المصفوفات:

هي عبارة عن مجموعة ذات حجم ثابت من العناصر التي لها نفس نوع البيانات ولكنها تختلف في القيم المُسندة إليها.

يحتاج المبرمج في كثير من الأحيان إلى تخزين البيانات بعد قراءتها بهدف الرجوع إليها في إجراء بعض العمليات، وعندها يحتاج إلى التصريح وبشكل منفصل عن عدة متحولات لها نفس نوع البيانات مثلاً num1, num2, ...num50 ، وهنا تبرز أهمية المصفوفات في تخزين قيم عدة متحولات من نفس النوع ضمن تسلسل معين مما يسهّل الوصول إليها والتعامل معها لاحقاً، حيث يتم تعريف متحول واحد من نوع مصفوفة وليكن على سبيل المثال numbers ، ثم يتم التعامل مع عناصر هذه المصفوفة بشكل مستقل عن طريق الـ Index (الدليل)الخاص بهذا العنصر

numbers[0], numbers[1], numbers[2],.....

أنواع المصفوفات:

- المصفوفات أحادية البعد (vector).
- المصفوفات ثنائية البعد.

التصريح عن مصفوفة أحادية البعد: يتم التصريح عنها بالشكل التالي:

```
dataType[]  arrayName = new dataType[n];
```

حيث:

dataType : نوع عناصر المصفوفة (int , float , string, ...).

arrayName : اسم المصفوفة.

n : عدد عناصر المصفوفة، ويسمى طول المصفوفة(length)

مثال:

```
int marks [ ] = new int [6] ;
```

هنا تم التصريح مصفوفة اسمها marks مؤلفة من 6 أعداد صحيحة.

للوصول إلى أي عنصر من عنصر المصفوفة لا بد من استخدام دليل يدل على هذا العنصر، علماً ان ترتيب عناصر المصفوفة يبدأ من الصفر، وبالتالي فإن:

ترتيب آخر عنصر من عناصر المصفوفة = عدد العناصر - ١

- يمكن اسناد قيم ابتدائية لعناصر المصفوفة مباشرة أثناء التصريح عنها كما في المثال التالي:

```
int marks [ ] = new int [6] {77, 99, 85, 66, 90} ;
```

في هذا المثال تم اسناد قيم لعناصر المصفوفة أثناء التصريح عنها، وبالتالي فإن:

marks[0]=77
marks[1]=99
marks[2]=85
marks[3]=66
marks[4]=90

إدخال قيم عناصر المصفوفة:

يتم التعامل مع عناصر المصفوفة كأي مجموعة من المتغيرات المشتركة بالنوع، لكن كونها تقع ضمن مصفوفة فإن هذا الأمر يقدم مجموعة من الميزات كإمكانية العودة إليها باستمرار كما ذكرنا سابقا، وكذلك إمكانية إدخال أو قراءة قيمها بشكل أبسط وأسرع من إدخال قيمة كل متغير على حدا، الأمر الذي يصبح معقدا جدا عند التعامل مع أعداد ضخمة من المتغيرات، يمكننا استخدام الحلقات للتعامل مع عناصر المصفوفة مثل For, Foreach، مع ملاحظة أن الحلقة foreach تستخدم فقط لقراءة عناصر المصفوفات وليس لتعديل هذه العناصر.

مثال:

```
static void Main(string[] args)
{

   string[ ] courses = new string[ 4] {"Programming 2" ,
                "Math 2" , "English", "Arabic" } ;

   foreach (  string  c  in  courses  )
   {
       Console.WriteLine( c  ) ;
   }
     }
```

مثال: اكتب برنامج يطلب من المستخدم إدخال عناصر مصفوفة مؤلفة من ١٠ عناصر، ثم يقوم بطباعة مجموع عناصر المصفوفة على الشاشة.

```
static void Main(string[] args)
{

   int[ ] myarr = new int[10];
   int elements_sum = 0;
   int  i ;

   Console.WriteLine("Enter the elements of array");
   for (  i = 0  ;  i < myarr.length  ;  i++)
   myarr [ i ] = int.Parse(Console.ReadLine());


   for (  i = 0 ; i < numbers.length ;  i++)
   elements_sum + = myarr [ i ] ;

   Console.WriteLine("Sum of array elements = {0}",
                        elements_sum ) ;
}
```

مثال: اكتب برنامج يطلب من المستخدم إدخال عناصر مصفوفة مؤلفة من ٣٠ عنصر، ثم يقوم بطباعة عدد مرات تكرار كل عنصر من عناصر المصفوفة.

```
static void Main(string[] args)
{
   int[ ] numbers = new int[30];
   int i , j , count = 0;

   Console.WriteLine("Enter the elements of array");
   for (  i = 0 ; i < numbers.length ; i++)
     numbers [ i ] = int.Parse(Console.ReadLine());

   for ( i = 0 ; i < numbers.length ;  i++)
   {
     for (  j = 0 ; j < numbers.length ;  j++)
     if (numbers[ i ] == numbers[ j ])
       count++;
     Console.WriteLine("count of element  {0} = {1} " ,
                numbers[i] , count);
     count = 0;
       }
 }
```

