المفهرسات والمجموعات والوكلاء والأحداث

Indexers, Collections, Delegates and Events

الكلمات المفتاحية

المفهرس، قائمة المعطيات، قاموس المعطيات، الوكيل، الحدث.

ملخص الفصل

خُصّص هذا الفصل لتوضيح المفهرسات والمجموعات والوكلاء والأحداث وطريقة عمل كلّ منها والسياق الذي يفرض استخدامها.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- المفهرس وكيفية استخدامه والغاية منه
- بعض المجموعات المستخدمة في تخزين المعطيات كالقوائم والقواميس
- الوكلاء ودورهم في التعامل مع الأحداث

مقدمة

في لغة C#، توجد العديد من البنى ذات الاستخدامات الخاصة. حيث تسمح المفهرسات بالوصول إلى عناصر صفّ ما بأسلوب مصفوفي، وتُشكِّل المجموعات في بعض الأحيان بديلاً جيداً عن المصفوفات التي تحتاج لتحديد عدد عناصرها قبل استخدامها. ويسمح الوكلاء بتمرير طريقة كوسيط دخل لطريقة الأخرى. وتقوم الأحداث بإرسال رسالة تنبيه عند وقوع حدث ما إلى الغرض الذي ينتظر وقوعه.

1. المفهرسات

عند استدعاء عناصر مصفوفة ما، يتمّ استخدام اسم المصفوفة ودليل تقابل كلّ قيمة فيه أحد العناصر، وتكون قيم الدليل من النمط int ومتزايدة بخطوة مساوية للواحد ابتداءً من القيمة (0). ويكون للقيمة المسترجعة نفس نمط المعطيات المستخدَم في تعريف المصفوفة. وقد يُغلِّف الصفّ قائمة من المعطيات التي نحتاج الوصول إليها من خارج الصفّ بأسلوب مصفوفي. ولهذه الغاية يمكن استخدام عناصر مشابهة للمصفوفات تُسمّى بالمفهرسات.

وعندها يصبح بالإمكان استخدام اسم الغرض للوصول إلى عناصر القائمة بنفس طريقة استخدام اسم المصفوفة.

يشبه المفهرس الخاصّية property من ناحية أنّه يحتوي على get و set.

يتمّ التصريح عن المفهرسات كما يلي:

```
element-type this[Parameter_List] {
          get  {
              // get block
                }
           set  {
             // set block
                }
       }
```

- يشير element-type إلى نمط معطيات القيم المرتجعة.
- تشير Parameter List إلى قائمة وسائط الدخل ويجب أن تحتوي وسيطاً واحداً على الأقل.
- يتمّ استخدام الكلمة المفتاحية this متبوعة بالقوسين [ ] اللذَين يضمّان قائمة وسائط الدخل Parameter_List.

أمثلة

مثال1:

في الرمّاز الآتي، نوضّح كيفية استخدام مفهرس كأحد أعضاء صفّ:

```
using System;
class NameIndex
{
    static public int size = 6;
    // internal data storage
    private string[] nameList = new string[size];
    // indexer
    public string this[int index]
    {
        get
        {
            if (index >= 0 && index <= size - 1)
                return nameList[index];
            else throw new IndexOutOfRangeException("Cannot store more than " +
size + " objects");
        } // end get
        set
        {
            if (index >= 0 && index <= size - 1) nameList[index] = value;
            else
                throw new IndexOutOfRangeException("Cannot store more than " +
size + " objects");
        } // end set
    }//end indexer
    static void Main()
    {
        NameIndex names = new NameIndex();
        names[0] = "Zyad"; names[1] = "Ramez"; names[2] = "Kefah";
        names[3] = "Aziz"; names[4] = "Rola"; names[5] = "Saber";
        for (int i = 0; i < NameIndex.size; i++)
            Console.WriteLine("  " + names[i]);
        Console.ReadKey();
    } // end Main
}// end NameIndex
```

