الدرس الثامن : الوسائل Method

لربما رأيت أو أدركت أنه كلما زادت مهام البرنامج ، زاد تعقيده ؛ فلهذا فكر بعض الأشخاص في تسهيل مهمة المبرمج فقالوا : لماذا لا نتبع سياسة فرق تسد ، و أخذوا العبرة من حكاية الجد الذي أعطى أحفاده رزمة من العصي فقال لكل واحد منهم اكسرها فلم يستطيعوا ، فقام بعد ذلك بإعطاء عصا لكل حفيد فقال لهم اكسروها الآن فكانت العلمية أسهل من ذي قبل .

هذا هو ملخص حكاية الوسائل **Method** و هو تقسيم العمل على أجزاء صغيرة (وسيلة) ، كل جزء يقوم بمهمة معينة و واحدة فقط .

بحيث لو كان عندنا على سبيل المثال برنامجاً يحسب ناتج ضرب عددين فإننا نستطيع أن نقسم هذا العمل إلى وسيلة تقرأ العددين ، و وسيلة تقوم بضرب العددين و ترجع الناتج ، و أخرى تقوم بإظهار ناتج الضرب في الشاشة ، أليس الأمر منظماً و مرتباً ؟! .

و الآن نرجع إلى لغة السي شارب ، حيث يتم تعريف الوسيلة كالتالي :

```
نوع الناتج الراجع    اسم الوسيلة ( المعامل الأول  ,  المعامل الثاني  , ... )
{
        محتوى الوسيلة
}
```

و الآن دعنا نطبق ما قلناه على المثال الذي يرجع ناتج ضرب عددين ، ركز البرنامج التالي :

```
using System;
class Multiply
{
      //الوسيلة الأولى، و هي الوسيلة الذي يبدأ البرنامج بتنفيذها أولا
      static void Main()
            {
                  Multiply Example = new Multiply();
                  double number1,number2,result;
                  number1 = Example.GetNumber(1);
                  number2 = Example.GetNumber(2);
                  result = Example.MultNumber(number1,number2);
                  Example.Display(result);
            }

             // الوسيلة الثانية و مهمتها إحضار العددين من عند المستخدم
      double GetNumber (int Number)
            {
                  double x;
                  Console.Write("Please Enter Number" + Number +":");
                  x = Double.Parse(Console.ReadLine());
                  return x;
            }
```

```
        // الوسيلة الثالثة و مهمتها إيجاد حاصل ضرب العددين
        double MultNumber (double x, double y)
              {
                    return x*y;
              }

              // الوسيلة الرابعة ومهمتها عرض المخرجات
        void Display (double result)
              {
              Console.WriteLine("The result of Multiply of tow   number= "
                                          + result);
                    return;
              }

}
```

قد ترى أننا أضفنا سطرا زيادة عما كنت تتوقع ألا و هو :

```
Multiply Example = new Multiply();
```

ففي هذا السطر قمنا بإنشاء كائن حقيقي من نوع الخلية **Multiply** و أعطيناه اسم **Example** ، و هذا الكائن له بعض الوسائل التي يمكن أن تصل إليها عن طريق معامل "." فالسطر :

```
Example.GetNumber(1);
```

يقوم بمناداة الوسيلة **GetNumber** من الكائن **Example** المنتمي إلى الخلية **Multiply** إلى هذا الحد يكفي ، سوف نتطرق إلى الخلايا بتعمق أكثر في الدروس القادمة بإذن الله تعالى .

و الآن نرجع إلى تعريف الوسائل ، هناك نوعين من الوسائل : وسيلة ترجع شيئا ، و أخرى لا ترجع شيئا ، فالتي لا ترجع شيئا تستخدم كلمة **void** للدلالة على ذلك كما في الوسيلة **Display** ، و أما التي ترجع شيئا فإنها تكتب بدل **void** نوع المُرجَع مثل الوسيلة **GetNumber** و **MultNumber** .

الأمر الثاني ألا وهو المعاملات ، وهي الأشياء التي سوف تحتاجها الوسيلة لتأدية عملها بالشكل المطلوب ، وتعرف بأن يكتب نوعها ثم اسمها ليستخدم داخل الوسيلة كمتغير ، و يمكنك أن تكتب ما تريد من المعاملات .