المصفوفات ثنائية البعد: تتكون من أكثر من سطر وأكثر من عمود ، ويتم التصريح عنها بالشكل التالي:

```
dataType[ , ]  arrayName = new dataType[n,m];
```

حيث:

dataType	:	نوع عناصر المصفوفة .
arrayName	:	اسم المصفوفة.
n	:	عدد أسطر المصفوفة.
m	:	عدد أعمدة المصفوفة.

مثال:

```
int  arr [ , ] = new  arr [5 , 3] ;
```

عرفنا مصفوفة اسمها arr مؤلفة من 5 أسطر و ٣ أعمدة ، عناصر هذه المصفوفة أعداد صحيحة.

للوصول إلى أي عنصر من عنصر المصفوفة الثنائية نحتاج إلى دليل للأسطر ودليل للأعمدة.

وكما في المصفوفات الأحادية يمكن إسناد القيم مباشرة لعناصر المصفوفة عند التصريح عنها.

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

مثال: اكتب برنامج يطلب من المستخدم ادخال عناصر مصفوفة ثنائية البعد على أن يتم تحديد عدد العناصر من قبل المستخدم أيضاً، ثم:

- طباعة عناصر المصفوفة بشكلها الرياضي
- طباعة اكبر عدد في المصفوفة، وكذلك أصغر عدد.
- طباعة مجموع عناصر المصفوفة بكاملها ثم المتوسط الحسابي للعناصر.
- طباعة مجموع عناصر القطر الرئيسي، ومجموع عناصر القطر الثانوي.
- طباعة مجموع عناصر السطر الاول والعمود الأول.

```
static void Main(string[ ] args)
{
  Console.WriteLine("Please enter the number of  Rows:");
  int n = int.Parse(Console.ReadLine());

  Console.WriteLine("Please enter the number of Columns:");
  int m = int.Parse(Console.ReadLine());

  int [ , ] x = new int [n , m] ;
  int  sum = 0, sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;

  Console.WriteLine("Please enter elements of  Array ");
  for ( int i= 0 ;  i < n ;  i++)
   for ( int  j = 0 ;  j < m ;  j++)
     {
        Console.Write  (" x[{0},{1}]=  ", i , j ) ;
         x[ i , j ] = int.Parse(Console.ReadLine());
      }

  for (int  i = 0; i < n; i++)
  {
    for ( int j = 0; j < m ; j++)
     {
           Console.Write  (x[i , j ] + "\t");
     }
    Console.WriteLine( );
  }

 int min = x[0, 0];
 int max = x[0, 0];

 for (int   i = 0 ; i < n ; i++)
 for (int  j = 0 ; j < m ; j++)
   if (x[ i, j ] < min)
            min = x[i, j];
  Console.WriteLine("minimum element in Array =  {0}" , min);

 for (int  i = 0; i < n; i++)
  for (int  j = 0; j < m; j++)
    if (x[ i ,  j ] > max)
           max = x[i, j];
 Console.WriteLine("maximum element in Array  =  {0}" , max);

for (int  i = 0 ; i < n ; i++ )
for ( int  j = 0 ;  j <m ;  j++)
       sum += x[ i , j ] ;
Console.WriteLine("Sum of  Array elements  =  " + sum ) ;
```

```
int  z = n * m;
double  avg = (double) sum / z;
Console.WriteLine("Avarage of Array elements  =  "+ avg);


for ( int  i = 0 ; i < n ; i++)
   sum1 +=  x [ i , i ] ;
Console.WriteLine("Sum of  the main  = " + sum1) ;

int k = m - 1;
for (int i = 0 ; i < n ; i++)
{
  if (k >= 0)
        sum2 += x[i, k];
  k-- ;
 }
Console.WriteLine("Sum of  secondary =  " + sum2);


for (int  j = 0 ; j < m ; j++)
    sum3 +=  x[ 0, j ] ;
Console.WriteLine("Sum of  the first row =  " + sum3);

for (int i = 0 ; i < n ; i++)
sum4 += x[ i , 0] ;
Console.WriteLine("Sum of  the first column = " + sum4);
}
```