نلاحظ أنّه تمّ استخدام اسم الغرض names من الصفّ NameIndex مباشرة للوصول إلى عناصر المصفوفة nameList المعرّفة ضمن نفس الصفّ. أي أنّه عوضاً عن كتابة names.nameList[0] للوصول إلى العنصر الأوّل في المصفوفة، تمّت كتابة names[0]. وبعد تنفيذ الرمّاز نحصل على الخرج:

```
Zyad
Ramez
Kefah
Aziz
Rola
Saber
```

مثال2:

يمكن استخدام المفهرسات مع البنى بنفس طريقة استخدامها مع الصفوف، ونوضّح ذلك من خلال الرمّاز الآتي الذي يعطي نفس خرج المثال السابق بعد تنفيذه:

```
using System;
class StructIndex
{
    struct NameStruct
    {
        public string[] data;
        //indexer
        public string this[int index]
        {
            get
            {
                if (index >= 0 && index <= data.Length - 1)
                    return data[index];
                else throw new IndexOutOfRangeException("Cannot store more than "
+ data.Length + " objects");
            } // end get
            set
            {
                if (index >= 0 && index <= data.Length - 1)
                    data[index] = value;
                else throw new IndexOutOfRangeException("Cannot store more than "
+ data.Length + " objects");
            } // end set
```

```
        } //end indexer
    } // end NameStruct
    public static void Main()
    {
        NameStruct names = new NameStruct();
        names.data = new string[6];
        names[0] = "Zyad"; names[1] = "Ramez"; names[2] = "Kefah";
        names[3] = "Aziz"; names[4] = "Rola"; names[5] = "Saber";
        for (int i = 0; i < names.data.Length; i++)
            System.Console.WriteLine("  " + names[i]);
        Console.ReadKey();
    } // end Main
} // end StructIndex
```

مثال3:

ليس بالضرورة أن يكون دليل المفهرس من النمط int، فيمكن أن يكون من النمط string، ونوضّح ذلك من خلال الرمّاز الآتي:

```
using System;
//Defining exception to handle "not found" situations
class ItemNotFoundException : ApplicationException
{

    public ItemNotFoundException() : base("Item does not found") { }
}// end Exception
class DayCollection
{
    readonly string[] days = { "Sunday", "Monday", "Tueseday", "Wednesday",
"Thursday", "Friday", "Saturday" };
    // The get accessor looks for  a given day and
    // returns its number if it is found;
    // if the day is not found it throws  ItemNotFoundExceptio exception
    // The indexer
    public int this[string day]
    {
        get
        {
            for (int j = 0; j < days.Length; j++)
```

```
                if (days[j] == day) return j;
            throw new ItemNotFoundException();
        }// end get
    }// end the indexer

    static void Main()
    {
        DayCollection week = new DayCollection();
        Console.WriteLine(week["Friday"]);
        Console.WriteLine(week["June"]);
        Console.ReadKey();
    } // end Main()
}// end DayCollection
```

وبعد التنفيذ، نحصل على الخرج الآتي الذي يشير إلى أنّ كلمة "Friday" موجودة ضمن مصفوفة الأيام days وقيمة دليلها (5)، أمّا الكلمة "June" فغير موجودة ولذلك تمّ قذف استثناء:

```
5

Unhandled Exception: ItemNotFoundException: Item does not found
```

مثال4:

يمكن القيام بتحميل زائد للمفهرس على نحو مشابهٍ للتحميل الزائد للطرائق، ففي الرمّاز الآتي، تمّ تعريف مفهرس دليله من النمط int ومفهرس دليله من النمط string. ويعيد المفهرس الأوّل قيمة من النمط string، أمّا المفهرس الثاني فيعيد قيمة من النمط int:

```
using System;
class NameIndex
{
    private string[] namelist = new string[size];
    static public int size = 8;
    public NameIndex()
    {
        for (int i = 0; i < size; i++) namelist[i] = "N. A.";
    }// end constructor NameIndex
    //first indexer
    public string this[int index]
```

```
    {
        get
        {
            string tmp;
            if (index >= 0 && index <= size - 1)
                tmp = namelist[index];
            else tmp = "";
            return tmp;
        } // end get
        set
        {
            if (index >= 0 && index <= size - 1) namelist[index] = value;
        } // end set
    }// end the first indexer
    // second indexer
    public int this[string name]
    {
        get
        {
            int index = 0;
            while (index < size)
            {
                if (namelist[index] == name)
                    return index;
                index++;
            }
            return index;
        }// end get
    }// end second indexer
    static void Main(string[] args)
    {
        NameIndex names = new NameIndex();
        names[0] = "Zyad"; names[1] = "Ramez"; names[2] = "Kefah";
        names[3] = "Aziz"; names[4] = "Rola"; names[5] = "Saber";
        //using the first indexer with int parameter
        for (int i = 0; i < NameIndex.size; i++)
        {
            Console.WriteLine("\t" + names[i]);
        }
```

```
        //using the second indexer with the string parameter
        Console.WriteLine("the order of Aziz is {0} ", names["Aziz"]);
        Console.ReadKey();
    } // end Main
}// end NameIndex
```