الأمر الثالث أنك تستخدم كلمة **return** لترجع القيمة المطلوبة فإذا لم تكن الوسيلة ترجع شيئا ، فضع فاصلة بعد كلمة **return** كما في الوسيلة **Display** .

و هناك أمر مهم وهو يجب أن يكون نوع المعاملات و المخرجات التي سوف نستعملها مطابقة لما في تعريف الوسيلة ، فإذا حاولت إرسال سلسلة حرفية بدلا من **double** ، فإن المترجم سوف يعطيك خطأ .

و هنا أود أن أناقش هذه القضية بتفصيل أكثر ، فهيا بنا .

أنواع البيانات بعمق

قد كنا في الدروس السابقة نمر على أنواع البيانات مرور الكرام ، نأخذ الذي نحتاج إليه فقط ، أما الآن فيجب علينا التوقف و الرجوع إليها بتعمق أكثر من ذي قبل .

قلنا أن هناك عدة أنواع من البيانات منها الأرقام و الحروف و السلاسل الحرفية ، فلنبدأ بالأرقام:

هناك نوعان من الأرقام : صحيحة و عشرية أو حقيقية .

الأعداد الصحيحة :

وهي التي لا تحوي فاصلة عشرية ، وتكتب بإستخدام مجموعة الرموز التالية ، والتي تستخدم لكتابة النظام العشري: 0 1 2 3 4 5 6 7 8 9

أما في النظام الستة عشر ، فإنك تستخدم أحد البادئتين التاليتين: **0x** و **0X** ، و تستخدم الرموز التالية:

0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

و توجد في السي شارب عدة أنواع مخصصة للأعداد الصحيحة ، وهي :

int, uint, long, ulong, byte, sbyte, short,ushort

و الجدول التالي يوضح الفرق بينهما من حيث سعة كل واحد منها :

النوع	حدودها		سعتها
	أصغر قيمة	أكبر قيمة	
sbyte	-128	127	عدد صحيح 8 بت
byte	0	255	عدد صحيح موجب 8 بت
short	-32,768	32,767	عدد صحيح 16 بت
ushort	0	65,535	عدد صحيح موجب 16 بت
int	-2,147,483,648	2,147,483,647	عدد صحيح 32 بت
uint	0	4,294,967,295	عدد صحيح موجب 32 بت
long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	عدد صحيح 64 بت
ulong	0	18,446,744,073,709,551,615	عدد صحيح موجب 64 بت

قد تكون الصورة الآن قد اتضحت ، لماذا عندنا أكثر من نوع ؟ والجواب حتى نجعل نوفر في استهلاك الذاكرة و في تحسين سرعة أداء البرنامج .

وتوجد في لغة السي شارب عدة لواحق للأعداد الصحيحة و هي :

```
U u L l UL Ul uL ul LU Lu lU lu
```

و لها عدة قواعد :-

1- إذا كتب الرقم بدون لاحقة ، فإن المترجم سوف يعتبره من النوع **int** فإذا كان أكبر منه فإنه يعتبره من نوع **uint** فإن لم يناسبه ، فإنه يعتبره من نوع **long** ، و إلا فإنه يعتبره من نوع **ulong** و الترتيب هنا مهم جداً .

2-إذا كانت اللاحقة **U** أو **u** فإنه يعتبر من نوع **uint** فإذا كان أكبر من ذلك النوع فهو من نوع **ulong** .

3- إذا كانت اللاحقة **L** أو **l** فإنه يعتبر من نوع **long** فإذا لم يسعه فهو من نوع **ulong**.

4- إذا كانت اللاحقة **UL** أو **Ul** أو **uL** أو **ul** أو **LU** أو **Lu** أو **lU** أو **lu** فإنه يعتبر من نوع **ulong** .

طيب سوف تقول لي إذا كان نوع **ulong** لا يسعه ماذا سيحدث ؟ سأجيب بكل بساطة سوف يعطيك خطأ لا أكثر و لا أقل .

طيب سأسألك أنا الآن سؤالا ؟ ما الذي يجري في الجملة التالية :

```
long x = 5;
```

ستجيب حسب القواعد الماضية أن 5 ستكون من نوع **int** ، لأنها لا توجد بها لاحقة ، و بما أننا نريد أن نسندها إلى نوع **long** فإننا سوف ننفذ عليها عملية تحويل ضمني من **int** إلى **long** ، تكمن فائدة هذه اللواحق في توفير الوقت اللازم لتحويل الضمني من **int** إلى النوع المراد ، مما يؤدي إلى تحسين أداء و سرعة البرنامج ، خذا المثال الثاني :

```
sbyte x = 5;
```

هنا سوف تتم عملية التحويل من **int** إلى **sbyte** ضمنيا مخالف للقواعد التحويل التي سوف نأخذها لاحقا ، حيث أن هذه قاعدة خاص بالرموز الصحيحة مثل : 1 ، 2 ، 3... إلخ فقط ، حيث يتم تحويلها من رمز إلى نوع **int** و من ثم إلى الأنواع التي أقل منه ضمنيا ، ولكي تتضح الصورة عندك خذ المثال التالي :

```
int x =5, y =4;
```

```
sbyte a= x , b = y;
```

ستجد أن المترجم يعطيك خطأ فحواه أنك لا تستطيع أن تحويل **int** إلى **sbyte** تحويلا ضمنيا ، لأن **x** و **y** هما متغيران و ليس رموز.

هذا الكلام ينطبق على جميع الأنواع التي هي أصغر من نوع **int** ، أقصد : **byte - sbyte - short - ushort**

وموضوع التحويل الضمني و الصريح بين الأنواع سوف نتناوله بالتفصيل لاحقا .

الأعداد الحقيقية :

و هي تنقسم إلى قسمين : قسم من نوع الفاصلة العائمة **float** و **double** ، و قسم من النوع العشري **decimal** .

لا تقلق الجدول التالي يبين لك تفصيلا عن هذه الأنواع

النوع		حدوده (تقريبا)	دقة العرض
الفاصلة العائمة	**float**	$\pm 1.5 \times 10^{-45}$ إلى $\pm 3.4 \times 10^{38}$	7 خانات
	double	$\pm 5.0 \times 10^{-324}$ إلى $\pm 1.7 \times 10^{308}$	15 إلى 16 خانة
العشري	**decimal**	$\pm 1.0 \times 10^{-28}$ إلى $\pm 7.9 \times 10^{28}$	28 إلى 29 خانة

و تستطيع أن تكتب الأعداد الحقيقية في لغة سي شارب بأن تضع في الأرقام فاصلة عشرية مثل :

```
-12.01 ، 12.0 ،1.5
```

أو أن تضيف إليها الأس العشري مثل :

```
-12.5 E10، 10 e-3
```

أو يمكنك أن تضيف إليها هذا اللواحق :

```
F f m M D d
```

مثل: **2f ، 10D**

و كما في الأعداد الصحيحة يقوم المترجم بإتباع الخطوات التالية للتميز بين الأنواع المختلفة من الأعداد الحقيقية :

1- إذا لم تكن هناك لواحق فإن الرقم يعتبر من النوع **double**

2- إذا كانت اللاحقة هي **f** أو **F** فإن الرقم يعتبر من النوع **float** مثال على ذلك :

```
1f ، 1.5f ، 1e10F
```

3- إذا كانت اللاحقة هي **d** أو **D** فهو من النوع **double** مثال على ذلك :

```
1d ، 1.5d ، 1e10d
```

4- إذا كانت اللاحقة هي **m** أو **M** فهو من النوع **decimal** مثال :

```
1m ، 1.5m ، 1e10M
```

الحروف :

يوجد نوع واحد يمثل الحروف ألا وهو **char** و هو من نوع الأعداد الصحيحة وهو يسع لجميع محارف الترميز العالمي **Unicode** كما يبين الجدول التالي :

نوع	الحدود	الحجم
char	U+0000 إلى U+ffff	Unicode 16 bit

و كنا قد أشارنا سابقا أن النوع يمكن أن يستقبل الحروف و السلاسل الحرفية بشرط أنتكون بين علامتي اقتباس مفردتين ' '

و يمكنك أن تكتب الحرف بما يعادله في النظام الستة عشر مثل :

```
'\x1120' ، '\x ffff'
```

و كذلك يمكنك أن تكتبه بما يعادلها في نظام الينوكود (نظام الترميز العالمي) مثال ذلك :

```
'\u0058' ، '\U0058'
```

المثال التالي يعطيك صورة شاملة :

```
char MyChar = 'X';          // الحروف العادية
char MyChar = '\x0058';     // نظام الستة عشر
char MyChar = (char)88;     // التحويل من عدد صحيح
char MyChar = '\u0058';     // نظام الترميز العالمي اليونكود
```