مثال: اكتب برنامج يطلب من المستخدم إدخال عناصر مصفوفة محارف أبعادها ٣*٣، ثم يقوم بالبحث ضمن عناصر المصفوفة عن عنصر يحدده المستخدم ويعيد عدد مرات تكرار هذا العنصر في حال وجوده.

```
static void Main(string[] args)
{
  char[,] myarr = new char[3, 3];
  int c = 0;

  Console.WriteLine("Please enter the char which you
                     search");
  char k = char.Parse(Console.ReadLine());

  Console.WriteLine("Enter the elements of array:");
  for(int i=0 ; i<3 ; i++)
```

```
for(int j=0 ; j<3 ; j++)
    myarr[i, j] = char.Parse(Console.ReadLine());

for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
  if (myarr[i, j] == k)
            c++;

if (c == 0)
  Console.WriteLine("Not Found");
else
  Console.WriteLine("Found, " + c + "Times");
}
```

الطرق (الدوال أو التوابع) Methods:

الطريقة هي جزء من البرنامج تقوم بأداء مهمة معينة وتعيد نتيجة ما، ويمكن أن تُمرر لها بارمترات من البرنامج المستدعي.

الهدف الأساسي من استخدام الطرق هو منع تكرار كتابة التعليمات البرمجية لأكثر من مرة، مما يؤدي إلى تقليل أسطر البرنامج، وبالتالي سهولة تتبع الأخطاء في حال حدوثها في البرنامج، هذا بالإضافة إلى أن الطرق تجعل البرنامج منظم بشكل أفضل.

يمكن استخدام الطريقة بعد التصريح عنها، عن طريق استدعاء هذه الطريقة في البرنامج أو في دالة أخرى، وذلك بذكر اسم الطريقة وتمرير قيم مناسبة للبارمترات الخاصة بهذه الطريقة.
عند استدعاء طريقة ما، ينتقل سير البرنامج إلى تلك الطريقة ويتم تنفيذ التعليمات الموجودة ضمنها، ثم تتم العودة إلى البرنامج الذي قام بالاستدعاء.

التصريح عن الطريقة (Method): يتم التصريح عن الطريقة أو الدالة بالشكل التالي:

```
accessModifier returnType methodName(Type Parameter1, Type Parameter2 ,…..)
{
    التصريح عن المتغيرات المحلية الخاصة بالطريقة
    .....
    تعليمات الطريقة
    .....
    return …….
}
```

accessModifier : محدد الوصول لتلك الطريقة.
returnType : نوع القيمة التي ستعيدها الطريقة، اذا كانت الطريقة تعيد قيم.
methodName : اسم الطريقة.
Type parameter1 : نوع واسم الوسيط الاول من وسطاء الطريقة.

Type parameter2 : نوع واسم الوسيط الثاني من وسطاء الطريقة.

.....

ملاحظات:

١. يجب أن يتطابق نوع القيمة التي ستتم إعادتها في تعليمة return مع النوع المحدد في القيمة المعادة returnType.

٢. في حال كانت الطريقة لا تعيد قيمة عندها تكون القيمة المعادة returnType هي **void**، وعندها لا نستخدم تعليمة return .

٣. يجب أن يتطابق عدد ونوع البارمترات المستخدمة عند التصريح عن الطريقة مع عدد ونوع البارمترات التي سيتم تمريرها إلى هذه الطريقة عند استدعائها.

المتغيرات المحلية (local) والمتغيرات العامة (global):

المتغيرات المحلية: هي المتغيرات التي يتم الإعلان عنها داخل الطريقة، وتسمى بهذا الاسم لأنها تكون غير معروفة خارج الطريقة المعرفة ضمنها، ومن الجدير بالذكر أن المتغير المحلي تنشأ له قيمة وكيان عند ابتداء تنفيذ الطريقة التي ينتمي إليها فقط، وتختفي عند الانتهاء من تنفيذ تلك الطريقة.

المتغيرات العامة: هي التي تكون قيمتها معروفة من أول البرنامج إلى آخره، ويمكن استعمالها في أي جزء منه، وتحتفظ بقيمتها أثناء تنفيذ البرنامج.

مثال: طريقة لجمع عددين صحيحين.

```
static int sum(int number1, int number2)
{
    int total = number1 + number2;
    return total;
}
```

وفي الـ main يتم استدعاء الطريقة بكتابة اسمها وتمرير قيم مناسبة للبارمترات.

```
int result = sum(5, 8);
Console.Write(result);
Console.ReadKey();
```

نلاحظ في هذا المثال أن المتغير total هو متغير محلي (local) ضمن الدالة sum وبالتالي لا نستطيع استخدامه خارج الطريقة.

مثال: اكتب طريقة تقوم بطباعة العبارة " Hello , in C# " على الشاشة :

```
static void print_hello()
 {
    Console.writeLine("Hello, in c#");
 }
Static void Main (string [ ] args)
  {
    Print_hello();
  }
```

نلاحظ من المثال السابق أن الأقواس ضرورية عند التصريح عن الطريقة وكذلك عند استدعائها حتى وان لم تكن هذه الطريقة تستخدم أية بارمترات.