وبعد تنفيذ الرمّاز السابق، نحصل على الخرج الآتي الذي يوضّح استخدام المفهرس الأوّل لأنّ الدليل المستخدم معه من النمط `int` ولم يتمّ استخدام المفهرس الثاني:

```
Zyad
Ramez
Kefah
Aziz
Rola
Saber
N. A.
N. A.
the order of Aziz is 3
```

2. المجموعات

في كثير من التطبيقات، نحتاج إلى التعامل مع مجموعة من الأغراض المتعلّقة ببعضها البعض. ويمكن تحقيق هذه الغاية إمّا باستخدام المصفوفات أو باستخدام المجموعات Collections. فتسمح المصفوفات بالتعامل مع عدد ثابت من الأغراض يتمّ تحديده قبل التخزين فيها، أمّا المجموعات فهي أكثر مرونة لأنّه يمكن أن يتغيّر عدد الأغراض المخزّنة فيها زيادة أو نقصاناً وفقاً للحاجة. وفي بعض المجموعات، يمكن مقابلة كلّ غرض بمفتاح يسمح باسترجاع الغرض عند الحاجة. وتفيد المجموعات في عدّة مجالات من بينها إدارة الذاكرة ديناميكياً والوصول إلى عناصر قائمة باستخدام دليل. وعادة ما تستخدم صفوف المجموعات أثناء تخزين المعطيات واسترجاعها، ويغلب استخدامها عند التعامل مع المكدِّسات Stacks والأرتال Queues والقوائم Lists وتوابع التقطيع Hash Tables.

مثال1

نوضّح من خلال الرمّاز الآتي استخدام المجموعات مع قائمة List حاوية على سلاسل محرفية، حيث تمّ استخدام الصفّ العام List<T> المنتمي لفضاء الأسماء System.Collections.Generic:

```
using System;
using System.Collections.Generic;
namespace Collections
{
    class CollectionsTester
    {
        static void Main()
        {
            // Create a list of strings.
            List<string> numbers = new List<string>();
            numbers.Add("One");
            numbers.Add("Two");
            numbers.Add("Three");
            numbers.Add("Four");
            // Iterate through the list.
            foreach (var num in numbers)
                Console.Write(num + " ");
            Console.WriteLine();
        }
    }
}
```

وبعد تنفيذه، نحصل على الخرج التالي:

```
One Two Three Four
```

مثال2:

نوضّح في الرمّاز التالي كيف يمكن تهيئة القائمة مباشرة أثناء التصريح عنها:

```
using System;
using System.Collections.Generic;
namespace Collections
{
    class CollectionsTester
    {
        static void Main()
        {
            // Create a list of strings
            // collection initializer
            List<string> numbers = new List<string> { "One", "Two", "Three" };
            // adding new string (object) to the list
            numbers.Add("Four");
            //removing specified string (object) from the list
            numbers.Remove("Two");
            // Iterate through the list using for statement
            for (var index = 0; index < numbers.Count; index++)
                Console.Write(numbers[index] + " ");
            Console.WriteLine();
        }//end Main
    }
}
```

نلاحظ أيضاً سهولة إضافة عنصر إلى القائمة من خلال استدعاء الطريقة Add وسهولة حذف عنصر آخر باستخدام الطريقة Remove، حيث أنّه لا حاجة إلى أي حلقة تكرارية لإضافة الكلمة "Four" ولا حتى لحذف الكلمة "Two". وبعد التنفيذ، نحصل على الخرج:

```
One Three Four
```

1.2. قاموس المعطيات Dictionary<TKey, TValue>

ينتمي الصفّ Dictionary<TKey, TValue> لفضاء الأسماء System.Collections.Generic وهو مجموعة يُخزّن فيها ثنائيات من المعطيات، تُسمّى الأولى TKey وتُسمّى الثانية TValue وهما متلازمتان. ويمكن أن تكونا من أي نمط معطيات.

