافة دالة + Operator

كود

```
public static Time operator +(Time r, Time L)
```

سأشرح لكم هذا الكود الذي يعتبر محور الدرس

أولا قمنا بتعريف ميثود اسمه Time من نوع Static

وقمنا بإضافة كلمة Operator+ إي إن هذا الكلاس يستخدم عملية الجمع

وبعدها قمنا بتعريف المتغيرين r , l إلى نوع Time

الآن قمنا بعملية تعريف لمتغير temp من نوع time

وقمنا بإعطاءه القيم

وبعدها يقوم بإرجاع قيمة الـ Time إلى الميثود

الآن انتهينا من شرح الميثود

وسوف نجرب الميثود في الـ main

اذهب الى الـ main

واكتب الكود التالي

كود

```
Time t1 = new Time();  
  
t1.Hour = t1.Minute = t1.Second = 4;  
  
Time t2 = new Time();  
  
t2.Hour = t2.Minute = t2.Second = 5;  
  
Time res = t1 + t2;  
  
Console.WriteLine(res.Minute);
```

أولا قمنا بعملية تعريف متغير t1 إلى الكلاس time

ومن ثم أعطينا القيم لها

وبعدها قمنا بجمع نواتج الكلاسات ووضعناها في المتغير res

ومن ثم طبعتها

لاحظ انه لم يعطنا إي خطأ يذكر عند استخدام هذه العملية مع إننا عرفنا كلاسات وجمعناها

وهنا تكمن أهمية ال **Operator Overloading**

هذا كان كل ما لدينا لهذا الدرس

لملاحظاتكم....

في حال وجود خطأ طباعي :

Y85.fox@hotmail.com

وفي حال وجود خطأ برمجي أنا ما دخلني على الأخ محمد جمال مد :

Mr.asp@Hotmail.com

يتبع