تمرير البارمترات للدالة أو الطريقة:

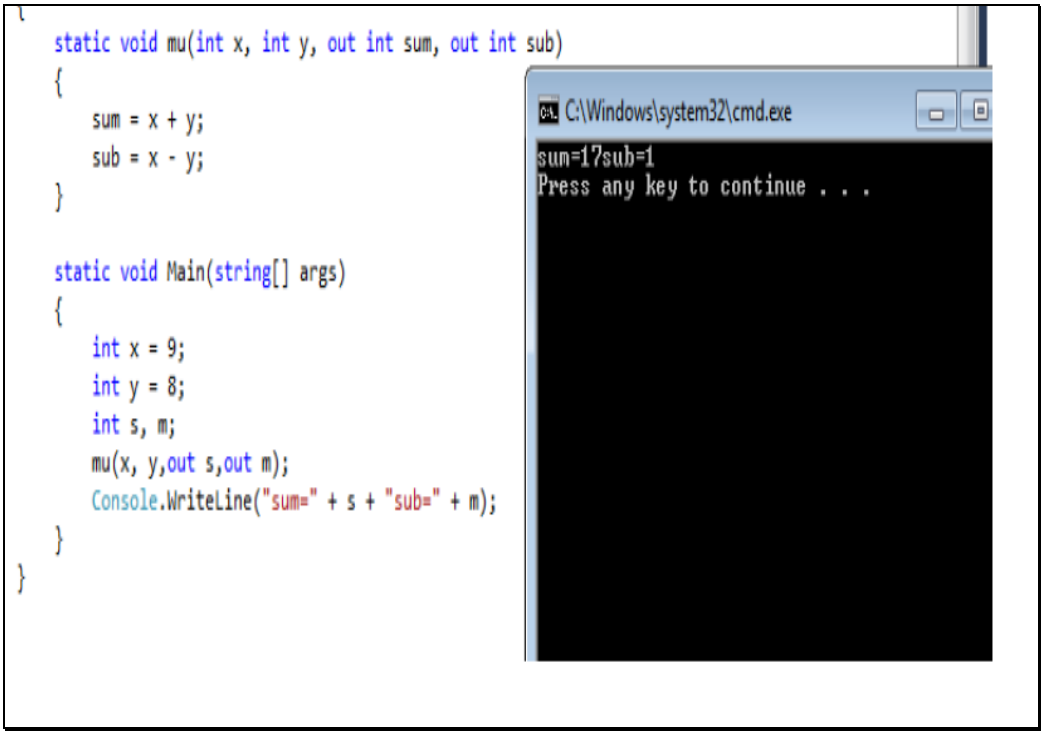
هناك عدة طرق لتمرير البارامترات إلى الطريقة، وهي:

١. التمرير بالقيمة (Passing By Value): في هذه الطريقة أي تعديلات تطرأ على قيمة البارامتر ضمن الطريقة لن تؤثر على القيمة الأساسية لهذا المتغير في البرنامج المستدعي، لأنه في هذه الحالة يتم عمل نسخة من البيانات وإرسالها إلى الطريقة، لتقوم الطريقة باستخدام هذه النسخة، وبالتالي أي تعديلات تطرأ ستؤثر على هذه النسخة ولن تتأثر القيمة الأصلية بها.

٢. التمرير بالمرجع (Passing By Refernce): في هذه الطريقة أي تعديلات تطرأ على قيمة البارامتر ضمن الطريقة ستُغير القيمة الأساسية لهذا المتغير في البرنامج المستدعي، للدلالة على أن هذا البارامتر سيُمرر بالمرجع لابد من استخدام الكلمة ref قبل البارامتر وذلك عند التصريح عن الطريقة وكذلك عند استدعائها.

٣. استخدام بارامترات الخرج (Out Parameters): بارامترات الخرج مشابهة إلى حد بعيد لبارامترات المرجع لكنها تختلف باستخدام الكلمة out قبل اسم البارامتر عند التصريح وعند الاستدعاء، وكذلك تختلف بأن بارامترات الخرج لا تحتاج إلى تهيئتها بقيم ابتدائية قبل استدعاء الطريقة.

في المثال التالي تم التصريح عن الطريقة mu التي تقوم بجمع عددين وتعيد ناتج الجمع في بارامتر الخرج sum وناتج الطرح في بارامتر الخرج sub



```
{
    static void mu(int x, int y, out int sum, out int sub)
    {
        sum = x + y;
        sub = x - y;
    }

    static void Main(string[] args)
    {
        int x = 9;
        int y = 8;
        int s, m;
        mu(x, y,out s,out m);
        Console.WriteLine("sum=" + s + "sub=" + m);
    }
}
```

```
C:\Windows\system32\cmd.exe
sum=17sub=1
Press any key to continue . . .
```

نلاحظ أنه في الطريقة الرئيسية للبرنامج (Main) تم تمرير البارمترات s, m للطريقة mu بدون أن يتم تهيئتهما بقيم ابتدائية.