مثال:

في الرمّاز الآتي، تمّ تعريف قاموس المعطيات people الذي يُستخدم بتخزين ثنائيات اسم وعمر مجموعة من الأشخاص، حيث يمثّل الاسم TKey ويمثّل العمر TValue.

```
using System;
using System.Collections.Generic;
namespace Collections
{
    class Dictionaries
    {
        public static void Main()
        {
            Dictionary<string, int> people = new Dictionary<string, int> {
                  {"Basem", 30 }, {"Mary", 35}, {"Younes", 40} };
            // Reading data
            Console.WriteLine(people["Basem"]); // 30
                                                //
Console.WriteLine(people["George"]); // throws KeyNotFoundException
            int age;
            if (people.TryGetValue("Mary", out age))
                Console.WriteLine(age); // 35

            // Adding and changing data
            try
            {
                people["John"] = 40; // Overwriting values this way is ok
                people.Add("John", 40); // Throws ArgumentException since
                                        //"John" already exists
            }
            catch (ArgumentException e)
            {
                Console.WriteLine(e.Message);
```

```
            }
            //Searching for a key
            Console.WriteLine("\nSearching for a key");
            if (people.ContainsKey("Michel"))
                Console.WriteLine("Key \"Michel\" is not found.");
            else
                Console.WriteLine("Key \"Michel\" is not found.");

            // Iterating through contents
            Console.WriteLine("\nThe dictionary content");
            foreach (KeyValuePair<string, int> person in people)
                Console.WriteLine("Name={0}, Age={1}", person.Key,
                                                        person.Value);

            Console.WriteLine("\n\nThe dictionary keys");

            foreach (string name in people.Keys)
                Console.WriteLine("Name={0}", name);

            Console.WriteLine("\n\nThe dictionary values");

            foreach (int i in people.Values)
                Console.WriteLine("Age={0}", i);
        }//end Main
    }// end class Dictionaries
}// end namespace Collections
```

وبعد التنفيذ، نحصل على الخرج الآتي:

```
30
35
An item with the same key has already been added. Key: John

Searching for a key
Key "Michel" is not found.

The dictionary content
Name=Basem, Age=30
Name=Mary, Age=35
Name=Younes, Age=40
Name=John, Age=40
```

```
The dictionary keys
Name=Basem
Name=Mary
Name=Younes
Name=John


The dictionary values
Age=30
Age=35
Age=40
Age=40
```

نورد فيما يلي بعض الملاحظات المتعلّقة بالخرج:

- تمّ الحصول على عمر "Basem" من خلال استدعاء القيمة له people["Basem"].
- تمّ استخدام الطريقة TryGetValue التي تعيد true إذا كان المفتاح موجوداً ضمن القاموس وتعيد القيمة الموافقة له، أمّا إذا كان غير موجود فتعيد false.
- يمكن إضافة ثنائية جديدة إلى القاموس من خلال الطريقة Add. وفي حال كانت الثنائية موجودة، يتمّ قذف استثناء من النمط ArgumentException.
- تمّ استخدام الطريقة ContainsKey التي تعيد إذا كان المفتاح موجوداً ضمن القاموس وتعيد false إذا كان غير موجود.
- تمّ المرور على جميع الثنائيات من خلال استخدام التعليمة foreach مع عداد person من النمط KeyValuePair<string,int> حيث يسمح person بالوصول إلى المفتاح والقيمة.
- تمّ المرور على جميع المفاتيح باستخدام التعليمة foreach.
- تمّ المرور على جميع القيم باستخدام التعليمة foreach.

3. الوكلاء

عادة ما يتمّ تمرير الحقول كوسطاء دخل للطرائق. وفي لغة C#، توجد إمكانية تمرير الطرائق كوسطاء دخل لطرائق أخرى من خلال استخدام الوكلاء Delegates. فيُستخدم الوكيل كمتحوّل لتخزين مرجع لطريقة، ويُمكن أن تتغيّر قيمته أثناء التنفيذ، ويرث جميع الوكلاء ضمنياً من الصفّ System.Delegate.

وعند التصريح عن وكيل يجب أن يكون له نفس توقيع الطريقة التي يشير إليها. فالتوقيع:

```
public delegate int MyDelegate (string s);
```

يمكن استخدامه للدلالة على أي طريقة تأخذ وسيط دخل واحد من النمط string وتعيد قيمة من النمط int. وغالباً ما يُستخدم الوكلاء مع الأحداث.

مثال:

نوضّح في الرمّاز التالي، كيفية استخدام الوكلاء:

```
using System;
delegate int ChangingNumber(int n);// delegate declaration
namespace Delegates
{
    class DelecgateNumber
    {
        static int i = 10;
        public static int Adding(int p)
        {
            i += p;
            return i;
        }
        public static int Multiplying(int q)
        {
            i *= q;
            return i;
        }
        public static int GetNumValue()
        {
            return i;
        }
```

```
    }// end class DelecgateNumber
    class DelegateTesting
    {
        static void Main()
        {
            //creating delegate instances
            ChangingNumber n1 = new ChangingNumber(DelecgateNumber.Adding);
            ChangingNumber n2 = new ChangingNumber(DelecgateNumber.Multiplying);
            //calling the methods using the delegate objects
            n1(25);
            Console.WriteLine("The new value of Num: {0}",
                                DelecgateNumber.GetNumValue());
            n2(5);
            Console.WriteLine("The new value of Num: {0}",
                                  DelecgateNumber.GetNumValue());
            Console.ReadKey();
        }//end Main
    }//end class DelegateTesting
}// end namespace Delegates
```

وبعد التنفيذ، يظهر الخرج الآتي:

```
The new value of Num: 35
The new value of Num: 175
```

4. الأحداث

تعريف الحدث

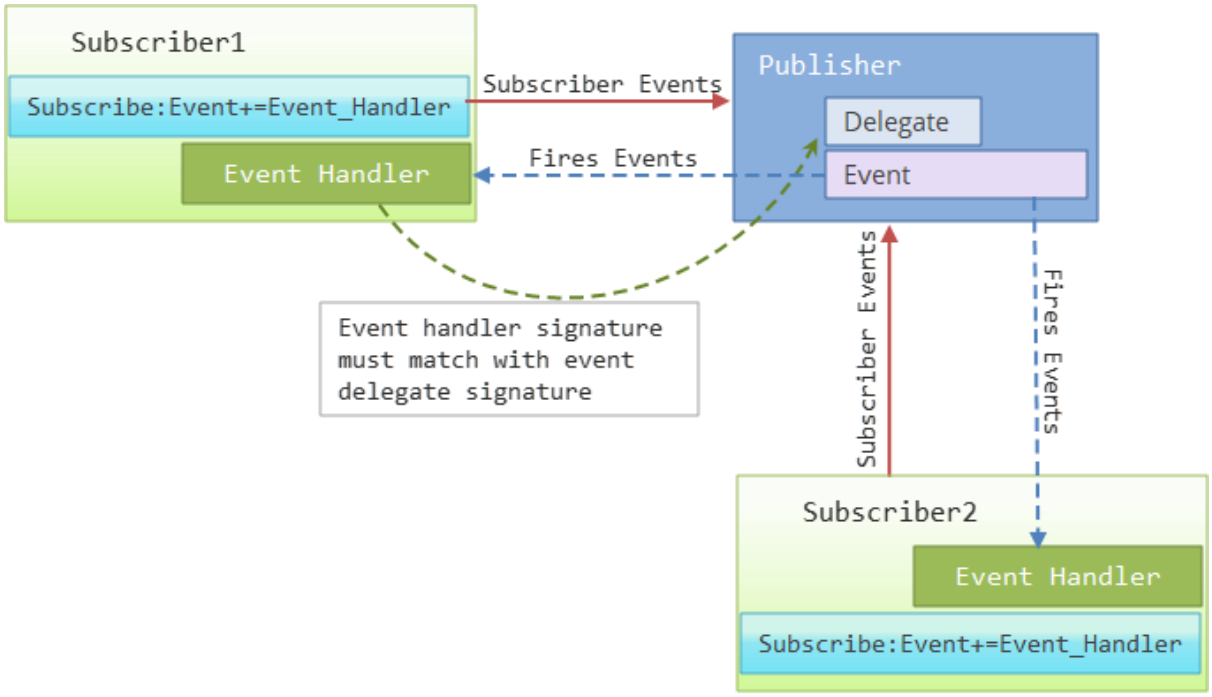
تسمح الأحداث Events بإرسال تنبيه بحدوث أمر ما من غرض(أو صفّ) إلى غرض(أو صفّ) آخر، ويسمّى الغرض المرسل للتنبيه بالناشر Publisher والغرض المستقبِل للتنبيه بالمشترِك Subscriber. ومن الممكن توافر أكثر من مشترك في نفس الحدث. ويجب على كلّ مشترك في حدث أن يوفّر معالج حدث Event Handler كما يجب على الناشر أن يحوي وكيل Delegate توكل إليه مهمّة استدعاء معالجات الأحداث في الصفوف المشترِكة في الحدث. وعند تصميم الواجهات الرسومية أو تطبيقات الويب، تشترك في الأحداث عناصر التحكم مثل الأزرار ومربعات القوائم. وتسمح بيئة التطوير المتكاملة C # IDE (Integrated Development Environment) بتصفّح الأحداث التي ينشرها عنصر تحكّم بسهولة، كما تسمح بإضافة طريقة معالجة حدث فارغة تلقائياً وتوفّر الرمّاز الخاصّ بالاشتراك في الحدث.

متى يظهر الحدث؟

يحدّد الناشِر متى يجب للحدث أن يظهر، ويقوم المشتركون بتحديد الأفعال الواجب القيام بها كردّ فعل على الحدث. ويمكن للناشر أن يعالج أحداثاً صادرة عن عدّة ناشرين. والأحداث التي ليس لها مشتركون لا تظهر أبداً. وعادة ما تظهر الأحداث بعد أفعال يقوم بها المستخدم كالضغط على زرّ ما أو تحديد أحد الخيارات قائمة منسدلة ضمن واجهة مستخدم رسومية.

علاقة الحدث بالوكيل

يعتمد الحدث على وكيل ويشكل غلافاً له، ويوفّر الوكيل توقيع الطريقة التي تقوم بمعالجة الحدث ضمن الصفّ الناشر. ويوضّح الشكل الآتي [1] العلاقة بين الناشر والمشتركين ودور الوكيل:



Event Publisher & Subscriber

التصريح عن الحدث:

يتمّ التصريح عن حدث ما باتباع الخطوتين الآتيتين:

1. التصريح عن وكيل
2. التصريح عن متغيّر من نمط الوكيل مسبوقاً بالكلمة event

```
// Declaring an Event
public delegate void Notify(); // delegate
public class ProcessBusinessLogic
{
public event Notify ProcessCompleted; // event
}
```

نسمّي الصفّ ProcessBusinessLogic ناشر ويضمّ الحدث ProcessCompleted المغلّف بالوكيل Notify الذي يحدّد توقيع الطريقة التي سنعالج الحدث ProcessCompleted عند وقوعه (ظهوره).

مثال عن الأحداث

يشير هذا المثال إلى أنّ الطريقة المعالجة للحدث في أحد المشتركين لن تعيد قيمة ولن يكون لها وسائط دخل.

```
using System;
public delegate void Notify();  // delegate
public class ProcessBusinessLogic
{
    public event Notify ProcessCompleted; // event

    public void StartProcess()
    {
        Console.WriteLine("Process Started!");
        // some code here..
        OnProcessCompleted();
    }

    protected virtual void OnProcessCompleted() //protected virtual method
    {
        //if ProcessCompleted is not null then call delegate
        ProcessCompleted?.Invoke();
    }
}
```

تقوم الطريقة StartProcess باستدعاء الطريقة onProcessCompleted التي توقع حدثاً وهي طريقة محمية افتراضية، ويجب استدعاؤها من قبل الصفّ الوراث للتحقّق من وقوع حدث. وتقوم هذه الطريقة بتفعيل الوكيل من خلال التعليمة ProcessCompleted?.Invoke(); وهذه بدورها تستدعي الطرائق المعالجة للحدث.

يقوم الصفّ Subscriber المشترك بالتسجيل على الحدث ProcessCompleted باستخدام المعامل (=+)، ويقوم بمعالجته بواسطة الطريقة bl_ProcessCompleted الموافقة للوكيل Notify، كما هو موضّح فيما يأتي:

```
class Subscriber
{
    public static void Main()
    {
        ProcessBusinessLogic bl = new ProcessBusinessLogic();
        bl.ProcessCompleted += bl_ProcessCompleted; // register with an event
        bl.StartProcess();
    }// end Main

    // event handler
    public static void bl_ProcessCompleted()
    {
        Console.WriteLine("Process Completed!");
    }// end event handler
}// end Class Subscriber
```

وبعد التنفيذ، ينتج الخرج الآتي:

```
Process Started!
Process Completed!
```

الأنشطة المرافقة

التمرين الأوّل: دليل الهاتف PhoneBook

نرغب في إنشاء تطبيق برمجي بلغة C# لإدارة دليل هاتف مكوّن من 26 فصلاً، حيث يوافق كلّ فصل حرف أبجدي من الأبجدية الإنكليزية، وكلّ فصل يتسع إلى 50 رقماً هاتفياً. ويسمح التطبيق بعمليات الإضافة والحذف والتعديل على أسماء الأشخاص وأرقام هواتفهم الموافقة. كما يسمح باسترجاع رقم هاتف شخص عند إعطاء اسمه كاملاً (الاسم والكنية)، وفي حال عدم تذكّر الاسم الكامل من قبل المستخدم، يكفي إعطاء سلسلة محرفية مؤلفة من ثلاثة محارف مرتبة بشكل صحيح من الاسم حتى يقوم التطبيق بإظهار جميع الأسماء التي تحتوي السلسلة المحرفية المعطاة وأرقام هواتفهم، يسمح التطبيق أيضاً باسترجاع اسم الشخص عند إدخال رقم هاتفه.

المطلوب:

قم بكتابة التطبيق المطلوب واختبره على أن يتمّ استخدام طريقة Method لكلّ وظيفة من وظائفه، وأن يتمّ تخزين معطيات دليل الهاتف ضمن بنية معطيات Data Structure مناسبة.

التمرين الثاني: مبيعات شركة CompanySales

يعمل لدى شركة بيع أدوية خمسة مندوبي مبيعات، ويقوم كلّ منهم بتوزيع ستة منتجات، وترغب الشركة بإنشاء تطبيق برمجي بلغة C# يتيح لها معرفة مبيعات كلّ مندوب خلال فترات زمنية محدّدة بالأيام.

المطلوب:

قم بكتابة التطبيق المطلوب واختبره على أن يتمّ إدخال مبلغ مبيعات كلّ مندوب من كلّ سلعة في كل يوم ضمن بنية معطياً Data Structure تسمح بتخزين بيانات شهر كامل، وأن يتمّ استخدام طريقة Method لكلّ وظيفة مما يأتي:

- إدخال مبيعات كلّ مندوب إلى بنية المعطيات
- إظهار مبيعات كلّ موظّف خلال فترة زمنية تفصل بين تاريخين محدّدين من نفس الشهر
- حساب المتوسط الحسابي الشهري والمتوسط الحسابي اليومي للمبيعات
- ترتيب المندوبين تنازلياً وفقاً لمبيعاتهم خلال الشهر

المراجع

1. C# – Events, https://www.tutorialsteacher.com/csharp/csharp-event, updated on April 28, 2020.
2. https://docs.microsoft.com/en-us/dotnet/csharp/.
3. C# – Events, https://www.tutorialsteacher.com/csharp/csharp-event, updated on April 28, 2020.