السلاسل الحرفية :

وهي تمثل بنوع **string** و قد تكون عندك فكرة مسبقة عنها ، و لكن أحببت أن أشير إلى أنه يمكنك أن تضيف حرف @ قبل علامتي الاقتباس " " حيث تظهر ما بداخلها كما هي مثال ذلك :

```
string x = "Welcome \t C#"   \\ Welcome C#
string x = @"Welcome \t C#"  \\ Welcome \t C#
```

التحويل بين الأنواع :

من خلال استعراضنا أنواع البيانات ، يمكننا أن نتصور أهمية التحويل نوع إلى نوع آخر حتى يعمل برنامجنا بالشكل المطلوب يوجد هناك نوعان من التحويل ، تحويل من نوع توسيع (الترقية) أي أنك تقوم بترقية ذلك الصنف من البيانات من صنف صغير إلى صنف كبير مثلا من **sbyte** إلى **int** و من **float** إلى **double** ، و النوع الآخر من التحويل التضييق حيث تقوم بخفض النوع الكبير إلى النوع الصغير مثل **long** إلى **int** و من **double** إلى **float** .

في الغالب الأعم يتم التحويل من نوع الترقية بشكل ضمني بدون تدخل المبرمج ، حيث يقوم المترجم بهذه المهمة ، خذ المثال التالي :

```
sbyte x = 5;
int total;
total = x + 6;
```

في هذا المثال سوف يقوم المترجم بعملية تحويل **x** من **sbyte** إلى **int** حتى يقوم بعملية الجمع من نوع **int** .

لا تنس أن المترجم يقوم بإنشاء نسخة من **x** ، ثم يقوم بتحويلها إلى **int** حتى يقوم بعملية الجمع فقط .

الجدول التالي يبين لك التحويلات الضمنية التي يمكن للمترجم أن يقوم بها إذا دعت الضرورة :

من	إلى
sbyte	short و int و long و float و double و decimal
byte	short و ushort و int و uint و long و ulong و float و double و decimal
short	int و long و float و double و decimal
ushort	int و uint و long و ulong و float و double و decimal
int	long و float و double و decimal
uint	long و ulong و float و double و decimal
long	float و double و decimal
char	ushort و int و uint و long و ulong و float و double و decimal
float	double
ulong	float و double و decimal

أما النوع الثاني من التحويلات و هو نوع التضييق فإن في الغالب يتطلب تصريح من المبرمج لفعل ذلك ، و سبب ذلك أنه قد يتسبب هذا النوع من التحويلات إلى فقدان في المعلومات ، تصور لو أننا حولنا 1.65 إلى نوع صحيح **int** ماذا سينتج ؟؟

سوف ينتج 1 فقط لذا خذ الحذر عندما تتعامل مع هذا النوع من التحويلات .

قبل أن أنسى هذا النوع يستعمل معامل التشكيل لإتمام مهمته ، فالمبرمج يجب عليه التصريح بأنه يريد التحويل التضييق في العادة باستخدام معامل التشكيل .

و التحويل الصريح يمكن أن يحول أي نوع إلى آخر فهو يستطيع أن يقوم بعمل التحويل الضمني (الجدول السابق) أضف إلى ذلك عكس عملية التحويل أي إذا كان ضمنيا يستطيع تحويل نوع **A** إلى النوع **B** فإنك تستطيع تحويل نوع **B** إلى نوع **A** باستخدام التحويل الصريح ، و الجدول التالي يبين لك ذلك :

من	إلى
sbyte	byte و ushort و uint و ulong وchar
byte	sbyte وchar
short	sbyte و byte و ushort و uint و ulong وchar
ushort	sbyte و byte و short وchar
int	sbyte و byte و short و ushort و uint و ulong وchar
uint	sbyte و byte و short و ushort و int وchar
long	sbyte و byte و short و ushort و int و uint و ulong وchar
ulong	sbyte و byte و short و ushort و int و uint و long و char
char	sbyte و byte وshort
float	sbyte و byte وshort و ushort و int وuint وlong و ulong وchar وdecimal
double	sbyte و byte و short و ushort و int و uint و longوulong وchar و float وdecimal
decimal	sbyte و byte و short و ushort و int و uint و long و ulong و char و float وdouble

و إلى هذه النقطة ينتهي هذا الدرس .

زايد السعيدي
منتدى الإبداع الإسلامي