مثال:

```
Static   void  set_value ( int  x  )
  {
     X = 100 ;
  }
Static void Main (string [ ] args)
  {
    int   my_value = 5 ;
```

```
    Set_value (  my_value ) ;
Console.writeLine ("The value is {0}", my_value);
  }
```

تم في هذا المثال تمرير البارامتر للطريقة بواسطة القيمة، وبالتالي فإن ناتج التنفيذ هو :

The value is 5 .

بينما لو قمنا بتمرير البارامتر بواسطة المرجع:

```
Static   void  set_value ( ref int  x )
 {
     X = 100 ;
 }
Static void Main (string [ ] args)
  {
    int   my_value = 5 ;
    Set_value ( ref my_value) ;
Console.writeLine ("The value is {0}", my_value);
  }
```

فإن ناتج التنفيذ هو :

The value is 100

التراكيب (السجلات) Structures:

السجل هو نوع بيانات يتم تعريفه من قبل المبرمج، وهو عبارة عن بنية معطيات تضم مجموعة متحولات من أنواع مختلفة، ويمكن أن يتضمن عدداً من الطرق Methods، متحولات وطرق السجل تسمى (أعضاء السجل).

التصريح عن التركيب (السجل): يجب أن يتم التصريح عن التركيب أو السجل خارج الدالة الرئيسية main، وله الشكل العام التالي:

```
struct  structure_name
 {
   access_modifier   dataType    var1 ;
   access_modifier   dataType    var2 ;
  ........
  ..........
 ........
 }
```

حيث:

structure_name: اسم التركيب أو السجل.

Access_modifier : محدد الوصول لهذا المتحول (العضو) من خارج السجل.

dataType var1 : نوع واسم العضو (المتحول) الاول من أعضاء السجل.

dataType var2 : نوع واسم المتحول الثاني من أعضاء السجل.

.....

مثال:

المطلوب بناء بنية معطيات تسمح بتخزين اسم المستخدم وعنوانه وعمره، ثم طباعة هذه المعلومات على الشاشة، عند تسجيل دخول مستخدم جديد.

```
namespace Person_struct
{
    0 references
    class Program
    {
        1 reference
        struct person
        {
            public string name;
            public string address;
            public int age;

        } 2
        0 references
        static void Main(string[] args)
        {
            person p;
            Console.Write("Please, Enter Your Name: ");
            p.name = Console.ReadLine();

            Console.Write("Please, Enter Your Address: ");
            p.address = Console.ReadLine();

            Console.Write("Please, Enter Your Age: ");
            p.age = int.Parse(Console.ReadLine());

            Console.WriteLine("************************************");
            Console.WriteLine("Hello {0}, ", p.name);
            Console.WriteLine("Your address is {0}, and you are {1} years old. ", p.address, p.age );

            Console.ReadKey();
        }
    } }
```

اللوائح أو القوائم (Lists) :

اللوائح تشبه المصفوفات لكن تختلف عنها بأن:

المصفوفات لها عدد عناصر ثابت يُحدد عند التصريح عنها ، بينما اللوائح متغيرة الحجم، و يمكن إضافة عناصر جديدة لها.

التصريح عن اللوائح:

```
List<Type>  list_name  =  new List <Type> ();
```

حيث:

Type : نوع عناصر اللائحة.

List_name : اسم اللائحة .

مثال:

```
List <string>  my_friends = new  list <string > ();
```

تم هنا التصريح عن لائحة اسمها my_friends عناصرها من نوع string .

- نلاحظ أنه لم يتم تحديد عدد عناصر اللائحة في التصريح وذلك لأنها متغيرة الحجم.
- يمكن اسناد عناصر للائحة مباشرة أثناء التصريح عنها:

```
List <string>  my_friends = new  list <string > {"Adel","Rana","Sami"};
```

- نستخدم الطريقة أو الدالة add لإضافة عناصر إلى اللائحة،

```
My_friends.add("friend1");
My_friends.add("friend2");
```

ملاحظة: العناصر التي تتم إضافتها باستخدام الطريقة add تضاف إلى نهاية اللائحة تلقائياً.

