دروس في C#.NET

الدرس الأول:
المتغيرات وأنواعها:

)أفترض هنا معرفة القارئ بأصول البرمجة في أحدى لغات البرمجة كالسي أو الجافا أو

الفيجوال بيسك أو غيرها .. المهم أن يكون القارئ على علم ولو بشي يسير(

كغيرها من لغات البرمجة تحتوي لغة C#.NET **على أنواع مبنية بداخل هذه اللغة**

كمتغيرات من نوع Integer **و غيرها**.
لكن مما ينبغي التنويه إليه أن جميع الأنواع الموجود في لغات .NET. **لها نفس الأساس**.
خذ مثال النوع Integer **في** C# **والنوع** Integer **في** Visual Basic **هما في الحقيقة**

مشتقان من أصل واحد الذي هو System.Int32 **وكذا الحال في أكثر الأنواع الموجودة**

بداخل اللغة.
وجيع هذه الأنواع وأي نوع آخر إنما هي في الأصل كائنات مشتقة من الكائن

System.Object

وسأورد هنا جدولا يبين الأنواع الأساسية للمتغيرات في: C#

Table 3-1. C# built-in value types

Type	Size (in bytes)	.NET Type	Description
byte	1	Byte	Unsigned (values 0-255).
char	1	Char	Unicode characters.
bool	1	Boolean	true or false.
sbyte	1	Sbyte	Signed (values -128 to 127).
short	2	Int16	Signed (short) (values -32,768 to 32,767).
ushort	2	Uint16	Unsigned (short) (values 0 to 65,535).
int	4	Int32	Signed integer values between -2,147,483,647 and 2,147,483,647.
uint	4	Uint32	Unsigned integer values between 0 and 4,294,967,295.
float	4	Single	Floating point number. Holds the values from approximately +/-1.5 * 10^{-45} to approximate +/-3.4 * 10^{38} with 7 significant figures.
double	8	Double	Double-precision floating point; holds the values from approximately +/-5.0 * 10^{-324} to approximate +/-1.7 * 10^{308} with 15-16 significant figures.
decimal	8	Decimal	Fixed-precision up to 28 digits and the position of the decimal point. This is typically used in financial calculations. Requires the suffix "m" or "M."
long	8	Int64	Signed integers ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
ulong	8	Uint64	Unsigned integers ranging from 0 to 0xffffffffffffffff.

وإذا أردنا أن نعرف متغير من string **مثلا فكل ما علينا كتابته هو**:

```
string myString ;
//هنا نقوم بالإسناد
myString = "Hello" ;
```

وهكذا الحال مع كل الأنواع الأخرى.
كما ذكر في الجدول كل نوع الأنواع المعرفة في C# **له نظير في بيئة** .NET. **فمثلا النوع**

string**في** C# **هو في الحقيقة نوع** System.String

ملاحظة هاامة:
بالنسبة لمبرمجي C **أو** c++ **فلا بد من الانتباه إلى أن المتغيرات المنطقية** Boolean)

(Variables**لا تستقبل المتغيرات من نوع** Integer **أي لا يمكنك فعل شيئا كهذا**

```
bool myBoolean ;
myBoolean = 1 ; // هذه العملية غير مسموح بها
```

المتغيرات المنطقية لا تستقبل قيما غير true or false **فقط** ! **وهذا له حسناته التي**

سوف نذكرها في جمل الشرط.

الثوابت وأنواعها:

كما قيل عن المتغيرات يقال عن الثوابت سوى انك لا تستطيع التحكم في المتغير بعد

تعريفه
أي عندما تريد تعريف ثابت من نوع Integer **تعمل كالآتي**:

```
const int myConstant = 147 ;
```

هكذا تكون قد عرفت الثابت myConstant **وتستطيع استخدمه لكن بدون الإسناد إليه**

.

جمل الشرط:

في أي برنامج مهما صغر حجمه لا بد من استخدام جمل الشرط.
وجمل الشرط في C# **تشابه إلى حد كبير جمل الشرط في**C or c++ or Java

```
if( هنا جملة الشرط )
// إذا تحقق الشرط
else
// الشرط إذا لم يتحقق
```

تذكر هنا انه لا بد من وضع جملة الشرط بين قوسين.
وليس من الضروري كتابة جملة else **إلا عند الحاجة لها في البرنامج**.
ونستطيع أن نضع جملة if **أخرى بداخل جملة** if **مثال**:

```
if( جملة شرط )
// إذا تحقق الشرط
else if( الأول جملة شرط أخرى إذا لم يتحقق الشرط )
// إذا تحقق الشرط
```

هنا خطأ متكرر.

```
if( جملة شرط )
// يكون هنا أكثر من جملة إذا تحقق الشرط
```

الحل لا بد من وضع أقواس:

```
if( جملة شرط )
{
// الشرط يسمح هنا بأكثر من جملة إذا تحقق
}
```

وكذا الحال مع جملة else **و**else if

ما هي جملة الشرط:
جملة الشرط هي عبارة عن جزين أو أكثر يتم المقارنة بينهما:
مثلا قد تكون جملة الشرط عبارة عن مقارنة بين عددين
مثلا:
لنفرض لدينا عددين a, b **كلاهما له نفس القيمة فإن صيغة جملة الشرط تكون**

```
int a = 10, b = 10 ;
if (a == b)
Console.WriteLine("متساويان") ;
```

لا حظ أننا هنا استخدمنا المعامل = = ولم نستخدم المعامل = لأن المعامل = يستخدم في

الإسناد والمعامل = = يستخدم في المقارنة.