مثال:

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        string friend;
        List<string> my_friends = new List<string>();
        Console.Write("Enter number of your friends: ");
        int friends_number = int.Parse(Console.ReadLine());

        Console.WriteLine("Please, Enter names of your friends");
        for (int i = 0; i < friends_number; i++)
        {
            friend = Console.ReadLine();
            my_friends.Add(friend);
        }
        Console.WriteLine("++++++++++++++++++++");
        Console.WriteLine("My friends are:");
        foreach (string f in my_friends)
        {
            Console.WriteLine(f);
        }
        Console.ReadKey();
    }
}
}
```

وعند تنفيذ البرنامج:

```
Please, Enter names of your friends
Soha
Rami
Lana
++++++++++++++++++++
My friends are:
Soha
Rami
Lana
```

بعض الخصائص و الطرق الخاصة باللوائح:

الخاصية أو الدالة	الاستخدام
Count	تحدد عدد عناصر اللائحة (طول) بعدد صحيح.
Sort	تقوم بترتيب عناصر اللائحة تصاعديا إذا كانت رقمية وترتيب هجائي إذا كانت نصية.
Reverse	تقوم بعكس ترتيب عناصر المصفوفة
add	يضيف عنصر للائحة يتوضع في نهاية اللائحة.
Insert	يضيف عنصر جديد للائحة بحيث يحدث إزاحة للعنصر الذي سيحل مكانه ويأخذ دليله.
Remove	لإزالة عنصر من اللائحة
Capacity	يحدد حجم اللائحة.
GetType	يحدد نوع عناصر اللائحة.
Sum	يقوم بجمع عناصر اللائحة إذا كانت من نوع int.
Contains	يقوم باختبار وجود قيمة ما ضمن اللائحة.
RemoveAt	يقوم بإزالة عنصر من اللائحة عن طريق تحديد دليله
RemoveRange	يقوم بمسح مجال من العناصر عن طريق تحديد عرض المجال ونقطة البداية.
Clear	يقوم بمسح جميع عناصر اللائحة.

مثال:

```
class Program
{
    static void printList(List <int> lists)
    {
        Console.WriteLine("\nElements of List is:\n");
        foreach (int i in lists)
        {
            Console.Write("\t{0}", i);
        }
        Console.WriteLine("\n");
    }

    static void Main(string[] args)

   {
      List<int> numbers = new List<int> { 14, 25, 33, 66, 78, 2, 8 };
      Console.WriteLine("Get length:\t{0}", numbers.Count);
      Console.WriteLine("Get Type:\t{0}", numbers.GetType());
      Console.WriteLine("Get Size:\t{0}", numbers.Capacity);
      Console.WriteLine("sum element:\t{0}", numbers.Sum());
      numbers .Sort();
      printList(numbers);
      numbers.Reverse();
      printList(numbers);
      numbers.Insert(1, 88);
      printList(numbers);
      numbers.Remove(8);
      printList(numbers);
      numbers.Add(96);
      printList(numbers);
      if (numbers.Contains  (25))
      {
          Console .WriteLine ("The Lists contains of 25");
      }
      numbers.RemoveAt(1);
      printList(numbers);
      numbers.RemoveRange(2, 4);
      printList(numbers);
      numbers.Clear();
      printList(numbers);
   }
}
```

الفئات (الأصناف) Classes:

يعتبر مفهوم الفئات واحد من أفضل ميزات لغة C#،

الفئة أو الكلاس هي عبارة عن مفهوم مجرد يُستخدم لتعريف نمط معطيات جديد، ويمكن أن يضم مجموعة من البيانات تسمى حقول (fields) ومجموعة من الدوال (methods) التي تعمل على هذه البيانات، بالإضافة إلى مجموعة من الخصائص (Properties) التي تطبق على حقول الفئة.

يبدأ تعريف الكلاس بالكلمة المحجوزة class يليها اسم اختياري للصف (يخضع لقاعدة تعريف الأسماء). يتم تحديد جسم الكلاس بواسطة قوسين { }، نعرّف ضمن هذين القوسين جميع أعضاء الكلاس.

التصريح عن الكلاس (الفئة): يتم التصريح عن الكلاس بالشكل التالي:

```
class class_name
{
    access_modifier  dataType   var1 ;
    access_modifier  dataType   var2 ;
   ..............
   access_modifier  dataType   Property1
   { get { return var ; }
     set {  var = value; }
    }
   ..............
   access_modifier  return_value  method_name  (parameters )
   {
     ........
   }

}
```

حيث:

class_name : اسم الكلاس .

access_modifier : محدد الوصول لمتحولات وطرائق الكلاس.

dataType var1 : نوع واسم الحقل (المتحول) الأول من متحولات الكلاس.

dataType var2 : نوع واسم الحقل (المتحول) الثاني من متحولات الكلاس.

.....

تجدر الإشارة إلى أن حقول الكلاس يمكن أن تكون من أي نوع ويمكن أن تكون أغراض مشتقة من صفوف أخرى أيضاً.

محدد الوصول (Access Modifier) :

يحدد محدد الوصول طريقة التعامل مع أي عضو من أعضاء الكلاس، ومن أهم محددات الوصول:

Private	أعضاء الكلاس التي نحدد محدد الوصول إليها كـ Private يتم التعامل معها فقط في الكلاس المُعرّفة ضمنه ولا يمكن الوصول إليها أبدا خارج هذا الكلاس، وهذا المحدد هو المحدد الافتراضي.
Public	أعضاء الكلاس التي نحدد محدد الوصول إليها كـ Public يمكن التعامل معها في الكلاس المعرّفة ضمنه وكذلك يمكن التعامل معها خارج هذا الكلاس.

كما ذُكر سابقاً فإن الكلاس هو نمط معطيات، لذلك نحتاج لتعريف متحولات نوعها هو هذا النمط، هذه المتحولات يُطلق عليها اسم أغراض (**objects**)،

نُعرّف الغرض(object) بالشكل التالي:

class_name object_name = **new** *class_name*( ) ;

حيث:

class_name هو اسم الكلاس.

object_name هو اسم الغرض *المُشتق* من الكلاس.

الخصائص (Properties):

نستخدمها للوصول إلى الحقول المُعرّفة كـ Private، وتُعرّف بالشكل العام التالي:

```
اسم الخاصة  نوع الخاصة  محدد الوصول
{
 get
  { .....  }
 set
  { .....  }
}
```

Get تقوم بإعادة القيمة الحالية لعضو الكلاس وذلك عن طريق تعليمة return.

Set تقوم بإسناد قيمة جديدة لعضو الكلاس وذلك عن طريق الكلمة المحجوزة value.

التابع الباني (Constructor): هو نوع خاص من الطرق (methods) يتميز بأنه:

- له نفس اسم الكلاس.
- لا يعيد أي قيمة
- نستخدمه لإعطاء قيم ابتدائية لحقول الكلاس.
- يتم استدعاؤه ضمنياً وبشكل تلقائي عند انشاء غرض من هذا الكلاس.

التابع الهادم (Destructor): عكس التابع الباني يتميز بأنه:

- له نفس اسم الصف مسبوقاً بـ ~.
- لا يعيد أي قيمة.
- لا يأخذ أي وسيط.
- يتم استدعاؤه ضمنياً وبشكل تلقائي عند الانتهاء من استخدام الغرض.
- من النادر استخدامه لأنه في C# يتم تحرير الذاكرة تلقائياً عن طريق ما يسمى بـ جامع النفايات (Garbage Collector).

مفهوم التحميل الزائد (OverLoading):

المقصود به إمكانية تعريف أكثر من طريقة أو دالة بنفس الاسم ضمن المجال نفسه ويكون الاختلاف **بعدد أو نوع** البارمترات (الوسائط) الممررة لهذه الدالة.

تطبيقاً لهذا المفهوم يمكن تعريف أكثر من تابع باني في نفس الكلاس.

الأعضاء الساكنة Static:

يتم الوصول إليها عن طريق اسم الكلاس مباشرة وليس عن طريق أي غرض من الأغراض المعرّفة من هذا الكلاس، لأن هذه الأعضاء تكون مشتركة بين جميع أغراض الكلاس.

مثال :

```
namespace  classes_example
{
    class  person
    {
        public  string  fname;
        public  string  lname;
        private  int      age;
        public  string  address;

        public   int      Age
        {
          get  { return  age  ; }
```

```
            set {
              if  (value > 0 && value < 120)
               { age = value; }
                else age = 0;
                }
          }

          //constructors
          // باني بدون بارمترات
          public  person()
          {
              fname  = "XX";
              lname  = "YY";
              age  =  20;
              address  =  "Hama";
              Console.WriteLine ( "Constructor Person" ) ;
          }
          // باني ببارامترين
          public  person ( string  fname ,  string  lname )
          {
              this.fname  =  fname;
              this.lname  =  lname;
              this.age  =  20;
              address  =  "Hama";
          }
          // باني بثلاثة بارمترات
          public person( string  fname, string lname, int Age )
          {
              this.fname  =  fname;
              this.lname  =  lname;
              this.Age  =  Age;
              address  = "Hama";
          }

        public  void  printinfo( )
          {
            Console.WriteLine(" Hi, I am Person >>>");
            Console.WriteLine(" Name is {0} {1},\n Age is
              {2},\n Address is {3} ", fname,lname, Age,
                                       address);
            Console.WriteLine ( "-----------------------" );
          }
    }