ملاحظة هاااااامة:
ملاحظة هامة لمبرمجي السي جمل الشرط لا بد أن تكون نتيجتها النهائية true or false

ولا يسمع بقيم غير ذلك أي أن الوضع يختلف عما هو عليه في السي حيث أن الجملة في

السي تكون صحيحة مادامت لا تساوي الصفر فإذا ساوت الصفر كانت خاطئة . فيمكن

فعل شيئا كالتالي:

```
if( i = 1) // هذه الجملة صحيحة دائما لأنها لا تساوي الصفر
```

أما في C# **فلا بد أن تكون النتيجة النهائية عبارة عن** true or false **ولا يصح**

استخدام العبارة

```
if( i = 1 ) // لا يسمح باستخدام هذه العبارة في سي شارب لأنها عبارة عن

عددي متغير
// ولكن تستبدل بهذه العبارة
if( i == 1 ) // هذه العبارة صحيحة
```

ونستطيع وضع أكثر من شرط في المقارنة في جملة الشرط مثلا:

```
if( i != 1 && i != 5 || i == 7 )
```

بعض المعاملات ومعانيها:

هذه تعني أو Or||
هذه تعني و And&&
هذه تستخدم في المساواة وليس الإسناد لاختبار إذا كان متغيرين متساويان فترجع = =

وإذا كانا غير متساويين فسترجعfalsetrue
لأختبار إذا كان المتغيرين غير متساويين فإذا كان المتغيرين غير متساويين فترجع true =!

وإذا كان غير ذلك فترجعfalse

جمل التكرار:

هناك ثلاثة أنواع من جمل التكرار:

1-**جملة التكرار**.. while

صيغة الجملة

```
while( هنا جملة التكرار )
{
..
..
}
```

يستمر التكرار حتى تكون قيمة جملة التكرار خطأ false **فإذا كانت خطأ فإن جملة التكرار تتوقف** . **ونستطيع الاستغناء عن الأقواس في جملة التكرار إذا كانت الجملة التي نريد تكرارها جملة واحدة فقط**.

مثال:

```
int a = 3 ;
while( a != 0 )
{
Console.WriteLine( a.ToString ) ;
a-- ;
}
```

هنا نقوم بطباعة المتغير a**ومن ثم انقاص قيمته واحد**.

2-**جملة**: do ... while

تختلف جملة do .. while **عن جملة** while **في أن الحلقة سوف يتم تنفيذه مرة واحدة على الأقل**.
مثال:

```
int a = 0 ;
do
{
Console.ToString( a ) ;
}while ( a != 0 )
```

هنا سوف يتم طباعة المتغير a **مع ان قيمته تساوي صفر لأننا استخدمنا الجملة** do ... while

3-**جملة**: for

تختلف الجمل السابقة عن جملة for **في أننا لا نعلم عدد المرات التي سوف يستمر فيها التكرار حتى يتوقف ، أما في جملة** for **فإننا غالبا ما نكون على علم بعدد المرات**.

تركيب جملة: for

```
for( هنا يتم تهيئة العداد ; شرط التكرار ; هنا مقدار الزيادة للعداد )
```

مثال:
سوف نستخدم المثال السابق:

```
int a = 10 ;
for( int i = 1 ; i <= 5 ; i++ )
{
a-- ;
Console.WriteLine( a.ToString ) ;
}
```

أخر قيمة سوف تكون للمتغير a **هي** 5 **حيث أننا وضعنا عداد اسمه** i **هذا العداد يستمر**

في الزيادة حتى يصل إلى الرقم 5 **ثم يتوقف**.

بعض المعاملات ومعانيها:

+ + **هذا العامل يستخدم للزيادة بمقدار واحد**
-- **هذا العامل يستخدم للنقصان بمقدار واحد**
< **أصغر من**
> **أكبر من**
= < **أصغر من أو يساوي**
= > **أكبر من أو يساوي**
= + **هذا المعامل لزيادة المتغير بقيمة معينة مثلاً**:

```
a = 10 ;
a += 20 ;
```

ستصبح القيمة الموجودة في a **هي** 30.
= - **للنقصان من المتغير كسابقه**.
= * **للضرب في المتغير نفس عمل السابق**.
= / **للقسمة**.
= % **باقي القسمة**.
* **ضرب**
+ **زائد**
/ **قسمة**
- **ناقص**
% **باقي القسمة**Mod

الدرس الثاني

بسم الله الرحمن الرحيم ..

الدرس الثاني :الدوال

بدأنا معكم في الدرس الأول بلمحة سريعة عن المتغيرات وطرق تعريفها في C# واليوم بإذن الله سوف نتطرق إلى الدوال وأنواعها.

ماهي الدوال:

الدوال هي كتل من الشيفرة نستطيع استخدامها في أماكن متعددة في البرنامج ولكن بدون إعادة كتابتها في كل مرة.

ما الفرق بين الاجراءات والدوال ؟

الدوال تقوم بارجاع قيمة معينة
أما الأجراءات فهي دوال لا ترجع قيمة.
مثال للإجراء:

```
void ShowMessage()
{
MessageBox.Show("Hello") ;
}
```

المثال السابق هو اجراء لأنه لا يرجع قيمة ونعرفه بكلمة void هذه الكلمة تشير إلى أن هذه الدالة لا ترجع قيمة ولهذا تسمى بالإجراء.

أما المثال على الدالة:

```
int ShowNumber()
{
MessageBox.Show("Hello") ;
return 10 ;
}
```

تلاحظ هنا أننا قمنا بتعريف الدالة بكلمة int وهذه تعني أن هذه الدالة تقوم بارجاع قيمة من نوع عدد صحيح وفي آخر الدالة تجد الكلمة return 10 هذه الكلمة تعني أن الدلة سوف ترجع القيمة 10.

نستطيع استخدام كلمة return في الاجراء لكن بدون وضع قيمة بعد كلمة. return
مثال:

```
void ShowMessage()
{
MessageBox.Show("Hello World") ;
return ;
}
```

وكلمة return في كل من الاجراءات والدوال تعني انهاء الإجراء او الدالة..

أنواع الدوال:

نستطيع تصنيف الدوال كالمتغيرات .. فكل نوع من انواع المتغيرات نستطيع ان نعرف به دالة.
مثلا: لو أردنا تعريف دالة من نوع عدد صحيح طويل long فإننا نضع الآتي:

```
long GetNum()
{
return 1111111111111111 ;//عدد صحيح طويل
}
```

فلذلك نقول أن قيمة الإرجاع لا بد أن تكون من نفس النوع الذي عرفت به الدالة ، أي لو عرفنا الدالة من نوع float فتكون القيمة التي بعد كلمة return هي من نوع. float

مثال:

```
float GetFloatNumber()
{
return 125.23 ; // الذي عرفنا به الدالة هنا نوع قيمة الارجاع هو نفسه النوع
}
```

وقد تكون القيمة التي بعد return **هي عملية حسابية . كما في الآتي:**

```
int GetSum()
{
return 10 + 52 + 98 ;
}
```

هنا الدالة سوف تقوم بإرجاع مجموع القيم 10 + 52 + 98.

طريقة تعريف الدوال:

```
[نوع متغيرات الدالة ()] [اسم الدالة] [الدالة ()]
{
//هنا تكتب الدالة
return [قيمة من نفس نوع الدالة] ;
}
```

نوع الدالة:
الأنواع التي توضع في تعريف الدالة هي نفسها الأنواع التي توضع لتعريف المتغيرات.

اسم الدالة:
ضع الاسم الذي تريد

متغيرات الدالة:
سوف نناقشها لاحقا.

تمرير القيم إلى الدوال أو الاجراءات:

تكمن الفائدة الفعلية للدوال هي أننا نستطيع نستطيع تمرير المتغيرات إليها .. كيف؟
مثلاً : لو كان لدينا دالة من نوع integer **تقوم بحساب عددين وإرجاع القيمة.**

```
int GetSum()
{
return 10 + 25 ;
}
```

هذه الدالة هي غير مفيدة إطلاقا ! لماذا؟
لأن القيمة التي سوف نحصل عليها من الدالة هي قيمة ثابتة دائما وتساوي 35 **أي كلما قمنا باستدعاء الدالة فسوف نحصل على القيمة** 35 **كقيمة مرجعة لهذه الدالة .**

كيف نجعل هذه الدالة تجمع لنا رقمين متغيرين ؟
نستطيع ذلك عن طريق تمرير القيم للدالة .. كيف يتم ذلك ؟
هناك ثلاث طرق في C# **لتمرير القيم للدالة .**

الطريقة الأولى التمرير بالقيمة:
أي نقوم بتمرير قيمة متغير معين إلى دالة معينة.
مثال:

```
int GetSum ( int num1, int num2 ) // صحيح قمنا هنا بوضع متغيرات للدالة من نوع عدد
// هذه المتغيرات هي عبارة عن الأعداد التي نريد جمعها
{
return num1+num2 ;
}
```

ولو أردنا استدعاء الدالة السابقة فاننا نكتب التالي:

```
int Sum ;
Sum = GetSum( 10, 20 ) // التي نريد جمعها قمنا هنا باستدعاء الدالة ووضع قيم للمتغيرات
Console.WriteLine( Sum.ToString() ) ;
```

وعندما نقول أن هذا التمرير هو تمرير بالقيمة فهذا يعني أن قيمة المتغير الذي نريد تمريره تنسخ إلى المتغير

الموجود في الدالة.
مثال:
نريد تمرير متغير من نوع عدد صحيح إلى الدالة السابقة فنعمل كالآتي:

```
int Sum , MyNumber1 = 20, MyNumber2 = 10 ;
Sum = GetSum( MyNumber1, MyNumber2 ) ;
Console.WriteLine( Sum.ToString() )
```

;
هنا المتغير MyNumber1 **و المتغير** MyNumber2 **قمنا بتمرير قيمهما إلى الدالة** GetSum **فقط** ! **ما معنى ذلك ؟**
معنى ذلك ان المتغير المسمى num1 **في الدالة السابقة سوف يأخذ نسخة من قيمة المتغير**MyNumber1 ، **وهكذا المتغير** num2 **مع المتغير**MyNumber2
أي لو قمنا بتعديل في الدالة السابقة بحيث تقوم بتغيير في قيم أحد متغيراتها.

```
int GetSum( int num1, int num2 )
{
int Sum ;
Sum = num1 + num2 ;
num1 = 50 ; // لاحظ هنا قمنا بتغيير قيمة المتغير num1
return Sum ;
}
```

هذه الدالة تقوم بتغيير في قيمة المتغير num1 **لكن مع هذا قيمة هذا المتغير لن تتغير لأننا قد مررناها بالقيمة** . **انظر المثال التالي**:
في هذا المثال نقوم باستدعاء الدالة. GetSum

```
int Sum, MyNumber1 = 30, MyNumber2 = 50 ;
Console.WriteLine( MyNumber1.ToString() ) ; // هنا نقوم بطباعة المتغير قبل تمريره للدالة
// للدالة نقوم بتمرير المتغير
Sum = GetSum( MyNumber1, MyNumer2 ) ;
Console.WriteLine( MyNumber1.ToString() ) ; // هنا نقوم بطباعة المتغير مرة أخرى بعد
للدالة التمرير
```

عندما قمنا بطباعة المتغير MyNumber1 **في المرة الأولى فستكون القيمة** 30 **ثم قمنا بتمرير المتغير** MyNumber1
إلى الدالة GetSum **وهذه الدالة تقوم بتغيير في قيمة المتغير لكن بما ان التمرير هو بالقيمة فإن المتغير** MyNumber1**سوف لن يتأثر ، لذلك فإنه قيمة المتغير** MyNumber1 **عندما قمنا بطباعته مرة أخرى هي نفسها لم تتغير** 30 . (**أرجو أن تكون الفكرة قد اتضحت**)

الطريقة الثانية التمرير بالإشارة:

معنى ذلك هو أن تقوم يتمرير عنوان المتغير في الذاكرة إلى الدالة.
أي لو أردنا القيام بتغيير في قيمة المتغير MyNumber1 **في المثال السابق فلا بد من كتابة المتغيرات في الدالة** GetSum**بصورة أخرى**.

```
int GetSum( ref int num1, int num2 )
{
int Sum ;
Sum = num1 + num2 ;
num1 = 50 ;
return Sum ;
}
```

قمنا هنا بزيادة كلمة في تعريف المتغير الأول (ref) **من متغيرات الدالة هذه الكلمة تعني أن نوع التمرير هو بالاشارة**.
ولو أردنا أن نقوم باستدعا الدالة السابقة فسنكتب شيئا كالتالي.

```
int Sum, MyNumber1 = 30, MyNumber2 = 20 ;
Console.WriteLine( MyNumber1.ToString() ) ;
Sum = GetSum( ref MyNumber1, MyNumber2 ) ;
Console.WriteLine( MyNumber1.ToString() ) ;
```

لاحظ هنا أننا حينما نريد تمرير متغير بالاشارة فلابد من كتابة كلمة ref **لكي تخبر المترجم بأن هذا التمرير هو بالاشارة**

.
هنا الحال يختلف عما هو عليه في طريقة التمرير بالقيمة فقيمة المتغير MyNumber1 **قبل التمرير إلى الدالة تختلف عن قيمته بعد التمرير إلى الدالة**..
أي أن قيمة المتغير MyNumber1 **قبل التمرير إلى الدالة هي** 30 **لكن بعد التمرير إلى الدالة أصحبت** 50 . **وبهذا يظهر الفرق بين التمرير بالقيمة والتمرير بالإشارة**.

الطريقة الثالثة التمرير بالاشارة لكن بدون قيم أولية:

في الطريقتين السابقتين كان لا بد من وضع قيم أولية في المتغير قبل تمرير إلى الدالة .. أي أنه لا يمكن لنا فعل شيئا كالتالي:

```
int MyNumber1, MyNumber2 , Sum ;
Sum = GetSum( MyNumber1, MyNumber2 ) // هذه العملية غير مسموح بها لأننا لم نقم بوضع قيم اولية للمتغيرين
```

والحل أن نضع كلمة out **عندما تعرف متغيرات الدالة وعند تمرير القيمة إليها**.

```
int GetSum( out num1, num2 )
{
return num1 + num2 ;
}
```

وعندما نريد استدعائها نقوم بالتالي

```
int Sum, MyNumber1, MyNumber2 = 10 ;
Sum = GetSum( out MyNumber1, MyNumber ) ;
```

كما قلنا سابقا فإن التمرير بـ out **هو في الحقيقة تمرير بالاشارة لكنه لا يحتاج إلى وضع قيم أولية**.

وبهذا نكون قد انتهينا من المبادئ الأساسية بمرور سريع على المتغيرات والدوال وطرق تعريفها وحلقات التكرار والشرط.

التحميل الزائد للدوال: (Functions Overloading)

ما معنى ذلك ؟

معنى التحميل الزائد للدوال هو أن يكون هناك دالتين بنفس الاسم ويختلفان في نوع البارامترات (متغيرات الدالة)

ما الفائدة منه ؟

لو فرضنا أن لدينا دالة تقوم بتحويل عدد صحيح إلى نص .. وهناك دالة تقوم بتحويل عدد حقيقي إلى نص ودالة ... أي جميع هذه الدوال لها العمل نفسه لكن المتغير هو نوع متغير الدالة ..
مثال:

```
void ConvertIntToString( int number ) // نص هذه دالة تحول عدد صحيح إلى
{
...
..
}

void ConvertFloatToString( float number ) // هذه دالة تحول عدد حقيقي إلى نص
{
..
..
}
```

نلاحظ من السابق أن كلا الدالتين لهما اسم مختلف مع أن عمل الدالتين واحد .. ماذا لو قمنا بتوحد الأسماء وجعل المترجم يتعرف عليها لوحده ؟.

نستطيع أن نعمل الآتي:

```
void ConvertToString( int number ) // هذه الدالة تقوم بتحويل رقم صحيح إلى نص
{
..
.
}

void ConvertToString( float number ) // بتحويل عدد حقيقي إلى نص هذه الدالة تقوم
{
```

```
..
.
}
```

لاحظ في المثالين السابقين أن اسم الدالتين واحد ولكن الذي يختلف هو متغير الدالة أو ما يسمى) البارامتر) ففي الدالة الأولى نجده من النوع int **والثانية نجده من النوع** .. float **أي عندما نريد استدعاء الدالة فإن المترجم ينظر إلى نوع البارامتر الذي وضعناه في الدالة ويستدعي الدالة المناسبة** ..
مثال:

```
int intNum = 123 ;
float floatNum = 147.25 ;

ConverToString( floatNum ) ; // هنا قمنا باستدعاء الدالة السابقة ووضعنا البارامتر من نوع
float
// المترجم سوف يستدعي لنا الدالة التي تقوم بتحويل عدد حقيقي إلى نص فهنا
```

أي أن التحميل الزائد للدوال هو أن يكون في البرنامج دالتين أو أكثر لهما نفس الاسم لكنهما يختلفان في أنواع البارامترات (متغيرات الدالة(

كتابة أول برنامج في: C#

أظنكم في شوق إلى كتابة أول برنامج في C# **لكنني أخرت ذلك لمناقشة بعض المفاهيم الأساسية في اللغة.**
البرنامج التي سوف نقوم بكتابتها من الآن وحتى الانتهاء من الفئات هي برامج من نوع Console Aplication **أي برامج** Dos
وبعد اتقان مفهوم الفئات سوف ننتقل بإذن الله إلى البرمجة في بيئة. Windows

اتجه إلى Microsoft Visual Studio .NET **ثم اعمل الآتي**.

```
New - > Project - > Visual C# - > ConsoleApplication .
```

بعد ذلك قم بالضغط على زر موافق وانتظر لحظات حتى تفتح امامك نافذة فيها بعض الكتابات قم بمسحها وألصق الكود التالي:

```
using System ;
public class HelloWorldProgram
{
static void Main()
{
Console.WriteLine("Hello World") ;
}
}
```

بعد ذلك قم بتشغيل البرنامج بالضغط على الزر الأزرق الموجود بالأعلى والذي يرمز إلى التشغيل.

التحميل الزائد للدوال: (Functions Overloading)

ما معنى ذلك ؟

معنى التحميل الزائد للدوال هو أن يكون هناك دالتين بنفس الاسم ويختلفان في نوع البارامترات (متغيرات الدالة(

ما الفائدة منه ؟

لو فرضنا أن لدينا دالة تقوم بتحويل عدد صحيح إلى نص .. وهناك دالة تقوم بتحويل عدد حقيقي إلى نص ودالة ... أي جميع هذه الدوال لها العمل نفسه لكن المتغير هو نوع متغير الدالة ..
مثال:

```
void ConvertIntToString( int number ) // نص هذه دالة تحول عدد صحيح إلى
{
...
..
}
```

```
void ConvertFloatToString( float number ) // هذه دالة تحول عدد حقيقي إلى نص
{
..
..
}
```

نلاحظ من السابق أن كلا الدالتين لهما اسم مختلف مع أن عمل الدالتين واحد .. ماذا لو قمنا بتوحد الأسماء وجعل المترجم يتعرف عليها لوحده ؟.

نستطيع أن نعمل الآتي:

```
void ConvertToString( int number ) // هذه الدالة تقوم بتحويل رقم صحيح إلى نص
{
..
.
}

void ConvertToString( float number ) // بتحويل عدد حقيقي إلى نص هذه الدالة تقوم
{
..
.
}
```

لاحظ في المثالين السابقين أن اسم الدالتين واحد ولكن الذي يختلف هو متغير الدالة أو ما يسمى) البارامتر) ففي الدالة الأولى نجده من النوع int **والثانية نجده من النوع** .. float **أي عندما نريد استدعاء الدالة فإن المترجم ينظر إلى نوع البارامتر الذي وضعناه في الدالة ويستدعي الدالة المناسبة ..**
مثال:

```
int intNum = 123 ;
float floatNum = 147.25 ;

ConverToString( floatNum ) ; // هنا قمنا باستدعاء الدالة السابقة ووضعنا البارامتر من نوع
float
// المترجم سوف يستدعي لنا الدالة التي تقوم بتحويل عدد حقيقي إلى نص فهنا
```

أي أن التحميل الزائد للدوال هو أن يكون في البرنامج دالتين أو أكثر لهما نفس الاسم لكنهما يختلفان في أنواع البارامترات (متغيرات الدالة(

الدرس الثالث

الدرس الثالث: الفئات

إعادة كتابة أول برنامج على: C#

```
namespace HelloProg
{
using System ;

class HelloWorld
{
public static void Main()
{
Console.WriteLine("Hello World") ;
}
}
}
```

سوف نعود لمناقشة هذا البرنامج لاحقا في هذا الدرس .. لكن دعونا الآن نتعرف على

الفئات..

ماهي الفئات ؟

في الدرس الماضي تكلمنا عن الدوال وطرق استخدامها ، وشاهدنا كيف يمكننا تقليص

الكثير من الكود باستخدام الدوال .. لكن هناك اشكالية تلازم هذه الدوال .. ما

هي هذه الاشكالية ؟
الاشكالية هي أن أي دالة من هذه الدوال غير منعزلة بشكل تام عن باقي أجزاء

البرنامج .. بمعنى أننا لو أردنا أن يكون هناك متغير مشترك بين ثلاثة دوال فقط ولا

يمكن للدوال الأخرى استخدامه فلن نستطيع ذلك !! إلا عن طريق التمرير عن طريق

البارامترات وسنضطر عندها إلى استخدام المؤشرات أو احدى الطرق التي ناقشناها في

الدرس الماضي .

أما باستخدام الفئات فيمكننا فعل السابق وبسهوله .. حيث أن كل فئة وكأنها

برنامج مستغل بذاته ، أي أن الفئة لها متغيرات خاصة بها ولها دوالها الخاصة بها يعني

كما قلنا كأنها برنامج مستقل بذاته .. ونحن في الحقيقة نستخدم هذه الدوال في البرمجة

بشكل مسسستمر .. مثلا لمبرمجي الفيجوال بيسك دائما يستخدمون "مربع النص" ومربع

النص هذا إنما هو مشتق من فئة اسمها TextBox **وهذه الفئة تحتوي على عناصر بداخلها**

كالدوال والمتغيرات والخصائص .. مثلاً نجد أن هذه الفئة تحتوي على دالة اسمها Find

هذه الدالة تقوم بالبحث عن كلمة معينة موجودة في مربع النص هذا .. وهناك خاصية

تسمى Text **هذه الخاصية نستطيع من خلالها رؤية وتغيير محتويات مربع النص.**

لكن لابد من الملاحظة .. أن الفئات لا يمكننا استخدامها مباشرة .. أي لا يمكننا

كتابة فئة معينة ثم نقوم باستخدامها مباشرة (إلا في حالات معينة سوف نناقشها

لاحقا) فلكي نكون قادرين على استخدام فئة معينة لابد أن نقوم باشتقاق كان من

هذه الفئة ثم نقوم باستخدام هذا الكائن .. وقد يتبادر إلى ذهن الشخص سؤال :

مالفرق بين الفئات والكائنات المستخلصة منها .. وللتوضيح نستطيع أن نشبه الفئات

بفصيلة الانسان .. وهذه الفئة يتفرع منها جميع البشر .. يعني أن فصيلة الانسان

تعتبر فئة والكائنات المتفرعة من هذه الفئة هي البشر .. يعني أن محمد وناصر وفهد

هم كائنات من فصيلة الانسان.. (أرجو ان تكون الفكرة قد اتضحت)

نظرة في البرنامج السابق: (Hello World)

1- NameSpaces (أسم المساحة) أو (أسماء النطاقات):

بصراحة لم أستطع حتى الآن أن اجد تسمية صالحة للـ Namespaces لكن على العموم

سوف أقوم بشرحها بدون التطرق لمعناها بالعربية.
ذكرنا قبل قليل أن الفئات عبارة عن مجموعة دوال ومتغيرات منعزلة عن باقي أجزاء

البرنامج بشكل تام.
أما أسماء المساحة Namespaces فهي عبارة عن مجموعة فئات مجتمعة مع بعضها .. كيف

؟؟
لنفرض أنك قمت بكتابة عدد من الفئات تقوم بمعالجة وضائف الإدخال والإخراج ..

فيمكننا وضع هذه الفئات تحت مسمى واحد ونسميه مثلاً InputOutputOperations هذا

الاسم هو مانطلق عليه "اسم المساحة" أو Namecpace أي بمعنى أوضح أسماء المساحة

عبارة عن مجموعة فئات) وليس دوال) يكون في الغالب لها وظيفة واحدة . ويمكن ان

يكون اسم مساحة بداخل اسم مساحة آخر.
وفي برنامجنا السابق كتبنا الجملة

```
namespace HelloWorldProg
{
..
..
}
```

في الحقيقة نستطيع الاستغناء عن اسم المساحة هذا لكنني وضعته هنا للفائدة

التعليمية فقط (لأن برنامجنا صغير جدا)

2- استخدام اسم المساحة: (Namespace)

في البرنامج السابق استخدمنا اسم المساحة System عن طريق الجملة .. using وفائدة

هذه الجملة هو أننا نكون قادرين على استخدام الفئات الموجودة بداخل اسم المساحة

System .. أي لو قمنا بحذف الجملة
using System ;
فلابد من تغيير الكود إلى الآتي.

```
namespace HelloWorldProg
{
class HelloWorld
{
public static void Main()
{
System.Console.WriteLine("Hello World") ;
}
```

```
}
}
```

لاحظ أننا هنا يوم قمنا بحذف الجملة using System **كان لزاما علينا ان نكتب المسار**

كاملا .. وتظهر فائد كلمة using **في هذا الموضع بأنها للاختصار فقط ..**

-3**الكلمة الأساسية**: class

في البرنامج السابق قمنا بتعريف الفئة HelloWorld **باستخدام الكلمة الأساسية**

class ..
طريقة تعريف الفئة:

```
class اسم الفئة
{
//هنا أعضاء الفئة
..
}
```

قمنا في البرنامج السابق بتعريف فئة تسمى HelloWorld **هذه الفئة نجد أنها تحتوي**

بداخلها دالة من نوع void **وهذه الدالة هي الدالة التي يقوم باستدعائها نظام**

التشغيل عندما تقوم بتشغيل البرنامج..
تلاحظ أننا وضعنا قبل تعريف الدالة Main **كلمة** public **و** static **وسوف نناقشها**

لاحقا.

دعونا الآن نغوص في أعماق الفئات وخصائصها

-1**تعريف الفئة**:
ناقشناها قبل قليل وسنضيف بعض التفصيل لاحقا.

-2**مصطلحات للفئة**:

قلنا أن الفئة هي جزء منعزل تماما عن باقي أجزاء البرنامج أي أن الفئة تحتوي على

دوال ومتغيرات وخصائص خاصة بها ، وهاكم توضيحا لبعض الأسماء:
أ- أعضاء الفئة: (Members)
أعضاء الفئة هي عبارة عن الدوال الموجودة بداخل فئة معينة.
ب- سجلات الفئة: (Fields)
سجلات الفئة عبارة عن المتغيرات المعرفة بداخل الفئة من أي نوع.
ج- خواص الفئة: (Propreties)
وهذه سوف نناقشها لاحقا.

نريد أن نبني فئة تسمى Calculator **هذه الفئة تحتوي على دالتين .. الأولى للجمع**

والثانية للضرب.

```
class Calculator //كما ناقشناها سابقا في تعريف الفئة
{
public int Plus( int num1, int num2 )
{
return num1 + num2 ;
}

public int Multiply( int num1, int num2 )
{
return num1 * num2 ;
```

```
}
}
```

الآن قمنا بتعريف فئة تسمى Calculator **هذه الفئة تحتوي على دالتين ، وكلا هاتين**

الدالتين يبدأ تعريفهما بكلمة public **وسوف نناقشها لاحقا . طبعا هذه الدوال**

تسمى في الفئة أعضاء.

كما ذكرنا سابقا لا يمكننا الاستفادة من الفئة مباشرة ، فلا بد من انتاج كائنات

من هذه الفئة ثم استخدام هذه الكائنات.. لكن ياترى كيف يمكننا ذلك ؟

نستطيع عمل ذلك بالآتي:

```
class HelloWorld
{
static void Main()
{
Calculator myCalc = new Calculator() ; //هنا نقوم بانشاء كائن

من الفئة
int sum, prod ;
sum = myCalc.Plus( 10, 20 ) ;
prod = myCalc.Multiply( 5, 3 ) ;
Console.WriteLine("{0}\n{1}", sum, prod ) ;
}
}
```

دقق في السطر

```
Calculator myCalc = new Calculator() ;
```

نستطيع ان نقسمه إلى سطرين..

```
Calculator myCalc ;
myCalc = new Calculator() ;
```

السطر الأول قمنا بتعريف متغير اسمه myCalc **هذه المتغير من نوع** .. Calculator **تماما**

بنفس الطريقة مع المتغيرات من نوع int, float, char **لكن هناك إختلاف في السطر**

الثاني ..

```
myCalc = new Calculator() ;
```

هذا السطر مهم جدا وبواسطة هذا السطر نكون قد أنشأنا كائن من الفئة

Calculator**واسم هذا الكائن**.. myCalc

لماذا كل هذه المتاعب ؟ ألا يكفي السطر الأول من التعريف ؟؟
ربما يتبادر إلى ذهن احدنا هذا السؤال ..
هناك أنواع من المتغيرات .. فهناك متغيرات قيمة ، وهناك متغيرات إشارة

(Pointers).
متغيرات القيمة يكفي فيها أن نقوم بالتعريف فقط . مثلا كـ int
أما متغيرات الإشارة فلايكفي فيها التعريف فقط .. فلابد من حجز مساحة في الذاكرة

ومن ثم نجعل المتغير يشير إلى هذه المساحة . (سوف نتطرق بتفصيل أكبر عن متغيرات القيمة

ومتغيرات الإشارة(
لكن كل مانريد قوله أن أي متغير تقوم بتعريفه عن طريق فئة (كما في مثالنا

السابق) فهو متغير ذو إشارة وليس متغير ذو قيمة.

إذا جميع المتغيرات التي نقوم بتعريفها عن طريق الفئات ماهي إلا متغيرات إشارة

(Pointers)وهناك أنواع أخرى من متغيرات الاشارة سوف نناقشها لاحقا .. وهناك

أيضا متغيرات ذات قيمة مثل. int, float, char

نرجع إلى المتغير myCalc

```
myCalc = new Calculator() ;
```

مافائدة الكلمة: new

كما قلنا سابقا بأن متغيرات الاشارة لكي نستفيد منها لابد من حجز مكان في الذاكرة

وجعل هذا المتغير يشير إلى هذا المكان .
كلمة new هنا تقوم بحجز مساحة في الذاكرة من

النوع .. Calculator لاحظ انه لابد من وضع قوسين بعد اسم الفئة كما في المثال

السابق .. هذه الأقواس سوف نتطرق لها عندما نتحدث عن المشيدات ..
بعدما قمنا بتكون كائن من الفئة Calculator نستطيع الآن استخدام الدوال

الموجودة بداخل الفئة Calculator كما في التالي:

```
myCalc = new Calculator() ;
myCalc.Plus( 10, 25 ) ;
...
.. .
```

الأعضاء الخاصة والأعضاء العامة:(public & private)

قلنا في السابق أن الفئة عبارة عن جزء معزول تماما عن باقي أجزاء البرنامج ويملك

دواله الخاصة به ومتغيراته الخاصة به وخواصه الخاصة به ، وقلنا بأننا إذا أردنا

استخدام أي دالة بداخل أي فئة فإننا نقوم بانشاء كائن من هذه الفئة واستخدام

الدوال عن طريق هذا الكائن لكن بهذه الطريقة سوف يكون من يستخدم الفئة قادر

على استخدام جميع الفئات والمتغيرات الموجودة بداخل هذه الفئة ، لكن ماذا لو

أردنا أن يكون هناك أعضاء ومتغيرات خاصة موجودة بداخل هذه الفئة ولكن لا يستطيع

مستخدم الفئة الوصول إلى هذه الأعضاء والمتغيرات ..
الحل هو بوجود الأعضاء الخاصة والأعضاء العامة!

كيف يمكن تعريف المتغيرات والأعضاء الخاصة والعامة ؟

نستخدم الكلمة private لجعل العضو أو المتغير خاص ولا يستطيع مستخدم الفئة

الوصول إليه.
ونستخدم الكلمة public لجعل العضو أو المتغير عام.

وتتم الطريقة كالآتي:

عندما نريد تعريف متغير أو عضو عام نضع الكلمة الأساسية public قبل العضو المراد

جعله عام.

لنفرض أن لدينا فئة تسمى Calculator **تحتوي على دالة للجمع ودالة للطرح ومتغير**

عام اسمه. result

```
class Calculator
{
public int result ; //عام هذا متغير

public void Plus( int num1, int num2 ) //

هذه دالة الجمع تضع المجموع في المتغير العام result
{
result = num1 + num ;
}

public void Minus( int num1, int num2 )// هذه

دالة الطرح تضع الفرق بين العددين في المتغير العام
{
result = num1 - num2 ;
}
}
```

في هذا المثال قمنا بتعريف جميع الأعضاء بالكلمة public **وهذه الكلمة كما ذكرنا**

سابقا تعني أن هذا العضو أو المتغير ظاهر لمستخدم الفئة .. أي كالتالي.
عندما نريد استخدام الفئة نكتب التالي.

```
Calculator myCalc = new Calculator() ;
myCalc.Plus( 10, 20 ) ; // هذه الدالة سوف تقوم 20 و 10 بجمع العدد
//ناتج عملية الجمع السابقة موجود في المتغير result
Console.WriteLine( myCalc.result ) ; // قمنا هنا بطباعة المتغير Result
//30 الذي يحتوي على مجموع العددين 10 و 20 أي سوف يكون الناتج

myCalc.Minus( 10, 50 ) ;//50 من العدد 10 وسوف هذه الدالة تقوم بطرح العدد

40- يكون الناتج
//الناتج سوف يكون في المتغير result
Console.WriteLine( myCalc.result ) ;
```

إذا تلاحظ من الأمثلة السابقة أن المتغير أو الدالة المعرفة بالكلمة public **يمكن**

استخدامها مباشرة بمجرد تكوين كائن من الفئة.

جيد .. ماذا لو أردنا أن نجعل أحد الأعضاء في الفئة خاص كأن نجعل المتغير

result**متغير خاص باستخدام الكلمة**.. private
دعنا نعدل المثال السابق لنتعرف على الطريقة..

```
class Calculator
{
private int result ; //هنا جعلنا المتغير خاص

public void Plus( int num1, int num2 )
{
result = num1 + num2 ;
}
public void Minus( int num1, int num2 )
{
result = num1 - num2 ;
}
}
```

وإذا أردنا استخدام هذه الفئة فنفس الطريقة السابقة إلا أن الأعضاء الخاصة لايمكن

لنا استخدامها.

```
Calculator myCalc = new Calculator() ;
myCalc.Plus( 15, 10 ) ; // result بتخزين المجموع في المتغير هذه الدالة سوف تقوم
Console.WriteLine( myCalc.result ) ; // result هذه العملية غير مسموح بها لأن

المتغير
// يمكننا الوصول إليه هو متغير خاص ولا
```

وكذلك الأمر ينطبق على الدوال .. لو قمنا بتعريف الدالة Plus **وجعلناها دالة**

خاصة كالتالي

```
private void Plus( int num2, int num2 )
{ ... }
```

فلايمكننا استخدامها كما في المثال السابق وإنما تتحول إلى دالة خاصة أي لايمكننا فعل

شيئا كالتالي.

```
Calculator myCalc = new Calcolator() ;
myCalc.Plus( 10, 25 ) ; //العملية لأننا عرفنا الدالة لا يمكننا فعل مثل هذه

على أنها دالة خاصة
```

مما سبق يتبين أن الدوال الخاصة لا يمكن لمستخدم الفئة استخدامها ولكن الدوال

العامة يمكن استخدامها.
لكن مما ينبغي التنبيه له بالنسبة لأنواع الدوال في داخل الفئة ليس لها

تأثير .. فنستطيع استخدام المتغير أو الدالة في داخل الفئة سواءاً أكان خاصا أم

عاماً..

<u>ملاحظة</u>:
لا يستحسن جعل المتغيرات الموجودة في داخل الفئة عامة وإنما يفضل استخدام الخصائص.

<u>ملاحظة أخرى</u>:
مما ينبغي التنبيه له أن المتغيرات الخاصة هي الافتراضية ، بمعنى أننا إن لم نقم

بتعريف المتغير بالكلمة private **فإن المتغير أو الدالة هو في الأصل هو متغير خاص ..**

أما إذا أردنا أن يكون عاما فلابد من استخدام كلمة public **والمثال التالي يوضح**

الفرق.

```
private int Var ; //هذا متغير خاص
int Var ; // وهذا المتغير أيضا خاص
public Var ; // هذا المتغير عام
```

طبعا الكلام الذي يقال في المتغيرات يقال في الدوال.

خواص الفئة: (Properties)

كما ذكرنا سابقا لايستحسن استخدام متغيرات الفئة بشكل عام لأن ذلك قد يسبب بعض

الارباك لبنية البرنامج وفتح المجال لأي تغيير قد يحصل للمتغيرات..
وبواسطة الخصائص نستطيع جعل بنية الفئة أفضل..

وسوف نقوم بتعديل في البرنامج السابق لكي نجعله يستخدم الخصائص.

```
class Calculator
{
int result ; // متغير خاص
```

```
public void Plus( int num1, int num2 )
{
result = num1 + num2 ;
}
public void Minus( int num1, int num2 )
{
result = num1 - num2 ;
}

// الفئة هنا سوف نقوم بتعريف

public int Result
{
get
{
return result ;
}
}
}
```

وعندما نريد استخدامها:

```
Calculator myCalc = new Calculator() ;
myCalc.Plus( 10, 25 ) ;
Console.WriteLine( myCalc.Result );// تلاحظ اننا هنا قمنا بطباعة الناتج عن

Result الخاصية طريق استخدام
```

تعريف الخصائص:

تعرف الخصائص بالأسلوب الآتي.

```
[public, private, static, ... ] [اسم الخاصية] [النوع]
{
get
{ ... }
set
{ ... }
}
```

نستنتج من التعريف السابق أن الخصائص تتكون من كلمتين أساسيتين وهما:
أ- الكلمة الأساسية: get
في مثالنا السابق عندما كتبنا الأمر التالي

```
Concole.WriteLine( myCalc.Result ) ;
```

هنا قمنا بطلب من الخاصية Result **أن تعطينا قيمة جمع أو طرح العددين .. فلذلك**

يتجه البرنامج إلى get **لكي يعطي المستخدم النتيجة**.
ب- الكلمة الأساسية: set
هذه الكلمة هي عكس الأولى فلو أردنا في مثالنا السابق أن نقوم بتعديل قيمة

المتغير result **فنستطيع ذلك عن طريق هذه الكلمة كالتالي**
سوف نقوم بتعديل الفئة السابقة:

```
class Calculator
{
private int result ;

public void Plus( int num1, in num2 )
{
result = num1 + num2 ;
}
public void Minus( int num1, int num2 )
{
result = num1 - num2 ;
```

```
}

public int Result
{
get
{
resturn result ;
}
set
{
result = value ;
}
}
}
```

وعند استخدام هذه الفئة نستخدم الآتي:

```
Calculator myCalc = new Calculator() ;
myCalc.Plus( 10, 25 ) ;
Console.WriteLine( myCalc.Result ) ; // 35 سوف يكون الناتج
// ونستطيع ان نغير الناتج
myCalc.Result = 10 ;
Console.WriteLine( myCalc.Result ) ; // الشاشة 10 سوف يطبع على
```

من المثال السابق نستفيد أن الخواص أفضل من استخدام متغيرات عامة بداخل الفئة.
فلو قمنا بحذف الجزء set **لأصبحت الخاصية للقراءة فقط**(Read Only)
الكلمة vlaue **تعني القيمة التي قام المستخدم بإسنادها للخاصية**.