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

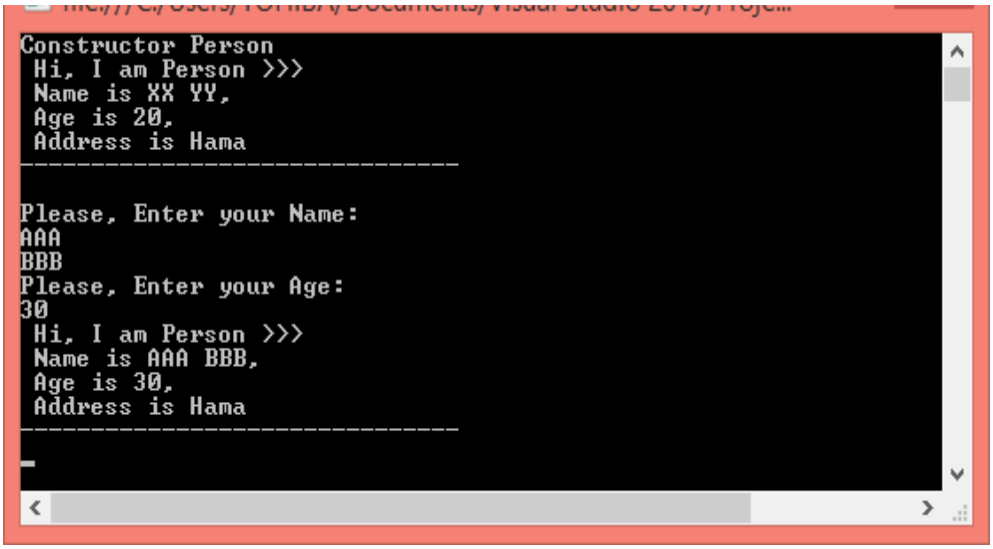
```
        {
            person p1 =  new person( );
            p1.printinfo( );

            Console.WriteLine( );
            Console.WriteLine( "Please, Enter your Name:" );
            string  p_first_name  = Console.ReadLine( );
            string  p_last_name  =  Console.ReadLine( );
            Console.WriteLine( "Please, Enter your Age:" );
            int  p_age  =  int.Parse (Console.ReadLine( ) );

            person p2  = new person (p_first_name,
                        p_last_name, p_age);
            p2.printinfo( );
            Console.ReadKey();

        }
    }
}
```

نتيجة التنفيذ:



مثال:

لدينا الكلاس CheckingAccount يقوم بتدقيق حساب مصرفي:

١- حقول هذا الكلاس: هي رقم الحساب account_num، اسم صاحب الحساب account_name، الرصيد الحالي balance .

٢- طرق الكلاس هي: طريقة لمعرفة الرصيد الحالي getBalance().

طريقة لسحب مبلغ get_money(amount).

طريقة لإيداع مبلغ deposit(amount).

```
class CheckingAccount
    {
      public string account_num;
      public string account_name;
      public double balance;

      //باني بدون بارامترات
      public CheckingAccount()
       {
         balance = 5000;
       }

      //باني بثلاثة بارامترات
      public CheckingAccount(string account_num, string
            account_name, double balance)
       {
         this.account_num = account_num;
         this.account_name = account_name;
         this.balance = balance;
       }

       //طريقة لمعرفة الرصيد الحالي
       public double getBalance()
       {
           return balance;
       }

       //طريقة لإيداع مبلغ
       public void deposit(double amount)
       {
           balance = balance + amount;
       }

       //طريقة لسحب مبلغ
```

```
   public void get_money (double amount)
    {
     //تتم عملية السحب بعد التأكد أنها أقل من المبلغ الموجود
       if (balance > amount)
            balance = balance - amount;
    }

// طريقة لطباعة معلومات الرصيد
public void display()
 {
   Console.WriteLine("**********************");
   console.WriteLine("Your name is {0},\n Your account number
     is {1},\n Your balance is {2}." , account_name,
                 account_num , balance);

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

 }
}

class Checking {
  public static void Main(string[] a)
   {
    CheckingAccount c1 = new CheckingAccount("123","Sara",100000);
      c1.display();  // طباعة معلومات الرصيد
      c1.deposit(20000);//إيداع مبلغ
      c1.get_money(15000); //سحب مبلغ
      c1.getBalance();//معرفة الرصيد الحالي
      c1.display();// طباعة معلومات الرصيد

    CheckingAccount c2 = new CheckingAccount();
      Console.Write(" Please, Enter your account number:");
      c2.account_num = Console.ReadLine();
      Console.Write(" Please, Enter your account name:");
      c2.account_name  = Console.ReadLine();
      c2.display();
      Console.Write(" How much money you want to deposit?");
      double amount1 = double.Parse(Console .ReadLine());
      c2.deposit(amount1);//مبلغ إيداع
      Console.WriteLine("Your balance is: {0} ",
                 c2.getBalance());

      Console.Write(" How much money you want to get?");
      double amount2 = double.Parse(Console.ReadLine());
      c2.get_money(amount2);

      c2.display();
```

```
        Console.ReadKey();
    }

}
```

نتيجة التنفيذ ستكون بالشكل التالي: