الجامعة الافتراضية السورية

SYRIAN VIRTUAL UNIVERSITY

برمجة التطبيقات الشبكية

د. صلاح الدوه جي

Books

برمجة تطبيقات شبكية

الدكتور صلاح الدوه جي

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

Network Application Programming

Salah DOWAJI

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

الفهرس

الفصل الثالث : خدمات الويب Web Services

الفصل الأول: تعددية النياسبMultithreading

1. مقدمة

إذا أمعنا النظر في طبيعة جسم الإنسان، فسوف نلاحظ أنه ينجز الكثير من العمليات معاً وبنفس الوقت، فمثلاً جميع العمليات الأساسية مثل التنفس والهضم والدورة الدموية هي عمليات لابد لها من أن تتم بنفس الوقت وعلى التوازي. لأنه من غير الممكن أن تنتظر إحدى هذه العمليات انتهاء عملية أخرى.

كذلك، تعمل الحواس الخمس للإنسان في وقت واحد، حيث يمكن له أن يسمع ويشاهد، مثلاً، في نفس الوقت.

وعلى غرار ذلك، فإننا إذا أمعنا النظر في الحاسب وكيفية قيامه بعملياته الأساسية، سنلاحظ أنه قادر على تنفيذ برنامج ما واستقبال بريد الكتروني من شبكة وتنفيذ أمر دخل/خرج (طباعة مثلاً) بنفس الوقت وبشكل متوازي.

يستخدم الحاسب بشكل عام عند إنجازه لعدد من العمليات بشكل متوازي ميزة تعددية النياسـ ب **Multithreading**. فإذا افترضنا مثلاً، وجود ثلاثة عمليات مختلفة تتطلب التنفيذ من قبل الحاسب بنفس الوقت، يشكّل الحاسب ثلاثة نياسب **Thread**، بحيث يكون كل نيسب مسؤولاً عن تنفيذ عملية معينة، ويتم تنفيذ النياسب الثلاثة في نفس الوقت.

إذ أنه من غير المنطقي أن تقتصر معالجة الحاسب على عملية واحدة فقط في وقت معين وبقاء جميع العمليات الأخرى معلقة حتى انتهاء هذه العملية.

ولإيضاح أهمية تعددية النياسب لنناقش المثال التالي: عندما يقوم المستخدم بتحميل برنامج أو ملف معين من الإنترنت (ليكن مقطع صوتي أو مقطع فيديو مثلاً). يطلب المستخدم تشغيل المقطع قبل انتهاء التحميل كاملاً. يحل الحاسب هذه المشكلة بتوظيف ميزة تعددية النياسب، حيث يقوم نيسب بتحميل المقطع بينما يقوم نيسب آخر بتشغيل المقطع.

وهنا سيطرح فوراً السؤال التالي:

> ماذا سيحدث إذا وصل نيسب التشغيل إلى جزء لم يحمله نيسب التحميل بعد؟ يكمن الحل هنا كما سنرى بمزامنة النيسبين **Threads Synchronization**.

ومن المثير للدهشة أنه رغم الفوائد الواضحة لتعددية النياسب، فقد كان توظيف هذه الميزة الدفاعوبناؤها صعباً في الماضي لأن لغات البرمجة القديمة لم تُقدم الأدوات اللازمة للتعامل مع النياسب، وكانت هذه الميزة محصورة على متخصصي برمجة النظم فقط. أدركت وزارة

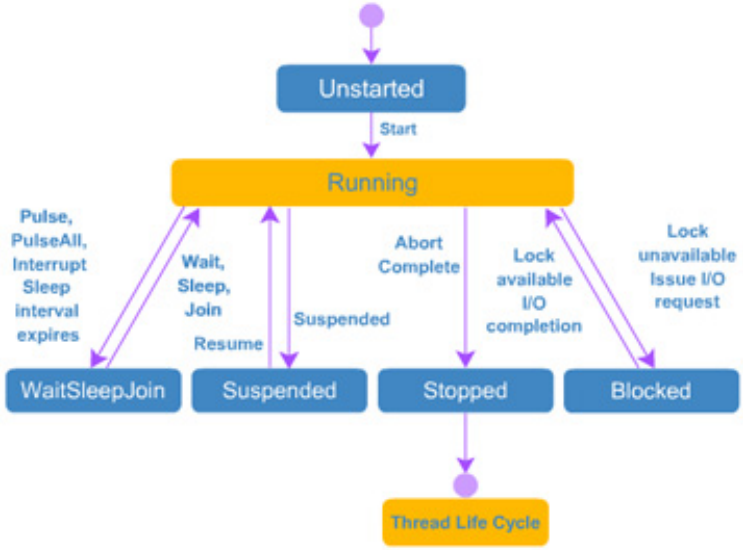
الأمريكية أهمية تعددية النياسب، فطورت لغة البرمجة **(Ada)** التي استخدمتها لخدمة مصالحها، إلا أنها كانت غير شائعة الاستخدام في المجال التعليمي أو المجال التجاري.

بعد ذلك ومنذ فترة قريبة قامت بيئة العمل **(.NET)** بتزويد جميع لغات البرمجة الموجودة ضمنها بالأدوات التي تُمكّن المبرمجين من بناء النياسب والتعامل معها، ووضعت هذه الأدوات في المكتبة **(System.Thread)**.

سنتحدث في هذا الفصل عن النياسب وكيف يتم بناؤها والتعامل معها برمجياً وكيف توظف ميزة تعددية النياسب.

2. حالات النيسب: دورة حياة النيسب Thread States: Life Cycle of a Thread

بعد أن تحدثنا بشكل عام عن مفهوم النيسب، سوف نتحدث الآن بالتفصيل عن النياسب ودورة حياتها في.**NET**. وسوف نتعرف على حالات النياسب **Threads States** وكيفية الانتقال فيما بينها. ومن الجدير بالذكر هنا أن المكتبة **(System.Threading)** تحوي الصفين **(Thread)** و**(Monitor)** اللذان يحويان الطرق اللازمة للتحكم في النياسب والتعامل معها.



تبدأ دورة حياة النيسب من حالة عدم الانطلاق (**Unstarted**) وذلك عندما يتم بناؤه بعد تمرير المفوض (**ThreadStart**) لباني النيسب **Constructor**.
يحدد هذا المفوض **Delegate** المهام التي يجب أن يقوم بها النيسب خلال فترة حياته. ويكون عبارة عن طريقة تُعيد لاشيء **void** وليس لها معاملات **parameters**.

يبقى النيسب في الحالة الإبتدائية (**Unstarted**) حتى يقوم البرنامج باستدعاء الطريقة (**Start**) التي تقوم بتغيير حالة النيسب إلى حالة التشغيل (**Running**). ثم بعد ذلك، يعاد التحكم مباشرة إلى التعليمة التالية في البرنامج بعد التعليمة التي استدعت الطريقة **Start**.

يمكن في هذه المرحلة، أن تُنفذ النياسب الموجودة في حالة التشغيل المتزامن في نظام المعالجات المتعددة أو تتشارك هذه النياسب التنفيذ في نظام المعالج الوحيد. مع ملاحظة أنه ليس بالضرورة أن ينفذ النيسب باستمرار إلا عندما يسند إلى المعالج.

يدخل النيسب في حالة الإيقاف أو الإحباط (**Stopped or Aborted**) عندما ينهي النيسب المهمة التي كانت موكلة إليه بشكل طبيعي، أو عندما يجبر البرنامج النيسب على ذلك باستدعاء طريقة الإحباط أو الإنهاء المفاجئ (**Abort**)، والتي تُدخل النيسب في حالة طلب الإحباط بإزالة النيسب(**AbortRequested**) حيث لايتم إنهاء النيسب فعلياً إلا بعد استقبال استثناء إحباط النيسب(**ThreadAbortException**). ثم يقوم جامع النفايات **Garbage Collector** من الذاكرة مالم تُسند إليه مهمة أخرى.

ومن الجدير بالذكر أنه عندما يكون النيسب في حالة أخرى غير حالة التشغيل، سوف يبقى في حالته الحالية ويدخل في حالة طلب الإحباط (**AbortRequested**) في نفس الوقت ولكنه لن يستجيب للإستثناء (**ThreadAbortException**) إلا بعد أن يخرج من حالته الحالية.

يعتبر النيسب مجمداً (**Blocked**) عندما يكون عاجزاً عن استخدام أي معالج حتى إذا وجد. فعلى سبيل المثال: يصبح النيسب مجمداً عندما تكون مهمته طلب دخل/خرج، حيث يقوم نظام التشغيل بحجبه ومنعه من متابعة التنفيذ حتى يتمكن نظام التشغيل من إنجاز طلب الدخل/الخرج الذي ينتظره النيسب. بعد ذلك يعود النيسب إلى حالة التشغيل ويمكن أن يعاود التنفيذ.

كذلك، فمن الأمثلة على تجميد النيسب في مزامنة النياسب: عندما يبدأ النيسب المزامنة يجب أن يمنح قفل معين (كما سنرى لاحقاً) باستدعاء الطريقة (**Enter**) الموجودة في الصف (**Monitor**). فإذا لم يكن القفل موجوداً، سيصبح النيسب مجمداً إلى أن يتاح القفل المطلوب له. يجب الملاحظة هنا أن حالة التجميد ليس حالة حقيقية فعلية في بيئة العمل (**NET.**) إلا أنه مفهـوم أو تعبير يطلق على النيسب عندما لا يكون في حالة التشغيل.

أما حالة انتظار/سبات/ضم (**WaitSleepJoin**) فيمكن أن يصل النيسب لها بثلاثة طرق:

1. **الطريقة الأولى**: عندما يواجه النيسب تعليمة لا يمكن تنفيذها حالياً (كأن تتطلب التعليمة حالة لم تتحقق بعد مثلاً) يتم استدعاء الطريقة (**Wait**) من الصف (**Monitor**)، وبالتالي يدخل النيسب في حالة الانتظار (**Wait**). بعدها، يعود النيسب إلى حالة التشغيل عندما يستدعي نيسب آخر طريقة الإنعاش (النبضة) (**Pulse**) أو إنعاش الكل (**PulseAll**) من الصف (**Monitor**). لاحظ أن طريقة الإنعاش (**Pulse**) تقوم بتغيير حالة النيسب المنتظر التالي إلى حالة التشغيل بينما تقوم طريقة (**PulseAll**) بتغيير حالات جميع النياسب المنتظرة إلى حالة التشغيل.
2. **الطريقة الثانية**: يمكن للنيسب الموجود في حالة التشغيل أن يستدعي طريقة السبات (**Sleep**) ليصبح في حالة انتظار/سبات/ضم (**WaitSleepJoin**) لفترة زمنية محددة (تكون الوحدة الميلي ثانية) تُعرف كدخل في الطريقة (**Sleep**). ويعود النيسب من حالة السبات إلى حالة التشغيل عندما ينتهي وقت السبات. لاحظ أن النيسب الذي يكون في حالة السبات لا يمكن أن يستخدم المعالج حتى وإن كان متاحاً.
3. **الطريقة الثالثة**: من الممكن أن يعتمد تنفيذ نيسب على نيسب آخر. وفي هذه الحالة يدخل النيسب حالة انتظار/سبات/ضم (**WaitSleepJoin**) حتى يستدعى النيسب المعتمد على نيسب آخر الطريقة (**Join**) والتي تقوم بربط وضم النيسبين وبعدها يخرج النيسب من تلك الحالة.

تعود النياسب الموجودة في حالة الانتظار (**Wait**) أو في حالة السبات (**Sleep**) إلى حالة التشغيل عندما تُستدعى طريقة المقاطعة (**Interrupt**) من قبل نيسب آخر في البرنامج. تقوم هذه الطريقة بإرسال استثناء مقاطعة النيسب (**ThreadInterruptionException**) إلى النيسب الأول.

عندما يتم استدعاء طريقة الإيقاف المؤقت (**Suspend**)، سوف تنتقل حالة النيسب من التشغيل إلى الإيقاف المؤقت. ويمكن إعادة النيسب إلى حالة التشغيل مرة أخرى عندما يستدعى البرنامج طريقة الإستئناف أو المتابعة (**Resume**).

و في الواقع، فإنه عندما تُستدعى طريقة الإيقاف المؤقت (**Suspend**) يدخل النيسب فعلياً في حالة طلب الإيقاف المؤقت (**SuspendRequested**). فإذا كان النيسب في حالة (**WaitSleepJoin**) أو في حالة التجميد عند استدعاء طلب الإيقاف المؤقت، فسوف يبقى النيسب في حالته السابقة وفي حالة انتظار الإيقاف المؤقت (**SuspendRequested**) ولا يمكنه الإستجابة لطلب الإيقاف المؤقت تماماً إلا بعد أن يغادر حالته الحالية. سوف نتكلم لاحقاً عن كيفية التحويل لهذه الحالة بالتفصيل في مزامنة النياسب.

وبشكل مختصر يبقى النيسب في حالة التشغيل إلا إذا صادفته أحد الحالات التالية:

1. أن يصل إلى النهاية ويفنى.
2. أن يجمد بسبب عمليات دخل/خرج (أو أي سبب آخر).
3. أن يحول إلى حالة السبات أو يستدعي من الصف (**Monitor**) الطرائق الانتظار (**Wait**) أو الضم (**Join**) أو السبات (**Sleep**).
4. أن يتسبب نيسب آخر ذو أفضلية في إيقاف عمله مؤقتاً.
5. انتهاء الكمة الزمنية (**Quantum**) المخصصة له.

كذلك يمكن أن يعمل نيسب ويحول إلى حالة التشغيل، على الرغم من وجود نيسب آخر في حالة التشغيل. وذلك بتأجيل عمل النيسب الآخر في الحالات التالية:

1. إذا تم إعادة نيسب إلى حالة التشغيل بعد أن كان في حالة السبات.
2. إذا انتهت معالجة عمليات دخل/خرج لنيسب كان قد جمد بسببها.
3. إذا تم استدعاء الطرق النبضة (**Pulse**) أو (**PulseAll**) لنيسب كان في حالة الانتظار.
4. عندما يعود نيسب للعمل بعد خروجه من حالة الإيقاف المؤقت.
5. أو بعد انتهاء تنفيذ نيسب ذو أولوية أكبر من النيسب الحالي.

3. أفضليات وتنظيم النياسب Thread Priorities and Scheduling

يملك كل نيسب درجة أفضلية **Priority** معينة من بين الأفضليات المتوفرة في التعداد **enumeration (ThreadPriority)** وتتراوح درجة هذه الأفضليات بين:

1. أفضلية دنيا **(Lowest)**.
2. أقل من الطبيعي **(BelowNormal)**.
3. طبيعي **(Normal)**، وهو الوضع الافتراضي للنياسب.
4. فوق الطبيعي **(AboveNormal)**.
5. أفضلية العليا **(Highest)**.

يدعم نظام التشغيل **Windows** ميزة اقتسام الوقت **(Timeslicing)**، والذي يمنح النياسب ذات الأولوية المتساوية الفرصة في مشاركة المعالج. لاحظ أنه بدون هذه الميزة سيعمل كل نيسب حتى انتهاء عمله (إلا إذا تغيرت حالة النيسب عن حالة التشغيل) دون أن يمنح فرصة للنياسب الأخرى التي تملك نفس الأولوية بالتنفيذ. أما مع ميزة تجزئة الوقت، فسيخصص لكل نيسب وقت معين من وقت المعالج يطلق عليه اسم الكمة **Quantum** يمكن أن ينفذ خلالها النيسب جزءاً من عمله. وعند انتهاء الكمة المخصصة لنيسب، يمنع هذا النيسب من استخدام المعالج، لتنتقل الفرصة أو الكمة إلى نيسب آخر إذا وجد وحتى لو لم ينتهي عمل النيسب الأول.

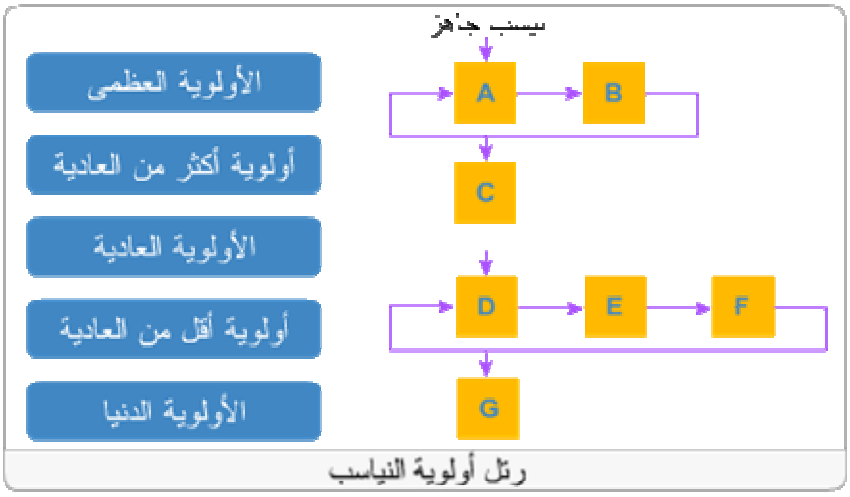
يتلخص عمل مجدول النياسب **(Thread Scheduler)** في أنه يحرص على أن تكون النياسب الأكثر أولوية في حالة التشغيل دائماً. وإذا وجد أكثر من نيسب له الأولوية الكبرى يعمل مجدول النياسب على ضمان عملها في كمات بطريقة تسمى طريقة روبن الدائرية **(Round-Robin Fashion)**.

يبين الشكل التالي تعدد مستويات الأولوية للنياسب بفرض أنها تُعالج في حاسب ذو معالج وحيد. تتم معالجة النيسب **(A)** والنيسب **(B)** بطريقة روبن الدائرية وذلك حتى تنتهي مهمة كل منهما. أي أن النيسب **(A)** سيأخذ كمة من الوقت ليعمل ثم يأخذ النيسب **(B)** كمة ثم يعود النيسب **(A)** ليأخذ كمة أخرى وهكذا حتى ينتهي عمل كل منهما.

يخصص المعالج بعد ذلك كل طاقاته للنياسب المتبقية في الأولويات الأدنى (إلا إذا بدأ عمل نيسب آخر في نفس الأولوية الأعلى). حيث يعمل النيسب **(C)** حتى ينتهي ثم تعمل النياسب **(D)** و **(E)** و **(F)** بنفس الطريقة الدائرية حتى تنتهي من التنفيذ.

وهنا يجب أن نلاحظ أن الاعتماد على نظام التشغيل في جدولة النياسب وتنفيذها حسب أولويتها سوف يمنح النياسب الجديدة ذات الأولوية الكبرى القدرة على تأجيل تنفيذ النياسب ذات الأولوية الدنيا. مما قد يؤدي هذا التأجيل غير المحدود في بعض الأحيان إلى تحطيم حظ وفرصة النياسب ذات الأولوية الدنيا من التنفيذ.

يمكن ضبط وتعريف أولوية النياسب من الخاصية **(Priority)** والتي تأخذ قيمها من مجموعة التعداد **(ThreadPriority)**. وفي حال تم تعريف قيمة غير معرفة تحت هذا التعداد سوف يظهر استثناء أو خطأ في القيمة التي تم تمريرها **(ArgumentException)**.



رتل أولوية النياسب

4. بناء وتنفيذ النياسب Creating and Executing Threads

لنبدأ بالبرنامج البسيط التالي والذي يقوم بتطبيق الطريقة **(Sleep)** على النياسب. يقوم هذا البرنامج ببناء ثلاثة نياسب مختلفة، لكل منها درجة الأولوية الافتراضية (الأولوية العادية). يظهر كل نيسب في البداية رسالة تشير إلى أن النيسب سوف يتحول إلى حالة السبات **(Sleep)** وذلك لفترة زمنية عشوائية ضمن المجال **(0-5000)** ميللي ثانية. ثم يذهب فعلياً إلى حالة السبات. وعند انتهاء الفترة الزمنية، سوف يظهر رسالة أخرى تُبين اسم النيسب وأنه قد خرج من حالة السبات، ثم ينتقل النيسب إلى حالة الإيقاف، سوف نلاحظ هنا أن تنفيذ البرنامج الفعلي ينتهي قبل انتهاء النياسب من تنفيذ مهماتها.

```
1  // ThreadTester.cs
2  // Multiple threads printing at different intervals.
3  using System;
4  using System.Threading;
5
6  // class ThreadTester demonstrates basic threading concepts
7  class ThreadTester
8  {
9    static void Main( string[] args )
10   {
11    // Create and name each thread. Use MessagePrinter's
12    // Print method as argument to ThreadStart delegate.
13    MessagePrinter printer1 = new MessagePrinter();
14    Thread thread1 = new Thread (new ThreadStart(printer1.Print));
15    thread1.Name = "thread1";
16
17    MessagePrinter printer2 = new MessagePrinter();
18    Thread thread2 = new Thread ( new ThreadStart(printer2.Print));
19    thread2.Name = "thread2";
20
21    MessagePrinter printer3 = new MessagePrinter();
22    Thread thread3 = new Thread ( new ThreadStart(printer3.Print));
23    thread3.Name = "thread3";
24
25    Console.WriteLine( "Starting threads" );
26
27    // call each thread's Start method to place each
28    // thread in Running state
29    thread1.Start();
30    thread2.Start();
31    thread3.Start();
32
33    Console.WriteLine( "Threads started\n" );
34   } // end method Main
35 } // end class ThreadTester
36
37  // Print method of this class used to control threads
38  class MessagePrinter
39  {
40   private int sleepTime;
41   private static Random random = new Random();
```

```
41
43    // constructor to initialize a MessagePrinter object
44    public MessagePrinter()
45    {
46       // pick random sleep time between 0 and 5 seconds
47       sleepTime = random.Next( 5001 ); // 5001 milliseconds
48    } // end constructor
49
50    // method Print controls thread that prints messages
51    public void Print()
52    {
53       // obtain reference to currently executing thread
54       Thread current = Thread.CurrentThread;
55
56       // put thread to sleep for sleepTime amount of time
57       Console.WriteLine( "{0} going to sleep for {1} milliseconds",
58          current.Name, sleepTime );
59      Thread.Sleep( sleepTime ); // sleep for sleepTime milliseconds
60
61       // print thread name
62       Console.WriteLine( "{0} done sleeping", current.Name );
63    } // end method Print
64  } // end class MessagePrinte
```

يتكون البرنامج من صفين:

- الصف **(ThreadTester)** في السطور **(7-35)**، والذي مهمته بناء النياسب الثلاثة.
- الصف **(MessagePrinter)** في السطور **(38-64)**، والذي مهمته تعريف الطريقة **(Print)** التي تحتوي المهمات التي ستنجزها وتعالجها النياسب.
- يقوم الباني **constructor** للصف **(MessagePrinter)** بتهيئة متغير زمن السبات **(SleepTime)** بإعطائه قيمة عشوائية ضمن المجال **(0 - 5000)**.
- تبدأ الطريقة **(Print)** للصف **(MessagePrinter)** بالوصول إلى النيسب الحالي(النيسب الذي استدعى تنفيذ الطريقة **(Print)**) عن طريق الطريقة **(Thread.CurrentThread)** السطر **(54)**، ثم تُظهر رسالة تحتوي اسم النيسب الحالي المنفذ وأنه سوف يقوم بتنفيذ الطريقة **(Sleep)** لمدة زمنية معينة السطور **(57-58)** لاحظ أن التعليمة في السطر **58** تقوم باستخدام الخاصية **(Name)** للنيسب المنفذ (تُحدد هذه الخاصية عند بناء النيسب).
- تتغير حالة النيسب السطر **(59)** إلى حالة انتظار/سبات/ضم **(WaitSleepJoint)**، وفي حالةهذه المرحلة سوف يفقد النيسب المعالج الذي كان يستخدمه للتنفيذ (فيفسح نظام التشغيل المجال لنيسب آخر إن وجد). وبعد أن يعود النيسب إلى حالة التشغيل ويخرج من السبات، ينتظر حتى يتم إسناده إلى معالج معين عن طريق مجدول النياسب، وبعدها سوف يظهر رسالة جديدة السطر **(62)** تتضمن اسم النيسب وإلى أنه قد أتم حالة السبات للفترة الزمنية التي حددت له وخرج منها.

- تكون الطريقة **(Main)** المسؤولة عن تنفيذ البرنامج فعلياً السطور **(9-34)**، حيث تبني أولاً ثلاثة أغراض من الصف **(MessagePrinter)** السطور **(21,17,13)**. ثم تخلق ثلاثة نياسب السطور **(22,18,14)** تكون مهمة كل منها تنفيذ الطريقة **Print** للغرض الموافق.
- يقوم المفوض **(ThreadStart)** في السطر**(14)** بإسناد النيسب **(thread1)** إلى الطريقة **(Print)**، يدخل بعد ذلك هذا النيسب في حالة التشغيل، مما يجبر الغرض **(printer1)** على تنفيذ المهمة المعرفة في محتوى الطريقة **(Print)**، وهكذا سوف يظهر النيسب **(thread1)** اسمه والفترة الزمنية التي سوف يبقى فيها في حالة السبات ويسبت فعلياً لهذه المدة ثم يعود ليظهر رسالة أخرى تشير إلى أن النيسب قد أنهى السبات، في هذه اللحظة سوف تنتهي الطريقة **(Print)** وسوف ينتقل النيسب إلى حالة الإيقاف، وتتكرر هذه العمليات على النيبسبن التاليين.

لاحظ أنه في هذا المثال سوف ينتهي النيسب ذو الوقت الأقل من السبات أولاً. سوف نناقش لاحقاً كيف نستطيع منع النيسب ذو الوقت الأقل من العودة والخروج أولاً من حالة السبات إلى حالة التشغيل.

تُبين الأشكال التالية ثلاثة تنفيذات مختلفة للبرنامج.

```
Starting threads
thread1 going to sleep for 3534 milliseconds
thread2 going to sleep for 1828 milliseconds
thread3 going to sleep for 4474 milliseconds
Threads started

thread2 done sleeping
thread1 done sleeping
thread3 done sleeping
Press any key to continue . . .
```

```
Starting threads
thread1 going to sleep for 1043 milliseconds
thread2 going to sleep for 1728 milliseconds
thread3 going to sleep for 3332 milliseconds
Threads started

thread1 done sleeping
thread2 done sleeping
thread3 done sleeping
Press any key to continue . . .
```

```
Starting threads
thread1 going to sleep for 1948 milliseconds
thread2 going to sleep for 1742 milliseconds
thread3 going to sleep for 382 milliseconds
Threads started

thread3 done sleeping
thread2 done sleeping
thread1 done sleeping
Press any key to continue . . .
```

5. مزامنة النياسب Thread Synchronization

إذا كانت لدينا عدة نياسب تتشارك في استخدام نفس البيانات. فيمكن لهذه النياسب أن تقرأ هذه البيانات أو أن تصل لأي جزء منها بدون أي مشكلة. ولكن عندما تتشارك عدة نياسب في بيانات معينة وتقوم بالتعديل على هذه البيانات، فستحدث مشاكل تضارب **collision** ما لم يتم مزامنة النياسب التي تشترك في هذه البيانات.

فمثلاً، لنفترض وجود نيسبان يقومان بتحديث بيانات معينة في وقت واحد ولكن لم تتم

مزامنتهما. سيتم بالطبع تحديث البيانات عن طريق آخر نيسب مع تجاهل ماقام به النيسب الأول من تعديلات. وإذا كانت هذه البيانات ممثلة بمصفوفة أو أي بنية معطيات أخرى وبحيث تُمكّن النياسب من تحديث أجزاء منفصلة من هذه البيانات في نفس الوقت، فمن الممكن أن نلاحظ أن جزءاً من هذه البيانات يعكس بيانات من نيسب والجزء الآخر يعكس بيانات من نيسب آخر، وسيكون من الصعب جداً أن يحدد البرنامج متى تم تحديث هذه البيانات.

يتم حل المشاكل السابقة بمزامنة **Synchronization** النياسب التي تريد مشاركة بيانات معينة لقراءتها والتعديل فيها. حيث نقصد بالمزامنة هنا، إعطاء الإمكانية لنيسب واحد فقط بمعالجة البيانات والسماح له بالتأثير فيها بشكل حصري **exclusive** وخلال هذا الوقت تكون النياسب الأخرى في حالة الانتظار. وعندما ينتهي النيسب المتحكم في البيانات، تُمنح الفرصة لأحد النياسب المنتظرة للتعامل معها، وبهذه الطريقة سيتعامل كل نيسب مع البيانات بشكل حصري وستُمنع النياسب الأخرى من التعامل مع البيانات في وقت واحد.

يقدم الصف **(Monitor)** عدداً من الطرق التي تُمكّن من مزامنة النياسب (مشاركة نياسب متعددة لبيانات معينة).

عندما يحتاج نيسب معين أن يتعامل مع غرض بشكل حصري، يجب أن يستدعي هذا النيسب الطريقة **(Enter)** من الصف **(Monitor)**، تقوم هذه الطريقة بتثبيت قفل على الغرض و بحيث يمكن لنيسب واحد فقط التعامل معها. يكون لكل غرض الخاصية **(SyncBlock)** التي تُحدد حالة قفل الغرض. وفي حال طلب أي نيسب آخر التعامل مع غرض مقفل فإنه يجمد **Blocked**.

عندما ينتهي النيسب من التعامل مع الغرض ينفذ الطريقة **(Exit)** لتحرير الغرض من القفل كما سيتم تحديث حالة قفل الغرض **(SyncBlock)** لتشير بأنه أصبح حراً ومتاحاً من جديد. وإذا وجد نيسب كان قد وضع في حالة التجميد فسوف يعاد إلى حالة التشغيل وتكرر العمليات السابقة، وهكذا، إلى أن تنتهي جميع النياسب التي تريد التعامل مع الأغراض نفسها.

يجب هنا الانتباه إلى الأخطاء البرمجية التالية الشائعة:

- يجب التأكد من أن جميع التعليمات التي تعمل على تعديل الغرض تطلب قفل الغرض قبل التعامل معه.
- تظهر حالة **(Deadlock)** عند عدم قدرة نيسب ما على التنفيذ بسبب انتظاره لنيسب آخر وفي نفس الوقت لاينفذ النيسب الثاني لأنه في انتظار تنفيذ النيسب الأول، أي أن هنالك نيسبان في انتظار بعضهما البعض.
- لا يمكن لنيسب في حالة انتظار /سبات/ ضم أن يعاد إلى حالة التشغيل ليكمل تنفيذ مهمته، إلا إذا قام نيسب آخر باستدعاء طريقة الإنعاش **(Pulse)** أو إنعاش الكل **(PulseAll)**، وإلا سيبقى النيسب منتظراً وهذا عملياً يشبه حالة **(Deadlock)**.

يمكن أيضاً تطبيق المفهوم السابق عملياً بلغة **(C#)** كالتالي: يتم وضع الغرض المراد قفله كمدخل للطريقة **lock**:

```
Lock (objectReference)
{ //التعليمات التي تطلب المزامنة تكتب هنا
}
```

تمنح التعليمة السابقة قفل للنيسب الذي يريد التعامل مع الغرض الذي مرر مرجعه بين القوسين، ويدخل الغرض ذاته وضع حظر القفل حتى لا يمكن لنيسب آخر أن يمنح قفل على الغرض نفسه. ويكون مرجع الغرض هو نفسه الذي يمرر بالعادة إلى طرائق الصف **(Monitor)**:

دخول، خروج، إنعاش، خروج

والتي تُستخدم عند مزامنة النياسب والتي سنتحدث عنها لاحقاً.

عند إنتهاء وضع حظر القفل لأي سبب من الأسباب يحرر **C#** الغرض من القفل الذي كان قد منح للنيسب. فمثلاً، عندما يتوقف عمل النيسب الذي يملك القفل على تحقق شرط معين ويدخل حالة الانتظار سوف يحرر الغرض من القفل. وعند تحقق الشروط التي ينتظرها النيسب ويعود إلى حالة التشغيل يمكنه إعادة طلب قفل على الغرض ليكمل مهمته.

ومما يجب أن يلاحظ هنا أنه يوجد فرق بين أن يكون النيسب منتظراً أن يمنح قفل على الغرض لأنه عملياً يكون النيسب في حالة التجميد وليس في حالة الانتظار وأن يكون في حالة انتظار /ضم/ سبات ينتظر تحقق شرط معين.

6. علاقة المنتج - المستهلك بدون مزامنة النياسب

Producer/Consumer Relationship without Thread Synchronization

ندعو علاقة منتج/مستهلك العلاقة بين جزئين من تطبيق أو برنامج ما. حيث يقوم الجزء الأول والمسمى بالمنتج بتوليد البيانات، بينما يقوم الجزء الثاني المسمى بالمستهلك بمعالجتها.

كذلك الأمر في حالة النياسب. حيث يقوم النيسب المنتج باستدعاء طريقة تولد البيانات وتحفظها في الذاكرة في منطقة مشتركة يمكن لأي نيسب الوصول إليها (تُدعى هذه المنطقة بالصوان **Buffer**). بينما يستدعي النيسب المستهلك طريقة مستهلكة تقرأ البيانات المحفوظة في الصوان وتعالجها.

عندما نُطبق هذه العلاقة على النياسب بدون مزامنتها سوف نواجه المشكلتين التاليتين:

1. **المشكلة الأولى:** تظهر عندما يريد النيسب المنتج حفظ بيانات جديدة في الصوان لكنه يجد أن النيسب المستهلك لم ينته بعد من معالجة البيانات القديمة. وهنا من الممكن أن يبدل النيسب المنتج المعلومات القديمة بأخرى جديدة، وبالتالي تُفقد البيانات القديمة دون أن يمنح النيسب المستهلك فرصة لقراءتها.
2. **المشكلة الثانية:** تظهر عندما يجد النيسب المستهلك الصوان فارغاً أو أنه يحتوي على بيانات قديمة تمت معالجتها سابقا. وفي هذه الحالة قد تُعالج معلومات قديمة مرة أخرى، وفي كلتا الحالتين السابقتين سوف تكون النتيجة غير صحيحة منطقياً في التطبيق.

تُحل المشكلتين السابقتين بإلزام النيسب الأحدث أن ينتظر انتهاء النيسب الأول من عمله. ويتم ذلك باستدعاء النيسب الأحدث لطريقة الانتظار، وبعد أن ينهي النيسب الأول مهمته يقوم باستدعاء طريقة الإنعاش لتُعيد النيسب الثاني لحالة التشغيل ليكون قادراً على إتمام مهمته.

وبمعنى آخر:

ففي حالة المشكلة الأولى، يستدعي النيسب المنتج طريقة الانتظار ويبقى منتظراً حتى ينتهي النيسب المستهلك من قراءة البيانات ومعالجتها. وعندما ينتهي هذا النيسب المستهلك يقوم باستدعاء طريقة الإنعاش ليعيد النيسب المنتج إلى حالة التشغيل ليعاود عمله.
أما في الحالة الثانية فيستدعي النيسب المستهلك طريقة الانتظار حتى ينتهي النيسب المنتج من عمله، و الذي يقوم باستدعاء طريقة الإنعاش لإحياء النيسب المستهلك.

ولتوضيح المسألة السابقة، ليكن المثال التالي والذي يفرض وجود نيسب منتج يولد رقم متسلسل من **1** إلى **10** في الصوان، وآخر مستهلك يقوم بقراءة الأرقام تباعاً وإظهارها.

يحوي البرنامج المثال التالي نيسب منتج وآخر مستهلك، يتعاملان مع متغير واحد من نوع عدد صحيح **(integer)** ولكن بدون مزامنتها معاً. ومن المفروض أن يقوم النيسب المنتج بكتابة رقم في المتغير، يقرؤه النيسب المستهلك ويعرضه مرة واحدة فقط. وبما أنه لا توجد مزامنة بين النيسبين، فمن الممكن أن تُفقد بيانات إذا وضع المنتج بيانات جديدة قبل أن يعالج المستهلك البيانات القديمة. أو أن تُعاد معالجة بيانات قديمة أكثر من مرة وذلك قبل أن تُنتج بيانات جديدة.

ولكي نتأكد من أن المستهلك قد عالج كل قيمة مرة واحدة فقط، ألزمناه بجمع قيم الأرقام التي

يقرؤها، وبما أن هذه الأرقام هي أرقام صحيحة متسلسلة من **1** إلى **10** سيكون مجموع القيم الكلي هو **(55)**. ولكن ماسنلاحظه هنا، وبالرغم من تنفيذ البرنامج عدد من المرات. فإنه نادراً ما سنحصل على القيمة **(55)** وربما لن نحصل عليها أبداً.

وحتى نؤكد على عشوائية عمل النيسبين لنُظهر المشكلة التي نريد توضيحها سوف نُدخل

النيسبين في حالة السبات لمدة عشوائية بين **(0- 3** ثانية**)** بين كل خطوة من مهماتهما، وبالتالي لا يمكن تحديد متى سيقوم المنتج بكتابة قيمة جديدة بالضبط أو تحديد متى سيقوم المستهلك بقراءتها.

لندخل الآن إلى التفاصيل البرمجية يحوي البرنامج التالي عدداً من الصفوف الهامة:
أولها الواجهة **interface (Buffer)**، بعدها نجد الصف المنتج **(Producer)** والصف المستهلك **(Consumer)**، ثم الصف **(UnsynchronizedBuffer)** والصف **(UnsynchronizedBufferTest)**، سوف نتحدث فيما يلي بالتفصيل عن جميع الصفوف السابقة وعن كيفية تنفيذ البرنامج والأخطاء والمشاكل المحتمل ظهورها.

تحوي الواجهة **(Buffer)** تصريح عن الخاصية **(Buffer)** من نوع عدد صحيح والتي تُلزم أي إسناد أو قراءة للمتغير أن يدخل أولاً في شروط الطريقتين **(get)** و **(set)**:

```
1 // Buffer.cs
2 // Interface for a shared buffer of int.
3 using System;
4
5 // this interface represents a shared buffer
6 public interface Buffer
7 {
8 // property Buffer
9 int Buffer
10 {
11 get;
12 set;
13 } // end property Buffer
14 } // end interface Buffer
```

يحوي الصف المنتج **(Producer)** متغيرين الأول **(sharedLocation)** من نوع **(Buffer)** والثاني **(randomSleepTime)** من نوع المتغير العشوائي. وكذلك يحوي البانى الذي يهيئ المتغيرين السابقين. تحوي الطريقة المنتجة **(Produce)** لهذا الصف المهمة التي سيقوم النيسب المنتج بتنفيذها، حيث أنها تحوي حلقة **(for)** تكرر عشر مرات، وفي كل مرة تُدخل النيسب حالة السبات باستخدام الطريقة **(Sleep)** لفترة زمنية عشوائية بين **(0-3)** ثواني)، وعند انتهاء الفترة يعود النيسب لحالة التشغيل ويسند قيمة المتغير **(Count)** إلى المتغير **(sharedLocation)**، وعند انتهاء الحلقة تماماً يعرض النيسب رسالة تشير إلى أنه قد أنهى مهمته، وتنتهي الطريقة المنتجة فعلياً ويدخل النيسب حالة الإيقاف.

```
1  // Producer.cs
2  // Producer produces 10 integer values in the shared buffer.
3  using System;
4  using System.Threading;
5
6  // class Producer's Produce method controls a thread that
7  // stores values from 1 to 10 in sharedLocation
8  public class Producer
```

```
9  {
10    private Buffer sharedLocation;
11    private Random randomSleepTime;
12
13    // constructor
14    public Producer( Buffer shared, Random random )
15    {
16       sharedLocation = shared;
17       randomSleepTime = random;
18    } // end constructor
19
20    // store values 1-10 in object sharedLocation
21    public void Produce()
22   {
23       // sleep for random interval up to 3000 milliseconds
24       // then set sharedLocation's Buffer property
25       for ( int count = 1; count <= 10; count++ )
26       {
27          Thread.Sleep( randomSleepTime.Next( 1, 3001 ) );
28          sharedLocation.Buffer = count;
29       } // end for
30
31       Console.WriteLine( " {0} done producing.\nTerminating {0}.",
                                32          Thread.CurrentThread.Name );
                                            33    } // end method Produce
                                              34 } // end class Producer
```

يحوي الصف المستهلك **(Consumer)** متغيرين الأول **(sharedLocation)** من نوع **(Buffer)** والثاني **(randomSleepTime)** من نوع المتغير العشوائي، وكذلك يحوي البانى الذي يهيئ المتغيرين السابقين.

تحوي الطريقة المستهلكة **(Consume)** لهذا الصف المهمة التي سيقوم النيسب المستهلك بتنفيذها.

تُهيئ الطريقة المنتجة **(Consume)** متغير الجمع **(Sum)** في البداية بالقيمة **(0)**، وتحوي حلقة **(for)** تكرار **10** مرات، في كل مرة تدخل النيسب المستهلك حالة السبات لمدة زمنية بين **(0- 3** ثواني) ثم تعيده لحالة التشغيل عند انتهاء الفترة وبعدها تضيف قيمة المتغير **(sharedLocation)** إلى القيمة السابقة في المتغير**(Sum)**، وعند انتهاء الحلقة تظهر رسالة تشير إلى مجموع القيم التي تمت قراءتها وأن النيسب قد أنهى مهمته، وتنتهي الطريقة فعلياً وينتقل النيسب إلى حالة الإيقاف.

```
1  // Consumer.cs
2  // Consumer consumes 10 integer values from the shared buffer.
3  using System;
4  using System.Threading;
5
6  // class Consumer's Consume method controls a thread that
7  // loops 10 times and reads a value from sharedLocation
8  public class Consumer
```

```
9  {
10    private Buffer sharedLocation;
11    private Random randomSleepTime;
12
13    // constructor
14    public Consumer( Buffer shared, Random random )
15    {
16       sharedLocation = shared;
17       randomSleepTime = random;
18    } // end constructor
19
20    // read sharedLocation's value ten times
21    public void Consume()
22    {
23       int sum = 0;
24
25       // sleep for random interval up to 3000 milliseconds then
26       // add sharedLocation's Buffer property value to sum
27       for ( int count = 1; count <= 10; count++ )
28       {
29          Thread.Sleep( randomSleepTime.Next( 1, 3001 ) );
30          sum += sharedLocation.Buffer;
31       } // end for
32
33       Console.WriteLine(
34          " {0} read values totaling: {1}.\nTerminating {0}.",
35          Thread.CurrentThread.Name, sum );
36    } // end method Consume
37 } // end class Consumer
```

لاحظ أن المثال يحوي نياسب بسيطة، فالمنتج يدخل في حلقة لعشر مرات ويحوي عبارة إسناد، والمستهلك يدخل في حلقة عشر مرات ويجمع قيم ويسندها للمتغير **(Sum)**، ولو أنا تخلينا عن تعليمة إدخال النياسب في السبات وإذا بدأ تنفيذ المنتج أولاً سيستمر التنفيذ حتى انتهاء الحلقة دون أن يعطي فرصة للمستهلك، وإذا بدأ المستهلك في التنفيذ فسوف يدخل في الحلقة ويعالج دائماً القيمة **(-1)** وينتهي قبل حتى أن يبدأ المنتج بإعطاء القيم الفعلية.

يشتق الصف **(UnsynchronizedBuffer)** من الواجهة **(Buffer)** ولكن مع تعديل الطريقتين **(get)** و **(set)** للخاصية **(Buffer)**. تقوم الطريقة **(get)** باستخدام الخاصية **(CurrentThread)** من الصف **(Thread)** للحصول على مرجع النيسب الذي ينفذ حالياً ومن ثم تصل لخاصية اسم النيسب الحالي **(Name)** حتى تحدد اسم النيسب الذي ينفذ قراءة الخاصية. وتطبع رسالة تحوي اسم النيسب القارئ وقيمة الخاصية المقروءة. كذلك تقوم الطريقة **(set)** بطباعة اسم النيسب المنتج والقيمة الموافقة.

```
1  // UnsynchronizedBuffer.cs
2  // An unsynchronized shared buffer implementation.
3  using System;
4  using System.Threading;
5
6  // this class represents a single shared int
```

```
7  public class UnsynchronizedBuffer : Buffer
8  {
9     // buffer shared by producer and consumer threads
10    private int buffer = -1;
11
12    // property Buffer
13    public int Buffer
14    {
15       get
16       {
17          Console.WriteLine( "{0} reads {1}",
18             Thread.CurrentThread.Name, buffer );
19          return buffer;
20       } // end get
21       set
22       {
23          Console.WriteLine( "{0} writes {1}",
24             Thread.CurrentThread.Name, value );
25          buffer = value;
26       } // end set
27    } // end property Buffer
28 } // end class UnsynchronizedBuffer
```

يحوي الصف **(UnsynchronizedBufferTest)** الطريقة الأساسية في البرنامج **(Main())**، والتي وتحوي على غرض من نوع **(UnsynchronizedBuffer)** وغرض من نوع عشوائي الذي يولد زمن السبات العشوائي. يستخدم هذين المتغيرين كدخل لباني غرض من نوع **(Producer)** **(Consumer)**.

```
1  // UnsynchronizedBufferTest.cs
2  // Showing multiple threads modifying a shared object without
3  // synchronization.
4  using System;
5  using System.Threading;
6
7  // this class creates producer and consumer threads
8  class UnsynchronizedBufferTest
9  {
10    // create producer and consumer threads and start them
11    static void Main( string[] args )
12    {
13       // create shared object used by threads
14       UnsynchronizedBuffer shared = new UnsynchronizedBuffer();
15
16       // Random object used by each thread
17       Random random = new Random();
18
19       // create Producer and Consumer objects
20       Producer producer = new Producer( shared, random );
21       Consumer consumer = new Consumer( shared, random );
22
23       // create threads for producer and consumer and set
24       // delegates for each thread
25       Thread producerThread =
```

```
26          new Thread( new ThreadStart( producer.Produce ) );
27       producerThread.Name = "Producer";
28
29       Thread consumerThread =
30         new Thread( new ThreadStart( consumer.Consume ) );
31       consumerThread.Name = "Consumer";
32
33       // start each thread
34       producerThread.Start();
35       consumerThread.Start();
36    } // end Main
37 } // end class UnsynchronizedBufferTest
```

يحوي الغرض **shared** من نوع **(UnsynchronizedBuffer)** البيانات المراد تشارك المنتج **(Producer)** والمستهلك **(Consumer)** معها. حيث سيقوم باني كل منهما بإسناد المتغير **shared** إلى المعامل المسمى **(SharedLocation)** الموجود في كل منهما.

بعد ذلك، ينشئ النيسب المنتج ويسمى **Producer** السطور **(25-27)** ويحدد المفوض **(ThreadStart)** للنيسب المنتج مهمته وهي أنه سينفذ الطريقة المنتجة من الصف المنتج **(Produce)**.

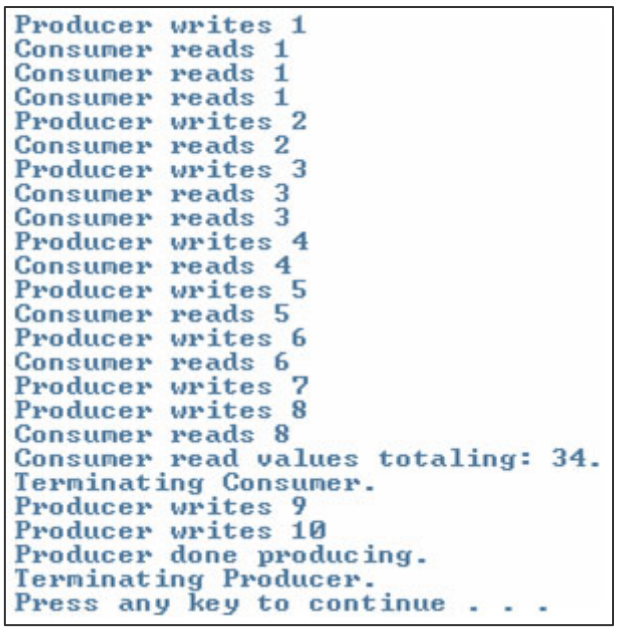
وبالمثل، ينشئ النيسب المستهلك ويسمى **Consumer** السطور **(29-31)** ويحدد المفوض **(ThreadStart)** للنيسب المستهلك مهمته وهي أنه سينفذ الطريقة المستهلكة من الصف المستهلك.

وفي نهاية الطريقة **(Main)** يوضع النيسبان في حالة التشغيل باستخدام الطريقة **(Start)** ويبدأ النيسبان بالتنفيذ الفعلي السطور **(34-35)**.

يكون الوضع المثالي والصحيح الذي نتمنى الحصول عليه هو أن تتم معالجة جميع القيمة المنتجة مرة واحدة فقط، لنلقي نظرة على خرج البرنامج الذي حصلنا عليه لتنفيذ البرنامج مرتين متتاليتين.

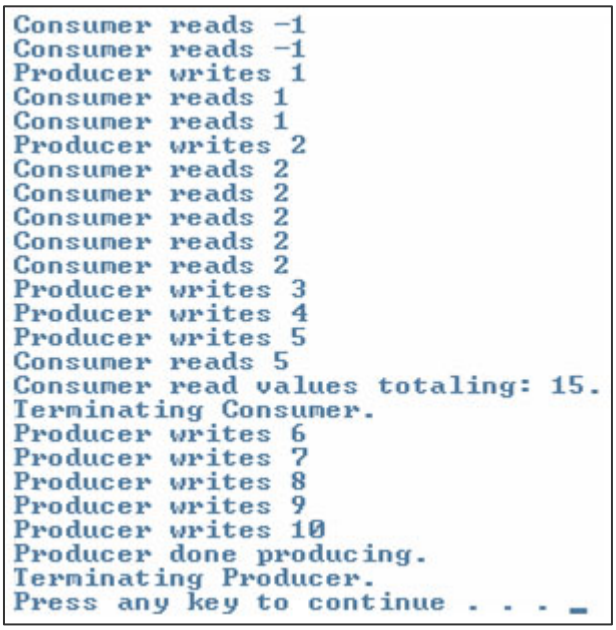
يبين الشكل التالي تنفيذ البرنامج للمرة الأولى:

في المرة الأولى من تنفيذ البرنامج نلاحظ أن القيم **(1,3)** قد عولجت من قبل النيسب المستهلك أكثر من مرة، وأن النيسب المنتج أظهر الرقمين **(7,8)** بالتتالي مباشرة ولم يعطي المجال للنيسب المستهلك في أن يعالج الرقم **(7)**، وبالتالي كان المجموع الناتج الذي حصلنا عليه هنا **34** غير صحيح (لايساوي **55**).



يبين الشكل التالي تنفيذ البرنامج لمرة ثانية:

نلاحظ أنه في المرة الثانية بدأ النيسب المستهلك بالعمل بمعالجة الرقم **(-1)** وذلك قبل أن يبدأ المنتج بإعطاء أي قيمة. كذلك فلم يترك النيسب المنتج المجال للمستهلك بأن يعالج الرقمين **(3,4)** لأنه أنتج الرقم **(4)** مباشرة بعد الرقم **(3)** وكذلك أنتج الرقم **(5)** مباشرة بعد الرقم **(4)**. كما نلاحظ أن النيسب المستهلك قد انتهى عمله قبل أن ينتهي النيسب المنتج وبالتالي لم تعالج القيم **(6,7,8,9,10)** لأن النيسب المنتج أنتجها بعد انتهاء النيسب المستهلك ودخوله حالة الإيقاف. ومما يلفت النظر أن القيمة **(2)** قد عولجت من قبل النيسب المستهلك خمسة مرات والقيمة **(1)** كذلك عولجت مرتين من قبل النيسب المنتج، ولجميع الأسباب السابقة ظهرت نتيجة الجمع **15** غير صحيحة.



ولحل المشاكل السابقة المتمثلة في معالجة نفس القيمة أكثر من مرة أو فقدان بعض القيم دون معالجتها، تتم مزامنة النيسبين المنتج والمستهلك وذلك باستخدام الصف **(Monitor)** وطرائقه المتمثلة في الدخول **(Enter)**، الانتظار **(Wait)**، الإنعاش **(Pulse)**، والخروج **(Exit)**. حيث أنه عند مزامنة النيسبين سوف يقفل الغرض المشترك ويمنح لنيسب واحد فقط وبالتالي سيتعامل معه هذا النيسب حصرياً ولايمكن لأي نيسب آخر التعامل مع نفس الغرض. هذا مانشرحه في الفقرة التالية.

7. علاقة المنتج - المستهلك مع مزامنة النياسب

Producer/Consumer Relationship with Thread Synchronization

بهدف توضيح آلية مزامنة النياسب ومعرفة الفائدة من ذلك، سوف نستعين بالبرنامج المثال التالي والذي يتضمن نيسب منتج ونيسب مستهلك يتعاملان مع خلايا معينة في الذاكرة بشكل متزامن، أي أنه لن يستهلك المستهلك قبل أن ينتج المنتج والعكس صحيح.

يحوي المثال الواجهة **(Buffer)** والصفوف **(Consumer)** و **(Producer)** التي تعاملنا معها في الفقرة السابقة، وكذلك لدينا الصف **(SynchronizedBuffer)** الذي يتضمن الواجهة **(Buffer)** والمتغير **(buffer)**. لاحظ أن شروط الخاصية **(Buffer)** في الطرق **(get)** و **(set)** تتعامل مع الصف **(Monitor)** وطرقه بهدف تحقيق التزامن.

يتضمن الصف **(SynchronizedBuffer)** المتغير **(occupiedBufferCount)** والذي يحدد للخاصية **(Buffer)** فيما إذا كان فيما إذا كان الدور في التنفيذ للنيسب المنتج أو للنيسب المستهلك. يأخذ هذا المتحول إحدى القيمتين:

1. القيمة **(0)**: في هذه الحالة يسمح الشرط للطريقة **(set)** بإسناد قيمة الخاصية **(Buffer)**، في حين لا يسمح الشرط للطريقة **(get)** من إعادة قيمة الخاصية.
2. القيمة **(1)**: في هذه الحالة يمكّن الشرط في الطريقة **(get)** من إرجاع قيمة المتغير **(buffer)**، إلا أنه لايمكن للطريقة **(set)** من إسناد قيمة جديدة.

لاحظ أن جميع الأغراض التي تكون من النوع **(SynchronizedBuffer)** سوف تملك الخاصية **(SyncBlock)**.

```
1  // SynchronizedBuffer.cs
2  // A synchronized shared buffer implementation.
3  using System;
4  using System.Threading;
5
6  // this class represents a single shared int
7  public class SynchronizedBuffer : Buffer
8  {
9     // buffer shared by producer and consumer threads
10    private int buffer = -1;
11
12    // occupiedBufferCount maintains count of occupied buffers
13    private int occupiedBufferCount = 0;
14
15    // property Buffer
16   public int Buffer
17    {
18      get
19       {
20         // obtain lock on this object
21          Monitor.Enter( this );
22
23        // if there is no data to read, place invoking
24         // thread in WaitSleepJoin state
25          if ( occupiedBufferCount == 0 )
26         {
27             Console.WriteLine(
28                Thread.CurrentThread.Name + " tries to read." );
29            DisplayState( "Buffer empty. " +
30                Thread.CurrentThread.Name + " waits." );
31             Monitor.Wait( this ); // enter WaitSleepJoin state
32        } // end if
33
34          // indicate that producer can store another value
35          // because consumer is about to retrieve a buffer value
36         --occupiedBufferCount;
37
38       DisplayState(Thread.CurrentThread.Name+" reads " + buffer );
```

```
39
40          // tell waiting thread (if there is one) to
41          // become ready to execute (Running state)
42          Monitor.Pulse( this );
43
44        // Get copy of buffer before releasing lock.
45        // It is possible that the producer could be
46        // assigned the processor immediately after the
47        // monitor is released and before the return
48        // statement executes. In this case, the producer
49        // would assign a new value to buffer before the
50        // return statement returns the value to the
51        // consumer. Thus, the consumer would receive the
52        // new value. Making a copy of buffer and
53        // returning the copy ensures that the
54        // consumer receives the proper value.
55           int bufferCopy = buffer;
56
57            // release lock on this object
58            Monitor.Exit( this );
59
60           return bufferCopy;
61         } // end get
62         set
63         {
64            // acquire lock for this object
65            Monitor.Enter( this );
66
67            // if there are no empty locations, place invoking
68            // thread in WaitSleepJoin state
69            if ( occupiedBufferCount == 1 )
70            {
71               Console.WriteLine(
72                  Thread.CurrentThread.Name + " tries to write." );
73               DisplayState( "Buffer full. " +
74                  Thread.CurrentThread.Name + " waits." );
75               Monitor.Wait( this );// enter WaitSleepJoin state
76            } // end if
77
78            // set new buffer value
79           buffer = value;
80
81            // indicate consumer can retrieve another value
82            // because producer has just stored a buffer value
83            ++occupiedBufferCount;
84
85         DisplayState(Thread.CurrentThread.Name+" writes "+ buffer );
86
87            // tell waiting thread (if there is one) to
88           // become ready to execute (Running state)
89           Monitor.Pulse( this );
90
91            // release lock on this object
92            Monitor.Exit( this );
93         } // end set
94      } // end property Buffer
95
96      // display current operation and buffer state
97      public void DisplayState( string operation )
```

```
98    {
99      Console.WriteLine( "{0,-35}{1,-9}{2}\n",
100         operation, buffer, occupiedBufferCount );
101   } // end method DisplayState
102} // end class SynchronizedBuffer
```

وكما في المثال السابق، سينفذ النيسب المنتج المهمة الموجودة في الطريقة المنتجة **(Produce)**، فعندما تُسند قيمة للخاصية **(Buffer)** في السطر **(28)** من تعليمات الصف المنتج، سوف يدخل البرنامج في الطريقة **(set)** السطور **(62-93)** حيث تُستدعى طريقة الدخول **(Enter)** من الصف **(Monitor)** مع **(this)** كدخل والتي تُشير إلى الغرض الحالي.

تؤدي هذه الطريقة **(Enter)** إلى طلب القفل على الغرض **(SynchronizedBuffer)**، ثم يدرس الشرط **(If)** قيمة المتغير **(occupiedBufferCount)** السطور **(69- 76)** فإذا كانت قيمته مساوية للواحد، فسوف تنفذ التعليمات في السطور **(71-74)** والتي تُظهر رسالة بأن النيسب المنتج يحاول كتابة القيمة في الصوان، ثم تُستدعى الطريقة **(DisplayState)** المعرفة بدورها في السطور**(97-101)** والتي تظهر رسالة أخرى مفادها أن الصوان ممتلئ وعلى النيسب المنتج أن ينتظر إذا أراد وضع قيمة جديدة.

لاحظ أن طريقة الانتظار السطر **(75)** تقوم بوضع النيسب الذي كان يتحكم في الغرض **(SynchronizedBuffer)** في حالة انتظار/ضم/ سبات وهذا بدوره يحرر الغرض **(SynchronizedBuffer)** من القفل، كما تُعدل وضعية حالة قفل الغرض. وبعدها يمكن لنيسب آخر أن يتحكم في الغرض **(SynchronizedBuffer)**.

يبقى النيسب المنتج في حالة انتظار /ضم/ سبات حتى يفعل، وذلك عند استدعاء طريقة الإنعاش من قبل النيسب المستهلك. وبهذا يعود النيسب المنتج لحالة التشغيل وينفذ تعليمات السطور **(79-92)** حيث تسند أولاً القيمة المعطاة من قبل النيسب المنتج، ثم تُزاد قيمة المتغير **(occupiedBufferCount)** بمقدار واحد (لتشير بأن الصوان الآن يحوي قيمة جديدة وعندها يمكن للنيسب المستهلك قراءتها ولايمكن للنيسب المنتج أن يضع قيمة جديدة فيها)، بعد ذلك تُستدعى الطريقة **(DisplayState)** والتي تُظهر رسالة تشير إلى أن النيسب المنتج يقوم بكتابة قيمة جديدة في الصوان، ثم تُستدعى طريقة الإنعاش التي تعيد النيسب المنتظر (إن وجد) إلى حالة التشغيل ليكمل مهمته.

وأخيراً، تُستدعى طريقة الخروج **(Exit)** والتي تحرر الغرض **(SynchronizedBuffer)** من القفل وتجعله متاحاً من جديد.

لاحظ أنه من الأخطاء البرمجية الشائعة عدم تحرير الغرض من القفل عند انتهاء تعامل النيسب معه، إذ أن ذلك سيمنع النياسب الأخرى من طلب قفل على نفس الغرض، وبالتالي فلا يمكنها إكمال مهمتها وستُجبر على الانتظار، ومن الممكن أن يؤدي هذا الانتظار إلى حالة **(DeadLock)** وتأجيل إلى اللانهاية.

أما بالنسبة للنيسب المستهلك، فسينفذ المهمة الموجودة في الطريقة **(Consume)** التابعة للصف **(Consumer)**، حيث سيأخذ القيم من الغرض **(SynchronizedBuffer)** ويقوم بمعالجتها.

تقوم بدايةً التعليمة في السطر **(30)** الموجودة في الصف المستهلك والتي تقرأ قيمة المتغير من النوع **(SynchronizedBuffer)** باستدعاء الطريقة **(get)** في السطور **(18-61)** من الصف **(SynchronizedBuffer)**.

حيث يبدأ التنفيذ باستدعاء طريقة الدخول **(Enter)** التي تطلب قفل الغرض **(SynchronizedBuffer)**، يلي ذلك عبارة الشرط **(if)** التي تتحقق من قيمة المتغير **(occupiedBufferCount)** فإذا ساوت الصفر فسوف تُنفذ السطور **(27-31)** حيث تظهر رسالة تشير إلى أن النيسب المستهلك يحاول قراءة القيمة من الصوان.

وبعدها تظهر رسالة أخرى ناتجة عن الطريقة **(DisplayState)** تشير إلى أن الصوان فارغ وعلى النيسب المستهلك الانتظار إذا أراد قراءة قيمة جديدة، ومن ثم يدخل حالة الانتظار */ضم/* سبات، وبعد الخروج من الشرط سوف تنقص قيمة المتغير **(occupiedBufferCount)** بمقدار واحد ليشير أن الصوان فارغ (أي لا يمكن للمستهلك أن يقرأ قيمة جديدة ولكن يمكن للمنتج أن يكتب قيمة جديدة)، ثم تظهر رسالة تشير إلى أن النيسب المستهلك يقوم بقراءة القيمة الموجودة في الصوان، ثم تُستدعى طريقة الإنعاش ليعاد النسيب المنتج إلى حالة التشغيل ويتمكن من إتمام مهمته.

أما السطر **(55)** فيأخذ نسخه من الصوان قبل أن يحرر الغرض من القفل، وهذا ضروري لأنه من الممكن أن يضع المنتج قيمة جديدة في الصوان مباشرةً بعد تحرير الغرض **(SynchronizedBuffer)** وذلك بعد استدعاء طريقة الخروج **(Exit)**، وفي النهاية تُعاد نسخة الصوان للجزء الذي استدعى التغيير السابق.

لننتقل الآن إلى الصف **(SynchronizedBufferTest)** المشابه للصف **(UnSynchronizedBufferTest)** من المثال السابق، والذي يحوي الطريقة الأساسية **(Main)** المسؤولة عن تنفيذ البرنامج والربط بين الصفوف السابقة.

إذا تأملنا الخرج الناتج عن تنفيذ البرنامج لمرتين مختلفتين، نلاحظ أن كل عدد قد أُنتج وعولج لمرة واحدة فقط. وذلك لأنه لا يمكن لكل من المنتج والمستهلك أن ينفذ مهمته إلا حينما يحين دوره. حيث لا ينتج المنتج حتى يستهلك المستهلك والعكس صحيح. حاول تنفيذ هذا البرنامج لعدة مرات للتأكد من أن كل ما ينتج يستهلك لمرة واحدة، ولاحظ السطور التي تشير أن كلاً من المنتج والمستهلك يجب عليها انتظار دورهما لينفذا مهمتهما.

ولدينا شكل يبين تنفيذ للبرنامج:

```
Operation                          Buffer   Occupied Count

Initial state                      -1       0

Consumer tries to read.
Buffer empty. Consumer waits.      -1       0

Producer writes 1                  1        1

Consumer reads 1                   1        0

Consumer tries to read.
Buffer empty. Consumer waits.      1        0

Producer writes 2                  2        1

Consumer reads 2                   2        0

Consumer tries to read.
Buffer empty. Consumer waits.      2        0

Producer writes 3                  3        1

Consumer reads 3                   3        0

Consumer tries to read.
Buffer empty. Consumer waits.      3        0

Producer writes 4                  4        1

Consumer reads 4                   4        0

Producer writes 5                  5        1

Consumer reads 5                   5        0

Producer writes 6                  6        1

Consumer reads 6                   6        0

Producer writes 7                  7        1

Consumer reads 7                   7        0

Producer writes 8                  8        1

Consumer reads 8                   8        0

Producer writes 9                  9        1

Consumer reads 9                   9        0

Consumer tries to read.
Buffer empty. Consumer waits.      9        0

Producer writes 10                 10       1

Consumer reads 10                  10       0

Consumer read values totaling: 55.
Terminating Consumer.
```

وشكل آخر يبين تنفيذ ثاني للبرنامج:

```
Operation                          Buffer   Occupied Count

Initial state                      -1       0

Producer writes 1                  1        1

Consumer reads 1                   1        0

Producer writes 2                  2        1

Consumer reads 2                   2        0

Producer writes 3                  3        1

Consumer reads 3                   3        0

Producer writes 4                  4        1

Consumer reads 4                   4        0

Consumer tries to read.
Buffer empty. Consumer waits.      4        0

Producer writes 5                  5        1

Consumer reads 5                   5        0

Producer writes 6                  6        1

Consumer reads 6                   6        0

Producer writes 7                  7        1

Consumer reads 7                   7        0

Producer writes 8                  8        1

Producer tries to write.
Buffer full. Producer waits.       8        1

Consumer reads 8                   8        0

Producer writes 9                  9        1

Producer tries to write.
Buffer full. Producer waits.       9        1

Consumer reads 9                   9        0

Producer writes 10                 10       1

Producer done producing.
Terminating Producer.
Consumer reads 10                  10       0

Consumer read values totaling: 55.
Terminating Consumer.
Press any key to continue . . .
```

8. علاقة المنتج/المستهلك: الصوان الدائري

Producer/Consumer Relationship: Circular Buffer

تم في المثال السابق استخدام مزامنة النياسب لضمان أن النيسبين الذين يتعاملان مع بيانات مشتركة في الصوان يتعاملان معها بشكل صحيح. إلا أن الأسلوب السابق المستخدم لا يعطي التطبيق الحل الأمثل بشكل عام. لأنه من الممكن أن ينفذ النيسبان مهامهما بسرع متفاوتة، وبهذا يقضي أحد النياسب معظم وقته منتظراً.

ففي البرنامج السابق مثلاً، تمت مشاركة رقم واحد صحيح من قبل نيسبان، فإذا أنتج النيسب المنتج القيم بسرعة أكبر من سرعة تنفيذ النيسب المستهلك، سينتظر النيسب المنتج لعدم وجود مكان آخر في الذاكرة توضع فيه القيم الجديدة.

وبالمثل، فإذا استهلك المستهلك أسرع من إنتاج المنتج للقيم، سينتظر المستهلك حتى يضع المنتج القيم الجديدة في الصوان. وحتى إذا وجدت نياسب تعالج في سرع متقاربة، فمن الممكن جداً أن تُصبح هذه النياسب في وقت معين خارج المزامنة، مما يسبب انتظار أحد النياسب الآخر.
وعندما تنتظر النياسب يصبح البرنامج وتفاعل البرنامج مع المستخدم أقل فاعلية، كما ستعاني الشبكة وتطبيقاتها من تأخير كبير، حيث أن المعالج لايعمل بشكل كامل.
وللتقليل من انتظار النياسب التي تتشارك المصادر لبعضها البعض لوقت طويل، ولنجعلها تُنفذ في سرع متساوية، يمكن استخدام:

الصوان الدائري أو الدوار (Circular Buffer)

والذي يعطي مراجع وأماكن إضافية في الذاكرة حيث يمكن للمنتج أن يضع فيها القيم التي ينتجها، أو يأخذ المستهلك منها القيم التي يريد معالجتها.

ولتقريب الفكرة، لنعتبر أن الصوان متضمن في مصفوفة حيث يبدأ عمل كل من المنتج

والمستهلك من بداية المصفوفة حتى الوصول إلى نهاية المصفوفة ثم تعاود من العنصر الأول في المصفوفة. وبهذه الطريقة، فإذا أنتج المنتج القيم بسرعة أكبر من السرعة التي يستهلك فيها المستهلك، لن يضطر المنتج أن ينتظر حتى ينهي المستهلك قراءة القيمة السابقة، حيث يمكنه كتابة قيمة جديدة في الصوان الإضافي (إذا وجدت خلايا وإلا سيضطر أن ينتظر المنتج في هذه الحالة).

يمكّن هذا الأسلوب النيسب المنتج من إكمال مهمته حتى إذا لم يكن المستهلك جاهزاً لاستقبال القيمة الجديدة التي تم إنتاجها.

وبالمثل، فإذا كان المستهلك يعالج القيم بسرعة أكبر من إنتاج المنتج لهذه القيم، فيمكن أن يقرأ المستهلك قيم إضافية من الصوان (إذا وجدت وإلا سيضطر المستهلك أن ينتظر في هذه الحالة)، مما يمكّن المستهلك من إكمال مهمته حتى إذا لم يكن المنتج جاهزاً لإنتاج قيم جديدة.
لاحظ أن الصوان الدائري سيكون غير ملائماً إذا كان المنتج والمستهلك يقومان بالتنفيذ بسرعات مختلفة. فإذا كان المستهلك يعمل دائماً بسرعة أكبر من المنتج، سيكون الصوان مع مكان تخزين واحد كافي حيث سيسبب الموضع الإضافي ضياعاً في الذاكرة بلا فائدة. أما إذا كان المنتج ينفذ بسرعة أكبر سيكون من الواجب وجود صوان بعدد لا نهائي من الأماكن ليخزن الإنتاج الزائد.

ومن الأمور الأساسية عند استخدام الصوان الدائري تحديد عدد الأماكن التي ستحوي الإنتاج الإضافي (أي طول المصفوفة في مثالنا) للتقليل من أزمنة الانتظار لأقصر حد ممكن.

نفترض في مثالنا أن المنتج ينتج بقدر ثلاث مرات مما يعالج المستهلك على الأكثر، وبهذا يمكننا أن نُعرف صوان بثلاثة خلايا على الأقل حتى تحوي الإنتاج الزائد. حيث أن هدفنا أن لا يكون الصوان صغيراً جداً لأن هذا قد يسبب انتظار النياسب أكثر، وفي الجهة المقابلة أن لا يكون كبيراً جداً لأن هذا يؤدي إلى ضياع الذاكرة.

يمكن للصفوف المنتجة والمستهلكة أن تتعامل مع الصوان الدائري (وفي مثالنا سنعتبره مصفوفة من ثلاثة عناصر) مع المزامنة بينها في علاقة المنتج/المستهلك، حيث سيستهلك المستهلك فقط عندما تكون المصفوفة غير فارغة، وينتج المنتج عندما تكون المصفوفة غير ممتلئة تماماً.

نستخدم في المثال التالي الواجهة **(Buffer)** والصفوف المنتجة والمستهلكة التي تعاملنا معها في المثالين السابقين. وكذلك التعليمات التي كونت أغراض النياسب في الطريقة الأساسية **(Main)** الموجودة في الصف **(UnSynchronizedBufferTest)** والصف **(SynchronizedBufferTest)** من الأمثلة السابقة.

تظهر التغييرات الأساسية عن المثالين السابقيين في الصف **(CircularBuffer)**. يحوي هذا الصف أربع متغيرات:

1. المصفوفة **(buffers)** **(السطر15)** وهي مصفوفة ذات بعد واحد وثلاثة عناصر تُشكّل الصوان الدائري.
2. المتغير **(occupiedBufferCount)** وهو عبارة عن عداد للعناصر المولدة في الصوان والتي لم تُقرأ بعد. يعبر هذا المتحول عملياً عن حالة كل من المنتج والمستهلك، أي فيما إذا كان المنتج قادراً على الكتابة في الصوان الدائري (في هذه الحالة يجب أن تكون قيمة **(occupiedBufferCount)** أقل من عدد عناصر المصفوفة)، وفيما إذا كان المستهلك قادراً على القراءة منه (وهنا يجب أن تكون قيمته أكبر من الصفر).
3. المتغير **(readLocation)** يحدد الموقع الذي سيقوم المستهلك بقراءة القيمة التالية منه.
4. المتغير **(writeLocation)** يحدد الموقع الذي سيقوم المنتج بكتابة القيمة التالية فيه.

يعالج شرط الإسناد **(set)** للخاصية **(Buffer)** السطور **(58-92)** نفس المهام التي عالجها في المثال السابق مع بعض التحسينات، فبدلاً من أن يستخدم الصف **(Monitor)** وطريقة الدخول **(Enter)** والخروج **(Exit)** لطلب وتحرير القفل على الغرض **(CircularBuffer)**، استخدمنا الطريقة **(Lock)** لقفل الغرض طوال فترة تنفيذ كتلة تعليمات. عندما ينفذ البرنامج هذه الكتلة يطلب النيسب المنفذ القفل على الغرض **(CircularBuffer)**، وعند انتهاء تنفيذ كتلة التعليمات سوف يحرر الغرض من القفل مباشرة.

لاحظ أنه إذا استخدمت طرائق الصف **(Monitor)** المتمثلة بطريقة الدخول **(Enter)** والخروج **(Exit)** للإدارة والسيطرة على قفل الغرض، وظهر استثناء أو خطأ قبل أن يصل البرنامج للتعليمة التي تستدعي طريقة الخروج ولم يتم مسك هذا الاستثناء وإظهاره لمعالجته، سوف تنتهي الطريقة قبل استدعاء الخروج، مما يعني أن الغرض لم يتحرر من القفل وبالتالي لا يمكن للنياسب الأخرى أن تتعامل معه.

ولتجنب هذا الخطأ الشائع يجب وضع التعليمات التي تتوقع أن ينتج منها استثناء بكتلة مسبوقة بالكلمة المفتاحية **(try)** ووضع التعليمة التي تستدعي الخروج في كتلة مسبوقة بالكلمة **(Finaly)** للتأكد من تحرير القفل.

عند استخدام الطريقة **(Lock)** لتنظيم قفل على غرض في المزامنة، يستبعد احتمال نسيان تحرير القفل باستدعاء طريقة الخروج، حيث تستدعي **C#** عملياً طريقة الخروج عندما تنتهي الكتلة **(Lock)** لأي سبب كان، وبهذا سيتحرر القفل حتى عند ظهور استثناء أو خطأ.
يختبر الشرط **(If)** في السطور**(66-71)** فيما إذا كان المنتج يجب أن ينتظر (عندما يكون الصوان ممتلئ)، وفي هذه الحالة تظهر رسالة تُعلم بأن المنتج ينتظر، ومن ثم تُستدعى طريقة الانتظار ليصبح النيسب في حالة انتظار /ضم/ سبات.

بعد إنعاش النيسب توضع القيمة الناتجة عن المنتج في الصوان الدائري في الموضع

(writeLocation)، ويقوم السطر **(81)** بإظهار رسالة تحوي القيمة المنتجة ثم تُزاد قيمة المتغير **(occupiedBufferCount)** لأنه يوجد الآن على الأقل قيمة واحدة في الصوان يمكن للمستهلك أن يقرأها، بعدها يحدث السطر **(85)** المتغير **(writeLocation)** للاستدعاء التالي لشرط الإسناد **(set)** للخاصية **(Buffer)**.

تقوم التعليمة في السطر **(86)** باستدعاء الطريقة **(CreateStateOutput)** المعرفة في

السطور**(96-135)** والتي تُظهر القيمة التي خُزنت في الصوان ومحتوى الصوان وقيمة المتغيرين **(writeLocation)** و**(readLocation)**. يستدعي السطر**(95)** طريقة الإنعاش حتى تُغير حالة النيسب الذي ينتظر الغرض **(CircularBuffer)** – إن وجد – إلى حالة التشغيل.

لاحظ أن الوصول إلى القوس الأيمن **{** من كتلة **(Lock)** في السطر **(91)** ستُسبب تحرير الغرض **(CircularBuffer)** من القفل.

```
1  // CircularBuffer.cs
2 // A circular shared buffer for the producer/consumer relationship.
3  using System;
4  using System.Threading;
5
6  // implement the an array of shared integers with synchronization
7  public class CircularBuffer : Buffer
8  {
9     // each array element is a buffer
10    private int[] buffers = { -1, -1, -1 };
11
12    // occupiedBufferCount maintains count of occupied buffers
13    private int occupiedBufferCount = 0;
14
15    private int readLocation = 0; // location of the next read
16    private int writeLocation = 0; // location of the next write
17
18    // property Buffer
19    public int Buffer
20    {
```

```
21       get
22       {
23          // lock this object while getting value
24          // from buffers array
25          lock ( this )
26          {
27             // if there is no data to read, place invoking
28             // thread in WaitSleepJoin state
29             if ( occupiedBufferCount == 0 )
30             {
31             Console.Write( "\nAll buffers empty. {0} waits.",
32               Thread.CurrentThread.Name );
33             Monitor.Wait( this ); // enter the WaitSleepJoin state
34             } // end if
35
36             // obtain value at current readLocation
37             int readValue = buffers[ readLocation ];
38
39             Console.Write( "\n{0} reads {1} ",
40              Thread.CurrentThread.Name, buffers[ readLocation ] );
41
42             // just consumed a value, so decrement number of
43             // occupied buffers
44             --occupiedBufferCount;
45
46             // update readLocation for future read operation,
47             // then add current state to output
48             readLocation = ( readLocation + 1 ) % buffers.Length;
49             Console.Write( CreateStateOutput() );
50
51             // return waiting thread (if there is one)
52             // to Running state
53             Monitor.Pulse( this );
54
55             return readValue;
56          } // end lock
57       } // end get
58       set
59       {
60          // lock this object while setting value
61          // in buffers array
62          lock ( this )
63          {
64             // if there are no empty locations, place invoking
65             // thread in WaitSleepJoin state
66            if ( occupiedBufferCount == buffers.Length )
67             {
68             Console.Write( "\nAll buffers full. {0} waits."
69                    Thread.CurrentThread.Name );
70             Monitor.Wait( this ); // enter the WaitSleepJoin state
71              } // end if
72
73              // place value in writeLocation of buffers
74              buffers[ writeLocation ] = value;
75
76             Console.Write( "\n{0} writes {1} ",
77             Thread.CurrentThread.Name, buffers[ writeLocation ] );
78
79              // just produced a value, so increment number of
```

```
80             // occupied buffers
81              ++occupiedBufferCount;
82
83              // update writeLocation for future write operation,
84             // then add current state to output
85            writeLocation = ( writeLocation + 1 ) % buffers.Length;
86            Console.Write( CreateStateOutput() );
87
88              // return waiting thread (if there is one)
89              // to Running state
90             Monitor.Pulse( this );
91          } // end lock
92       } // end set
93    } // end property Buffer
94
95   // create state output
96    public string CreateStateOutput()
97    {
98       // display first line of state information
99       string output = "(buffers occupied: " +
100         occupiedBufferCount + ")\nbuffers: ";
101
102     for ( int i = 0; i < buffers.Length; i++ )
103   output += " " + string.Format( "{0,2}", buffers[ i ] ) + "  ";
104
105     output += "\n";
106
107     // display second line of state information
108     output += "         ";
109
110     for ( int i = 0; i < buffers.Length; i++ )
111        output += "---- ";
112
113     output += "\n";
114
115     // display third line of state information
116     output += "         ";
117
118     // display readLocation (R) and writeLocation (W)
119     // indicators below appropriate buffer locations
120     for ( int i = 0; i < buffers.Length; i++ )
121     {
122        if ( i == writeLocation &&
123           writeLocation == readLocation )
124           output += " WR  ";
125        else if ( i == writeLocation )
126           output += " W   ";
127        else if  ( i == readLocation )
128           output += "  R  ";
129        else
130           output += "     ";
131     } // end for
132
133     output += "\n";
134     return output;
135  } // end method CreateStateOutput
136} // end class HoldIntegerSynchronized
```

أما بالنسبة لشرط **(get)** للخاصية **(Buffer)** في السطور **(21 - 57)**، فتُنفذ عمليات مشابهة للمثال السابق، ولكن مع بعض التعديلات الأساسية. حيث نستخدم مرة أخرى الطريقة **(Lock)** لطلب قفل وتحرير الغرض **(Circular Buffer)** بدلاً من استخدام الصف **(Monitor)** وطرائقه.

يختبر الشرط **(If)** في السطور **(29-34)** فيما إذا كان المستهلك يجب أن ينتظر (عدم وجود أي قيمة في الصوان)، وإذا تحقق هذا الشرط فستُنفذ التعليمة في السطور **(31-32)** لطباعة رسالة مفادها أن المستهلك ينتظر، يستدعي بعدها السطر **(33)** طريقة الانتظار لوضع النيسب المستهلك في حالة انتظار/ضم/سبات. بعد ذلك تُسند قيمة العنصر الموجود في الموضع **(readLocation)** في الصوان إلى المتغير **(readValue)**، أما السطور **(39-40)** فتُظهر القيمة المستهلكة (التي تعالج حالياً). يحدث السطر **(44)** قيمة المتغير **(occupiedBufferCount)** لأن الصوان يحوي الآن موضع يمكن للمنتج أن يضع فيه قيمة جديدة، ثم يحدث السطر **(48)** الموضع **(readLocation)** للاستدعاء التالي.

يستدعي السطر **(49)** الطريقة **(CreateStateOutput)** لتُظهر محتوى الصوان وقيمة كل من **(readLocation)** و**(writeLocation)**، ثم يستدعي السطر **(53)** تعليمة الإنعاش لتحول حالة النيسب التالي للتشغيل إذا وجد ليكمل مهمته، وأخيراً تُعاد القيمة التي عولجت من قبل النيسب المستهلك.

```
1  // CircularBufferTest.cs
2  // Implementing the producer/consumer relationship with a
3  // circular buffer.
4  using System;
5  using System.Threading;
6
7  class CircularBufferTest
8  {
9     // create producer and consumer threads and start them
10    static void Main( string[] args )
11    {
12       // create shared object used by threads
13       CircularBuffer shared = new CircularBuffer();
14
15       // Random object used by each thread
16       Random random = new Random();
17
18       // display shared state before producer
19       // and consumer threads begin execution
20       Console.Write( shared.CreateStateOutput() );
21
22       // create Producer and Consumer objects
23       Producer producer = new Producer( shared, random );
24       Consumer consumer = new Consumer( shared, random );
25
26       // create threads for producer and consumer and set
```

```
27         // delegates for each thread
28         Thread producerThread =
29            new Thread( new ThreadStart( producer.Produce ) );
30         producerThread.Name = "Producer";
31
32         Thread consumerThread =
33            new Thread( new ThreadStart( consumer.Consume ) );
34         consumerThread.Name = "Consumer";
35
36         // start each thread
37         producerThread.Start();
38         consumerThread.Start();
39      } // end Main
40 } // end class CircularBufferTest
```

تقوم الطريقة **CreateStateOutput** بطباعة حالة الصوان الدائري، تتضمن القيمة الحالية للمتغير **(occupiedBufferCount)** ومحتوى الصوان والقيمة الحالية لكل من **(readLocation)** و**(writeLocation)**، يمثل الحرف **(W)** في الخرج قيمة **(writeLocation)** أما الحرف **(R)** فيمثل قيمة **(readLocation)**.

يحوي الصف **(CircularBufferTest)** التالي الطريقة الأساسية في البرنامج **(Main)**. والتي تُعرف في البداية الغرض **(Shared)** من الصف **(CircularBuffer)** لكل من المنتج والمستهلك.

يبين الشكل تنفيذ للبرنامج:

```
(buffers occupied: 0)
buffers:  -1   -1   -1
          ---- ---- ----
           WR

All buffers empty. Consumer waits.
Producer writes 1 (buffers occupied: 1)
buffers:   1   -1   -1
          ---- ---- ----
             R   W

Consumer reads 1 (buffers occupied: 0)
buffers:   1   -1   -1
          ---- ---- ----
                WR

All buffers empty. Consumer waits.
Producer writes 2 (buffers occupied: 1)
buffers:   1    2   -1
          ---- ---- ----
                R   W

Consumer reads 2 (buffers occupied: 0)
buffers:   1    2   -1
          ---- ---- ----
                     WR

All buffers empty. Consumer waits.
Producer writes 3 (buffers occupied: 1)
buffers:   1    2    3
          ---- ---- ----
           W           R

Consumer reads 3 (buffers occupied: 0)
buffers:   1    2    3
          ---- ---- ----
           WR

Producer writes 4 (buffers occupied: 1)
buffers:   4    2    3
          ---- ---- ----
             R   W

Consumer reads 4 (buffers occupied: 0)
buffers:   4    2    3
          ---- ---- ----
                WR

Producer writes 5 (buffers occupied: 1)
buffers:   4    5    3
          ---- ---- ----
                R   W

Consumer reads 5 (buffers occupied: 0)
buffers:   4    5    3
```

```
Producer writes 6 (buffers occupied: 1)
buffers:   4    5    6
          ---- ---- ----
           W         R

Producer writes 7 (buffers occupied: 2)
buffers:   7    5    6
          ---- ---- ----
                W    R

Consumer reads 6 (buffers occupied: 1)
buffers:   7    5    6
          ---- ---- ----
             R  W

Producer writes 8 (buffers occupied: 2)
buffers:   7    8    6
          ---- ---- ----
             R       W

Consumer reads 7 (buffers occupied: 1)
buffers:   7    8    6
          ---- ---- ----
                R  W

Producer writes 9 (buffers occupied: 2)
buffers:   7    8    9
          ---- ---- ----
           W    R

Consumer reads 8 (buffers occupied: 1)
buffers:   7    8    9
          ---- ---- ----
           W         R

Producer writes 10 (buffers occupied: 2)
buffers:  10    8    9
          ---- ---- ----
                W    R
Producer done producing.
Terminating Producer.

Consumer reads 9 (buffers occupied: 1)
buffers:  10    8    9
          ---- ---- ----
             R  W

Consumer reads 10 (buffers occupied: 0)
buffers:  10    8    9
          ---- ---- ----
               WR
Consumer read values totaling: 55.
Terminating Consumer.
Press any key to continue . . .
```

9. تعددية النياسب في واجهات المستخدم التخاطبية (GUIs)

تمنع طبيعة البرامج المتعددة النياسب عملياً من معرفة متى سيتم تنفيذ النيسب فعلياً، كما لايمكن بسهولة متابعة التنفيذ والتغيرات التي تطرأ على النياسب.

ومن المهم معرفته، أن مكونات واجهات **Windows** ليست آمنة للنياسب **(thread Safety)**، فإذا ما تم استخدام النياسب للتحكم في مكونات واجهات **Windows**، فيمكن أن لا تكون النتائج صحيحة.

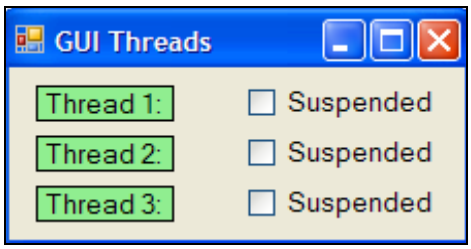
ولكي نتجنب الحصول على نتائج غير صحيحة أو غير دقيقة عند استخدام النياسب للتعامل مع مكونات الواجهات التفاعلية، لابد من استخدام نيسب واجهات المستخدم **(UI thread)**. يعمل هذا النيسب الخاص على صيانة واجهات المستخدم التخاطبية التي تتعامل مع النياسب.

يحوي الصف **(Control)** (وهو الصف الأب لكل عناصر الواجهات مثل **TextBox, Label, ... ,CheckBox**) الطريقة **(Invoke)** التي تقوم بتعريف نيسب واجهات المستخدم والذي تكون مهمته تنفيذ مجموعة من التعليمات على عنصر تحكم محدد.
يتم استدعاء الطريقة **(Invoke)** على عنصر تحكم مع تمرير معاملي دخل لهذه الطريقة هما مفوض ومصفوفة من الأغراض. يحوي المفوض التعليمات التي تقوم بتحديث عنصر التحكم.

عندما تُستدعى الطريقة **(Invoke)**، سينفذ نيسب واجهات المستخدم تعليمات المفوض بعد تمرير عناصر المصفوفة كدخل له.

نستخدم في المثال التالي نياسب منفصلة لتعديل المحتوى الذي سيعرض في واجهات المستخدم التخاطبية، سيوضح هذا المثال كيف يمكن استخدام مزامنة النياسب لإيقاف نيسب مؤقتاً (منعه من إكمال عمله لفترة قصيرة)، وكيف يعاد النيسب الذي وضع في الإيقاف المؤقت إلى العمل.

يحوي النموذج في مثالنا ثلاثة تسميات **(Label)** وثلاثة صناديق تحقق **(CheckBox)**، عند النقر على صندوق تحقق محاذي لاسم نيسب معين يتم إيقاف النيسب مؤقتاً أو إعادته للتشغيل من جديد.



لنستعرض أولاً الصف **(RandomLetters)** والذي يحوي الطريقة **(GenerateRandomCharacters)** التي ليس لها دخل أو خرج **السطور (33-58)**. تُستخدم خاصية الغرض نيسب **(CurrentThread)** في **السطر (36)** لتحديد النيسب الذي ينفذ حالياً، ومن ثم تُستخدم خاصية اسم النيسب **(Name)** للحصول على اسم هذا النيسب وإسناده إلى المتغير **(threadName)**. (نقوم بإعطاء اسم لكل نيسب في البرنامج الأساسي **(Main)**).

نُشكّل في السطور **(38-57)** حلقة لانهائية، رغم أنه من المعروف أن الحلقات اللانهائية تُعتبر سيئة برمجياً لأن البرنامج الذي يحويها لن ينتهي أبداً. ولكن في حالتنا هذه، توجد الحلقة اللانهائية في نياسب منفصلة عن النيسب الأصلي، وعندما يغلق التطبيق ستُغلق وتنتهي كل النياسب التي كونت عن طريق النيسب الأصلي ومنها النيسب الذي ينفذ الحلقة اللانهائية.

لاحظ أن في كل مرة أو دورة من الحلقة سوف ينام النيسب لوقت اختير عشوائياً ضمن المجال من **1** إلى **5** ثانية (السطر**41**). وعندما يعود النيسب للعمل ويخرج من حالة السبات، يقفل الغرض **(RandomLetters)** في السطر **(43)** حتى نتمكن من أن نُحدد فيما إذا كان النيسب قد أوقف مؤقتاً (عندما يضغط المستخدم على صندوق التحقق المقابل للنيسب)، عندما يبقى المتغير المنطقي **(suspend)** في الحلقة مساوي لـ **(True)**، تُستدعى طريقة الانتظار حتى يتحرر الغرض **(RandomLetters)** مؤقتاً ويوضع النيسب الذي كان يملك القفل في حالة انتظار/ضم/سبات.

وعندما ينعش هذا النيسب (عند ضغط المستخدم لصندوق التحقق المقابل له) يعود إلى حالة التشغيل. إذا كانت قيمة المتغير المنطقي **(suspend)** مساوي لـ **(True)** سوف يستمر تنفيذ الحلقة ويعاد وضع النيسب إلى حالة انتظار/ضم/سبات.

```
1  /// RandomLetters.cs
2  /// Writes a random letter to a label
3  using System;
4  using System.Windows.Forms;
5  using System.Drawing;
6  using System.Threading;
7
8  public class RandomLetters
9   {
10     private static Random generator=new Random();//random letters
11     private bool suspended = false; // true if thread is suspended
12     private Label output; // Label to display output
13     private string threadName; // name of the current thread
14
15     // RandomLetters constructor
16     public RandomLetters( Label label )
17     {
18        output = label;
```

```
19      } // end
20
21      // delegate that allows method DisplayCharacter to be called
22      // in the thread that creates and maintains the GUI
23      private delegate void DisplayDelegate( char displayChar );
24
25      // method DisplayCharacter sets the Label's Text property
26      private void DisplayCharacter( char displayChar )
27      {
28         // output character in Label
29         output.Text = threadName + ": " + displayChar;
30      } // end method DisplayCharacter
31
32      // place random characters in GUI
33      public void GenerateRandomCharacters()
34      {
35        // get name of executing thread
36         threadName = Thread.CurrentThread.Name;
37
38        while (true) //infinite loop;will be terminated from outside
39         {
40            // sleep for up to 1 second
41           Thread.Sleep( generator.Next( 1001 ) );
42
43            lock ( this ) // obtain lock
44            {
45              while ( suspended ) // loop until not suspended
46               {
47                  Monitor.Wait( this ); // suspend thread execution
48               } // end while
49            } // end lock
50
51            // select random uppercase letter
52          char displayChar = ( char ) ( generator.Next( 26 ) + 65 );
53
54           // display character on corresponding Label
55            output.Invoke( new DisplayDelegate( DisplayCharacter ),
56               new object[] { displayChar } );
57         } // end while
58      } // end method GenerateRandomCharacters
59
60      // change the suspended/running state
61     public void Toggle()
62     {
63        suspended = !suspended; // toggle bool controlling state
64
65        // change label color on suspend/resume
66        output.BackColor = suspended ? Color.Red : Color.LightGreen;
67
68        lock ( this ) // obtain lock
69         {
70            if ( !suspended ) // if thread resumed
71               Monitor.Pulse( this );
72         } // end lock
73      } // end method Toggle
74   } // end class RandomLetters
```

تولد التعليمة في السطر **(52)** حرف عشوائي، ثم تستدعى الطريقة **(Invoke)** بعد تمرير الدخل الممثل بالمفوض **(DisplayDelegate)** الذي يحوي الطريقة **(DisplayCharacter)**، ومصفوفة تحوي الحرف الذي تم توليده. حيث تم تعريف المفوض **(DisplayDelegate)** في السطر**(23)** على شكل طريقة بدون دخل أو خرج. أما الطريقة **(DisplayCharacter)** فقد تم تعريفها في السطور **(30-26)** والتي تستقبل الدخل الممثل بالمتغير **(displayChar)** لكن لا تُعيد خرج.

أما السطور **(56-55)** فتقوم باستدعاء الطريقة **(Invoke)** وهذا سيدفع نيسب واجهات المستخدم لاستدعاء الطريقة **(DisplayCharacter)** مع الدخل الممثل بالحرف الذي سبق توليده عشوائياً. لاحظ أنه عندما ينقر المستخدم على صندوق التحقق فسوف يسبب هذا إيقاف مؤقت للنيسب المقابل أو يعاد إلى حالة التشغيل (حسب الحالة السابقة لاختيار الصندوق).

تُعرف السطور **(73-61)** الطريقة **(Toogle)** التي تغير حالة النيسب الحالي بين حالتي الإيقاف المؤقت وإعادة التشغيل، حيث يستقبل السطر **(63)** قيمة المتغير المنطقي **(suspended)**، ثم في السطر **(66)** يتم تغيير لون التسمية للون الأحمر **(Color.Red)** إذا كان النيسب قد أوقف مؤقتاً، وللون الأخضر **(Color.LightGreen)** إذا كان النيسب قد أُعيد لحالة التشغيل، ثم يقوم السطر **(68)** بقفل الغرض **(RandomLetters)** ومن ثم إنعاش النيسب إذا كان صندوق التحقق الموافق محدداً.

يحوي الصف **(GUIThreads)** التالي ثلاث تسميات وثلاثة صناديق تحقق، وثلاث نياسب منفصلة يرتبط كل منها بزوج مكون من تسمية وصندوق تحقق. يقوم كل نيسب بعرض و بشكل عشوائي أحرف في التسمية الموافقة له.

تُعرف السطور **(35،28،21)** ثلاثة أغراض من الصف **(RandomLetters)**، أما السطور **(22-23)** و **(30-29)** و **(37-36)** فتُشكّل ثلاثة نياسب مهمة كل منها تنفيذ الطريقة **(GenerateRandomCharacters)**. تُمنح النياسب

أسماء في السطور **(38،31،24)**، وأخيراً يبدأ عمل النياسب بعد تنفيذ السطور **(39،32،25)**.

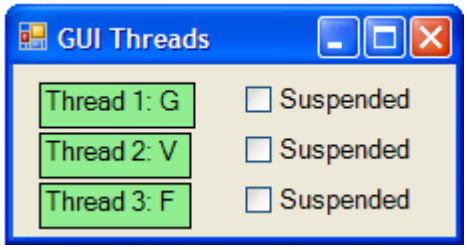
```
1  // GUIThreads.cs
2  // Demonstrates using threads in a GUI
3  using System;
4  using System.Windows.Forms;
5  using System.Threading;
6
7  public partial class GUIThreadsForm : Form
8  {
9     public GUIThreadsForm()
10    {
11       InitializeComponent();
```

```
12    } // end constructor
13
14    private RandomLetters letter1; // first RandomLetters object
15    private RandomLetters letter2; // second RandomLetters object
16    private RandomLetters letter3; // third RandomLetters object
17
18    private void GUIThreadsForm_Load( object sender, EventArgs e )
19    {
20       // create first thread
21       letter1 = new RandomLetters( thread1Label );
22       Thread firstThread = new Thread(
23          new ThreadStart( letter1.GenerateRandomCharacters ) );
24       firstThread.Name = "Thread 1";
25       firstThread.Start();
26
27       // create second thread
28       letter2 = new RandomLetters( thread2Label );
29       Thread secondThread = new Thread(
30          new ThreadStart( letter2.GenerateRandomCharacters ) );
31       secondThread.Name = "Thread 2";
32       secondThread.Start();
33
34       // create third thread
35       letter3 = new RandomLetters( thread3Label );
36       Thread thirdThread = new Thread(
37          new ThreadStart( letter3.GenerateRandomCharacters ) );
38       thirdThread.Name = "Thread 3";
39       thirdThread.Start();
40    } // end method GUIThreadsForm_Load
41
42    // close all threads associated with this application
43    private void GUIThreadsForm_FormClosing( object sender,
44       FormClosingEventArgs e )
45    {
46      System.Environment.Exit( System.Environment.ExitCode );
47    } // end method GUIThreadsForm_FormClosing
48
49    // suspend or resume the corresponding thread
50    private void threadCheckBox_CheckedChanged( object sender,
51       EventArgs e )
52    {
53       if ( sender == thread1CheckBox )
54          letter1.Toggle();
55       else if ( sender == thread2CheckBox )
56          letter2.Toggle();
57       else if ( sender == thread3CheckBox )
58          letter3.Toggle();
59    } // end method threadCheckBox_CheckedChanged
60 } // end class GUIThreadsForm
```

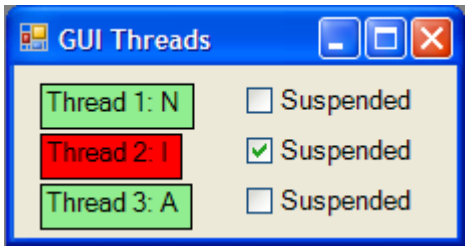
فإذا نقر المستخدم على أحد صناديق التحقق سوف تُنفذ الطريقة **(threadCheckBox_CheckedChanged)** والتي تُحدد أولاً أي من الصناديق قد استدعى الطريقة، ثم يتم استدعاء الطريقة **(Toggle)** لتحديد النيسب المقابل لصندوق التحقق ويتم إيقاف النيسب أو إعادته للتشغيل.

تُعرف السطور **(47-43)** الطريقة **(GUIThreadsForm_FormClosing)** التي تقوم باستدعاء طريقة الخروج **(Exit)** من الصف **(System.Environment)** مع الخاصية **(ExitCode)**، تقوم هذه الطريقة بإنهاء جميع النياسب الموجودة في هذا التطبيق عند الخروج منه، وبدون هذه الطريقة سينتهي نيسب واجهات المستخدم فقط وتنفذ النياسب الأخرى للأبد.

يبين الشكل التالي تنفيذ البرنامج حيث سيستمر بشكل متتال توليد أحرف عشوائية في كل تسمية:



أما إذا نقر المستخدم على أحد صناديق التحقق فسيتم إيقاف توليد الأحرف العشوائية في التسمية الموافقة..



10. الخلاصة

عرضنا في هذا الفصل الأدوات الأساسية في بيئة العمل **(.NET)** التي تسمح بتكوين وتعريف المهمات المتزامنة في البرامج، و ناقشنا كيف يمكن تكوين نياسب متعددة باستخدام الصف **(Thread)** والمفوض **(ThreadStart)** الموجودين في المكتبة **(System.Threading)**.

ناقشنا تطبيقات عديدة للنياسب المتزامنة (العمليات التي تُنفذ في نفس الوقت)، وتعرفنا على المشاكل المحتمل ظهورها عند تحكم نياسب متعددة في بيانات مشتركة وكيفية معالجتها.

عرضنا بعد ذلك، الحالة الخاصة للواجهات التخاطبية حيث يجب أن تُعالج كل التغيرات في هذه الواجهات من قبل نيسب الواجهات التخاطبية. ووضحنا كيف نستخدم الطريقة **(Invoke)** من الصف **(Control)** لتعريف المهمات التي سينفذها نيسب الواجهات.

الفصل الثاني : برمجة التطبيقات الشبكية زبون / مخدم

1. مقدمة

يمنح التعامل مع شبكة الانترنت وعالم الويب الكثير من الإمكانات نظراً لما تُقدمه من خدمات متنوعة، إضافةً إلى سهولة الحصول على المعلومات من أي مكان في العالم. ولذا فقد أدركت الشركات أهمية تشارك البيانات على شبكة الانترنت لتحسين الأداء.

تُقدم المكتبة البرمجية **.NET FCL (Framework Class Library)** عدداً من الإجرائيات الجاهزة الخاصة بالتشبيك والتي تُسهل عملية تطوير تطبيقات الويب وتطبيقات الانترنت. وبحيث تستطيع البرامج باستخدام الويب البحث عن المعلومات في كل مكان من العالم، كما تستطيع مشاركة برامج أخرى تعمل على حواسب أخرى في أي منطقة بعيدة.

سنقوم في الفصل القادم بدراسة خدمات الويب **Web Services** المستخدمة في بناء التطبيقات الموزعة وذلك على المستوى العالي **high-level**.

أما في هذا الفصل، فسنبدأ بعرض تقانات الاتصالات المستخدمة لإرسال البيانات عن طريق شبكة الانترنت، ثم نقوم بعرض الطرق الأساسية لتأسيس اتصال بين تطبيقين يستخدمان قناتين لتدفق البيانات بطريقة شبيهة لطريقة كتابة وقراءة الملفات **file I/O**. يدعى هذا الاتصال بالاتصال المرتبط **Connection Oriented** والذي يمكّن برنامجين من الاتصال ونقل البيانات بينهما بطريقة مشابهة للكتابة والقراءة من الملفات على الأقراص الصلبة. ثم نقوم بعرض تطبيق محادثة **Chatting** يستخدم هذه التقانات ويقوم بتبادل رسائل بين الزبون والمخدم.

نقوم بعد ذلك، بعرض مثال عن تقانة الاتصال عديم الارتباط **Connectionless** لنقل البيانات بين التطبيقات. والتي هي تقانة أقل وثوقية من الاتصال المرتبط إلا أنها أخف حملاً على الشبكة. تُستخدم هذه التقانة عادةً في تطبيقات نقل الصوت والفيديو عبر الانترنت.

ثم نعرض مثال على برنامج (زبون/مخدم) (لعبة بين لاعبين على الشبكة) نُبين من خلاله طريقة بناء مخدم متعدد النياسب.

ندرس بعدها مكون التحكم الجديد **WebBrowser** والذي يسمح بإضافة آليات تصفح لأي تطبيق يستخدم الويب.

ينتهي هذا الفصل بعرض سريع عن تقانة التشبيك عن بعد **.NET Remoting** والتي تؤمن خدمات مشابهة لخدمات الويب المدروسة في الفصل القادم والتي تسمح بالمعالجة الموزعة عبر الشبكة.

2. مقارنة بين الاتصال المرتبط والاتصال عديم الارتباط

Connection-Oriented vs. Connectionless Communication

يوجد طريقتين رئيسيتين لتحقيق الاتصال بين التطبيقات هما:

1. الاتصال المرتبط.
2. الاتصال عديم الارتباط.

يشابه الاتصال المرتبط النظام الهاتفي، حيث يتم تأسيس الاتصال وإبقاءه مفتوحاً طالما هنالك تبادل للمعلومات بين الطرفين. أي يبقى الاتصال مفتوحاً حتى إنهاء الاتصال من قبل أحد الطرفين.

بينما يشابه الاتصال عديم الارتباط الخدمة البريدية، حيث إذا قُمنا بإرسال رسالتين من مصدر ما إلى مكان ما آخر، فيمكن فعلياً أن تأخذ هاتين الرسالتين مسارين مختلفين في هذا النظام، كما يمكن أن تصلا في أوقات مختلفة، كما يمكن أن لاتصل إحداهما أبداً.

يتم في الاتصال المرتبط إرسال معلومات تحكم بين الحواسب لتفعيل الاتصال بين الطرفين. تُسمى هذه التقانة بالمصافحة **Handshaking**.

تُعتبر شبكة الانترنت شبكة غير موثوقة، مما يعني أن المعلومات أو البيانات المرسلة عبر الانترنت يمكن أن تُفقد أو أن تتأذى خلال عملية النقل.

تُرسل البيانات عملياً في رزم **Packets** تحوي، إضافةً لمحتواها، مجموعة معلومات تساعد الانترنت في توجيه هذه الرزم لتصل إلى المكان المقصود والصحيح. لا تضمن الانترنت لأحد ماذا سيحدث للرزم المرسلة حيث يمكن أن تصل هذه الرزم معطوبة أو متأخرة أو فارغة أو مكررة كما يمكن أن لا تصل أبداً. يقتصر دور الانترنت على القيام بما في وسعها لإيصال الرزم المرسلة.

يضمن الاتصال المرتبط تراسل معطيات موثوق ضمن شبكة غير موثوقة، ويقوم بتحقيق ضمان وصول الرزم المرسلة إلى المستقبل صحيحة بدون أضرار، كما يقوم بتجميعها بالتسلسل الصحيح.

لا يقوم الحاسبان في الاتصال عديم الارتباط بالمصافحة قبل إرسال البيانات، ولا يضمن أحد وثوقية وصول هذه البيانات. يمكن للمعلومات أن لا تصل أبداً إلى المستقبل المقصود. وبهذا

النحو، يقوم الاتصال عديم الارتباط بتخفيف الحمل على الشبكة وذلك بتجنب إرسال معلومات التحكم في تقنية المصافحة لضمان الوثوقية. وبذا فسنحتاج لإرسال معلومات أقل بين الحواسب.

3. بروتوكولات نقل البيانات

يوجد العديد من البروتوكولات لتعريف كيفية الاتصال بين التطبيقات. يمكن تعريف البروتوكول بأنه مجموعة من القواعد التي تَحكُم كيفية التخاطب بين مكونين (يمكن أن تكون هذه المكونات عبارة عن حواسب). سنقوم في هذا الفصل بالتركيز على بروتوكول التحكم بالنقل **Transmission Control Protocol TCP** وبروتوكول برقية البيانات للمستخدم **User Datagram Protocol UDP**.

توفر **NET.** إمكانيات التعامل مع البروتوكولات **TCP** و **UDP** ضمن فضاء الأسماء **System.Net.Sockets**.

يعتبر البروتوكول **TCP** بروتوكول مرتبط لأنه:

> يضمن وصول رزم البيانات إلى الوجهة المقصودة بالتسلسل المرسل الصحيح وبدون أي فقدان أو ضياع في البيانات المرسلة.

يسمح البروتوكول **TCP** للبروتوكولات الأعلى مثل البروتوكول **HTTP** بإرسال المعلومات عبر شبكة الانترنت بطريقة مشابهه لكتابة الملفات على القرص الصلب للحاسب. يقوم البروتوكول **TCP** بإعادة إرسال رزم البيانات في حال ضياعها أو عدم وصولها إلى الوجهة المقصودة، كما يضمن تجميع هذه الرزم بالتسلسل الصحيح وبصورة واضحة للتطبيق المستلم، كما يقوم البروتوكول باستبعاد الرزم المكررة.

يستخدم الاتصال عديم الارتباط **UDP** من قبل التطبيقات التي لا تتطلب بالضرورة تبادل

معطيات موثوقة بين طرفي الاتصال. حيث يضمن البروتوكول **UDP** الحد الأدنى من نقل

البيانات الضروري لتأمين الاتصال بين التطبيقات. لا يقدم البروتوكول **UDP** أي ضمان بأن رزم البيانات، والتي تسمى برقيات البيانات **Datagrams**، ستصل إلى الوجهة المقصودة أو بأنها ستصل بالترتيب التي أُرسلت فيه.

يوجد العديد من الفوائد التي يمكن الحصول عليها عند استخدام البروتوكول **UDP** عوضاً عن البروتوكول **TCP**. فعند استخدام البروتوكول **UDP** تُشكّل عملية نقل البيانات بين طرفي

الاتصال حملاً أقل على الشبكة، إذ أن رزم البيانات لا تحتاج لأن تحمل معلومات التحكم التي يحتاجها البروتوكول **TCP** لضمان وثوقية وصول المعلومات.

كما أن البروتوكول **UDP** يقلل حركة البيانات على الشبكة نظراً لعدم اهتمامه بعملية المصافحة وإعادة إرسال البيانات في حال عدم وصولها.
يمكن استخدام الاتصالات غير الموثوقة في العديد من الحالات العملية وحيث تكون الوثوقية غير مطلوبة وحيث نرغب بالتخلص من الحمل الزائد الذي تفرضه البروتوكولات التي تضمن وثوقية وصول البيانات.

فمثلاً يمكن لبعض التطبيقات التي تستخدم نقل الصوت والفيديو أن تتغاضى عن فقدان بعض برقيات البيانات. يمكن بالطبع أن ينتج عن هذا الفقدان توقف مؤقت لعرض الصوت أو الفيديو. إلا أنه إذا كانت هذه التطبيقات تستخدم البروتوكول **TCP** فإن فقدان جزء من البيانات سيؤدي إلى توقف كبير وملحوظ حيث أن التطبيق سينتظر حتى يتم إعادة إرسال الجزء المفقود عند النقل واستقباله كي يتم مواصلة عمل التطبيق.

إضافةً إلى ذلك، فيمكن لبعض التطبيقات الحساسة والتي تحتاج لأن تعتمد تقنيات وثوقية خاصة مختلفة عن تلك التي يقدمها البروتوكول **TCP** أن تقوم بتطوير هذه التقنيات ابتداء من البروتوكول **UDP**.

4. بناء مخدم TCP باستخدام مقابس تدفق البيانات Stream Sockets

يقوم المخدم عادةً بانتظار طلب الاتصال من الزبون. حيث يحتوي برنامج المخدم غالباً عبارات تحكم أو كتل برمجية تُنفذ باستمرار حتى يستقبل المخدم الطلب. عند وصول الطلب إلى المخدم يقوم بتأسيس الاتصال مع الزبون.

بعدها، يقوم المخدم باستخدام هذا الاتصال لمعالجة الطلبات المستقبلية للزبون وإرسال المعلومات إليه. تتم إدارة هذا الاتصال من خلال غرض من الصف **Socket** كما سنرى.

تقوم البرامج التي تتصل عن طريق البروتوكول **TCP** بمعالجة البيانات التي تستقبلها وترسلها كتدفق من البايتات **Bytes**.

يتطلب تأسيس مخدم بسيط مع البروتوكول **TCP** ومقابس تدفق البيانات **Stream Sockets** خمسة مراحل:

1. المرحلة الأولى: إنشاء متنصت **Listener**.
2. المرحلة الثانية: بدء الاستماع لطلبات الاتصال.
3. المرحلة الثالثة: تأسيس الاتصال بين المخدم والزبون.
4. المرحلة الرابعة: معالجة الاتصال.
5. المرحلة الخامسة: إنهاء الاتصال.

المرحلة الأولى: إنشاء متنصت Listener

يجب أولاً إنشاء غرض من الصف **TcpListener** من فضاء الأسماء **System.Net.Sockets**.
يمثّل هذا الصف مقبس تدفق البيانات باستخدام **TCP** والذي يمكن للمخدم من خلاله الاستماع إلى الطلبات المرسلة إليه.

يتم إنشاء منتسخ جديد من الصف **TcpListener** بالتعليمة التالية:

```
TcpListener server = new TcpListener( ipAddress , port )
```

تقوم التعليمة السابقة بربط تطبيق المخدم برقم منفذ محدد **port**. حيث يكون رقم المنفذ عبارة عن محدد رقمي يستخدم من قبل التطبيق لتعريف نفسه كمرافق لعنوان شبكة معطى.

يمكن أن تأخذ أرقام المنافذ قيم بين **0** و **65535**. تقوم معظم أنظمة التشغيل بحجز أرقام المنافذ التي أرقامها أقل من **1024** وذلك من أجل خدمات النظام (مثل البريد الإلكتروني وخدمات الويب). ويجب أن تُمنح هذه التطبيقات مزايا خاصة للحصول على أرقام المنافذ المحجوزة.

يكون المعامل الأول للتعليمة السابقة **ipAddress** عنوان **IP Internet Protocol Address**.
تقوم عناوين **IP** بتعريف الحواسب على شبكة الانترنت.

وفي الحقيقة، فإن أسماء المواقع مثل **www.scs-org.sy** هي مرادفات لعناوين **IP**. يمكن تعريف عناوين الشبكة من خلال الصف **IPAddress** من فضاء الأسماء **System.Net**.

يجب على أي تطبيق يقوم بعمليات على الشبكة أن يعرف نفسه عن طريق الزوج عنوان **IP**-رقم منفذ. ولا يمكن لتطبيقين أن يأخذا رقم المنفذ ذاته لعنوان **IP** معطى.
سنرى أن ربط المقبس برقم منفذ (باستخدام الطريقة **Bind** من الصف **Socket**) هو غير ضروري، لأن الصف **TcpListener** والصفوف الأخرى التي سنناقشها في هذا الفصل تقوم بذلك بشكل آلي.

المرحلة الثانية: بدء الاستماع لطلبات الاتصال

يجب استدعاء الطريقة **start** من الصف **TcpListener** كي يبدأ الغرض **TcpListener** بالاستماع لطلبات الإتصال.

المرحلة الثالثة: تأسيس الإتصال بين المخدم والزبون

يستمر المخدم بالاستماع للطلبات المرسلة إليه بصورة مستمرة حتى يقوم أحد الزبائن بمحاولة الاتصال. بعدها ينشئ المخدم اتصال اعتماداً على الطلب المرسل.

يقوم غرض من الصف **System.Net.Sockets.Socket** بإدارة هذا الاتصال مع الزبون. تقوم الطريقة **AcceptSocket** من الصف **TcpListener** بقبول طلب الاتصال. تُعيد هذه الطريقة غرض من النوع **Socket** نتيجة لإنشاء الاتصال:

```
Socket connection = server.AcceptSocket();
```

تقوم الطريقة **AcceptSocket** عند استقبال المخدم للطلب باستدعاء الطريقة **Accept** من الصف **TcpListener** لجعل الاتصال مفعلاً.

يخفي استخدام لغة **C#** التفاصيل المعقدة التي تتم على مستوى الشبكة عن المبرمج. حيث يمكن للمبرمج كتابة كود بسيط ومفهوم في برنامج المخدم والسماح للصنوف في فضاء الأسماء **System.Net.Socket** بمعالجة تفاصيل قبول الطلبات وتأسيس الإتصال.

المرحلة الرابعة: معالجة الاتصال

يتم في هذه المرحلة استقبال وإرسال الطلبات والمعلومات بين المخدم والزبون عن طريق الطريقتين **Send** و **Receive** من الصف **Socket**. يجب ملاحظة أن هاتين الطريقتين يمكن استخدامهما فقط في حالة تحقيق الإتصال فعلياً بين المخدم والزبون. بينما يمكن استخدام مقابس برقية البيانات والبروتوكول **UDP** من خلال التوابع **SendTo** و **RecieveFrom** من الصف **Socket** في حال عدم وجود اتصال كما سنرى لاحقاً.

المرحلة الخامسة: إنهاء الاتصال

عند انتهاء المخدم والزبون من عملية استقبال وإرسال الطلبات والمعلومات، يستخدم المخدم الطريقة **Close** من الصف **Socket** لإنهاء الاتصال. بعدها يعود المخدم إلى المرحلة الثانية (انتظار طلب اتصال آخر من قبل الزبون).

يوجد مشكلة في المخطط السابق لتأسيس الاتصال بين المخدم والزبون ذلك أنه عند معالجة طلب من قبل المخدم في المرحلة الرابعة، تقوم هذه المعالجة بحجب الطلبات الأخرى من أن تُخدم. وبهذا، فلا يمكن لأي زبون أن يتصل مع المخدم ما دام المخدم منهمكاً في معالجة طلبات الزبون الأول المتصل. ولحل هذه المشكلة، يجب استخدام البرمجة متعددة النياسب **Multi Threading** والتي درسناها في الفصل السابق. فعند استقبال طلب اتصال من قبل المخدم، يقوم الأخير بإنشاء نيسب **Thread** جديد لتخديم الطلب الجديد مما يسمح للمقبس أو **TcpListener** من استقبال اتصالات أخرى.

المرحلة الخامسة: إنهاء الاتصال

لاحظ أن استخدام الإمكانيات التي تقدمها تقنية تعددية النياسب في **C#** تجعل في الإمكان بناء مخدمات تقوم بإدارة اتصالات متزامنة مع أكثر من زبون. تُستخدم منهجية تعددية النياسب بشكل كبير في مخدمات الشبكة لأنظمة التشغيل **Windows** و **UNIX**.

يمكن أن يزود المخدم متعدد النياسب في الأنظمة ذات الأداء العالي والذواكر الوفيرة بالقدرة على بناء بركة من النياسب **Pool**. يمكن لهذه النياسب معالجة عمليات الدخل/الخرج على الشبكة بصورة سريعة وفعالة عبر مقابس متعددة، وبحيث يستطيع المخدم معالجة الاتصالات الواردة إليه دون الحاجة لبناء نيسب جديد عند كل اتصال مما يساعد على زيادة الأداء.

5. بناء زبون TCP باستخدام مقابس تدفق البيانات Stream Sockets

يتطلب بناء تطبيق زبون يعتمد على مقابس تدفق البيانات **Stream Sockets** أربعة مراحل:

1. المرحلة الأولى: إنشاء زبون **TCP**

يتم في هذه المرحلة بناء غرض من الصف **TcpClient** (والذي ينتمي إلى فضاء الأسماء **System.Net.Socket**) للاتصال بالمخدم.

يؤسس هذا الاتصال باستخدام الطريقة **Connect** من الصف **TcpClient**. والتي تملك معاملي دخل: الأول هو عنوان المخدم **IP**، والثاني هو رقم المنفذ.

```
TcpClient client = new TcpClient ();
client.Connect( serverAddress, serverPort );
```

حيث يكون المعامل **serverPort** رقم منفذ، أما المعامل **serverAddress** فيمكن أن يكون منتسخ من الصف **IPAddress** (الذي يغلف عنوان **IP** للمخدم) أو سلسلة محرفية تُعرف اسم المخدم.

يمكن أيضاً استدعاء الطريقة **Connect** مع معامل دخل وحيد من النمط مرجع لغرض من الصف **IPEndPoint** و الذي يمثل الزوج عنوان-منفذ لنسخة أخرى من التحميل الزائد للطريقة **Connect**.

تستدعي الطريقة **Connect** من الصف **TcpClient** الطريقة **Connect** من الصف **Socket** لتأسيس الاتصال. إذا نجحت عملية الاتصال تقوم الطريقة **TcpClient.Connect** بإعادة رقم موجب وإلا فتُعيد **0**.

2. المرحلة الثانية: توفير تدفق معطيات شبكي

يستخدم الصف **TcpClient** في هذه المرحلة الطريقة **GetStream** للحصول على تدفق معطيات شبكي **NetworkStream** والذي يسمح له بالقراءة من والكتابة إلى المخدم.

ويمكن استخدام الطريقتين **WriteByte** و **Write** لكتابة بايت وحيد أو سلسلة من البايتات على منفذ تدفق البيانات على الشبكة لتصل إلى المخدم. كما يمكن استخدام الطريقتين **ReadByte** **Read** لقراءة بايت أو سلسلة من البايتات من كالمخدم.

3. المرحلة الثالثة: معالجة الاتصال

يتم في هذه المرحلة استقبال وإرسال الطلبات والمعلومات بين المخدم والزبون. يستخدم الزبون في هذه المرحة الطرائق **Read** و **ReadByte** و **Write** و **WriteByte** من الصف **NetworkStream** لإنجاز العمليات المطلوبة خلال الاتصال.

يمكن للزبون أيضاً استخدام النياسب لمنع عملية حجز الاتصال مع المخدمات الأخرى بينما يتم تخديم ومعالجة أحد الاتصالات.

4. المرحلة الرابعة: إغلاق الاتصال

بعد إنهاء عملية نقل البيانات، تتطلب المرحلة الرابعة من الزبون إغلاق الاتصال باستخدام

الطريقة **Close** من الغرض **NetworkStream** والتي تقوم بإغلاق المقبس الموافق لعملية الاتصال (إذا كان الغرض **NetworkStream** يحوي مرجعاً لذاك المقبس). بعدها يقوم الزبون باستدعاء الطريقة **Close** من الصف **TcpClient** لإنهاء الاتصال **TCP**. يمكن بعد ذلك طلب وتأسيس اتصال جديد باستدعاء الطريقة **Connect** من جديد.

6. تفاعل زبون/مخدم باستخدام اتصالات مقابس تدفق البيانات

تستخدم التطبيقات التالية الصفوف والتقانات المناقشة في الفقرتين السابقتين لبناء تطبيق محادثة زبون/مخدم بسيط. حيث ينتظر المخدم طلبات الزبون لتحقيق اتصال. وعندما يتصل تطبيق الزبون بالمخدم، يرسل تطبيق المخدم سلسلة من البايتات إلى الزبون تشير إلى نجاح عملية الاتصال. يعرض عندها الزبون رسالة لإعلام المستخدم بنجاح هذه العملية وإتمام عملية الاتصال.

يحوي، في المثال التالي، كل من تطبيق المخدم والزبون على مربع نص يمنح المستخدم إمكانية كتابة رسالة وإرسالها الى التطبيق الآخر. إذا أرسل تطبيق المخدم أو تطبيق الزبون الرسالة **TERMINATE** فإن الاتصال بين المخدم والزبون ينتهي.

المخدم

يكون في هذا التطبيق كود المخدم:

```
1 // ChatServer.cs
2 // Set up a server that will receive a connection from a client
3 // send a string to client,chat with the client and close
4 using System;
5 using System.Windows.Forms;
6 using System.Threading;
7 using System.Net;
8 using System.Net.Sockets;
9 using System.IO;
10
11 public partial class ChatServerForm : Form
12 {
13  public ChatServerForm()
14   {
15     InitializeComponent();
16  } // end constructor
17
18   private Socket connection; // Socket for accepting a connection
19   private Thread readThread; //Thread for processing incoming msgs
20   private NetworkStream socketStream; // network data stream
21   private BinaryWriter writer; //facilitates writing to the stream
22  private BinaryReader reader;//facilitates reading from the stream
23
24   // initialize thread for reading
25 private void ChatServerForm_Load( object sender, EventArgs e )
26   {
27      readThread = new Thread( new ThreadStart( RunServer ) );
28      readThread.Start();
29   } // end method CharServerForm_Load
30
31   // close all threads associated with this application
32   private void ChatServerForm_FormClosing( object sender,
```

```
33       FormClosingEventArgs e )
34    {
35       System.Environment.Exit( System.Environment.ExitCode );
36    } // end method CharServerForm_FormClosing
37
38    // delegate that allows method DisplayMessage to be called
39    // in the thread that creates and maintains the GUI
40    private delegate void DisplayDelegate( string message );
41
42 // method DisplayMessage sets displayTextBox's Text property
43    // in a thread-safe manner
44    private void DisplayMessage( string message )
45    {
46       // if modifying displayTextBox is not thread safe
47       if ( displayTextBox.InvokeRequired )
48       {
49          // use inherited method Invoke to execute DisplayMessage
50          // via a delegate
51          Invoke( new DisplayDelegate( DisplayMessage ),
52             new object[] { message } );
53       } // end if
54       else // OK to modify displayTextBox in current thread
55          displayTextBox.Text += message;
56    } // end method DisplayMessage
57
58    // delegate that allows method DisableInput to be called
59    // in the thread that creates and maintains the GUI
60    private delegate void DisableInputDelegate( bool value );
61
62    // method DisableInput sets inputTextBox's ReadOnly property
63    // in a thread-safe manner
64    private void DisableInput( bool value )
65    {
66       // if modifying inputTextBox is not thread safe
67       if ( inputTextBox.InvokeRequired )
68       {
69          // use inherited method Invoke to execute DisableInput
70          // via a delegate
71          Invoke( new DisableInputDelegate( DisableInput ),
72             new object[] { value } );
73       } // end if
74       else // OK to modify inputTextBox in current thread
75          inputTextBox.ReadOnly = value;
76    } // end method DisableInput
77
78    // send the text typed at the server to the client
79    private void inputTextBox_KeyDown(object sender,KeyEventArgs e )
80    {
81       // send the text to the client
82       try
83       {
84       if ( e.KeyCode == Keys.Enter && inputTextBox.ReadOnly==false)
85        {
86         writer.Write( "SERVER>>> " + inputTextBox.Text );
87        displayTextBox.Text += "\r\nSERVER>>> " + inputTextBox.Text;
88
89             // if the user at the server signaled termination
90             // sever the connection to the client
91             if ( inputTextBox.Text == "TERMINATE" )
```

```
92                connection.Close();
93
94             inputTextBox.Clear(); // clear the user's input
95          } // end if
96       } // end try
97       catch ( SocketException )
98       {
99          displayTextBox.Text += "\nError writing object";
100       } // end catch
101    } // end method inputTextBox_KeyDown
102
103    // allows a client to connect; displays text the client sends
104    public void RunServer()
105    {
106       TcpListener listener;
107       int counter = 1;
108
109       // wait for a client connection and display the text
110       // that the client sends
111       try
112       {
113          // Step 1: create TcpListener
114          IPAddress local = IPAddress.Parse( "127.0.0.1" );
115          listener = new TcpListener( local, 50000 );
116
117          // Step 2: TcpListener waits for connection request
118          listener.Start();
119
120          // Step 3: establish connection upon client request
121          while ( true )
122          {
123             DisplayMessage( "Waiting for connection\r\n" );
124
125             // accept an incoming connection
126             connection = listener.AcceptSocket();
127
128             // create NetworkStream object associated with socket
129             socketStream = new NetworkStream( connection );
130
131             // create objects for transferring data across stream
132             writer = new BinaryWriter( socketStream );
133             reader = new BinaryReader( socketStream );
134
135            DisplayMessage("Connection "+counter+" received.\r\n" );
136
137             // inform client that connection was successfull
138             writer.Write( "SERVER>>> Connection successful" );
139
140             DisableInput( false ); // enable inputTextBox
141
142             string theReply = "";
143
144             // Step 4: read string data sent from client
145             do
146             {
147                try
148                {
149                   // read the string sent to the server
150                   theReply = reader.ReadString();
```

```
151
152                  // display the message
153                  DisplayMessage( "\r\n" + theReply );
154               } // end try
155               catch ( Exception )
156               {
157                  // handle exception if error reading data
158                  break;
159               } // end catch
160            } while ( theReply != "CLIENT>>> TERMINATE"  &&
161               connection.Connected );
162
163            DisplayMessage( "\r\nUser terminated connection\r\n" );
164
165             // Step 5: close connection
166             writer.Close();
167             reader.Close();
168             socketStream.Close();
169             connection.Close();
170
171             DisableInput( true ); // disable InputTextBox
172             counter++;
173          } // end while
174       } // end try
175       catch ( Exception error )
176       {
177          MessageBox.Show( error.ToString() );
178       } // end catch
179    } // end method RunServer
180 } // end class ChatServerForm
```

قبول الاتصال وتأسيس مجرى تدفق البيانات

تُعرف الأسطر **121-173** حلقة لا نهائية تبدأ بتأسيس الاتصال المطلوب من قبل الزبون (المرحلة **3**). يستدعى السطر **126** الطريقة **AcceptSocket** للغرض **TcpListener** والذي يعيد **Socket** نتيجة للاتصال الناجح.

يتم حجز النيسب الذي تستدعيه الطريقة **AcceptSocket** كي يتم تأسيس الاتصال. يدير الغرض المعاد من نمط **Socket** عملية الاتصال.

يقوم السطر **129** بتمرير الغرض **Socket** كمعامل لباني الغرض **NetworkStream** والذي يزود إمكانية النفاذ الى قنوات تدفق البيانات عبر الشبكة.

يستخدم الغرض **NetworkStream** في هذا المثال قنوات تدفق البيانات لمقبس محدد.

تقوم الأسطر **132-133** ببناء منتسخات **instances** من الصفوف **BinaryWriter** و **BinaryReader** من أجل كتابة وقراءة البيانات، حيث يتم تمرير الغرض **NetworkStream** كمعامل عند بناء هذه المنتسخات.

يمكن للكائن **BinaryWriter** كتابة بايتات على مجرى التدفق الشبكي **NetworkStream** و **BinaryReader** كما يمكنه القراءة منها.

يقوم السطر **135** باستدعاء الطريقة **DisplayMessage** مشيراً إلى وصول طلب الإتصال الى المخدم، بعدها يتم إرسال رسالة للزبون لإعلامه بوصول طلب الإتصال.

يوجد للطريقة **Write** الخاصة بالصف **BinaryWriter** عدة نسخ من التحميل الزائد **(overload versions)** يمكنها كتابة البيانات على مجرى تدفق البيانات بطرق مختلفة.

يستخدم السطر **138** الطريقة **Write** لإرسال سلسلة محرفية للزبون تقوم بإعلامه بنجاح عملية الإتصال. وبهذا تكون انتهت المرحلة الثالثة.

استقبال رسائل من الزبون

بعد قبول الاتصال تبدأ مرحلة المعالجة (المرحلة 4). تُعرف الأسطر **145-161** حلقة **do... while** يتم تنفيذها بصورة مكررة إلى أن يستقبل المخدم رسالة تشير إلى إنهاء الاتصال (هنا **TERMINATE <<<CLIENT**).

يستخدم السطر **150** الطريقة **ReadString** من الصف **BinaryReader** لقراءة سلسلة محرفية من مجرى تدفق الشبكة. يتم تجميد الطريقة **ReadString** حتى تتم عملية القراءة، وهذا هو سبب تنفيذ الطريقة **RunServer** ضمن نيسب مستقل (يتم بناء هذا النيسب في الأسطر **27-28** عند تحميل النموذج). يضمن هذا النيسب لمستخدم تطبيق المخدم بالتفاعل مع واجهة التطبيق وإرسال رسائل إلى الزبون حتى في حال تجميد النيسب المسؤول عن عملية القراءة وهو في حالة انتظار رسالة من الزبون.

تعديل عناصر التحكم في واجهة التطبيق من خلال نياسب منفصلة

بينا في الفصل السابق بأن عناصر التحكم في النماذج **Forms** غير آمنة للنياسب **(thread safe)**، أي أنه عند تعديل عنصر التحكم من قبل أكثر من نيسب فلا يمكن ضمان إجراء هذا التعديل بصورة صحيحة.

كما عرضنا في الفصل السابق، وجوب استخدام الطريقة **Invoke** للصف **Control** لتحقيق سلامة التعديل. تملك الطريقة **Invoke** معاملي دخل: المعامل الأول هو عبارة عن مفوض **delegate** يمثل الطريقة التي سيتم من خلالها تعديل واجهة التطبيق، والثاني مصفوفة من الأغراض تُمثل معاملات هذه الطريقة.

يعرف السطر **40** المفوض **DisplayDelegate** الذي يمثل طريقة تأخذ سلسلة محرفية كمعامل دخل ولا تُعيد أي قيمة.

توافق الطريقة **DisplayMessage** (الأسطر **44-56**) المفوض السابق حيث تستقبل سلسلة محرفية كمعامل باسم **message** ولا تُعيد أي قيمة.

تقوم التعليمة الشرطية **if** في السطر **47** بفحص الخاصية **InvokeRequired** لمربع النص **DisplayTextBox** (يتم وراثة هذه الخاصية من الصف **Control**) والتي تعيد القيمة **True** في حال كان من غير المسموح للنيسب الحالي التعديل على عنصر التحكم مباشرة وإلا يعيد القيمة **False**.

إذا كان النيسب الحالي والذي يقوم بتنفيذ الطريقة **DisplayMessage** مغاير للنيسب الذي قام ببناء واجهة التطبيق، فعندها تُعطي التعليمة **if** القيمة **true**، وتستدعي الأسطر **51-52** الطريقة **invoke** ممررةً لها **DisplayDelegate** الذي يمثل الطريقة **DisplayMessage** نفسها كمعامل أول ومصفوفة جديدة من الأغراض تتكون من سلسلة محرفية كمعامل ثاني.

مما يجعل النيسب الذي قام ببناء واجهة التطبيق يقوم باستدعاء الطريقة **DisplayMessage** ثانية في وقت لاحق مع نفس المعامل للاستدعاء الأصلي. عندما يحدث ذلك الاستدعاء من النيسب الذي قام ببناء واجهة التطبيق، فإن الطريقة قادرة حينها على تعديل **DisplayTextBox** مباشرة و يتم تنفيذ الترميز ضمن عبارة **else** (السطر **55**) وإضافة رسالة إلى النص الخاص بـ **DisplayTextBox**.

تقوم الأسطر **60-76** بتعريف المفوض **DisableInputDelegate** والطريقة **DisableInput** والتي تسمح للنيسب بتعديل الخاصية **ReadOnly** للعنصر **DisplayTextBox** باستخدام نفس التقنية السابقة. يستدعي النيسب الطريقة **DisableInput** مع تمرير معامل من نوع **bool** (**True** مقابل لفعال و **False** مقابل لغير فعال).

إذا كان من غير المسموح للطريقة **DisableInput** بتعديل عناصر التحكم أثناء تنفيذ النيسب الحالي، فعندها تقوم الطريقة **DisableInput** باستدعاء الطريقة **Invoke**. مما يسمح للنيسب الذي قام ببناء واجهة التطبيق بتعديل قيمة الخاصية **inputTextBox.ReadOnly** حسب القيمة الممررة في وقت لاحق.

إنهاء الإتصال مع الزبون

عند انتهاء عملية المحادثة. يتم في الأسطر **166-169** إغلاق كل من **BinaryWriter** و **BinaryReader** و **NetworkStream** و **Socket** عن طريق استدعاء الطريقة **Close** لكل منهم. بعدها يقوم المخدم بانتظار طلب اتصال آخر عن طريق العودة لتنفيذ حلقة **while** من البداية.

إرسال رسالة إلى الزبون

عند إدخال مستخدم تطبيق المخدم سلسلة محرفية في مربع النص **inputTextBox** والقيام بضغط المفتاح **Enter**، يقوم معالج الحدث **inputTextBox_KeyDown** (الأسطر **79-101**) بقراءة السلسلة المحرفية وإرسالها باستخدام الطريقة **Write** من الصف **BinaryWriter**. إذا أنهى المستخدم تطبيق المخدم، يقوم السطر **92** باستدعاء الطريقة **Close** للغرض **Socket** لإنهاء الاتصال.

إنهاء تطبيق المخدم

تُعرف الأسطر **32-36** معالج الحدث **ChatServerForm_FormClosing** من أجل تنفيذه عند إغلاق التطبيق. يقوم هذا الحدث بإغلاق التطبيق عن طريق استدعاء الطريقة **Exit** من الصف **Environment** مع تمرير المعامل **ExitCode** لإنهاء كل النياسب الخاصة بهذا التطبيق.

الزبون

يكون في هذا التطبيق كود الزبون:

```
1 // ChatClient.cs
2 // Set up a client that will send information to and
3 // read information from a server.
4 using System;
5 using System.Windows.Forms;
6 using System.Threading;
7 using System.Net.Sockets;
8 using System.IO;
9
10  public partial class ChatClientForm : Form
11 {
12   public ChatClientForm()
13   {
14     InitializeComponent();
15   } // end constructor
16
17  private NetworkStream output;//stream for receiving data
```

```
18  private BinaryWriter writer; //facilitates writing to the stream
19  private BinaryReader reader;//facilitates reading from the stream
20  private Thread readThread;//Thread for incoming messages
21  private string message = "";
22
23   // initialize thread for reading
24   private void ChatClientForm_Load( object sender, EventArgs e )
25   {
26      readThread = new Thread( new ThreadStart( RunClient ) );
27      readThread.Start();
28   } // end method ChatClientForm_Load
29
30   // close all threads associated with this application
31   private void ChatClientForm_FormClosing( object sender,
32      FormClosingEventArgs e
33   {
34      System.Environment.Exit( System.Environment.ExitCode );
35   } // end method ChatClientForm_FormClosing
36
37   // delegate that allows method DisplayMessage to be called
38   // in the thread that creates and maintains the GUI
39   private delegate void DisplayDelegate( string message );
40
41   // method DisplayMessage sets displayTextBox's Text property
42   // in a thread-safe manner
43   private void DisplayMessage( string message )
44   {
45      // if modifying displayTextBox is not thread safe
46      if ( displayTextBox.InvokeRequired )
47      {
48         // use inherited method Invoke to execute DisplayMessage
49         // via a delegate
50         Invoke( new DisplayDelegate( DisplayMessage ),
51            new object[] { message } );
52      } // end if
53      else // OK to modify displayTextBox in current thread
54         displayTextBox.Text += message;
55   } // end method DisplayMessage
56
57   // delegate that allows method DisableInput to be called
58   // in the thread that creates and maintains the GUI
59   private delegate void DisableInputDelegate( bool value );
60
61   // method DisableInput sets inputTextBox's ReadOnly property
62   // in a thread-safe manner
63   private void DisableInput( bool value )
64   {
65      // if modifying inputTextBox is not thread safe
66      if ( inputTextBox.InvokeRequired )
67      {
68         // use inherited method Invoke to execute DisableInput
69         // via a delegate
70         Invoke( new DisableInputDelegate( DisableInput ),
71            new object[] { value } );
72      } // end if
73      else // OK to modify inputTextBox in current thread
74         inputTextBox.ReadOnly = value;
75   } // end method DisableInput
76
```

```
77   // sends text the user typed to server
78   private void inputTextBox_KeyDown(object sender,KeyEventArgs e)
79   {
80      try
81      {
82         if (e.KeyCode==Keys.Enter && inputTextBox.ReadOnly==false)
83         {
84           writer.Write( "CLIENT>>> " + inputTextBox.Text );
85         displayTextBox.Text += "\r\nCLIENT>>> "+inputTextBox.Text;
86            inputTextBox.Clear();
87         } // end if
88      } // end try
89      catch ( SocketException )
90      {
91         displayTextBox.Text += "\nError writing object";
92      } // end catch
93   } // end method inputTextBox_KeyDown
94
95   // connect to server and display server-generated text
96   public void RunClient()
97   {
98      TcpClient client;
99
100     // instantiate TcpClient for sending data to server
101     try
102     {
103        DisplayMessage( "Attempting connection\r\n" );
104
105        // Step 1: create TcpClient and connect to server
106        client = new TcpClient();
107        client.Connect( "127.0.0.1", 50000 );
108
109        // Step 2: get NetworkStream associated with TcpClient
110        output = client.GetStream();
111
112        // create objects for writing and reading across stream
113        writer = new BinaryWriter( output );
114        reader = new BinaryReader( output );
115
116        DisplayMessage( "\r\nGot I/O streams\r\n" );
117        DisableInput( false ); // enable inputTextBox
118
119        // loop until server signals termination
120        do
121        {
122           // Step 3: processing phase
123           try
124           {
125              // read message from server
126              message = reader.ReadString();
127              DisplayMessage( "\r\n" + message );
128           } // end try
129           catch ( Exception )
130           {
131              // handle exception if error in reading server data
132           System.Environment.Exit(System.Environment.ExitCode);
133           } // end catch
134        } while ( message != "SERVER>>> TERMINATE" );
135
```

```
136         // Step 4: close connection
137         writer.Close();
138         reader.Close();
139         output.Close();
140         client.Close();
141
142         Application.Exit();
143      } // end try
144      catch ( Exception error )
145      {
146         // handle exception if error in establishing connection
147         MessageBox.Show( error.ToString(), "Connection Error",
148            MessageBoxButtons.OK, MessageBoxIcon.Error );
149         System.Environment.Exit( System.Environment.ExitCode );
150      } // end catch
151   } // end method RunClient
152 } // end class ChatClientForm
```

يقوم **ChatClientForm** ببناء نيسب **(الأسطر 96-151)** لمعالجة الرسائل الواردة.

تتصل الطريقة **RunClient** بالصف **ChatServerForm** حيث تقوم باستقبال البيانات منه وإرسال البيانات إليه.

تقوم الأسطر **106-107** بتعريف وتهيئة الغرض **TcpClient** ثم استدعاء الطريقة **Connect** لتأسيس عملية الإتصال مع المخدم (المرحلة **1**). يكون المعامل الأول للطريقة **Connect** هو اسم المخدم (في حالتنا هذه اسم المخدم هو **localhost** مما يعني أن المخدم والزبون يتوضعان على نفس الحاسب). يطابق **localhost** لعنوان **IP** للحاسب المضيف وهو **127.0.0.1**. أما المعامل الثاني فهو رقم المنفذ والذي ينتظر المخدم من خلاله لاستقبال طلبات الاتصال.

يستخدم **ChatClientForm** غرض من الصف **NetworkStream** لاستقبال وإرسال البيانات إلى المخدم.

يتضمن الزبون في السطر **110** على غرض من الصف **NetworkStream** نتيجة لاستدعاء الطريقة **GetStream** من الصف **TcpClient** (المرحلة **2**). يستمر تنفيذ الحلقة **do...while** في الأسطر **120-134** بصورة متكررة حتى يتم استقبال رسالة **TERMINATE** من قبل المخدم.

يستخدم السطر **126** الطريقة **ReadString** من الصف **BinaryReader** للحصول على الرسالة التالية من المخدم (المرحلة **3**). يعرض السطر **127** هذه الرسالة، ثم تقوم الأسطر **137-140** بإغلاق الاتصال لكل من الإغراض **BinaryWriter** و **BinaryReader** و **NetworkStream** و **TcpClient** (المرحلة **4**).

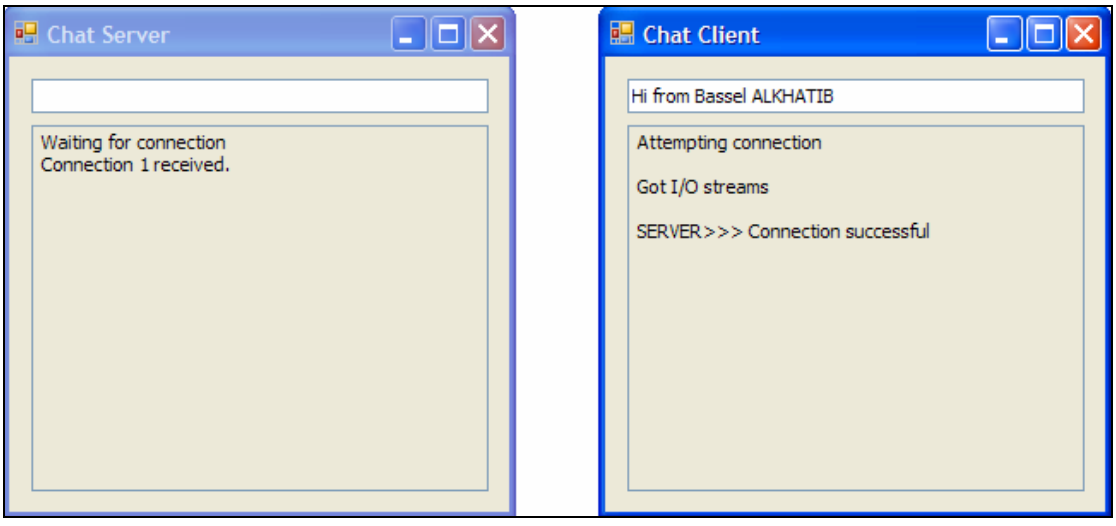
تُعرف الأسطر **75-39** كل من **DisplayDelegate** و **DisplayMessage** و **DisableInputDelegate** و **DisableInput** بصورة مشابهة تماماً لما هو معرف سابقاً في الأسطر **76-40** من كود المخدم. تُستخدم هذه الطرائق لضمان تعديل واجهة التطبيق من النيسب الذي قام ببناء واجهة التطبيق فقط.

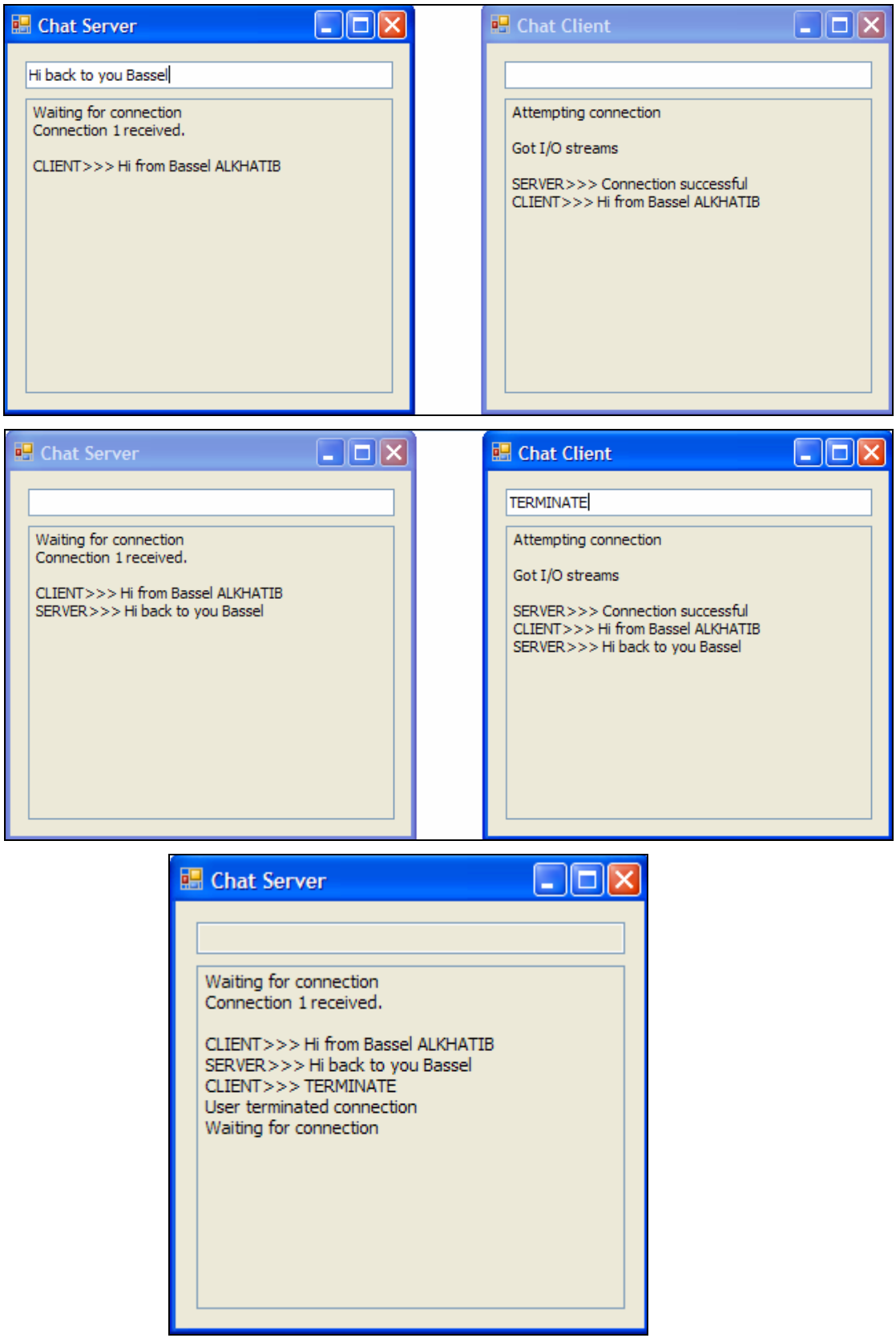
عندما يقوم مستخدم تطبيق الزبون بإدخال سلسلة محرفية في مربع النص ومن ثم ضغط المفتاح **Enter**، يقوم معالج الحدث **inputTextBox_KeyDown** بقراءة السلسلة المحرفية وإرسالها باستخدام الطريقة **Write** من الصف **BinaryWriter**.

نلاحظ هنا أن:

ChatServerForm يستقبل الاتصال ثم يقوم بمعالجته، بعدها يتم إغلاق الاتصال لينتظر من جديد من أجل طلب اتصال جديد. لكن إذا كنا نتحدث عن تطبيق واقعي فيقوم المخدم باستقبال الاتصال ثم إسناد معالجة لنيسب منفصل ليقوم بتنفيذ عملية الاتصال، ليتابع انتظار طلب اتصال جديد. تقوم النياسب المنفصلة بمعالجة الاتصالات الحالية بينما يقوم المخدم بالتركيز على طلبات الاتصال الجديدة.

تُبين الأشكال التالية مختلف مراحل تفاعل المخدم والزبون:




Chat Server
Hi back to you Bassel
Waiting for connection
Connection 1 received.
CLIENT>>> Hi from Bassel ALKHATIB
Chat Client
Attempting connection
Got I/O streams
SERVER>>> Connection successful
CLIENT>>> Hi from Bassel ALKHATIB
Chat Server
Waiting for connection
Connection 1 received.
CLIENT>>> Hi from Bassel ALKHATIB
SERVER>>> Hi back to you Bassel
Chat Client
TERMINATE
Attempting connection
Got I/O streams
SERVER>>> Connection successful
CLIENT>>> Hi from Bassel ALKHATIB
SERVER>>> Hi back to you Bassel
Chat Server
Waiting for connection
Connection 1 received.
CLIENT>>> Hi from Bassel ALKHATIB
SERVER>>> Hi back to you Bassel
CLIENT>>> TERMINATE
User terminated connection
Waiting for connection

7. تفاعل زبون/مخدم في الاتصال عديم الارتباط مع برقيات البيانات

Connectionless Client/Server Interaction With Datagrams

عرضنا في الفقرة السابقة الاتصال المرتبط ونقل البيانات اعتماداً على تدفق البيانات **Streams-Based** باستخدام البروتوكول **TCP** وذلك لضمان وصول رزم البيانات إلى الوجهة المقصودة بصورة موثوقة.

نعتبر فيما يلي أن إرسال البيانات يتم باستخدام الاتصال عديم الارتباط أي باستخدام برقية البيانات **Datagram** والبروتوكول **UDP**.

تُشابه عملية إرسال البيانات باستخدام الاتصال عديم الارتباط وباستخدام برقيات البيانات عملية تسليم الرسائل في خدمة البريد.

تتم معالجة البيانات وإرسالها عند استخدام الاتصال عديم الارتباط كرزم معطيات تُدعى برقيات البيانات.

يمكن تشبيه برقية البيانات بالرسائل المرسلة عن طريق البريد. في حال عدم القدرة على إرسال رسالة كبيرة ضمن مغلف واحد، يتم تقسيم هذه الرسالة إلى أقسام، ثم يتم إرسالها ضمن مغلفات منفصلة وبصورة متسلسلة.

كما تُرسل كل هذه الرسائل مرة واحدة. يمكن أن تصل هذه الرسائل بصورة مرتبة أو غير

مرتبة أو أن لا تصل أبداً. عند استقبال الرسائل من قبل الشخص عند نقطة الاستقبال يقوم بتجميع الأجزاء المرسلة بالترتيب الصحيح قبل أن يحاول قراءة هذه الرسالة. إذا كانت الرسالة صغيرة كفاية بحيث يمكن إرسالها ضمن مغلف واحد، فعندها لا يبقى مشكلة في تسلسل الإرسال، لكن تبقى إمكانية عدم وصول الرسالة أبداً بعد عملية الإرسال.

تُوفر **C#** الصف **UdpClient** من أجل إجراء نقل البيانات في الاتصال عديم الارتباط. يستخدم الصف **UdpClient** طرائق من الصف **Socket** بطريقة مشابهة لاستخدام الصفوف **TcpListener** و **TcpClient** لطرائق من الصف **Socket** كما رأينا سابقا. تقوم الطرائق **Send** و **Receive** الخاصة بالصف **UdpClient** باستخدام الطرائق **SendTo** و **ReceiveForm** من الصف **Socket** لإرسال واستقبال البيانات.

تستخدم البرامج المعروضة فيما يلي برقية البيانات لإرسال رزم من البيانات بين تطبيقات الزبون والمخدم. يقوم المستخدم في التطبيق **PacketClient** بكتابة رسالة في مربع النص **TextBox** ثم يقوم بضغط المفتاح **Enter**.

يقوم الزبون بتحويل هذه الرسالة إلى سلسلة من البايتات ويرسلها إلى المخدم. يقوم المخدم باستلام رزمة البيانات وعرض المعلومات المرسلة التي تحويها هذه الرزمة و من ثم إعادتها إلى الزبون.

يقوم الزبون عند استلام رزمة البيانات بعرض المعلومات التي تتضمنها. تكون في هذا المثال طريقة تنفيذ الترميز للصفوف **PacketClientForm** و **PacketServerForm** متشابهة.

المخدم

نعرض فيما يلي الكود الخاص بالصف **PacketServerForm**:

```
1 // PacketServer.cs
2 // Set up a server that will receive packets from a
3// client and send the packets back to the client.
4 using System;
5 using System.Windows.Forms;
6 using System.Net;
7 using System.Net.Sockets;
8 using System.Threading;
9
10 public partial class PacketServerForm : Form
11 {
12   public PacketServerForm()
13   {
14      InitializeComponent();
15   } // end constructor
16
17   private UdpClient client;
18   private IPEndPoint receivePoint;
19
20   // initialize variables and thread for receiving packets
21   private void PacketServerForm_Load( object sender, EventArgs e )
22   {
23      client = new UdpClient( 50000 );
24      receivePoint = new IPEndPoint( new IPAddress( 0 ), 0 );
25      Thread readThread =
26         new Thread( new ThreadStart( WaitForPackets ) );
27      readThread.Start();
28   } // end method PacketServerForm_Load
29
30   // shut down the server
31   private void PacketServerForm_FormClosing( object sender,
32      FormClosingEventArgs e )
33   {
34      System.Environment.Exit( System.Environment.ExitCode );
35   } // end method PacketServerForm_FormClosing
36
```

```
37    // delegate that allows method DisplayMessage to be called
38    // in the thread that creates and maintains the GUI
39    private delegate void DisplayDelegate( string message );
40
41    // method DisplayMessage sets displayTextBox's Text property
42    // in a thread-safe manner
43    private void DisplayMessage( string message )
44    {
45
46        // if modifying displayTextBox is not thread safe
47       if ( displayTextBox.InvokeRequired )
48       {
49          // use inherited method Invoke to execute DisplayMessage
50          // via a delegate
51          Invoke( new DisplayDelegate( DisplayMessage ),
52             new object[] { message } );
53       } // end if
54       else // OK to modify displayTextBox in current thread
55          displayTextBox.Text += message;
56    } // end method DisplayMessage
57
58    // wait for a packet to arrive
59    public void WaitForPackets()
60    {
61       while ( true )
62       {
63          // set up packet
64          byte[] data = client.Receive( ref receivePoint );
65          DisplayMessage( "\r\nPacket received:" +
66             "\r\nLength: " + data.Length +
67             "\r\nContaining: " +
68             System.Text.Encoding.ASCII.GetString( data ) );
69
70          // echo information from packet back to client
71          DisplayMessage( "\r\n\r\nEcho data back to client..." );
72          client.Send( data, data.Length, receivePoint );
73          DisplayMessage( "\r\nPacket sent\r\n" );
74       } // end while
75    } // end method WaitForPackets
76 } // end class PacketServerForm
```

يقوم معالج الحدث **Load** في السطر **23** من الصف **PacketServerForm** ببناء منتسخ من الصف **UdpClient** المسؤول عن استلام البيانات على المنفذ **50000**. تقوم هذه العملية بتهيئة المقبس **(Socket)** لتنفيذ عملية الاتصال.

يقوم السطر **24** ببناء منتسخ من الصف **IPEndPoint** ليحتفظ بعنوان **IP** و رقم المنفذ الذي يتم من خلالهما إرسال البيانات إلى البرنامج **PacketServerForm** من قبل الزبائن.

يكون المعامل الأول لباني **IPEndPoint** غرض من الصف **IPAddress** أما المعامل الثاني فهو رقم المنفذ. يكون للمعاملات السابقة القيمة **0** لأننا نقوم بتهيئة غرض **IPEndPoint** فارغ. عند

وصول برقية معطيات من الزبون إلى المخدم فإنه يتم نسخ العنوان **IP** ورقم المنفذ إلى الغرض **IPEndPoint**.

يتم تعريف الطرائق **DisplayDelegate**و **DisplayMessage** في الأسطر **39-55** لإعطاء أي نيسب القدرة على تعديل الخاصية **Text** للعنصر **DisplayTextBox**.

تقوم الطريقة **WaitForPackets** بتنفيذ حلقة لا نهائية يتم من خلالها انتظار وصول البيانات إلى التطبيق. عند وصول البيانات تقوم الطريقة **Receive** (السطر **64**) من الصف **UdpClient** باستلام سلسلة من البايتات من الزبون.

يتم تمرير الغرض **IPEndPoint** للطريقة **Receive** والذي تم إنشائه عن طريق الباني. وبهذا يمكن للطريقة **Receive** نسخ عنوان **IP** ورقم المنفذ إلى الغرض **IPEndPoint** الممرر. وبهذا الأسلوب يمكن ترجمة وتنفيذ البرنامج بدون أن يعطي أي استثناء **(Exception)** لأن الطريقة **Receive** تقوم بتهيئة الغرض **IPEndPoint** وإعطاءه قيمة لا تساوي **Null**.

تقوم الأسطر **64-67** بتعديل واجهة الإظهار لتتضمن بيانات رزم البيانات المرسلة.

يقوم السطر **71** بإعادة البيانات المرسلة إلى الزبون باستخدام الطريقة **Send** للصف **UpdClient**. تملك الطريقة **Send** ثلاثة معاملات:

- سلسلة البايتات المعدة للإرسال، رقم من نمط **int** يمثل طول السلسلة.
- والغرض **IPEndPoint** الذي يمثل وجهة الإرسال.

نستخدم المصفوفة **data** التي تُعيدها الطريقة **Receive** كمعامل أول للطريقة **Send** وتُمثل البيانات المعدة للإرسال، أما طول المصفوفة **data** فهو المعامل الثاني، أما المعامل الثالث فهو **IPEndPoint** الممررة للطريقة **Receive**.

يتم تخزين عنوان **IP** ورقم المنفذ للزبون المرسل في الغرض **ReceivePoint**، وذلك عن طريق تمرير هذا الغرض إلى الطريقة **Send** مما يسمح للبرنامج **PacketServerForm** بالاستجابة والرد على الزبون.

الزبون

نعرض فيما يلي الكود الخاص بالصف **PacketClientForm**:

```
1 // PacketClient.cs
2 // Set up a client that sends packets to a server and receives
3 // packets from a server.
4 using System;
5 using System.Windows.Forms;
6 using System.Net;
7 using System.Net.Sockets;
8 Using System.Threading;
9
10 public partial class PacketClientForm : Form
11 {
12   public PacketClientForm()
13   {
14      InitializeComponent();
15   } // end constructor
16
17   private UdpClient client;
18   private IPEndPoint receivePoint;
19
20   // initialize variables and thread for receiving packets
21   private void PacketClientForm_Load( object sender, EventArgs e )
22   {
23      receivePoint = new IPEndPoint( new IPAddress( 0 ), 0 );
24      client = new UdpClient( 50001 );
25      Thread thread =
26         new Thread( new ThreadStart( WaitForPackets ) );
27      thread.Start();
28   } // end method PacketClientForm_Load
29
30   // shut down the client
31   private void PacketClientForm_FormClosing( object sender,
32      FormClosingEventArgs e )
33   {
34      System.Environment.Exit( System.Environment.ExitCode );
35   } // end method PacketClientForm_FormClosing
36
37   // delegate that allows method DisplayMessage to be called
38   // in the thread that creates and maintains the GUI
39   private delegate void DisplayDelegate( string message );
40
41   // method DisplayMessage sets displayTextBox's Text property
42   // in a thread-safe manner
43   private void DisplayMessage( string message )
44   {
45      // if modifying displayTextBox is not thread safe
46      if ( displayTextBox.InvokeRequired )
47      {
48         // use inherited method Invoke to execute DisplayMessage
49         // via a delegate
50         Invoke( new DisplayDelegate( DisplayMessage ),
51            new object[] { message } );
```

```
52       } // end if
53       else // OK to modify displayTextBox in current thread
54          displayTextBox.Text += message;
55    } // end method DisplayMessage
56
57    // send a packet
58    private void inputTextBox_KeyDown( object sender, KeyEventArgs e )
59    {
60       if ( e.KeyCode == Keys.Enter )
61       {
62          // create packet (datagram) as string
63          string packet = inputTextBox.Text;
64          displayTextBox.Text +=
65             "\r\nSending packet containing: " + packet;
66
67          // convert packet to byte array
68          byte[] data = System.Text.Encoding.ASCII.GetBytes( packet );
69
70          // send packet to server on port 50000
71          client.Send( data, data.Length, "127.0.0.1", 50000 );
72          displayTextBox.Text += "\r\nPacket sent\r\n";
73          inputTextBox.Clear();
74       } // end if
75    } // end method inputTextBox_KeyDown
76
77    // wait for packets to arrive
78    public void WaitForPackets()
79    {
80       while ( true )
81       {
82          // receive byte array from server
83          byte[] data = client.Receive( ref receivePoint );
84
85          // output packet data to TextBox
86          DisplayMessage( "\r\nPacket received:" +
87             "\r\nLength: " + data.Length + "\r\nContaining: " +
88             System.Text.Encoding.ASCII.GetString( data ) + "\r\n" );
89       } // end while
90    } // end method WaitForPackets
91 } // end class PacketClientForm
```

يعمل الصف **PacketClientForm** بطريقة مشابهة للصف **PacketServerForm** عدا أن الغرض **Client** يقوم بإرسال رزم البيانات فقط وذلك عندما يقوم المستخدم بكتابة رسالة في مربع النص ومن ثم ضغط المفتاح **Enter**. إذ يقوم البرنامج باستدعاء معالج الحدث **inputTextBox_KeyDown** (الأسطر **58-75**).

يقوم السطر **68** بتحويل السلسلة المحرفية التي أدخلها المستخدم في مربع نص إلى مصفوفة من البايتات. يستدعي السطر **71** الطريقة **Send** من الصف **UdpClient** لإرسال مصفوفة البايتات إلى البرنامج **PacketServerForm**.

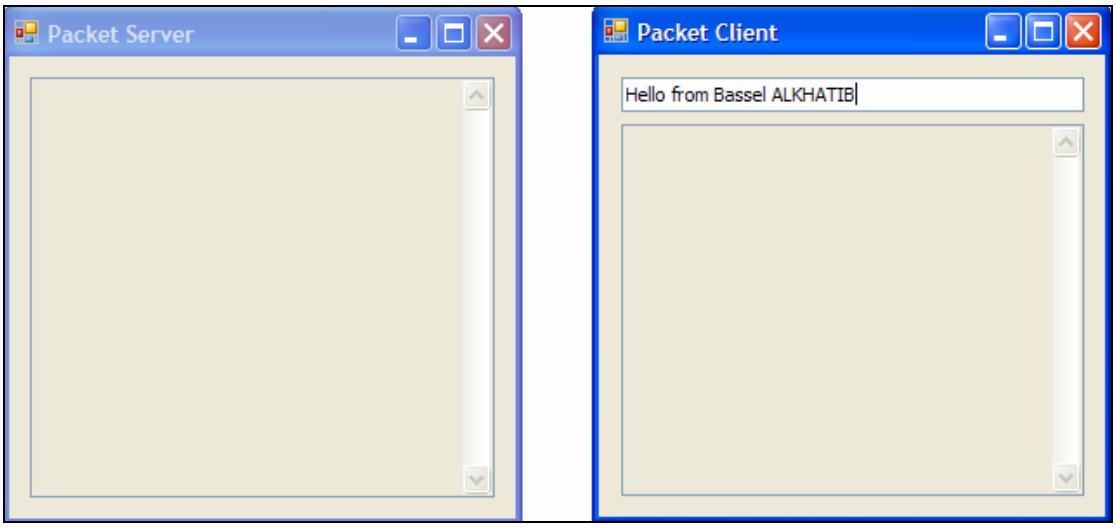
يتم تعريف الطرائق **DisplayDelegate** و **DisplayMessage** في الأسطر **39-55** لإعطاء أي نيسب القدرة على تعديل الخاصية **Text** للعنصر **DisplayTextBox**.

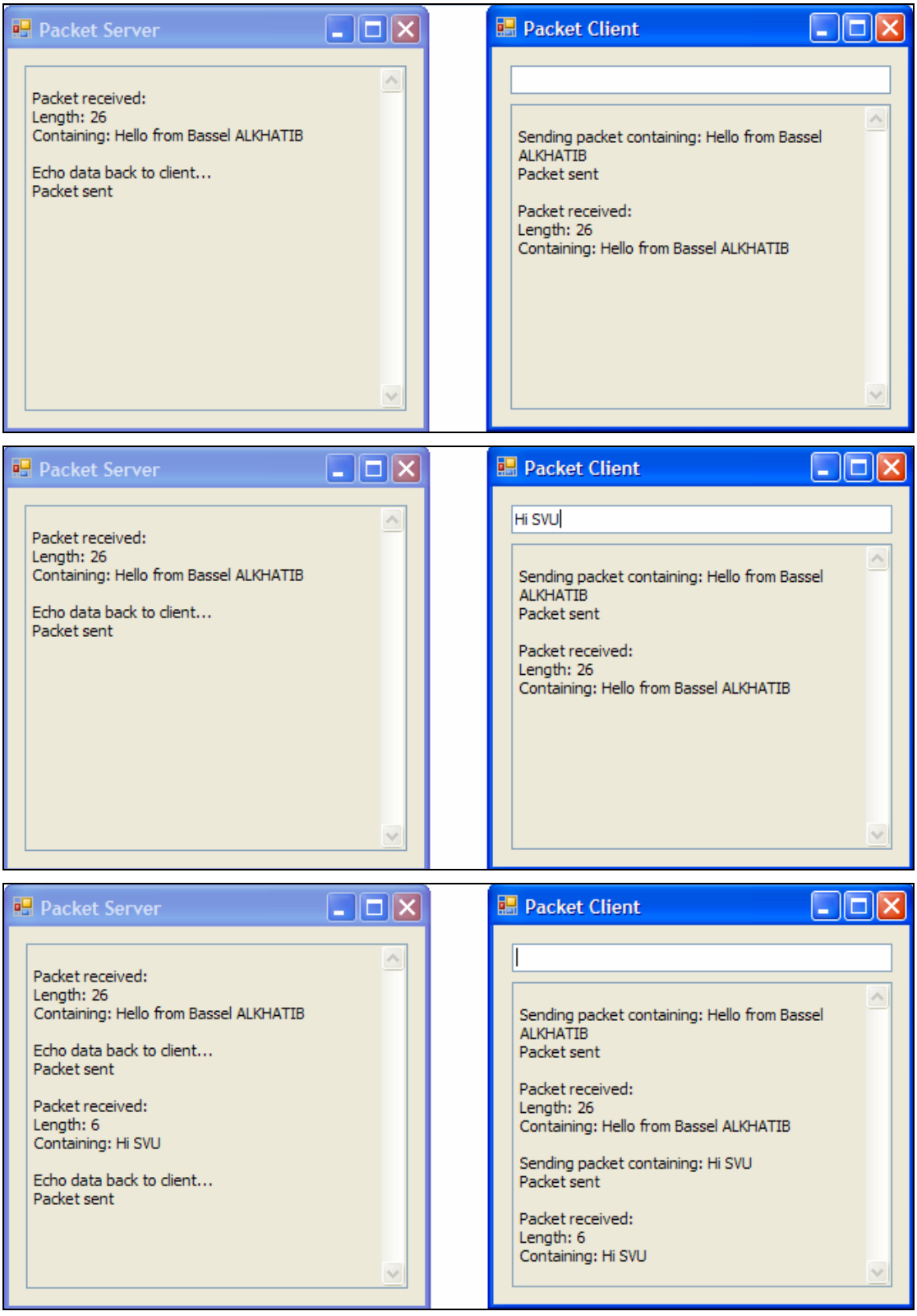
يقوم السطر **24** بتهيئة الغرض **UdpClient** لاستقبال رزم البيانات على المنفذ **50001** (استخدمنا رقم المنفذ **50001** لأن **PacketServerForm** حجز رقم المنفذ **50000** مسبقاً).

تستخدم الطريقة **WaitForPackets** من الصف **PacketClientForm** (الأسطر **78-90**) حلقة لا نهائية لانتظار رزم البيانات المرسلة. يتم تجميد الطريقة **Receive** من الصف **UdpClient** حتى يتم استقبال رزمة بيانات (السطر **83**). إلا أن التجميد المنجز من قبل الطريقة **Receive** لا يمنع الصف **PacketClientForm** من تنفيذ خدمات أخرى لأن نيسب آخر يقوم بتنفيذ الطريقة **WaitForPackets**.

عند وصول رزمة البيانات، تعرض الأسطر **68-88** مكونات هذه الرزمة ضمن مربع النص. يمكن للمستخدم كتابة المعلومات المراد إرسالها إلى تطبيق المخدم في مربع النص ثم يقوم بضغط المفتاح **Enter** في أي وقت حتى في حال وصول رسالة من المخدم.

تُبين الأشكال التالية مختلف مراحل تفاعل المخدم والزبون:




Packet Server
Packet received:
Length: 26
Containing: Hello from Bassel ALKHATIB
Echo data back to client...
Packet sent
Packet Client
Sending packet containing: Hello from Bassel ALKHATIB
Packet sent
Packet received:
Length: 26
Containing: Hello from Bassel ALKHATIB
Packet Server
Packet received:
Length: 26
Containing: Hello from Bassel ALKHATIB
Echo data back to client...
Packet sent
Packet Client
Hi SVU
Sending packet containing: Hello from Bassel ALKHATIB
Packet sent
Packet received:
Length: 26
Containing: Hello from Bassel ALKHATIB
Packet Server
Packet received:
Length: 26
Containing: Hello from Bassel ALKHATIB
Echo data back to client...
Packet sent
Packet received:
Length: 6
Containing: Hi SVU
Echo data back to client...
Packet sent
Packet Client
Sending packet containing: Hello from Bassel ALKHATIB
Packet sent
Packet received:
Length: 26
Containing: Hello from Bassel ALKHATIB
Sending packet containing: Hi SVU
Packet sent
Packet received:
Length: 6
Containing: Hi SVU

8. بناء تطبيق زبون/مخدم باستخدام مخدم متعدد النياسب

نقوم في هذه الفقرة بدراسة وعرض كيفية تصميم نسخة شبكية من اللعبة الشائعة **Tic-Tac-Toe** (والتي تُعرف أيضاً بلعبة **XO**) وذلك عن طريق استخدام تقانة مقابس تدفق البيانات في بيئة زبون/مخدم.

يتكون البرنامج بشكل أساسي من الصف **TicTacToeServer** للمخدم، ومن الصف **TicTacToeClient** للزبون. يسمح صف المخدم **TicTacToeServer** لمنتسخين من صف الزبون **TicTacToeClient** بالاتصال بالمخدم وباللعب مع بعضهما البعض.

يقوم المخدم كما سنرى ببناء منتسخ من الصف **Player** ليسمح بمعالجة اتصال كل زبون وتنفيذه ضمن نيسب منفصل. مما يمكّن المخدم من معالجة الطلبات المرسلة من الزبونين.

يقوم المخدم بتحديد القيمة "**X**" للزبون الأول المتصل (اللاعب **X** يقوم بالحركة الأولى)، ثم تحديد القيمة "**O**" للزبون الثاني.

يحتفظ المخدم أثناء عملية اللعب بمعلومات تحوي حالة واجهة تطبيق الزبونين ليتمكن المخدم من اختبار صحة حركة اللاعبين. يمكن لكل من المخدم والزبون معرفة اللاعب الفائز. (تركنا في هذا المثال الطريقة **GameOver** تُعيد دوماً القيمة **False**، أي لم نضع الكود اللازم لتحديد فوز أحد الزبونين).

يقوم كل زبون بعرض واجهة اللعبة الخاصة به ويقوم فقط بوضع علامة في مربع فارغ على اللوحة في واجهة المستخدم. يستخدم الصف **Square** لتعريف المربعات على لوحة اللعب.

المخدم

نعرض فيما يلي الكود الخاص بالصف **TicTacToeServerForm**:

```
1 // TicTacToeServer.cs
2 // This class maintains a game of Tic-Tac-Toe for two
3 // client applications.
4 using System;
5 using System.Windows.Forms;
6 using System.Net;
7 using System.Net.Sockets;
8 using System.Threading;
9 using System.IO;
10
11 public partial class TicTacToeServerForm : Form
12 {
13    public TicTacToeServerForm()
14       {
15          InitializeComponent();
```

```
16       } // end constructor
17
18 private byte[] board; //the local representation of the game board
19 private Player[] players; // two Player objects
20 private Thread[] playerThreads; // Threads for client interaction
21 private TcpListener listener; // listen for client connection
22 private int currentPlayer; // keep track of whose turn it is
23 private Thread getPlayers;//Thread acquiring client connections
24 internal bool disconnected = false; // true if the server closes
25
26 // initialize variables and thread for receiving clients
27 private void TicTacToeServerForm_Load(object sender, EventArgs e )
28       {
29          board = new byte[ 9 ];
30          players = new Player[ 2 ];
31          playerThreads = new Thread[ 2 ];
32          currentPlayer = 0;
33
34          // accept connections on a different thread
35          getPlayers = new Thread( new ThreadStart( SetUp ) );
36          getPlayers.Start();
37       } // end method TicTacToeServerForm_Load
38
39       // notify Players to stop Running
40       private void TicTacToeServerForm_FormClosing( object sender,
41          FormClosingEventArgs e )
42       {
43          disconnected = true;
44          System.Environment.Exit( System.Environment.ExitCode );
45       } // end method TicTacToeServerForm_FormClosing
46
47       // delegate that allows method DisplayMessage to be called
48       // in the thread that creates and maintains the GUI
49       private delegate void DisplayDelegate( string message );
50
51       // method DisplayMessage sets displayTextBox's Text property
52       // in a thread-safe manner
53       internal void DisplayMessage( string message )
54       {
55          // if modifying displayTextBox is not thread safe
56          if ( displayTextBox.InvokeRequired )
57          {
58             // use inherited method Invoke to execute
59             // DisplayMessage  via a delegate
60             Invoke( new DisplayDelegate( DisplayMessage ),
61                new object[] { message } );
62          } // end if
63          else // OK to modify displayTextBox in current thread
64             displayTextBox.Text += message;
65       } // end method DisplayMessage
66
67       // accepts connections from 2 players
68       public void SetUp()
69       {
70          DisplayMessage( "Waiting for players...\r\n" );
71
72          // set up Socket
73          listener =
74          new TcpListener( IPAddress.Parse( "127.0.0.1" ), 50000 );
```

```
75          listener.Start();
76
77          // accept first player and start a player thread
78          players[0]=new Player(listener.AcceptSocket(), this, 0 );
79          playerThreads[ 0 ] =
80             new Thread( new ThreadStart( players[ 0 ].Run ) );
81          playerThreads[ 0 ].Start();
82
83          // accept second player and start another player thread
84          players[1]=newPlayer(listener.AcceptSocket(), this, 1 );
85          playerThreads[ 1 ] =
86             new Thread( new ThreadStart( players[ 1 ].Run ) );
87          playerThreads[ 1 ].Start();
88
89          //let the 1 player know that the 2 player has connected
90          lock ( players[ 0 ] )
91          {
92             players[ 0 ].threadSuspended = false;
93             Monitor.Pulse( players[ 0 ] );
94          } // end lock
95       } // end method SetUp
96
97       // determine if a move is valid
98       public bool ValidMove( int location, int player )
99       {
100         // prevent another thread from making a move
101         lock ( this )
102         {
103            // while it is not the current player's turn, wait
104            while ( player != currentPlayer )
105               Monitor.Wait( this );
106
107            // if the desired square is not occupied
108            if ( !IsOccupied( location ) )
109            {
110             // set the board to contain the current player's mark
111               board[ location ] = ( byte ) ( currentPlayer == 0 ?
112                  'X' : 'O' );
113
114               // set the currentPlayer to be the other player
115               currentPlayer = ( currentPlayer + 1 ) % 2;
116
117               // notify the other player of the move
118            players[ currentPlayer ].OtherPlayerMoved(location );
119
120               // alert the other player that it's time to move
121               Monitor.Pulse( this );
122               return true;
123            } // end if
124            else
125               return false;
126         } // end lock
127      } // end method ValidMove
128
129      // determines whether the specified square is occupied
130      public bool IsOccupied( int location )
131      {
132         if (board[ location ]== 'X' || board[ location ] == 'O' )
133            return true;
```

```
134         else
135            return false;
136      } // end method IsOccupied
137
138      // determines if the game is over
139      public bool GameOver()
140      {
141         // place code here to test for a winner of the game
142         return false;
143      } // end method GameOver
144   } // end class TicTacToeServerForm
145
146   // class Player represents a tic-tac-toe player
147   public class Player
148   {
149    internal Socket connection;//Socket for accepting a connection
150    private NetworkStream socketStream; // network data stream
151    private TicTacToeServerForm server; // reference to server
152    private BinaryWriter writer;//facilitates writing to stream
153    private BinaryReader reader;//facilitates reading from stream
154    private int number; // player number
155    private char mark; // player's mark on the board
156    internal bool threadSuspended=true;//if wait for other player
157
158      // constructor requiring Socket, TicTacToeServerForm and int
159      // objects as arguments
160    public Player( Socket socket, TicTacToeServerForm serverValue,
161         int newNumber )
162      {
163         mark = (newNumber == 0 ? 'X' : 'O');
164         connection = socket;
165         server = serverValue;
166         number = newNumber;
167
168         // create NetworkStream object for Socket
169         socketStream = new NetworkStream( connection );
170
171         // create Streams for reading/writing bytes
172         writer = new BinaryWriter( socketStream );
173         reader = new BinaryReader( socketStream );
174      } // end constructor
175
176      // signal other player of move
177      public void OtherPlayerMoved( int location )
178      {
179         // signal that opponent moved
180         writer.Write( "Opponent moved." );
181         writer.Write( location ); // send location of move
182      } // end method OtherPlayerMoved
183
184      // allows the players to make moves and receive moves
185      // from the other player
186      public void Run()
187      {
188         bool done = false;
189
190         // display on the server that a connection was made
191    server.DisplayMessage( "Player " + ( number == 0 ? 'X' : 'O' )
192            + " connected\r\n" );
```

```
193
194         // send the current player's mark to the client
195         writer.Write( mark );
196
197         // if number equals 0 then this player is X,
198         // otherwise O must wait for X's first move
199         writer.Write( "Player " + ( number == 0 ?
200         "X connected.\r\n" : "O connected, please wait.\r\n" ) );
201
202         // X must wait for another player to arrive
203         if ( mark == 'X' )
204         {
205            writer.Write( "Waiting for another player." );
206
207            // wait for notification from server that another
208            // player has connected
209            lock ( this )
210            {
211               while ( threadSuspended )
212                  Monitor.Wait( this );
213            } // end lock
214
215            writer.Write( "Other player connected. Your move." );
216         } // end if
217
218         // play game
219         while ( !done )
220         {
221            // wait for data to become available
222            while ( connection.Available == 0 )
223            {
224               Thread.Sleep( 1000 );
225
226               if ( server.disconnected )
227                  return;
228            } // end while
229
230            // receive data
231            int location = reader.ReadInt32();
232
233            // if the move is valid, display the move on the
234            // server and signal that the move is valid
235            if ( server.ValidMove( location, number ) )
236            {
237             server.DisplayMessage( "loc: " + location + "\r\n" );
238               writer.Write( "Valid move." );
239            } // end if
240            else // signal that the move is invalid
241               writer.Write( "Invalid move, try again." );
242
243            // if game over, set done to true to exit while loop
244            if ( server.GameOver() )
245               done = true;
246         } // end while loop
247
248         // close the socket connection
249         writer.Close();
250         reader.Close();
251         socketStream.Close();
```

```
252         connection.Close();
253      } // end method Run
254   } // end class Player
```

يستخدم الصف **TicTacToeServerForm** معالج الحدث **Load** لبناء مصفوفة **board** من البايتات لتخزين الحركات التي يقوم بها كل لاعب **(**السطر **29)**.

كما يقوم ببناء مصفوفة من عنصرين من النمط **Player (**السطر **30)** ومصفوفة من عنصرين من النمط **Thread (**السطر **31)**. يمثل كل عنصر من المصفوفتين السابقتين لاعب في البرنامج.

يتم تهيئة المتحول **currentPlayer** بالقيمة **0 (**السطر **32)** ويعبر هذا المتحول عن اللاعب **"X"** وهو الذي يقوم بالحركة الأولى.

تقوم الأسطر **35-36** ببناء وتشغيل النيسب **getPlayers** والذي سيستخدمه الصف **TicTacToeServerForm** لقبول طلبات الاتصال، وبهذا فلا يتم تجميد النيسب الحالي بينما يتم انتظار الاعبين.

تُعرف الأسطر **49-65** كل من **DisplayDelegate** و **DisplayMessage** لتسمح لأي نيسب بالتعديل على الخاصية **Text** للعنصر **displayTextBox** وذلك بشكل آمن. تُعرف الطريقة **DisplayMessage** على أنها **internal** وبهذا يمكن استدعاءها فقط داخل طريقة من الصف **Player.**

ينفذ النيسب **getPlayer** الطريقة **SetUp (**الأسطر **68-95)**، والتي تقوم ببناء الغرض **TcpListener**، والذي يقوم بدوره بالاستماع لطلبات الاتصال من اللاعب الأول واللاعب الثاني على المنفذ **50000 (**الأسطر **73-75)**.

تقوم الأسطر **78 و 84** بتهيئة المصفوفة **Player** (من عنصرين) والتي تُمثل اللاعبين.
كما تقوم الأسطر **79-81 و 85-87** ببناء نيسبين يتم من خلالهما تنفيذ الطريقة **Run** لكل عنصر من المصفوفة **Player.**

يتم تمرير ثلاثة معاملات لباني الصف **Player** (الأسطر **68-174**):

- المعامل الأول هو مرجع إلى الغرض **Socket**.
- والمعامل الثاني هو مرجع للغرض **TicTacToeServerForm**.
- والمعامل الثالث هو رقم من النمط **int** يمثل رقم اللاعب يمكن من خلال هذا الرقم استنتاج الخطوة التي لعبها اللاعب إن كانت "**X**" (قيمة المعامل **0**) أو "**O**" (قيمة المعامل **1**).

يستدعي **TicTacToeServerForm** الطريقة **Run** (الأسطر **186-253**) بعد تهيئة الغرض **Player**. تُعلم الأسطر **191-200** المخدم بتأسيس اتصال ناجح وإرسال محرف إلى الزبون ليقوم بوضعه على اللوحة عند قيامه بحركة.

إذا تم تنفيذ الطريقة **Run** للاعب "**X**"، تُنفذ الأسطر **205-215** ليتم انتظار إتمام اتصال اللاعب الثاني.

تُعرف الأسطر **211-212** حلقة **while** تقوم بوقف تنفيذ نيسب اللاعب "**X**" حتى يقوم المخدم بإعلام اللاعب "**O**" بأنه متصل. يتم معرفة أن اللاعب متصل من خلال إسناد القيمة **False** إلى المتحول **threadSuspended** من الصف **Player** (السطر **92**). عندما تصبح قيمة المتحول **threadSuspended** تساوي **False** يتم إنهاء الحلقة **while** في الأسطر **211-212**.

تُنفذ الطريقة **Run** حلقة **while** في الأسطر **219-224** ممكنةً المستخدم من اللعب. تنتظر كل دورة من هذه الحلقة من المستخدم أن يرسل رقم من النمط **int** محدداً مكان اللعب على اللوحة ليتم وضع "**X**" أو "**O**". عند وضع اللاعب العلامة على لوحة التطبيق، يتم اختبار صلاحية هذه الحركة (لا يحوي المربع مسبقاً أي علامة).

يمكن ملاحظة أن الحلقة **while** تستمر في التنفيذ ما دامت قيمة المتحول المنطقي **done** هي **False**. يتم تعديل قيمة هذا المتحول إلى القيمة **True** عن طريق معالج الحدث **TicTacToeServerForm_FormClosing** والذي يتم استدعائه عند إغلاق المخدم للاتصال.
يبدأ السطر **222** بتنفيذ حلقة **while** ويستمر تنفيذ هذه الحلقة حتى تشير الخاصية **Available** بأن هنالك معلومات يجب استلامها من المقبس. إذا لم يكن هنالك أية معلومات يذهب النيسب إلى الحالة **sleep** لمدة ثانية واحدة.

يستخدم النسيب عند إيقاظه الخاصية **Disconnectd** لفحص فيما إذا كانت قيمة متحول المخدم **Disconnectd** مساوية **True**. إذا كانت القيمة مساوية **True** يتم إنهاء النيسب وإلا تستمر الحلقة في التنفيذ.

إذا كانت قيمة الخاصية **Available** تشير إلى وجود معلومات يجب استقبالها، فعندها يتم إنهاء تنفيذ الحلقة **while** في الأسطر **222-228** ليتمكن البرنامج من معالجة هذه البيانات. تحوي هذه البيانات على رقم من النمط **int** يمثل المكان الذي اختاره الزبون لوضع العلامة.

يستدعي السطر **231** الطريقة **Readint32** من الصف **BinaryReader** لتقوم بقراءة هذا الرقم. يقوم السطر **235** بعدها بتمرير هذا الرقم إلى الطريقة **ValidMove** من الصف **TicTacToeServerForm**. تقوم هذه الطريقة بفحص صلاحية هذه الحركة، فإذا كانت ممكنة يقوم الصف **Player** بوضع العلامة في المكان المحدد. تُرسل الطريقة **ValidMove** (الأسطر **98-127**) رسالة إلى الزبون مشيرة إلى إمكانية هذه الحركة.
تُمثل الأماكن على اللوحة في واجهة المستخدم بالأرقام من **0** إلى **8**. تُغلف كل تعليمات الطريقة **ValidMove** بالتعليمة **lock** والتي تسمح بمحاولة لعب حركة واحدة فقط في نفس الوقت.

هذا يمنع اللاعبين من تعديل معلومات الحالة الحالية للعبة بصورة متزامنة. عند محاولة لاعب فحص إمكانية الحركة وهو ليس اللاعب الحالي (أي ليس دوره)، يتم وضع هذا اللاعب في حالة انتظار ريثما يحين دوره للعب.

إذا حاول المستخدم وضع علامة في مكان مشغول سابقاً، تقوم الطريقة **ValidMove** بإعادة القيمة **False**. إذا قام المستخدم بوضع العلامة في مكان خال (السطر **108**)، تقوم الأسطر **111-112** بوضع العلامة بتمثيلها الصحيح على لوح واجهة المستخدم. يعلم السطر **118** اللاعب الأخر بإتمام الحركة من اللاعب الأول، كما يستدعي السطر **121** الطريقة **Pluse** بحيث يمكن اللاعب المنتظر من أن يقوم بالحركة.

الزبون

يقوم **TicTacToeServerClient** ببناء مربع نص لإظهار الرسائل المرسلة من قبل المخدم وإظهار اللعبة على اللوحة.

تتكون واجهة المستخدم من تسعة أغرض من الصف **Square**، وتتضمن هذه الأغراض ألواح **Panals** والتي سيقوم المستخدم بالنقر عليها مشيراً إلى الموقع الذي سيضع عليه العلامة عند إجراء الحركة.

يقوم معالج الحدث **Load** للصف **TicTacToeServerClient** بفتح اتصال مع المخدم **(**السطر **50)** ويحتفظ بمرجع للاتصال المرتبط مع الغرض **NetworkStream** من الصف **TcpClient** **(**السطر **51)**.

تبدأ الأسطر **56-57** بتشغيل نيسب لقراءة الرسائل المرسلة من المخدم إلى الزبون.

يمرر المخدم الرسائل إلى الطريقة **ProcessMessage** **(**الأسطر **179-215)**. إذا كانت الرسالة تشير إلى إمكانية إجراء الحركة **(**السطر **184)** يقوم الزبون بوضع العلامة في المربع الحالي (المربع الذي نقر عليه المستخدم) ثم يقوم بإعادة رسم الواجهة لإظهار التعديل. إذا كانت الرسالة تشير إلى عدم إمكانية القيام بالحركة **(**السطر **190)** يطلب الزبون من المستخدم النقر على مربع آخر. إذا كانت الرسالة تشير إلى حركة مقابلة **(**السطر **197)**، يقوم السطر **200** بقراءة رقم من المخدم يحدد المكان على لوحة الزبون لوضع العلامة المقابلة.

```
1 // TicTacToeClient.cs
2 // Client for the TicTacToe program.
3 using System;
4 using System.Drawing;
5 using System.Windows.Forms;
6 using System.Net.Sockets;
7 using System.Threading;
8 using System.IO;
9
10 public partial class TicTacToeClientForm : Form
11 {
12   public TicTacToeClientForm()
13   {
14      InitializeComponent();
15   } // end constructor
16
17   private Square[ , ] board;// representation of the game board
18   private Square currentSquare; //the Square that this player chose
19   private Thread outputThread;//Thread for receiving data server
20   private TcpClient connection; // client to establish connection
21   private NetworkStream stream; // network data stream
22   private BinaryWriter writer; // facilitates writing to the stream
23   private BinaryReader reader; // facilitates reading the stream
24   private char myMark; // player's mark on the board
25   private bool myTurn; // is it this player's turn?
26   private SolidBrush brush; // brush for drawing X's and O's
27   private bool done = false; // true when game is over
28
29   // initialize variables and thread for connecting to server
30   private void TicTacToeClientForm_Load(object sender,EventArgs e)
31   {
32      board = new Square[ 3, 3 ];
33
34      // create 9 Square objects and place them on the board
35      board[ 0, 0 ] = new Square( board0Panel, ' ', 0 );
36      board[ 0, 1 ] = new Square( board1Panel, ' ', 1 );
37      board[ 0, 2 ] = new Square( board2Panel, ' ', 2 );
```

```
38          board[ 1, 0 ] = new Square( board3Panel, ' ', 3 );
39          board[ 1, 1 ] = new Square( board4Panel, ' ', 4 );
40          board[ 1, 2 ] = new Square( board5Panel, ' ', 5 );
41          board[ 2, 0 ] = new Square( board6Panel, ' ', 6 );
42          board[ 2, 1 ] = new Square( board7Panel, ' ', 7 );
43          board[ 2, 2 ] = new Square( board8Panel, ' ', 8 );
44
45          // create a SolidBrush for writing on the Squares
46          brush = new SolidBrush( Color.Black );
47
48          // make connection to server and get the associated
49          // network stream
50          connection = new TcpClient( "127.0.0.1", 50000 );
51          stream = connection.GetStream();
52          writer = new BinaryWriter( stream );
53          reader = new BinaryReader( stream );
54
55          // start a new thread for sending and receiving messages
56          outputThread = new Thread( new ThreadStart( Run ) );
57          outputThread.Start();
58       } // end method TicTacToeClientForm_Load
59
60       // repaint the Squares
61       private void TicTacToeClientForm_Paint( object sender,
62          PaintEventArgs e )
63       {
64          PaintSquares();
65       } // end method TicTacToeClientForm_Load
66
67       // game is over
68       private void TicTacToeClientForm_FormClosing( object sender,
69          FormClosingEventArgs e )
70       {
71          done = true;
72          System.Environment.Exit( System.Environment.ExitCode );
73       } // end TicTacToeClientForm_FormClosing
74
75       // delegate that allows method DisplayMessage to be called
76       // in the thread that creates and maintains the GUI
77       private delegate void DisplayDelegate( string message );
78
79       // method DisplayMessage sets displayTextBox's Text property
80       // in a thread-safe manner
81       private void DisplayMessage( string message )
82       {
83          // if modifying displayTextBox is not thread safe
84          if ( displayTextBox.InvokeRequired )
85          {
86             // use inherited method Invoke to execute DisplayMessage
87             // via a delegate
88             Invoke( new DisplayDelegate( DisplayMessage ),
89                new object[] { message } );
90          } // end if
91          else // OK to modify displayTextBox in current thread
92             displayTextBox.Text += message;
93       } // end method DisplayMessage
94
95       // delegate that allows method ChangeIdLabel to be called
96       // in the thread that creates and maintains the GUI
```

```
97   private delegate void ChangeIdLabelDelegate( string message );
98
99   // method ChangeIdLabel sets displayTextBox's Text property
100   // in a thread-safe manner
101   private void ChangeIdLabel( string label )
102   {
103      // if modifying idLabel is not thread safe
104      if ( idLabel.InvokeRequired )
105      {
106         // use inherited method Invoke to execute ChangeIdLabel
107         // via a delegate
108         Invoke( new ChangeIdLabelDelegate( ChangeIdLabel ),
109            new object[] { label } );
110      } // end if
111      else // OK to modify idLabel in current thread
112         idLabel.Text = label;
113   } // end method ChangeIdLabel
114
115   // draws the mark of each square
116   public void PaintSquares()
117   {
118      Graphics g;
119
120      // draw the appropriate mark on each panel
121      for ( int row = 0; row < 3; row++ )
122      {
123         for ( int column = 0; column < 3; column++ )
124         {
125            // get the Graphics for each Panel
126            g = board[ row, column ].SquarePanel.CreateGraphics();
127
128            // draw the appropriate letter on the panel
129            g.DrawString( board[ row, column ].Mark.ToString(),
130               board0Panel.Font, brush, 10, 8 );
131         } // end for
132      } // end for
133   } // end method PaintSquares
134
135   // send location of the clicked square to server
136   private void square_MouseUp( object sender,
137      System.Windows.Forms.MouseEventArgs e )
138   {
139      // for each square check if that square was clicked
140      for ( int row = 0; row < 3; row++ )
141      {
142         for ( int column = 0; column < 3; column++ )
143         {
144            if ( board[ row, column ].SquarePanel == sender )
145            {
146               CurrentSquare = board[ row, column ];
147
148               // send the move to the server
149               SendClickedSquare( board[ row, column ].Location );
150            } // end if
151         } // end for
152      } // end for
153   } // end method square_MouseUp
154
155   // control thread that allows continuous update of the
```

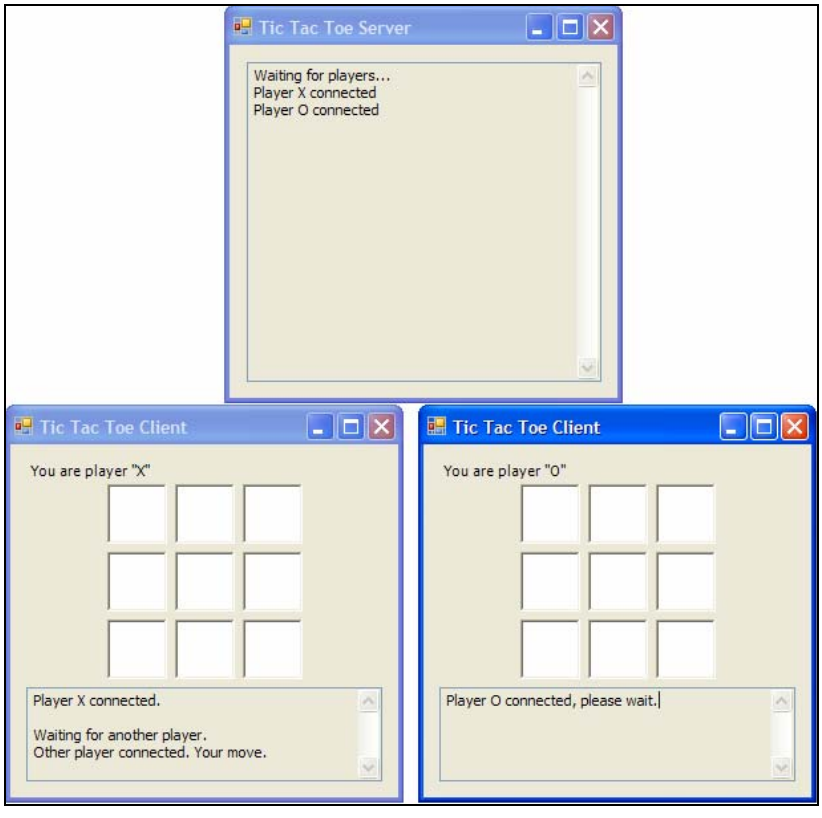
```
156   // TextBox display
157   public void Run()
158   {
159      // first get players's mark (X or O)
160      myMark = reader.ReadChar();
161      ChangeIdLabel( "You are player \"" + myMark + "\"" );
162      myTurn = ( myMark == 'X' ? true : false );
163
164      // process incoming messages
165      try
166      {
167         // receive messages sent to client
168         while ( !done )
169            ProcessMessage( reader.ReadString() );
170      } // end try
171      catch ( IOException )
172      {
173         MessageBox.Show( "Server is down, game over", "Error",
174            MessageBoxButtons.OK, MessageBoxIcon.Error );
175      } // end catch
176   } // end method Run
177
178   // process messages sent to client
179   public void ProcessMessage( string message )
180   {
181      // if the move the player sent to the server is valid
182     // update the display, set that square's mark to be
183      // the mark of the current player and repaint the board
184      if ( message == "Valid move." )
185      {
186         DisplayMessage( "Valid move, please wait.\r\n" );
187         currentSquare.Mark = myMark;
188         PaintSquares();
189      } // end if
190      else if ( message == "Invalid move, try again." )
191      {
192         // if the move is invalid, display that and it is now
193         // this player's turn again
194         DisplayMessage( message + "\r\n" );
195         myTurn = true;
196      } // end else if
197      else if ( message == "Opponent moved." )
198      {
199         // if opponent moved, find location of their move
200         int location = reader.ReadInt32();
201
202         // set that square to have the opponents mark and
203         // repaint the board
204         board[ location / 3, location % 3 ].Mark =
205            ( myMark == 'X' ? 'O' : 'X' );
206         PaintSquares();
207
208         DisplayMessage( "Opponent moved.  Your turn.\r\n" );
209
210         // it is now this player's turn
211         myTurn = true;
212      } // end else if
213      else
214         DisplayMessage( message + "\r\n" ); // display message
```

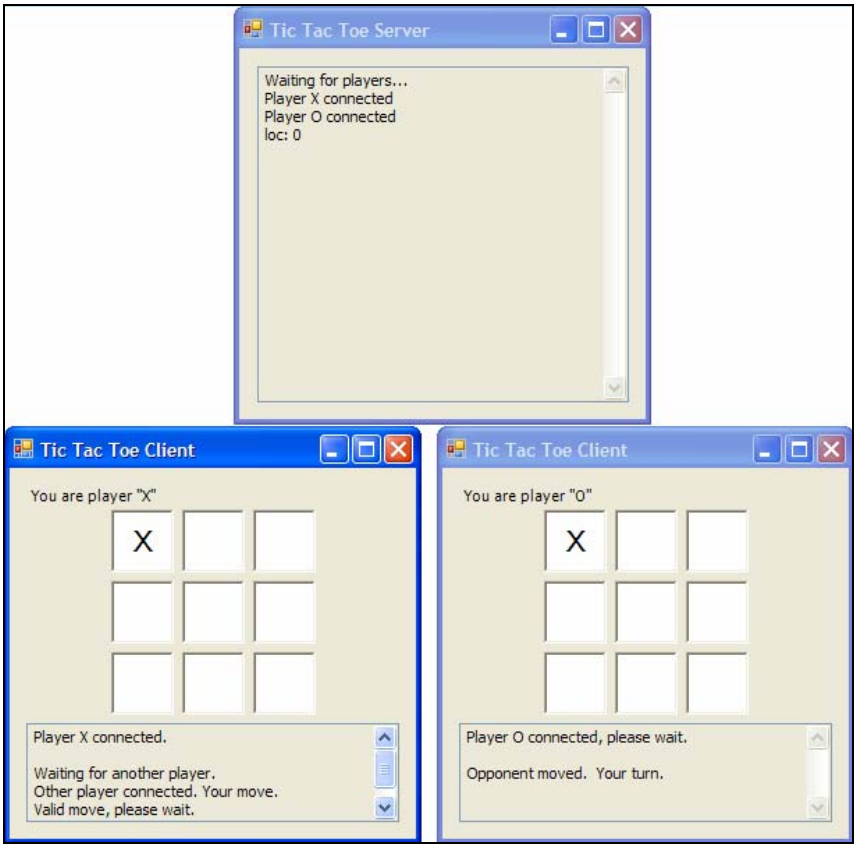
```
215    } // end method ProcessMessage
216
217    // sends the server the number of the clicked square
218    public void SendClickedSquare( int location )
219    {
220       // if it is the current player's move right now
221       if ( myTurn )
222       {
223          // send the location of the move to the server
224          writer.Write( location );
225
226          // it is now the other player's turn
227          myTurn = false;
228       } // end if
229    } // end method SendClickedSquare
230
231    // write-only property for the current square
232    public Square CurrentSquare
233    {
234       set
235       {
236          currentSquare = value;
237       } // end set
238    } // end property CurrentSquare
239 } // end class TicTacToeClientForm
```

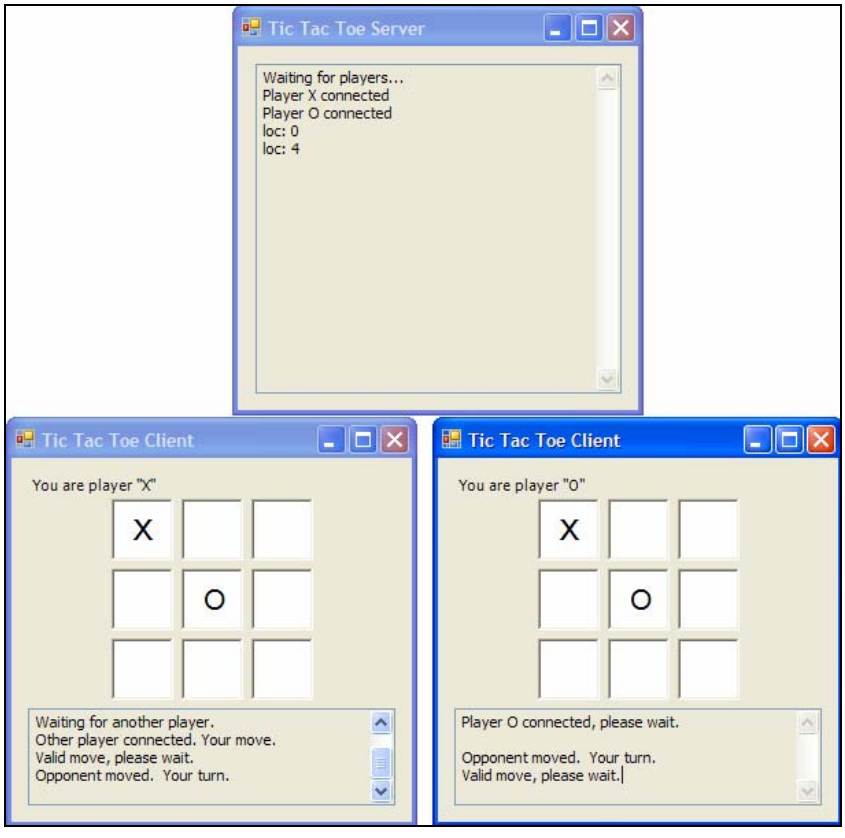
```
1     // Square.cs
2     // A Square on the TicTacToe board.
3     using System.Windows.Forms;
4
5     // the representation of a square in a tic-tac-toe grid
6     public class Square
7     {
8        private Panel panel; // GUI Panel that represents this Square
9        private char mark; // player's mark on this Square (if any)
10       private int location; // location on the board of this Square
11
12       // constructor
13       public Square(Panel newPanel, char newMark, int newLocation )
14       {
15          panel = newPanel;
16          mark = newMark;
17          location = newLocation;
18       } // end constructor
19
20       // property SquarePanel;the panel which the square represents
21       public Panel SquarePanel
22       {
23          get
24          {
25             return panel;
26          } // end get
27       } // end property SquarePanel
28
29       // property Mark; the mark on the square
30       public char Mark
31       {
32          get
```

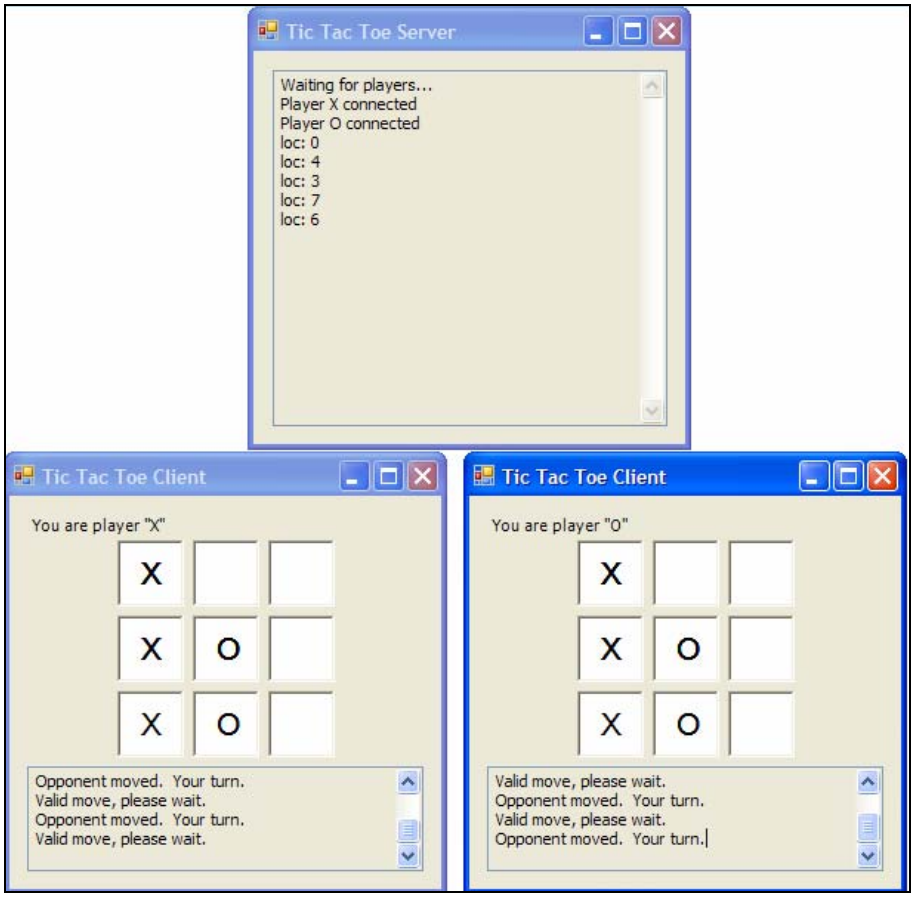
```
33          {
34             return mark;
35          } // end get
36          set
37          {
38             mark = value;
39          } // end set
40       } // end property Mark
41
42       // property Location; the square's location on the board
43       public int Location
44       {
45          get
46          {
47             return location;
48          } // end get
49       } // end property Location
50    } // end class Square
```

تُبين الأشكال التالية تنفيذ البرنامج السابق مع لاعبين:




Tic Tac Toe Server
Waiting for players...
Player X connected
Player O connected
loc: 0
Tic Tac Toe Client
You are player "X"
X
Player X connected.
Waiting for another player.
Other player connected. Your move.
Valid move, please wait.
Tic Tac Toe Client
You are player "O"
X
Player O connected, please wait.
Opponent moved. Your turn.


Tic Tac Toe Server
Waiting for players...
Player X connected
Player O connected
loc: 0
loc: 4
Tic Tac Toe Client
You are player "X"
X
O
Waiting for another player.
Other player connected. Your move.
Valid move, please wait.
Opponent moved. Your turn.
Tic Tac Toe Client
You are player "O"
X
O
Player O connected, please wait.
Opponent moved. Your turn.
Valid move, please wait.



9. عنصر التحكم WebBrowser

تتضمن **FCL 2.0** عنصر التحكم **WebBrowser** والذي يمكّن التطبيق من دمج إمكانات استعراض الويب. يحوي عنصر التحكم **WebBrowser** طرائق لتصفح الوب والاحتفاظ بعناوين مواقع الويب التي يتصفحها المستخدم. كما يقدم إمكانية تفاعل المستخدم مع مكونات الصفحة المعروضة ضمن هذا العنصر. وبهذا يمكن للتطبيق الاستجابة والتفاعل مع المستخدم عند النقر على رابط ضمن صفحة الانترنت مثلا.

يعرض الشكل التالي إمكانات عنصر التحكم **WebBrowser**. يزود الصف **BrowserForm** الوظائف الأساسية لهذا العنصر.

مثلاً:

السماح للمستخدم باستعراض عنوان موقع على شبكة الانترنت **(URL)** والتنقل بين الصفحات التي تمت زيارتها مسبقاً أو تحديث الصفحة الحالية.


```
1     // Browser.cs
2     // WebBrowser control example.
3     using System;
4     using System.Windows.Forms;
5
6     public partial class BrowserForm : Form
7     {
8        public BrowserForm()
9        {
10          InitializeComponent();
11       } // end constructor
12
13       // navigate back one page
14       private void backButton_Click( object sender, EventArgs e )
15       {
16          webBrowser.GoBack();
17       } // end method backButton_Click
18
19       // navigate forward one page
20       private void forwardButton_Click( object sender, EventArgs e )
```

```
21        {
22           webBrowser.GoForward();
23        } // end method forwardButton_Click
24
25        // stop loading the current page
26        private void stopButton_Click( object sender, EventArgs e )
27        {
28           webBrowser.Stop();
29        } // end method stopButton_Click
30
31        // reload the current page
32        private void reloadButton_Click( object sender, EventArgs e )
33        {
34           webBrowser.Refresh();
35        } // end method reloadButton_Click
36
37        // navigate to the user's home page
38        private void homeButton_Click( object sender, EventArgs e )
39        {
40           webBrowser.GoHome();
41        } // end method homeButton_Click
42
43        // if the user pressed enter, navigate to the specified URL
44        private void navigationTextBox_KeyDown( object sender,
45           KeyEventArgs e )
46        {
47           if ( e.KeyCode == Keys.Enter )
48              webBrowser.Navigate( navigationTextBox.Text );
49        } // end method navigationTextBox_KeyDown
50
51        // enable stopButton while the current page is loading
52        private void webBrowser_Navigating( object sender,
53           WebBrowserNavigatingEventArgs e )
54        {
55           stopButton.Enabled = true;
56        } // end method webBrowser_Navigating
57
58        // update the status text
59        private void webBrowser_StatusTextChanged( object sender,
60           EventArgs e )
61        {
62           statusTextBox.Text = webBrowser.StatusText;
63        } // end method webBrowser_StatusTextChanged
64
65        // update the ProgressBar
66        private void webBrowser_ProgressChanged( object sender,
67           WebBrowserProgressChangedEventArgs e )
68        {
69           pageProgressBar.Value =
70              (int) ((100 * e.CurrentProgress ) / e.MaximumProgress );
71        } // end method webBrowser_ProgressChanged
72
73        // update the web browser's controls appropriately
74        private void webBrowser_DocumentCompleted( object sender,
75           WebBrowserDocumentCompletedEventArgs e )
76        {
77          //set the text in navigationTextBox to the current page's URL
78           navigationTextBox.Text = webBrowser.Url.ToString();
79
```

```
80          // enable or disable backButton and forwardButton
81          backButton.Enabled = webBrowser.CanGoBack;
82          forwardButton.Enabled = webBrowser.CanGoForward;
83
84          // disable stopButton
85          stopButton.Enabled = false;
86
87          // clear the pageProgressBar
88          pageProgressBar.Value = 0;
89       } // end method webBrowser_DocumentCompleted
90
91       // update the title of the Browser
92       private void webBrowser_DocumentTitleChanged( object sender,
93          EventArgs e )
94       {
95          this.Text = webBrowser.DocumentTitle + " - Browser";
96       } // end method webBrowser_DocumentTitleChanged
97    } // end class BrowserForm
```

تُعرف الأسطر **14-41** معالج الحدث **Click** لكل من أزرار الاستعراض الخمسة الموجودة أعلى الشكل. يقوم كل معالج حدث باستدعاء طريقة من طرق عنصر التحكم **WebBrowser** الموافقة له.

- تنقل الطريقة **GoBack** (السطر **16**) عنصر التحكم لاستعراض الصفحة السابقة.
- تقوم الطريقة **GoForward** (السطر **22**) بالانتقال إلى الصفحة التالية التي تمت زيارتها من قبل المستخدم.
- توقف الطريقة **Stop** (السطر **28**) تحميل الصفحة الحالية.
- تُعيد الطريقة **Refresh** (السطر **34**) تحميل الصفحة الحالية.
- تقوم الطريقة **GoHome** باستعراض الصفحة الافتراضية.

يسمح مربع النص الموجود إلى يمين أزرار التحكم للمستخدم بإدخال عنوان موقع على شبكة الانترنت **URL**. عندما يقوم المستخدم بكتابة العنوان في هذا الحقل ومن ثم النقر على **Enter**، يتم تنفيذ معالج الحدث في الأسطر **44-49**. يستدعي السطر **48** الطريقة **Navigate** لتحميل الصفحة وفق العنوان المدخل.

يقوم **WebBrowser** بتوليد الحدث **Navigating** عندما يبدأ بتحميل صفحة جديدة، وعندها يتم تنفيذ معالج الحدث في الأسطر **52-56**. يقوم السطر **55** بتفعيل **stopButton** بحيث يمكن للمستخدم وقف تحميل الصفحة.

عادةً، يستطيع المستخدم رؤية حالة تحميل الصفحة في أسفل واجهة المتصفح. لهذا السبب نضع العنصرين **TextBox** و **ProgressBar** في أسفل النموذج ولهما الأسماء **StatusTextBox** و

PageProgressBar على التوالي. يقوم **WebBrowser** بتوليد الحدث **StatusTextChanged+** وذلك عند تغيير الخاصية **StatusText**، يقوم معالج الحدث الخاص بالحدث المولد **(الأسطر 59-63)** بوضع محتويات الخاصية **StatusText** ضمن الخاصية **Text** للعنصر **StatusTextBox (السطر 62)**، وبهذا يستطيع المستخدم مشاهدة الرسائل الناتجة عن حالة **WebBrowser**. يقوم معالج الحدث **ProgressChanged (الأسطر 66-71)** بتحديث قيمة الخاصية **Value** للعنصر **PageProgressBar** ليعكس الحجم الذي تم تحميله من الصفحة المطلوبة.

عند انتهاء **WebBrowser** من تحميل الصفحة، يقوم بتوليد الحدث **DocumentCompleted**. ينتج عن ذلك تنفيذ معالج الحدث في الأسطر **74-89**.

يقوم السطر **78** بتحديث محتويات العنصر **NavigationTextBox** بحيث يظهر العنوان الجاري تحميله وهذا ضروري خصوصاً في حال النقر من قبل المستخدم على ارتباط موجود ضمن الصفحة الحالية.

تستخدم الأسطر **81-82** الخصائص **CanGoBack** و **CanGoForward** للتحكم بتفعيل وإلغاء تفعيل الأزرار أثناء استعراض الصفحات. عند إتمام تحميل الصفحة فإنه يتم إلغاء تفعيل الزر **StopButton**.

يقوم السطر **88** بإسناد القيمة **0** للخاصية **Value** لتشير إلى انتهاء تحميل الصفحة.

تُعرف الأسطر **92-96** معالج الحدث للحدث **DocumentTitleChanged** والذي يتم تنفيذه عند تحميل صفحة جديدة.

10. التشبيك عن بعد

تحوي منصة **.NET** تقانات معالجة موزعة تُدعى التشبيك عن بعد **.NET Remoting**. تسمح للبرنامج بالنفاذ إلى أغراض موجودة على حواسب أخرى عبر الشبكة.

يشابه التشبيك عن بعد طريقة الاستدعاء عن بعد **RMI Remote Method Invocation** في **Java** واستدعاء الإجرائيات عن بعد **RPC Remote Procedure Call** في لغات البرمجة الإجرائية. كما يشابه أيضا خدمات الوب **Web Services** المعروضة في الفصل القادم مع بعض الاختلافات. إذ يقوم تطبيق الزبون في خدمات الويب بالاتصال مع خدمة الويب المستضافة من قبل مخدم الويب.

يمكن أن يكتب الزبون وخدمة الويب باستخدام لغات برمجية مختلفة طالما يمكنهما إرسال رسائل باستخدام البروتوكول **SOAP**. أما في التشبيك عن بعد، فيقوم تطبيق الزبون بإجراء الاتصال مع تطبيق المخدم، إلا أن كل منهما يجب أن يكون مكتوباً باستخدام**NET.** .

عند استخدام التشبيك عن بعد **NET.** يقوم كل من الزبون والمخدم بإجراء الاتصال بينهما عن طريق استدعاء الطرائق وإرسال الأغراض بين تطبيقيهما. تُدعى هذه العملية الإدارة بالأغراض **Marshaling.**

وتشمل:

1. القنوات **Channels.**
2. الإدارة بالأغراض **Marshaling.**
3. تطبيق.

القنوات Channels

يمكن للمخدم والزبون الاتصال وتبادل المعلومات باستخدام طريقة القنوات.

تستخدم القنوات إما البروتوكول **HTTP** أو البروتوكول **TCP** لإرسال الرسائل. تمتاز قناة **HTTP** في أن جدران النار تسمح لها بالاتصال بشكل افتراضي، بينما تقوم بتجميد الاتصالات **TCP** غير المألوفة. إلا أن القناة **TCP** تُعتبر أفضل أداء من قناة **HTTP.**

يجب في تطبيق التشبيك عن بعد التي تقدمه.**NET** أن يقوم كل من الزبون والمخدم بفتح قناتين تستخدمان البروتوكول نفسه للاتصال وتبادل المعطيات بينهما. نستخدم في مثالنا البروتوكول **HTTP.**

الإدارة بالأغراض Marshaling

يوجد طريقتين لإدارة غرض، وذلك عن طريق القيمة **(by value)** أو عن طريق المرجع **(by reference).**

تتطلب إدارة غرض عن طريق القيمة من الغرض أن يكون قابلاً للسلسلة **Serializable**. بحيث يوفر إمكانية تمثيله كرسالة موحدة يمكن أن تُرسل بين التطبيقات عبر القناة. تقوم الجهة المستقبلة في الجهة المقابلة من القناة بإجراء العملية العكسية للسلسلة **Deserialize** للغرض لتشكيل نسخة من الغرض الأصلي.

ولتمكين غرض من أن يمتلك الخاصية **Serialized** و **Deserialized** يجب تزويد الصف الخاص بهذا الغرض بالصفة **[Serializable]** أو تزويده بالواجهة **ISerializable (interface)**.

تتطلب إدارة غرض عن طريق المرجع من الصف الخاص بالغرض التوسيع **(extend)** من الصف **MarshalByRefObject** من فضاء الأسماء **System**.

يشار إلى الغرض المدار عن طريق المرجع بالغرض البعيد **(remote object)**، وإلى الصف الذي ينتمي إليه بالصف البعيد **(remote class)**.

لا يتم إرسال الغرض نفسه عند استخدام الإدارة عن طريق المرجع، بل يتم بناء غرضين وكيلين يدعى الأول بالوكيل الشفاف **transparent proxy** والثاني بالوكيل الحقيقي **real proxy**. يزود الوكيل الأول الزبون بكل الخدمات العامة **(public)** للغرض البعيد.

فعلياً، عندما يستدعي الزبون طريقة أو خاصية من الوكيل الأول يبدو كما لو أنه يستدعيها من الغرض البعيد. وعندها يستدعي الوكيل الأول الطريقة **Invoke** للوكيل الثاني الذي يرسل الرسالة المناسبة من قناة الزبون إلى قناة المخدم. يستلم المخدم الرسالة ويقوم بتنفيذ استدعاء الطريقة المطلوبة أو الوصول إلى الخاصية المحددة للغرض الفعلي الموجود على المخدم.

تطبيق

يتألف التطبيق المثال التالي، والذي يهدف إلى عرض معلومات عن حالة الطقس باستخدام التشبيك عن بعد، من خمسة مكونات:

1. الصف القابل للسلسلة **CityWeather** والذي يمثل تقرير الطقس لمدينة واحدة.
2. الواجهة **Report** والتي تُعرف الخاصية **Reports** وهي غرض مدار يحتفظ بمجموعة من الأغراض من الصف **CityWeather** يتم تزويد تطبيق الزبون به.
3. الصف **ReportInfo** الذي يوسع الصف **MarshalByRefObject** ويرث الواجهة **Report** وسيتم تهيئته فقط في المخدم.
4. تطبيق المخدم **WeatherServer** الذي سيقوم بإعداد قناة المخدم وجعل الصف **RoportInfo** متاحاً ضمن عنوان **URL** خاص.
5. تطبيق الزبون **WeatherClient** الذي سيقوم بإعداد الزبون ويطلب غرض **ReportInfo** من المخدم للحصول على معلومات الطقس ليوم محدد.

يستخدم هذا التطبيق الموقع **Traveler's Foracast** التالي الذي يزود المعلومات عن حالة الطقس:

http://iwin.nws.noaa.gov/iwin/us/traveler.html

الصف CityWeather

يحتوي الصف **CityWeather** على معلومات الطقس لمدينة واحدة. يتم التصريح عن هذا الصف ضمن فضاء الأسماء **Weahter** (السطر **5**)، كما وسينشر ضمن ملف مكتبة الصف **Weaher.dll** والتي يجب الإشارة إليها **reference** ضمن تطبيق المخدم وتطبيق الزبون.

ولهذا فيجب وضع هذا الصف (والواجهة **Report**) ضمن مشروع مكتبة الصف **class library project**. يتم التصريح عن الصف **CityWeather** باستخدام الواصفة **[Serializable]** (**السطر 7**) والتي تشير إلى أن الغرض من الصف **CityWeather** يمكن أن يدار عن طريق القيمة. وهو أمر ضروري لأن الأغراض من الصف **CityWeather** سيتم إعادتها عن طريق الخاصية **Reports** من الصف **ReportInfo**. يجب أن تتم إدارة القيم المعادة من الطرائق والخصائص المصرح عنها في الصف البعيد عن طريق القيمة عند إرسالها من المخدم إلى الزبون.

وبهذا فعندما يقوم الزبون باستدعاء طريقة النفاذ **get** التي تُعيد غرض **CityWeather**، ستقوم قناة المخدم بسلسلة **serialize** للغرض في رسالة، يمكن للزبون أن يعكس هذه السلسلة **deserialize** لبناء نسخة من الغرض الأصلي. يورث الصف **CityWeather** من الصف **IComparable** (السطر **8**) بحيث يمكن للخاصية من النمط **ArrayList** التي ستحوي أغراض من الصف **CityWeather** ترتيبها أبجديا.

```
1 // CityWeather.cs
2 // Class representing the weather information for one city.
3 using System;
4
5 namespace Weather
6 {
7   [ Serializable ]
8   public class CityWeather : IComparable
9   {
10      private string cityName;
11      private string description;
12      private string temperature;
13
14      public CityWeather( string city, string information,
15          string degrees )
16      {
17         cityName = city;
18         description = information;
19         temperature = degrees;
20      } // end constructor
```

```
21
22      // read-only property that gets city's name
23      public string CityName
24      {
25         get
26         {
27            return cityName;
28         } // end get
29      } // end property CityName
30
31      // read-only property that gets city's weather description
32      public string Description
33      {
34         get
35         {
36            return description;
37         } // end get
38      } // end property Description
39
40      // read-only property that gets city's temperature
41      public string Temperature
42      {
43         get
44         {
45            return temperature;
46         } // end get
37      } // end property Temperature
48
49      // implementation of CompareTo method for alphabetizing
50      public int CompareTo( object other )
51      {
52         return string.Compare(
53            CityName, ( ( CityWeather ) other ).CityName );
54      } // end method Compare
55
56      // return string representation of this CityWeather object
57      // to display the weather report on the server console
58      public override string ToString()
59      {
60      return cityName + " | " + temperature + " | " + description;
62      } // end method ToString
63   } // end class CityWeather
64 } // end namespace Weather
```

يحتوي الصف **CityWeather** حقول **(الأسطر 10-12)** لتخزين اسم المدينة، درجة الحرارة الدنيا والعليا، ووصف حالة الطقس.

يقوم باني الصف **CityWeather** **(الأسطر 14-20)** بتهيئة هذه المتحولات.

تُعرف الأسطر **23-47** ثلاثة خصائص للقراءة فقط للسماح بالحصول على قيم الحقول الثلاثة السابقة. يستخدم الصف **CityWeather** الطريقة **CompareTo** من الواجهة **IComparable** وتأخذ هذه الطريقة مرجع لغرض وتعيد رقم من النمط **int** **(الأسطر 50-54)**.

حيث نرغب بترتيب أغراض الصف **CityWeather** أبجدياً حسب اسم المدينة. لهذا تستدعي الطريقة **CompareTo** الطريقة **Compare** الخاصة بالنمط **string** وتقوم بتمرير **CityName** كمعامل دخل. يعيد الصف **CityWeather** تعريف الطريقة **ToString** لعرض معلومات المدينة **(الأسطر 58-61)**.

تُستخدم الطريقة **ToString** من قبل تطبيق المخدم لعرض معلومات الطقس التي يحصل عليها من صفحة الوب **Traveler's Foracast**.

الواجهة Report

يتم التصريح عن هذه الواجهة ضمن فضاء الأسماء **Weather** (السطر 7) ويجب تضمينها مع الصف **CityWeather** في ملف مكتبة الصف **Weather.Dll** وبهذا يمكن استخدامها في تطبيق المخدم وتطبيق الزبون. تُصرح الواجهة **Report** عن خاصية للقراءة فقط مع الطريقة **get** الذي تُعيد قيمة من النمط **ArrayList**. تحوي هذه الخاصية أغراض من الصف **CityWeather**.
يستخدم التطبيق هذه الخاصية للحصول على المعلومات في تقرير الطقس: اسم المدينة، درجة الحرارة الدنيا والعليا، وحالة الطقس.

```
1 // Report.cs
2 // Interface that defines a property for getting
3 // the information in a weather report.
4 using System;
5 using System.Collections;
6
7 namespace Weather
8 {
9   public interface Report
10   {
11      ArrayList Reports
12      {
13         get;
14      } // end property Reports
15   } // end interface Report
16 } // end namespace Weather
```

الصف ReportInfo

يقوم الصف البعيد **ReportInfo** بوراثة الواجهة **Report** السطر **10** في فضاء الأسماء **Weather**. كما يوسع **(extend)** الصف **MarshalByRefObject**. يكون الصف **ReportInfo** جزءاً من تطبيق المخدم ولا يمكن الوصول إليه مباشرة من قبل تطبيق الزبون.

```
1 // ReportInfo.cs
2 // Class that implements interface Report, retrieves
3 // and returns data on weather
4 using System;
5 using System.Collections;
6 using System.IO;
7 using System.Net;
8 using Weather;
9
10 public class ReportInfo : MarshalByRefObject, Report
11 {
12   private ArrayList cityList; //cities,temperatures, descriptions
13
14   public ReportInfo()
15   {
16      cityList = new ArrayList();
17
18      // create WebClient to get access to Web page
19      WebClient myClient = new WebClient();
20
21      // get StreamReader for response so we can read page
22      StreamReader input = new StreamReader( myClient.OpenRead(
23         "http://iwin.nws.noaa.gov/iwin/us/traveler.html" ) );
24
25      string separator1="TAV12";//indicates first batch of cities
26      string separator2="TAV13";//indicates second batch of cities
27
28      // locate separator1 in Web page
29   while ( !input.ReadLine().StartsWith(separator1));// do nothing
30      ReadCities( input ); // read the first batch of cities
31
32      // locate separator2 in Web page
33   while (!input.ReadLine().StartsWith(separator2)); // do nothing
34      ReadCities( input ); // read the second batch of cities
35
36      cityList.Sort(); //sort list of cities by alphabetical order
37      input.Close(); // close StreamReader to NWS server
38
39      // display the data on the server side
40      Console.WriteLine( "Data from NWS Web site:" );
41
42      foreach ( CityWeather city in cityList )
43      {
44         Console.WriteLine( city );
45      } // end foreach
46   } // end constructor
47
48   // utility method that reads a batch of cities
49   private void ReadCities( StreamReader input )
```

```
50    {
51       // day format and night format
52       string dayFormat =
53          "CITY            WEA     HI/LO   WEA     HI/LO";
54       string nightFormat =
55          "CITY            WEA     LO/HI   WEA     LO/HI";
56       string inputLine = "";
57
58       // locate header that begins weather information
59       do
60       {
61          inputLine = input.ReadLine();
62       } while ( !inputLine.Equals( dayFormat ) &&
63          !inputLine.Equals( nightFormat ) );
64
65       inputLine = input.ReadLine(); // get first city's data
66
67       // while there are more cities to read
68       while ( inputLine.Length > 28 )
69       {
70          // create CityWeather object for city
71          CityWeather weather = new CityWeather(
72             inputLine.Substring( 0, 16 ),
73             inputLine.Substring( 16, 7 ),
74             inputLine.Substring( 23, 7 ) );
75
76          cityList.Add( weather ); // add to ArrayList
77          inputLine = input.ReadLine(); // get next city's data
78       } // end while
79    } // end method ReadCities
80
81    // property for getting the cities' weather reports
82    public ArrayList Reports
83    {
84       get
85       {
86          return cityList;
87       } // end get
88    } // end property Reports
89 } // end class ReportInfo
```

تُعرف الأسطر **14-46** باني الصف **ReportInfo**. يقوم السطر **19** ببناء الغرض **WebClient** للتواصل مع مصدر المعلومات المعرف عن طريق عنوان **URL**. نستخدم في مثالنا عنوان **URL** لصفحة الانترنت **Forcast Traveler's NWS (http://iwin.nws.noaa.gov/iwin/us/traveler.html)**.

تستدعي الأسطر **22-23** الطريقة **OpenRead** والتي تُعيد غرض من الصف **Stream** الذي يستطيع البرنامج استخدامه في قراءة البيانات التي تحوي معلومات الطقس من العنوان السابق.

يستخدم **Stream** لبناء غرض من الصف **StreamReader** وبهذا يمكن للبرنامج قراءة الصفحات المكتوبة بلغة **HTML** سطر سطر.
نهتم من صفحة الويب فقط بالقسم الذي يحوي جزئين من معلومات المدن **Albany** حتى **Reno** ومدينة **Lacke Salt** حتى **Washington**. تبدأ معلومات الجزء الأول عند ظهور السلسلة المحرفية **"TAV12"**، أما الجزء الثاني فتبدأ معلوماته عند ظهور السلسلة المحرفية **"TAV13"**.

نقوم بتعريف المتحولين **separator1** و **separator2** لتخزين هاتين السلسلتين المحرفيتين. في السطر **29**، يتم قراءة الصفحة المكتوبة بلغة **HTML** سطر في كل مرة ليتم العثور على **"TAV12"**. عند الوصول إلى **"TAV12"** يقوم البرنامج باستدعاء الطريقة **ReadCities** لتقوم بقراءة معلومات المدن ووضعها في الغرض **cityList**.

يقوم السطر **33** بقراء الصفحة سطر في كل مرة ليتم العثور على **"TAV13"**. عند الوصول إلى **"TAV13"** يقوم السطر **34** باستدعاء آخر للطريقة **ReadCities** لتقوم بقراءة معلومات المدن.
يقوم السطر **36** باستدعاء الطريقة **Sort** من الصف **ArrayList** لتقوم بترتيب الأغراض من الصف **CityWeather** أبجدياً حسب اسم المدينة. يقوم السطر **37** بإغلاق الاتصال مع الموقع المستخدم من قبل **StreamReader**. تقوم الأسطر **42-45** بعرض معلومات الطقس لكل مدينة على واجهة تطبيق المخدم.

تُصرح الأسطر **49-79** عن الطريقة **ReadCities** والتي تأخذ الغرض **StreamReader** كمعامل وتقوم بقراءة معلومات كل مدينة، ثم تقوم ببناء الغرض **CityWeather** لتخزين المعلومات ضمنها وتخزين الغرض الناتج ضمن **cityList**.

تستمر الحلقة **do ... while** **(الأسطر 59-63)** في قراءة الصفحة سطر كل مرة حتى تجد السطر الرئيسي الذي يعتبر ترويسة جدول الطقس.

يبدأ هذا السطر إما بتصميم نهاري **(الأسطر 52-53)** مشيراً إلى ترويسة معلومات فترة النهار. أو يبدأ بتصميم ليلي **(الأسطر 54-55)** مشيراً إلى ترويسة معلومات فترة الليل. يقوم السطر **65** بقراءة السطر التالي من الصفحة والذي يعتبر أول سطر يحوي معلومات الحرارة.
تقوم حلقة **while** **(الأسطر 68-78)** ببناء غرض جديد من الصف **CityWeather** الذي يمثل المدينة الحالية. تقوم هذه الحلقة بسبر السلسلة المحرفية التي تحوي معلومات الطقس للمدينة الحالية حيث تقوم بفصل اسم المدينة، حالة الطقس، ودرجة الحرارة. ثم يتم إضافة غرض الصف **CityWeather** إلى **cityList**.

بعدها، يتم قراءة سطر آخر ويتم تخزينه في **inputLine** من أجل المعالجة. تستمر هذه العملية مادام طول السطر الذي يتم قرائته من الصفحة أكبر من **28** (الأسطر التي تحوي معلومات الطقس طولها أكبر من **28** محرف).

تُعيد الخاصية **Reports** (الأسطر **82-88**) **cityList**. يقوم تطبيق الزبون باستدعاء هذه الخاصية عن بعد للحصول على تقرير الطقس اليومي.

الصف WeatherServer

تُحدد الأسطر **5-7** فضاءات الأسماء المستخدمة في تطبيق المخدم حيث يحوي السطر **5** والسطر **6** فضاء الأسماء المستخدم في التشبيك عن بعد في **NET**. أما السطر **7** فيستخدم من أجل قنوات البروتوكول **HTTP**. يحدد السطر **8** فضاء الأسماء **Weather** والذي يحوي الواجهة **Report**.

```
1 // WeatherServer.cs
2 // Server application that uses .NET remoting to send
3 // weather report information to a client
4 using System;
5 using System.Runtime.Remoting;
6 using System.Runtime.Remoting.Channels;
7 using System.Runtime.Remoting.Channels.Http;
8 using Weather;

9 class WeatherServer
10 {
11   static void Main( string[] args )
12   {
13      // establish HTTP channel
14      HttpChannel channel = new HttpChannel( 50000 );
15      ChannelServices.RegisterChannel( channel, false );
16
17      // register ReportInfo class
18      RemotingConfiguration.RegisterWellKnownServiceType(
19         typeof( ReportInfo ), "Report",
20         WellKnownObjectMode.Singleton );
21
22      Console.WriteLine( "Press Enter to terminate server." );
23      Console.ReadLine();
24   } // end Main
25 } // end class WeatherServer
```

تقوم الأسطر **15-16** في الطريقة **Main** بتسجيل **(register)** قناة **HTTP** على الحاسب الحالي على المنفذ **50000**. سيستخدم الزبون رقم هذا المنفذ للاتصال مع **WeatherServer** عن بعد. يشير المعامل **False** عند السطر **16** إلى عدم الرغبة بتفعيل الحماية. تقوم الأسطر **19-21** بتسجيل نمط الصف **ReportInfo** عند **"Report"** كصف بعيد من النمط **Singleton**. إذا سجل الصف البعيد على أنه **Singleton** فإنه سيتم بناء غرض بعيد واحد عندما يطلب الزبون الأول ذاك الصف البعيد. وهذا الغرض البعيد سيقوم بتخديم كل الزبائن.

إذا سجل الصف على أنه **SingleCall** سيتم بناء غرض لكل استدعاء طريقة عن بعد لذلك الصف البعيد.

لاتكون فترة حياة الغرض البعيد من النمط **Singleton** دائمة، بل يتم تحريره عن طريق جامع النفايات **(Garbage Collector)** عند عدم تخديمه أي استدعاء لفترة **5** دقائق. وفي حال طُلب الغرض من قبل الزبون في وقت لاحق يتم بناء غرض جديد من النمط **Singleton**.

يمكن طلب الصف **ReportInfo** الآن من قبل الزبون على العنوان **URL**:
http://IPAddress:50000/Report حيث **IPAddress** هو عنوان **IP** للحاسب الذي يشغل تطبيق الزبون. تبقى القناة مفتوحة مادام تطبيق المخدم قيد العمل. ينتظر السطر **24** مستخدم التطبيق المخدم ليقوم بالضغط على المفتاح **Enter** قبل إنهاء التطبيق.

الصف WeatherClientForm

وهو واجهة التطبيق التي تستخدم التشبيك عن بعد للحصول على معلومات الطقس من **WeatherServer** لعرض هذه المعلومات بصورة بيانية سهلة القراءة. تحتوي واجهة المستخدم على **43** عنصر من النوع **Label**لكل مدينة. تتوضع كل هذه العناصر على لوحة من النمط **Panal**.

```
1 // WeatherClient.cs
2 // Client that uses .NET remoting to retrieve a weather report.
3 using System;
4 using System.Collections;
5 using System.Drawing;
6 using System.Windows.Forms;
7 using System.Runtime.Remoting;
8 using System.Runtime.Remoting.Channels;
9 using System.Runtime.Remoting.Channels.Http;
10 using Weather;
11
12 public partial class WeatherClientForm : Form
13 {
14   public WeatherClientForm()
15   {
16      InitializeComponent();
17   } // end constructor
18
19   // retrieve weather data
20   private void WeatherClientForm_Load(object sender,EventArgs e)
21
21   {
22      //setup HTTP channel, does not need to provide a port number
23      HttpChannel channel = new HttpChannel();
24      ChannelServices.RegisterChannel( channel, false );
25
26   //obtain a proxy for an object that implements interface Report
```

```
27       Report info = ( Report ) RemotingServices.Connect(
28          typeof( Report ), "http://localhost:50000/Report" );
29
30       // retrieve an ArrayList of CityWeather objects
31       ArrayList cities = info.Reports;
32
33       // create array and populate it with every Label
34       Label[] cityLabels = new Label[ 43 ];
35       int labelCounter = 0;
36
37       foreach ( Control control in displayPanel.Controls )
38       {
39          if ( control is Label )
40          {
41             cityLabels[ labelCounter ] = ( Label ) control;
42             ++labelCounter; // increment Label counter
43          } // end if
44       } // end foreach
45
46       // create Hashtable and populate with all weather conditions
47       Hashtable weather = new Hashtable();
48       weather.Add( "SUNNY", "sunny" );
49       weather.Add( "PTCLDY", "pcloudy" );
50       weather.Add( "CLOUDY", "mcloudy" );
51       weather.Add( "MOCLDY", "mcloudy" );
52       weather.Add( "TSTRMS", "rain" );
53       weather.Add( "RAIN", "rain" );
54       weather.Add( "SNOW", "snow" );
55       weather.Add( "VRYHOT", "vryhot" );
56       weather.Add( "FAIR", "fair" );
57       weather.Add( "RNSNOW", "rnsnow" );
58       weather.Add( "SHWRS", "showers" );
59       weather.Add( "WINDY", "windy" );
60       weather.Add( "NOINFO", "noinfo" );
61       weather.Add( "MISG", "noinfo" );
62       weather.Add( "DRZL", "rain" );
63       weather.Add( "HAZE", "noinfo" );
64       weather.Add( "SMOKE", "mcloudy" );
65       weather.Add( "SNOWSHWRS", "snow" );
66       weather.Add( "FLRRYS", "snow" );
67       weather.Add( "FOG", "noinfo" );
68
69       // create the font for the text output
70       Font font = new Font( "Courier New", 8, FontStyle.Bold );
71
72       // for every city
73       for ( int i = 0; i < cities.Count; i++ )
74       {
75          // use array cityLabels to find the next Label
76          Label currentCity = cityLabels[ i ];
77
78         //use ArrayList cities to find the next CityWeather object
79          CityWeather city = ( CityWeather ) cities[ i ];
80
81          // set current Label's image to image
82          // corresponding to the city's weather condition -
83          // find correct image name in Hashtable weather
84          currentCity.Image = new Bitmap( @"images\" +
85             weather[ city.Description.Trim() ] + ".png" );
```

```
86          currentCity.Font = font; // set font of Label
87          currentCity.ForeColor = Color.White;
88
89          // set Label's text to city name and temperature
90       currentCity.Text="\r\n " + city.CityName + city.Temperature;
91       } // end for
92    } // end method WeatherClientForm_Load
93 } // end class WeatherClientForm
```

تقوم الطريقة **WeatherClientForm_Load** **(**الأسطر **20-92)** عند تحميل واجهة التطبيق بالحصول على معلومات الطقس. يقوم السطر **23** ببناء قناة **HTTP** بدون تحديد رقم المنفذ. هذا يجعل باني الصف **HTTPChannal** يقوم باختيار أي رقم منفذ متوفر على حاسب الزبون.

لايكون رقم المنفذ ضرورياً لأن هذا التطبيق لا يمتلك الزبون الخاص به الذي يحتاج أن يعرف رقم المنفذ. يسجل السطر **24** القناة على حاسب الزبون. هذا يسمح للمخدم بإعادة إرسال المعلومات إلى الزبون.

تقوم الأسطر **27-28** بتعريف المتحول **Report** ويقوم بالإسناد له الوكيل للغرض **Report** الذي تمت تهيئته من قبل المخدم. يسمح الوكيل للزبون بالاستدعاء عن بعد خصائص الصف **ReportInfo** عن طريق تحويل الاستدعاءات إلى المخدم. تقوم الطريق **Connect** من الصف **RemotingServices** بالاتصال بالمخدم وإعادة مرجع إلى وكيل الغرض **Report**.

نقوم هنا بتنفيذ تطبيق المخدم وتطبيق الزبون على نفس الحاسب ولهذا نستخدم **localhost** في عنوان **URL** الذي يمثل تطبيق المخدم. للإتصال بتطبيق **WeatherServer** يتم تنفيذه على حاسب مختلف، يجب تعديل عنوان **URL** السابق. يقوم السطر **31** بالحصول على المصفوفة **ArrayList** من الأغراض **CityWeather** التي تولد من قبل باني الصف **ReportInfo** **(**الأسطر **14-64)**. يشير المتحول **cities** إلى مصفوفة الأغراض **CityWeather** التي تحوي المعلومات من الصفحة **Traveler's Forcast**.

بما أن التطبيق يقوم بعرض معلومات الطقس لعدة مدن، يجب الانتباه إلى طريقة تنظيم

المعلومات على اللواصق **(Labels)** وضمان أن كل وصف لمعلومات الطقس موافق للصورة المناسبة.

يستخدم البرنامج مصفوفة لتخزين اللواصق وغرض من الصف **Hashtable** لتخزين وصف الطقس وأسماء الصور الموافقة للوصف.

يقوم السطر **34** ببناء مصفوفة مراجع للواصق.

تضع الأسطر **35-44** اللواصق التي تم بناؤها في واجهة التصميم في مصفوفة وبهذا يمكن الوصول إليها برمجياً لعرض معلومات الطقس لكل مدينة.

يقوم السطر **47** ببناء غرض من الصف **Hashtable** والذي يقوم بتخزين الزوج حالة الطقس - الصورة الموافقة لهذه الحالة.

تقوم الأسطر **73-91** بتهيئة اللواصق بحيث تحوي اسم المدينة، درجة الحرارة الحالية والصورة الموافقة لحالة الطقس لهذه المدينة.

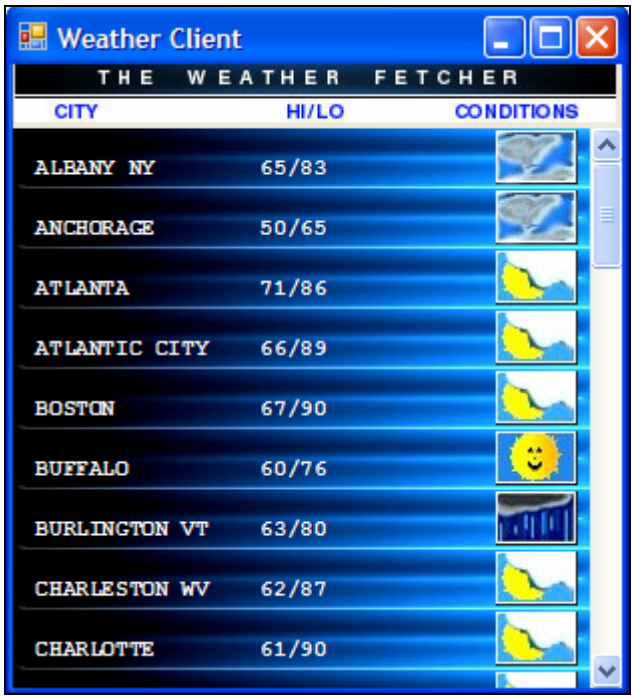
يقوم السطر **76** بالحصول على اللاصقة من أجل عرض معلومات الطقس للمدينة التالية.

يستخدم السطر **79** المصفوفة **cities** للحصول على الأغراض **CityWeather** التي تحوي معلومات الطقس لكل مدينة.

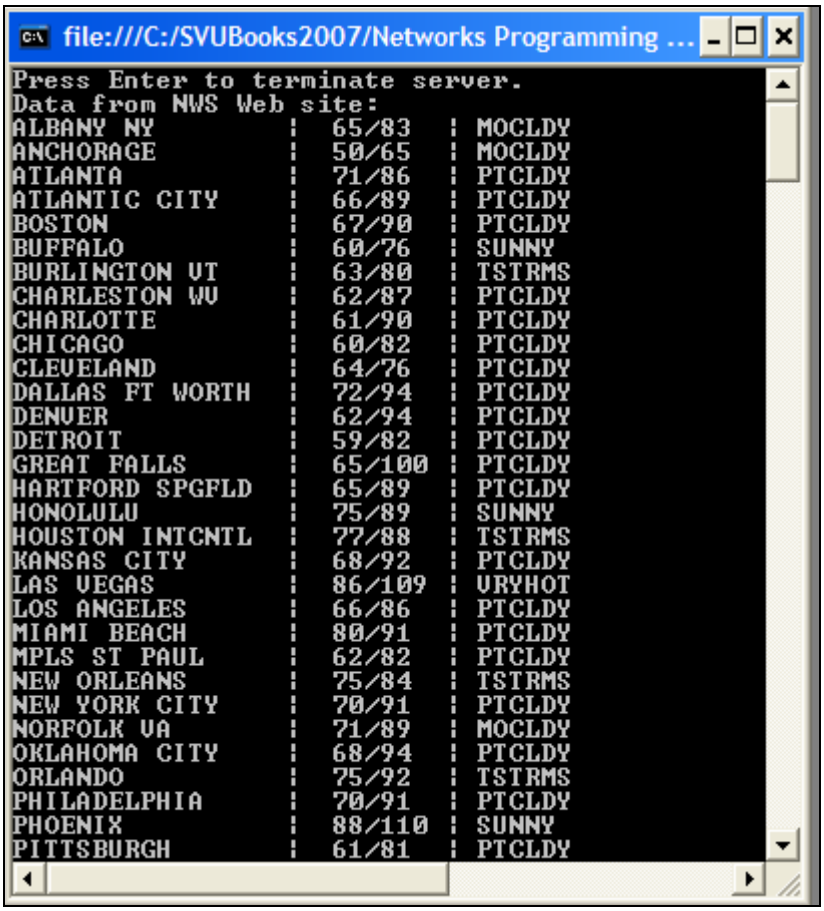
تقوم الأسطر **86-87** بتهيئة خصائص اللواصق ليتم عرض التأثيرات المرئية.

يقوم السطر **90** بتهيئة الخاصية **Text** لعرض اسم المدينة ودرجة الحرارة الدنيا والعليا.

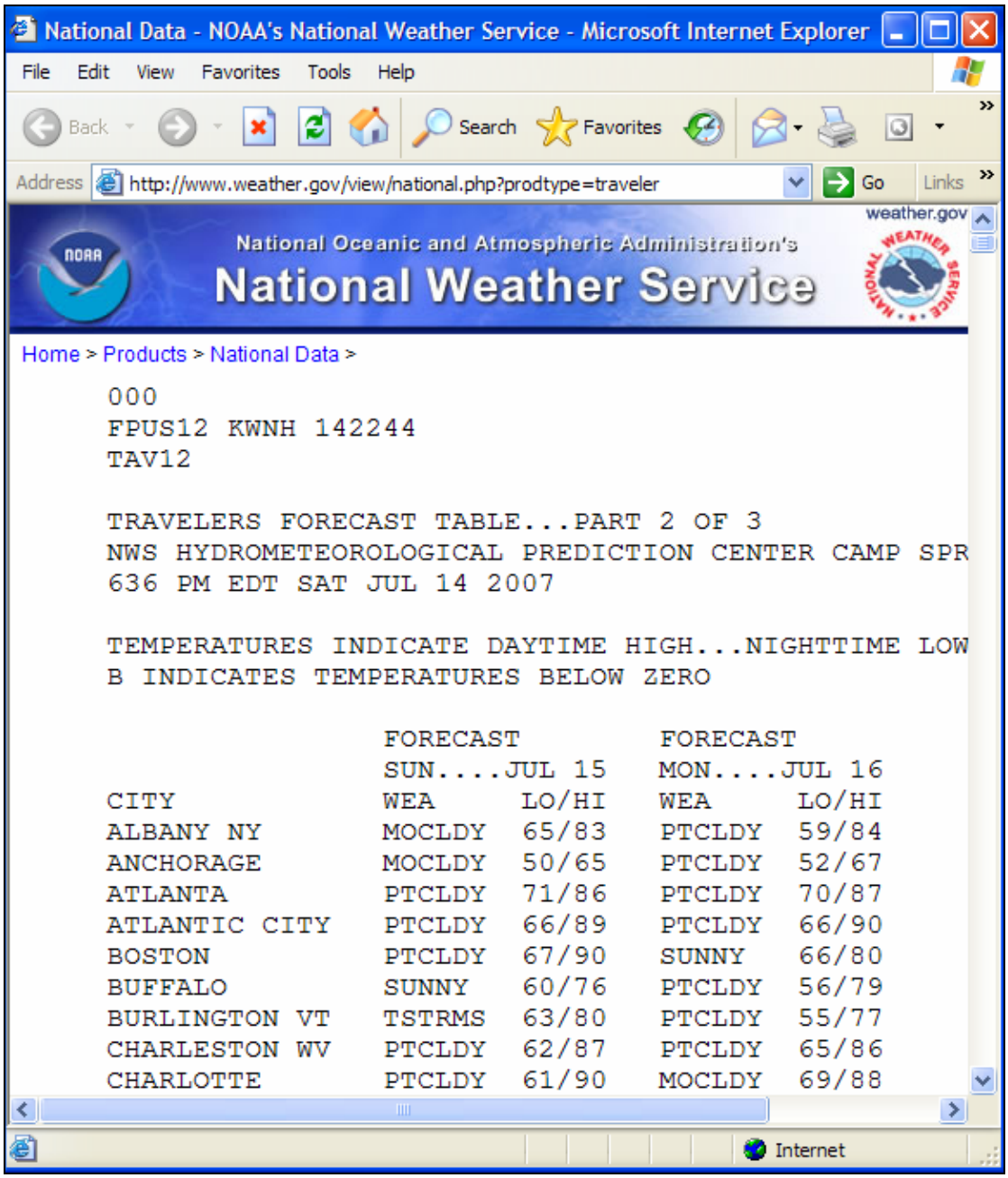
يبين الشكل التالي واجهة الزبون:



يبين الشكل التالي واجهة المخدم:



يبين الشكل التالي الموقع **National Weather Service**:



الفصل الثالث : خدمات الويب Web Services

1. مقدمة

يستخدم المبرمجون عادةً مكتبات الربط الديناميكية **Dynamic Link Libraries (DLLs)** لتسهيل عملية إعادة استخدام البرمجيات. إلا أن هذه المكتبات يجب أن توضع على نفس الحاسب الذي يستخدم أو يستدعي هذه البرمجيات.

ندرس في هذا الفصل استخدام تقانات خدمات الويب للسماح بإعادة استثمار البرمجيات في الأنظمة الموزعة. تسمح تقانات الأنظمة الموزعة بتنفيذ التطبيقات على عدة حواسب ضمن شبكة.

تسمح خدمة ويب **Web Service** بالقيام بالحسابات الموزعة وذلك من خلال السماح لحاسب من استدعاء طريقة **method** موجودة على حاسب آخر عبر صيغ بيانات مشتركة وبروتوكولات معيارية مثل **XML** و **HTTP**.

تُنفّذ عمليات الاستدعاء عبر الشبكة للطرق في **NET.** باستخدام بروتوكول الوصول للأغراض البسيط **SOAP**. والذي هو بروتوكول مبني على **XML** ويقوم بتوصيف كيفية تأشير **mark up** الطلبات **requests** والأجوبة **responses** وبحيث يتم تناقلها عبر بروتوكولات معيارية مثل **HTTP**.

يمكّن استخدام **SOAP** التطبيقات من تمثيل وتناقل البيانات في قوالب معيارية **XML**.

تُشجع **Microsoft** مسوقي البرمجيات والأعمال الالكترونية على نشر وتشارك خدمات الويب. إذ طالما تتزايد يومياً أعداد المتصلين بالانترنت، فإن استدعاء التطبيقات للطرق عبر الانترنت يصبح مربحاً.

ومن منظور البرمجة غرضية التوجه، تُمثل خدمات الويب الجيل الثاني من البرمجة غرضية التوجه. إذ عوضاً عن برمجة مجموعة قليلة من الصفوف في مكان واحد، سيصبح المبرمجون قادرون على استخدام عدد هائل من الصفوف تتواجد في أماكن متفرقة متباعدة.

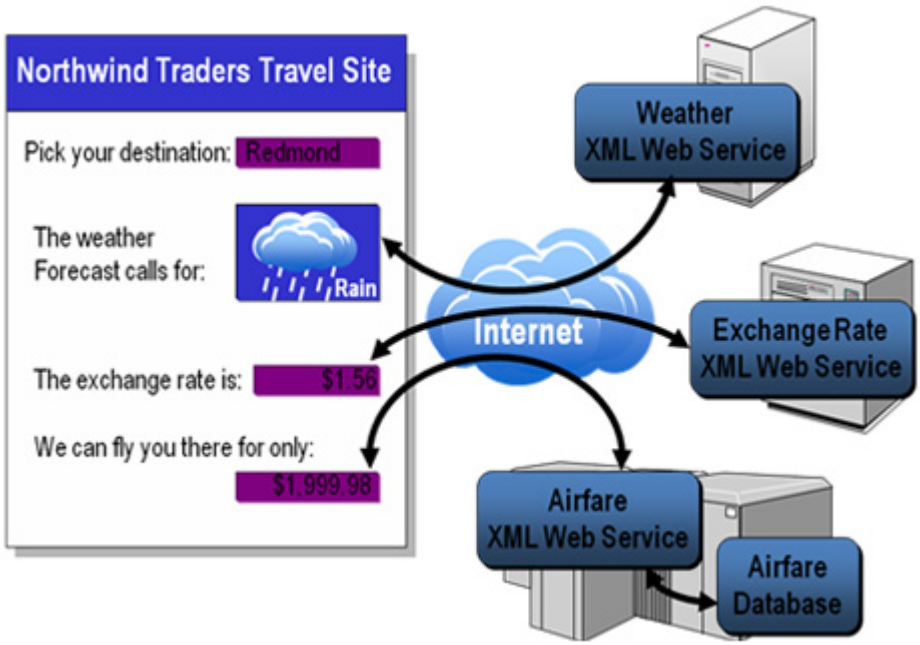
تسمح هذه التقانة بتسهيل عملية تعاون تطبيقات الأعمال الضخمة ونموها بشكل سريع. إذ بالتشارك في خدمات الويب الموافقة، ستتمكن الشركات العاملة في تطوير المنتجات البرمجية من اختصار الزمن اللازم للتطوير والالتفات دائماً لتطوير خدمات جديدة.

لنأخذ كمثال بسيط، حالة شركة تبيع على الويب أقراص موسيقية كما تُعطي معلومات عن الفنانين. ولنفرض أن شركة أخرى تختص ببيع بطاقات الحفلات تُقدم خدمة ويب تسمح بمعرفة تواريخ الحفلات القادمة لمختلف الفنانين وتسمح للمستخدمين بشراء بطاقات هذه الحفلات. يمكن للشركتين أن تتعاونا وبحيث يتمكن مستخدمي موقع الشركة الأولى من شراء بطاقات الحفلات مما سيزيد من عدد زوار موقع الشركة الأولى.

كذلك فإن الشركة الثانية ستزداد مبيعاتها إذ أنها ستستفيد من زبائن الشركة الأولى كما يمكن لها أن تطلب عمولة من الشركة الأولى على البطاقات المباعة.

يمثل الشكل التالي مثلاً صفحة ويب لشركة سفر وسياحة تستفيد من عدة خدمات ويب لمساعدة زبونها:

- خدمة ويب لمعرفة حالة الطقس للوجهة المسافر إليها.
- خدمة ويب لمعرفة سعر الصرف للعملة المحلية.
- خدمة ويب لمعرفة ثمن التذكرة.



2. خدمات الويب Web Services

تكون خدمة الويب عبارة عن تطبيق موجود على حاسب وبحيث يمكن لأي حاسب آخر على الشبكة من الوصول إلى هذا التطبيق. يدعى عادة الحاسب الحاضن للتطبيق بالحاسب البعيد **remote machine**.

يقوم التطبيق الذي يريد الاستفادة من خدمة الويب بإرسال طلب استدعاء لطريقة توفرها خدمة الويب للحاسب البعيد، والذي يقوم بدوره بمعالجة هذا الطلب وإرسال الجواب إلى التطبيق المستدعي.

يفيد هذا النمط من الحسابات الموزعة العديد من الأنظمة، بما فيها تلك الأنظمة التي لا يمكن لها الوصول إلى بيانات معينة أو تلك التي لاتملك قدرات كافية للقيام ببعض الحسابات المعينة. إذ أنها ستتمكن عبر استخدامها لخدمات الويب من الاستفادة من مصادر حواسيب ضخمة.

سنرى أن خدمة الويب هي في النهاية صف **Class** يحوي مجموعة من الطرق **Methods** ويتواجد على حاسب بعيد.

يتمّ نقل الطلبات والأجوبة باستخدام البروتوكول **SOAP**. مما يعني أن أي زبون قادر على توليد ومعالجة رسائل **SOAP** يستطيع استخدام خدمة الويب، وذلك بغض النظر عن اللغة التي كُتبت بها خدمة الويب. كما يبين الشكل التالي:

يمكن لخدمة الويب أن تقصر خدماتها على زبائن مخولين **authorized** فقط.

وبما أن خدمات الويب والبروتوكول **SOAP** مستقلين عن منصات التشغيل واللغات المستخدمة، فيمكن للشركات أن تتعاون إذاً عبر خدمات الويب دون أن تقلق لتوافقية البرمجيات أو العتاد أو تقانات الاتصال.

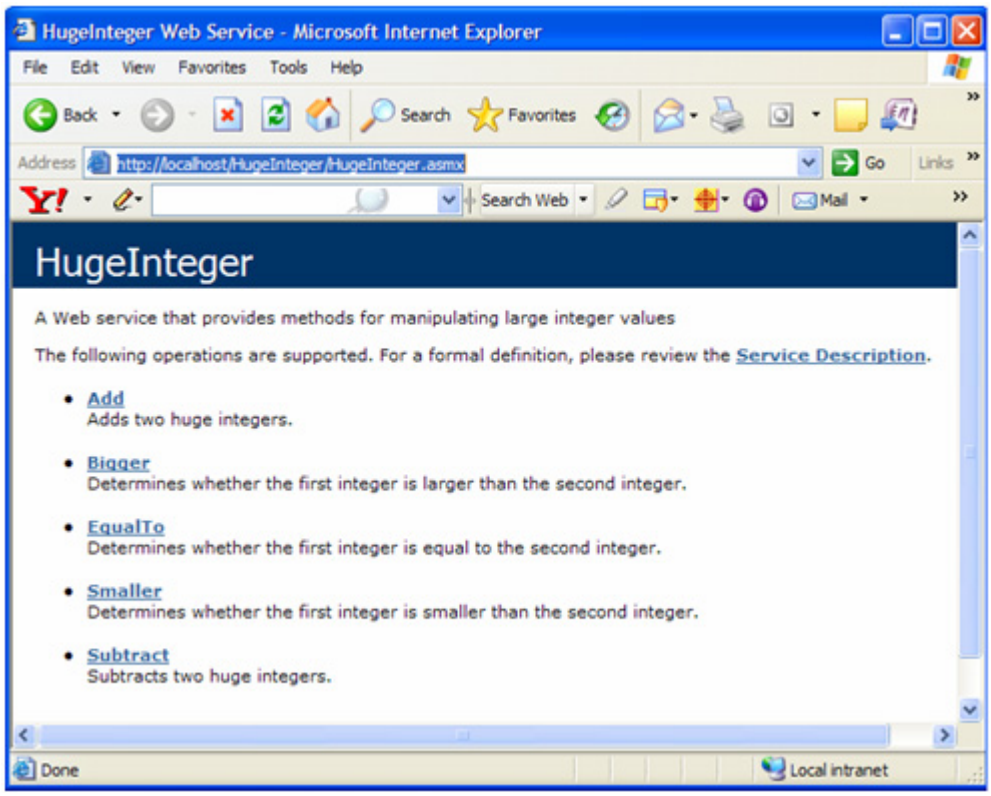
تتعاون مثلاً الشركات مثل **Amazon** و **Google** و **eBay** عبر تشارك مجموعة من خدمات الويب.

تُستدعى الطرق الموجودة في خدمة ويب عن بعد باستخدام إجرائية الاستدعاء البعيدة **Remote Procedure Call (RPC)**. يتمّ تأشير هذه الطرق باستخدام الواصفة **WebMethod** كما سنرى لاحقاً.

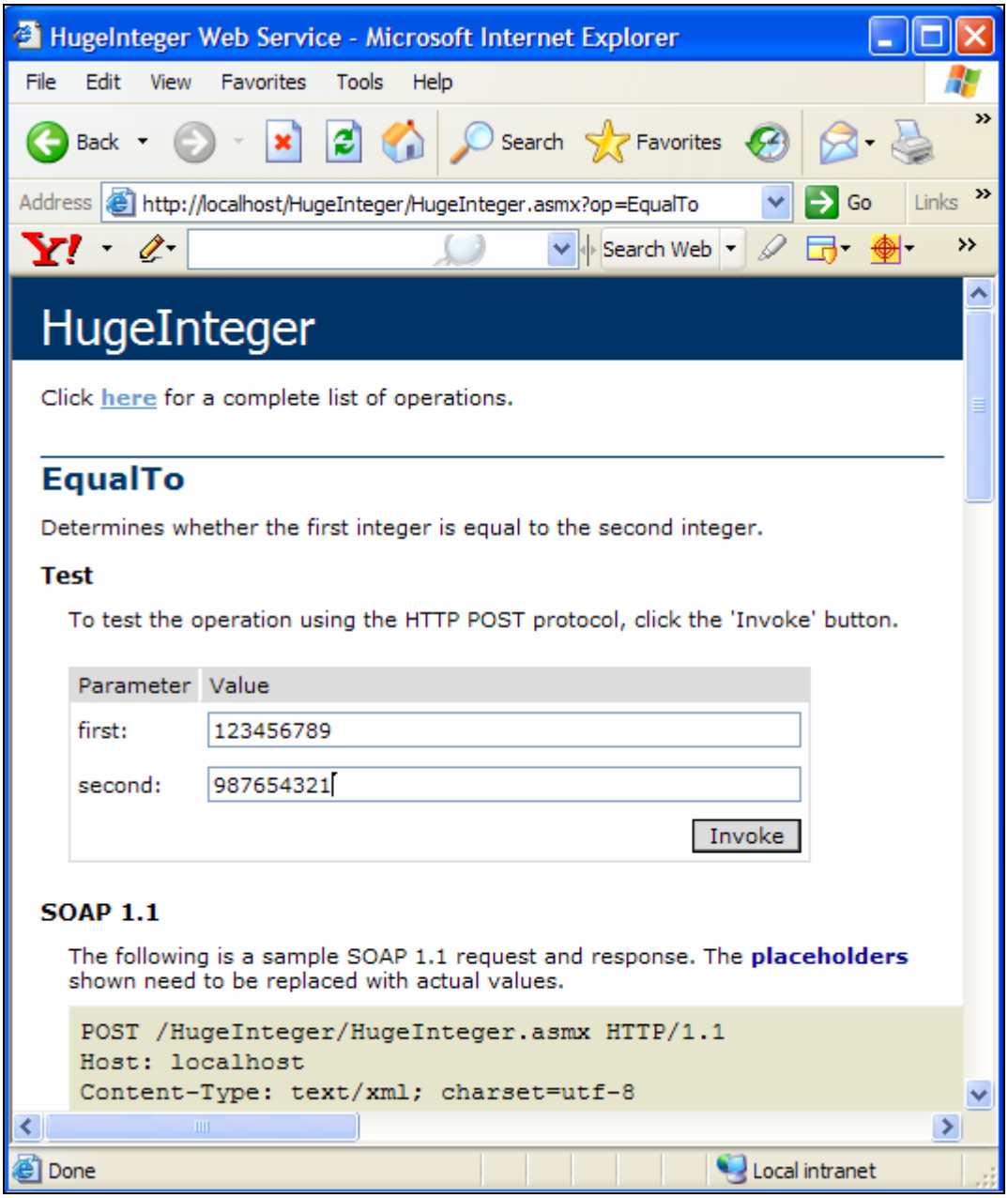
يجب لإنشاء خدمة ويب باستخدام **Visual Web Developer** أن تنشئ أولاً مشروع من نوع **ASP.NET Web Service** والذي سيولد مايلي:

- الملفات التي ستضم كود خدمة الويب (مختلف الطرق المؤلفة للخدمة).
- ملف **asmx** والذي يؤمن الوصول إلى خدمة الويب.

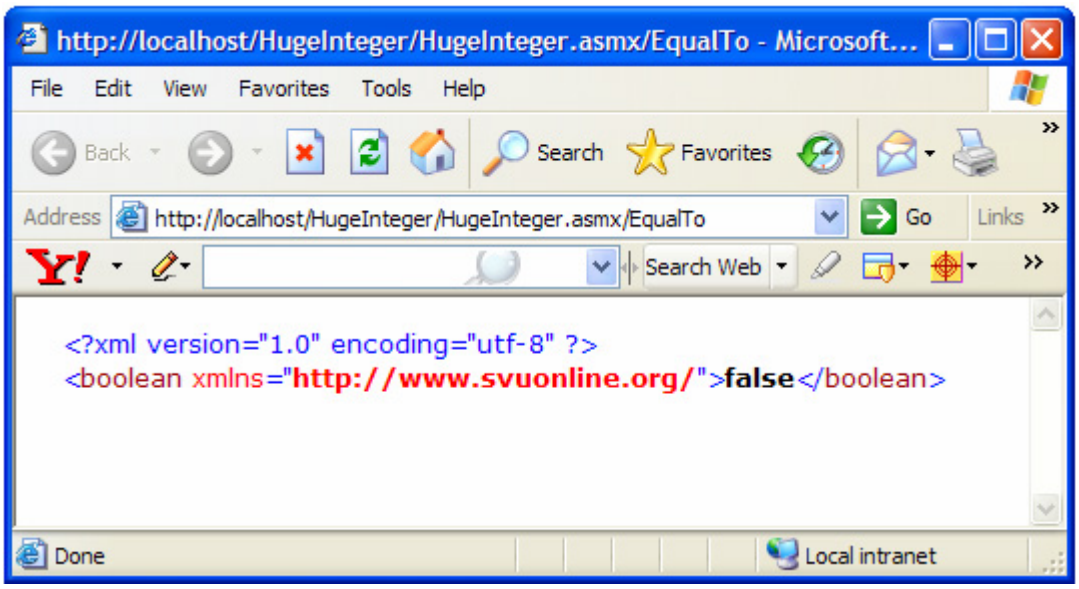
يبين الشكل التالي مثلاً مايعيده استدعاء ملف **asmx** من قبل المتصفح:



تحوي الصفحة ارتباطات إلى مختلف الطرق المتاحة في خدمة الويب. يمكن بالنقر على أي ارتباط لطريقة فتح الصفحة التي تسمح بإدخال معاملات هذه الطريقة. مثلاً لو أردنا اختبار الطريقة **EqualTo**:



وبالنقر على الزر **Invoke** يظهر لنا ناتج تنفيذ الطريقة:



يحوي رأس الصفحة الأولى لخدمة الويب الارتباط **Service Description** والذي يفتح ملف توصيف الخدمة. يكون ملف توصيف الخدمة عبارة عن ملف **XML** مطابق للغة المعيارية المستخدمة لتوصيف خدمات الويب **Web Services Description Language (WSDL)**. والتي هي لغة **XML** تُستخدم لتوصيف الطرق المتاحة في خدمة ويب وكيفية التعامل مع هذه الطرق.

3. البروتوكول البسيط للوصول للأغـراض Simple Object Access Protocol (SOAP)

يمكن تعريف البروتوكول البسيط للوصول للأغراض **SOAP** بأنه بروتوكول مستقل عن منصة التشغيل يستخدم **XML** لإنشاء الاستدعاءات البعيدة للإجرائيات عبر البروتوكول **HTTP**.

يحزم كل طلب **request** وجواب **response** في رسالة **SOAP** أي رسالة **XML** تحوي المعلومات اللازمة لمعالجة محتواها.

لقد تمّ اختيار البروتوكول **HTTP** لإرسال رسائل **SOAP**، لأنه بروتوكول معياري لتراسل البيانات على الانترنت.

يمكّن استخدام كل من **HTTP** و **XML** نظم التشغيل المختلفة من إرسال واستقبال رسائل **SOAP**. لاكما أن من الميزات الإضافية الأخرى للبروتوكول **HTTP** هو أنه يمكن استخدامه في الشبكات التي تحوي جدران نارية **fire walls** تُقيد الاتصالات في الشبكة. إذ أن هذه الجدران الأمنية تحد عملياً من حركة **HTTP**.

يدعم البروتوكول **SOAP** مجموعة كبيرة من أنماط البيانات **Data Types**. إذ يجب على صيغة النقل **wire format** المستخدمة لتبادل الطلبات والأجوبة أن تدعم جميع أنماط البيانات المستخدمة في التطبيقات.

تدعم خدمات الويب التي تستخدم **SOAP** أنماط بيانات أكثر من خدمات الويب التي تستخدم صيغاً أخرى للنقل.

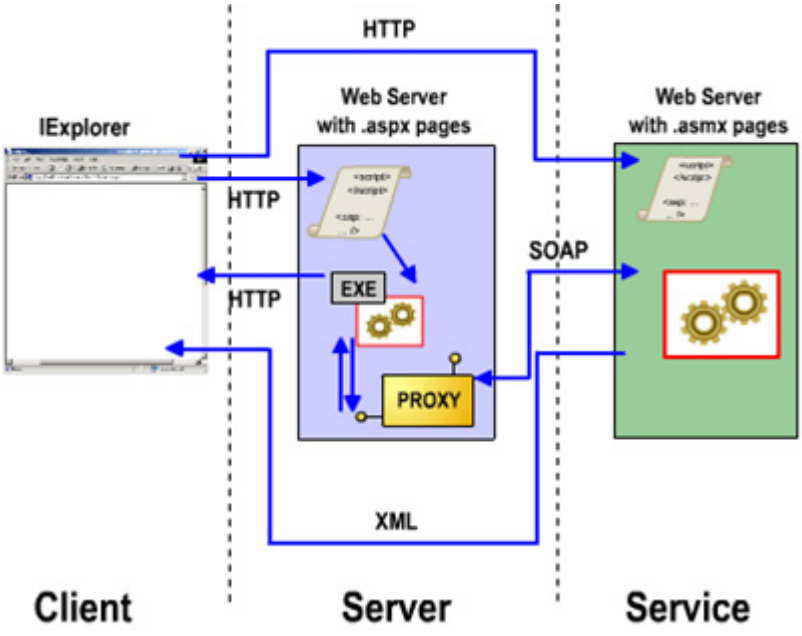
تتضمن أنماط البيانات في **SOAP** جميع أنماط المعطيات الأساسية (مثل **integer, double,. .**)، كذلك تتضمن الأنماط المركبة (مثل **DataSet, DateTime, XmlNode**). كما يسمح البروتوكول **SOAP** بتبادل مصفوفات **arrays** من البيانات من جميع الأنماط. كذلك يمكن استخدام الأنماط المعرّفة من قبل المستخدم **user-defined types**.

تقوم التطبيقات بإرسال الطلبات والأجوبة إلى ومن خدمات الويب عبر رسائل **SOAP**. عندما يستدعي برنامج طريقة من خدمة ويب، فإن الطلب وجميع المعلومات المتعلقة به تُحزم في رسالة **SOAP**، وتُرسل إلى الجهة الموافقة. عندما تستقبل خدمة الويب الرسالة تبدأ بمعالجة محتواها (المدعو بالمغلف **envelope**) والذي يحدد الطريقة التي يريد الزبون تنفيذها والمعاملات **parameters** الواجب تمريرها إلى هذه الطريقة. تُستدعى بعد ذلك الطريقة مع المعاملات الموافقة ويتمّ إرسال ناتج تنفيذ الطريقة إلى الزبون في رسالة **SOAP** أخرى. يقوم الزبون بمعالجة هذه الرسالة للوصول إلى نتائج الطريقة.

يتألف التطبيق الذي يستهلك خدمة ويب عملياً من قسمين: صف وكيل **proxy** يمثل خدمة الويب، وتطبيق زبون يصل إلى خدمة الويب من خلال غرض من الصف السابق.

يعهد إلى صف الوكيل مهمة نقل المعاملات لطريقة خدمة الويب من التطبيق الزبون إلى خدمة الويب، كما يقوم بنقل نتائج طريقة خدمة الويب إلى التطبيق الزبون. يقوم **Visual Studio** بتوليد صفوف **proxy** كما سوف نرى.

يبين الشكل التالي مخطط العمليات السابقة:



4. نشر واستهلاك خدمات الويب Publishing and Consuming Web Services

وتشمل:

1. تعريف خدمة الويب
2. إنشاء خدمة ويب.
3. نشر خدمة الويب.
4. إنشاء زبون لاستهلاك خدمة الويب.
5. استهلاك خدمة ويب.

تعريف خدمة الويب

نعرض فيما يلي الكود الخلفي لخدمة الويب المثال. صممت هذه الخدمة للقيام بالحسابات على أعداد صحيحة تحوي على الأكثر **100** خانة. (لا يمكن للمتحولات من النوع **long** أن تصل لهذا الحجم). تُقدم خدمة الويب الطرق التالية والتي تأخذ كل منها رقمين صحيحين ضخمين كمعاملي دخل:

- **Bigger** لاختبار أن العدد الضخم الأول أكبر من العدد الضخم الثاني.
- **EqualTo** لاختبار أن العددين الضخمين متساويين.
- **Smaller** لاختبار أن العدد الضخم الأول أصغر من العدد الضخم الثاني.
- **Add** لجمع العددين الضخمين.

- **Substract** لطرح العدد الضخم الثاني من الأول.

يمكن لأي تطبيق الوصول إلى هذه الخدمة عبر الويب والاستفادة من الطرق المتوفر فيها دون أن يضطر المبرمج لكتابة الكود اللازم لهذه الطرق:

```
1 // HugeInteger.cs
2 // HugeInteger Web service performs operations on large integers.
3 using System;
4 using System.Web;
5 using System.Web.Services;
6 using System.Web.Services.Protocols;
7
8 [WebService(Namespace = "http://www.svuonline.org/",
9      Description = "A Web service that provides methods for" +
10      " manipulating large integer values")]
11 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
12 public class HugeInteger : System.Web.Services.WebService
13 {
14       private const int MAXIMUM = 100; // maximum number of digits
15       public int[] number; // array representing the huge integer
16
```

```
17        // default constructor
18        public HugeInteger()
19        {
20            number = new int[MAXIMUM];
21        } // end default constructor
22
23        // indexer that accepts an integer parameter
24        public int this[int index]
25        {
26            get
27            {
28                return number[index];
29            } // end get
30
31            set
32            {
33                number[index] = value;
34            } // end set
35        } // end indexer
36
37        // returns string representation of HugeInteger
38        public override string ToString()
39        {
40            string returnString = "";
41
42            foreach (int i in number)
43                returnString = i + returnString;
44
45            return returnString;
46        } // end method ToString
47
48        // creates HugeInteger based on argument
```

```
49         public static HugeInteger FromString(string value)
50         {
51        // create temporary HugeInteger to be returned by the method
52             HugeInteger parsedInteger = new HugeInteger();
53
54             for (int i = 0; i < value.Length; i++)
55                 parsedInteger[i] = Int32.Parse(
56                    value[value.Length - i - 1].ToString());
57
58             return parsedInteger;
59         } // end method FromString
60
61// WebMethod that adds integers represented by the String arguments
62         [WebMethod(Description = "Adds two huge integers.")]
63         public string Add(string first, string second)
64         {
65             int carry = 0;
66             HugeInteger operand1 = HugeInteger.FromString(first);
67             HugeInteger operand2 = HugeInteger.FromString(second);
68             HugeInteger result = new HugeInteger();
69
70             // perform addition algorithm for each digit
71             for (int i = 0; i < MAXIMUM; i++)
72             {
73                 // add two digits in same column,
74                 // result is their sum plus carry from
75                 // previous operation, modulo 10
76                 result[i] =
77                    (operand1[i] + operand2[i] + carry) % 10;
78
79     // set carry to remainder of dividing sums of two digits by 10
80                 carry = (operand1[i] + operand2[i] + carry) / 10;
81             } // end for
82
83             return result.ToString();
84         } // end method Add
85
86         // WebMethod that subtracts integers
87         // represented by the string arguments
88         [WebMethod(Description = "Subtracts two huge integers.")]
89         public string Subtract(string first, string second)
90         {
91             HugeInteger operand1 = HugeInteger.FromString(first);
92             HugeInteger operand2 = HugeInteger.FromString(second);
93             HugeInteger result = new HugeInteger();
94
95             // subtract bottom digit from top digit
96             for (int i = 0; i < MAXIMUM; i++)
97             {
98     // if top digit is smaller than bottom digit we need to borrow
99                 if (operand1[i] < operand2[i])
100                    Borrow(operand1, i);
101
102                // subtract bottom from top
103                result[i] = operand1[i] - operand2[i];
104            } // end for
105
106            return result.ToString();
107        } // end method Subtract
```

```
108
109         // borrow 1 from next digit
110         private void Borrow(HugeInteger hugeInteger, int place)
111         {
112             // if no place to borrow from, signal problem
113             if (place >= MAXIMUM - 1)
114                 throw new ArgumentException();
115
116       // otherwise if next digit is zero, borrow from column to left
117             else if (hugeInteger[place + 1] == 0)
118                 Borrow(hugeInteger, place + 1);
119
120       // add ten to current place because we borrowed and subtract
121       // one from previous digit--this is the digit we borrowed from
122             hugeInteger[place] += 10;
123             hugeInteger[place + 1]--;
124         } // end method Borrow
125
126     // returns true if first integer is bigger than second
127     [WebMethod(Description = " whether the first integer is " +
128           "larger than the second integer.")]
129         public bool Bigger(string first, string second)
130         {
131             char[] zeros = { '0' };
132
133             try
134             {
135             // if elimination of all zeros from result
136             // of subtraction is an empty string,
137         // numbers are equal, so return false, otherwise return true
138                 if (Subtract(first, second).Trim(zeros) == "")
139                     return false;
140                 else
141                     return true;
142             } // end try
143             // if ArgumentException occurs, first
144             // number was smaller, so return false
145             catch (ArgumentException exception)
146             {
147                 return false;
148             } // end catch
149         } // end method Bigger
150
151         // returns true if first integer is smaller than second
152         [WebMethod(Description = " whether the first integer " +
153           "is smaller than the second integer.")]
154         public bool Smaller(string first, string second)
155         {
156
157             return Bigger(second, first);
158         } // end method Smaller
159
160         // WebMethod that returns true if two integers are equal
161         [WebMethod(Description = " whether the first integer " +
162           "is equal to the second integer.")]
163         public bool EqualTo(string first, string second)
164         {
165             // if either first is bigger than second, or first is
166             // smaller than second, they are not equal
```

```
167             if (Bigger(first, second) || Smaller(first, second))
168                 return false;
169             else
170                 return true;
171         } // end method EqualTo
172     } // end class HugeInteger
```

تحوي الأسطر **8-11** الواصفة **WebService**. تضم هذه الواصفة الخصائص التالية:

- **Namespace** فضاء العمل: وأعطيناه القيمة **http://www.svuonline.org/**.
- **Description** لتوصيف خدمة الويب. يعرض الملف **asmx** هذا الوصف.
- **WebServiceBinding** تُضاف تلقائياً وتُحدد معيار خدمة الويب.

يشتق كل صف خدمة ويب جديد تلقائياً السطر **(12)** من الصف

System.Web.Services.WebService والذي يحوي طرق مفيدة لتحديد معلومات عن الزبون وعن خدمة الويب نفسها.

تمّ تأشير العديد من الطرق في الصف بالواصفة **WebMethod** لتحديد بأن هذه الطرق هي الطرق المتاحة في خدمة الويب الأسطر **(62, 88, 127, 152, 161)** أي يمكن استدعائها عن بعد.

عندما لا توضع هذه الواصفة لطريقة، لا تكون هذه الطريقة متاحة للزبائن عن بعد. تحوي هذه الواصفة أيضاً الخاصية **Description** والتي ستسمح للملف **asmx** بعرض وصف للطريقة.

تُعرّف الأسطر **24-35** مفهرس **indexer** يمكّن من الوصول لأي خانة في **HugeInteger**.

تُعرّف الأسطر **62-84** طريقة الويب **Add**. كما تُعرّف الأسطر **88-107** طريقة الويب **Substract**.

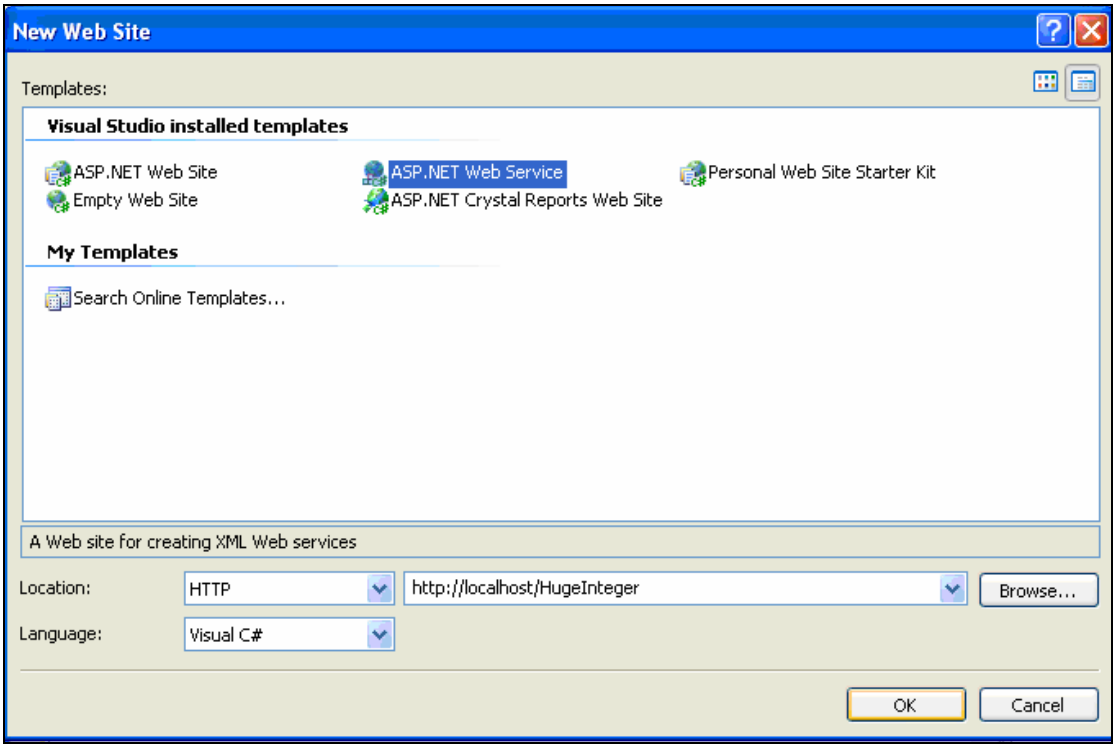
تُعرّف الأسطر **110-124** طريقة **Borrow** (ليست طريقة ويب) والتي تُعالج حالة طرح خانة أكبر من خانة أصغر.

(مثلاً، عندما نطرح **19** من **32** نجد أن الرقم **2** هو أصغر من **9** و بالتالي فإننا نضيف **10** إلى **2** مما يعطينا **12** ، بعد الاستعارة نستطيع طرح **9** من **12** ويكون الناتج هو **3** ، ثم نطرح **1** من **2**).

إنشاء خدمة ويب

تُبين الخطوات التالية الخطوات اللازمة لإنشاء خدمة الويب **HugeInteger** باستخدام **Visual Studio** على مخدم الانترنت المحلي **Internet Information Services IIS**:

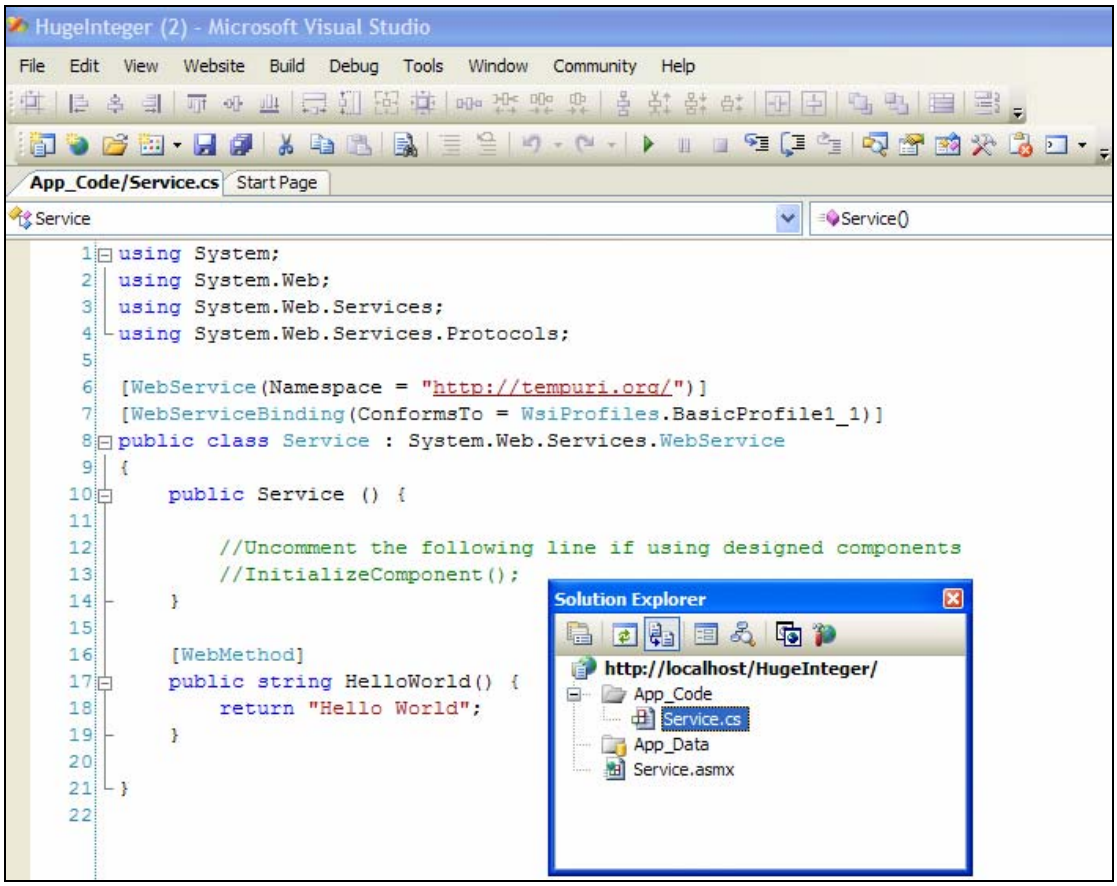
1. الخطوة الأولى: إنشاء المشروع.
   - اختيار **File → New Web Site**.
   - اختيار القالب **ASP.NET Web Service**.
   - اختيار **Location: HTTP**.
   - العنوان **http://localhost/HugeInteger**.
   - اللغة **Visual C#**.



2. الخطوة الثانية: فحص الخدمة الأولية المنشأة:

يقوم **Visual Studio** بإنشاء خدمة ويب بسيطة تتألف مما يلي:

- الصف **Service.cs** الذي يحوي الطريقة **HelloWorld**، وهو صف مشتق من الصف **System.Web.Services.WebService**.



- الملف **Service.asmx** والذي يحوي:

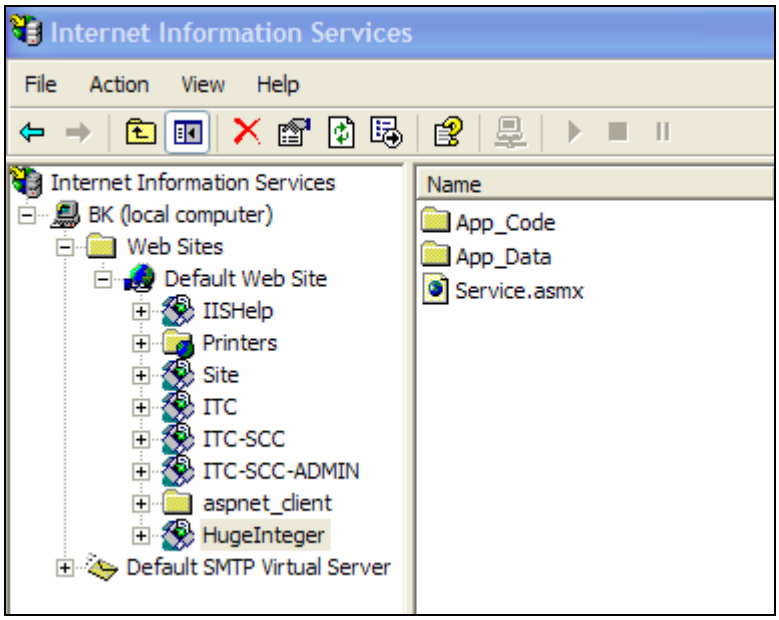
```
<%@ WebService Language="C#"
 CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

محدداً لغة البرمجة المستخدمة، مسار الكود الخلفي، واسم الصف.

(عند طلب الملف **asmx.** عبر **IIS** كما سنرى، تُستخدم هذه المعلومات لتوليد الناتج)

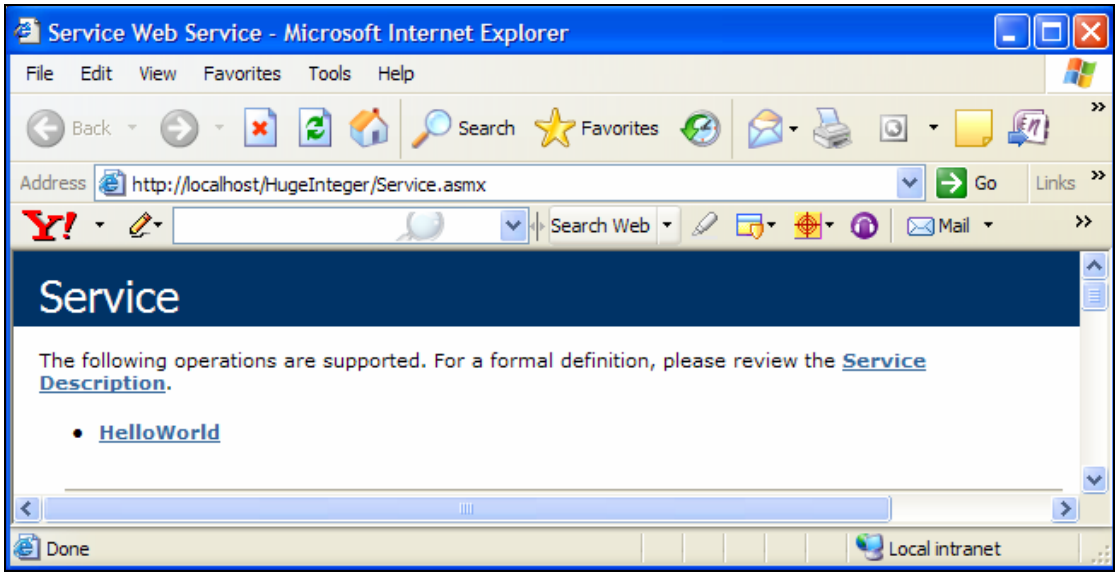
3. الخطوة الثالثة: التأكد من استضافة خدمة الويب

يمكن فتح **IIS** للتأكد من استضافة خدمة الويب المنشأة على الحاسب كما يبين الشكل:

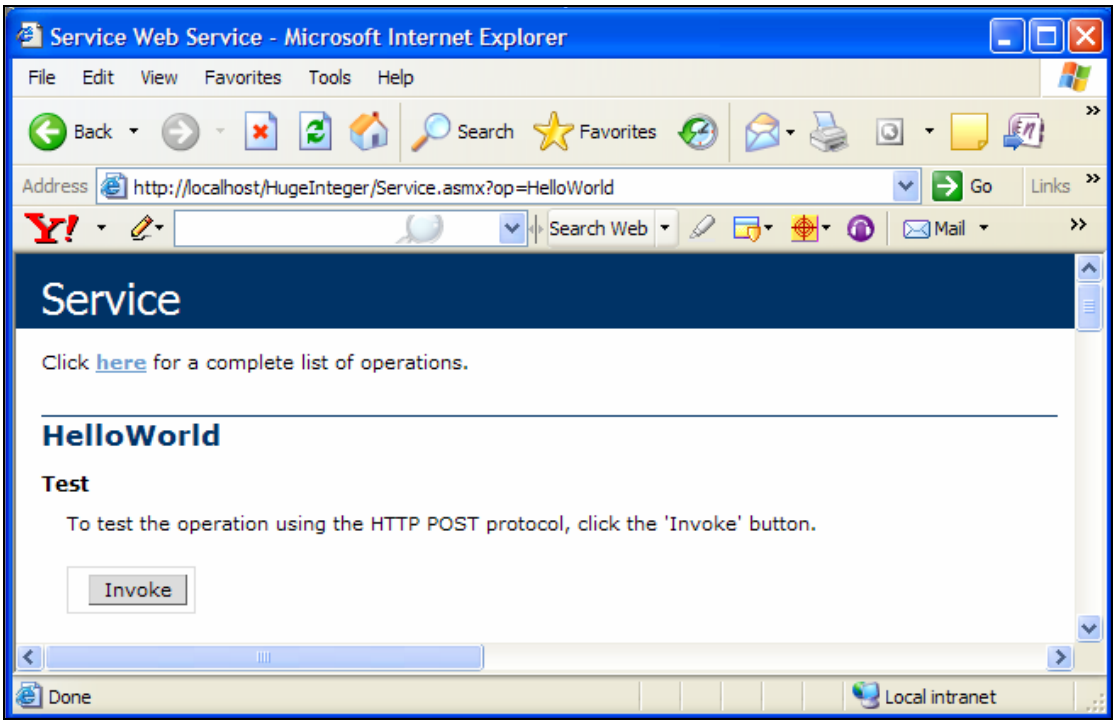


4. الخطوة الرابعة: تجريب خدمة الويب

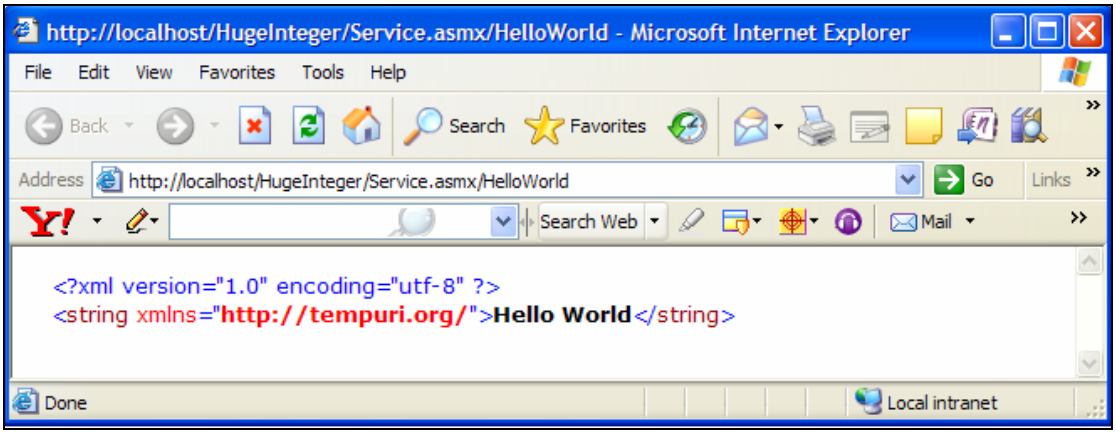
يمكن فتح المستعرض الآن مع العنوان **//localhost/HugeInteger/Service.asmx** لتشغيل خدمة الويب:



وبالنقر على الارتباط **HelloWord** تظهر النافذة التالية:



وبالنقر على الزر **Invoke** يظهر ناتج تنفيذ طريقة الويب:



5. الخطوة الخامسة: تعديل الخدمة الأولية إلى الخدمة المطلوبة:

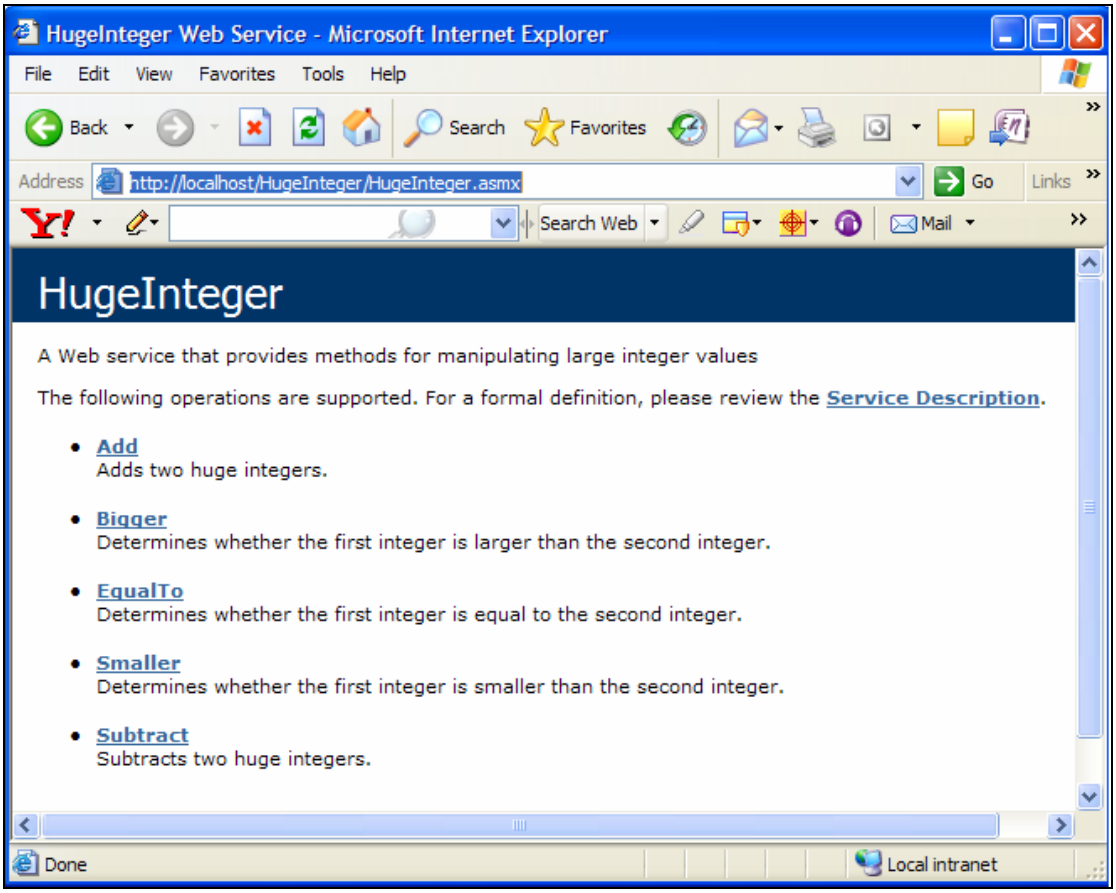
تتضمن هذه الخطوة:

- كتابة الكود المطلوب للخدمة **HugeInteger** (راجع الفقرة السابقة).
- تعديل اسم الملف **Service.cs** إلى **HugeInteger.cs**.
- تعديل اسم الملف **Service.asmx** إلى **HugeInteger.asmx**.
- تعديل محتوى الملف **HugeInteger.asmx** إلى:

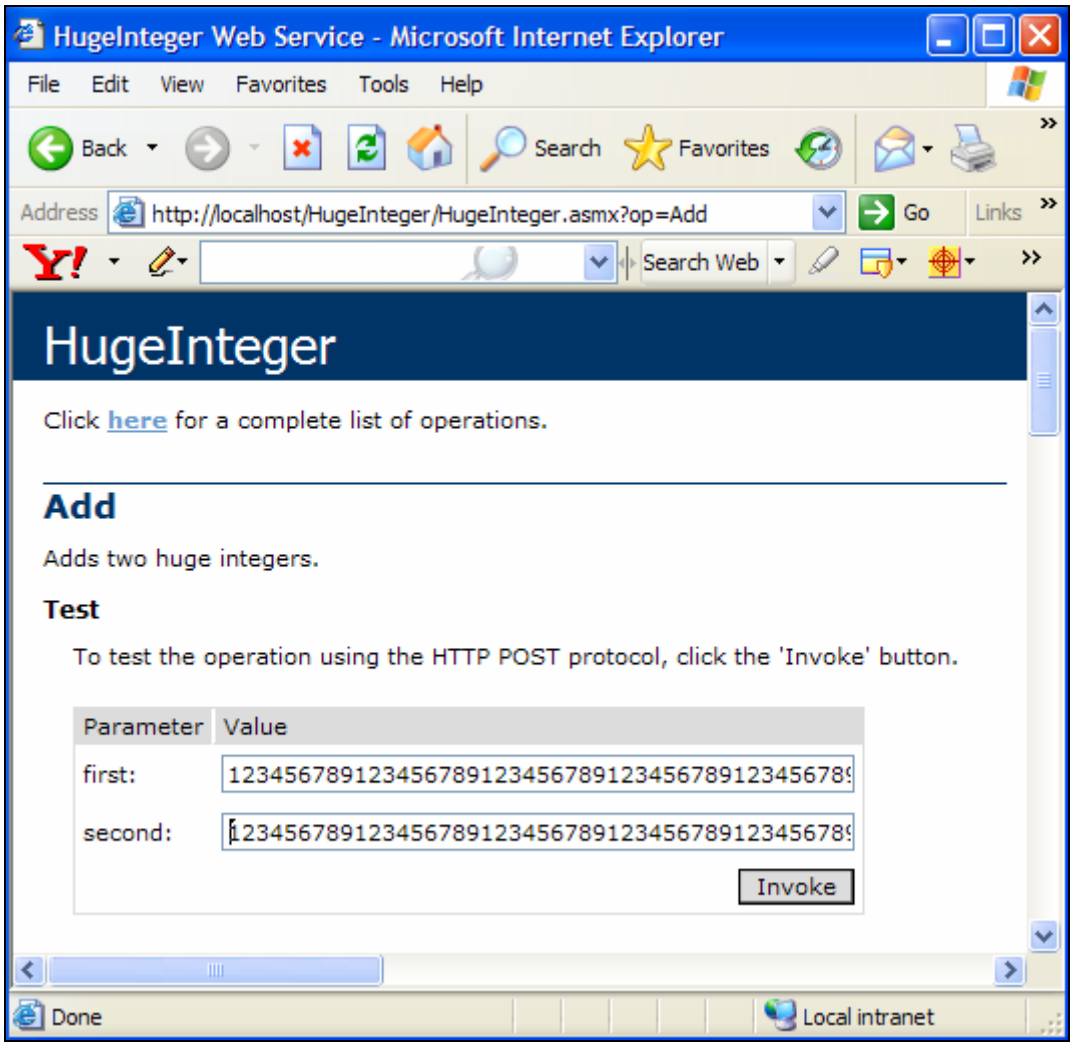
```
<%@ WebService Language="C#" CodeBehind="~/App_Code/HugeInteger.cs"
Class="HugeInteger" %>
```

6. الخطوة السادسة: تجريب خدمة الويب من المستعرض

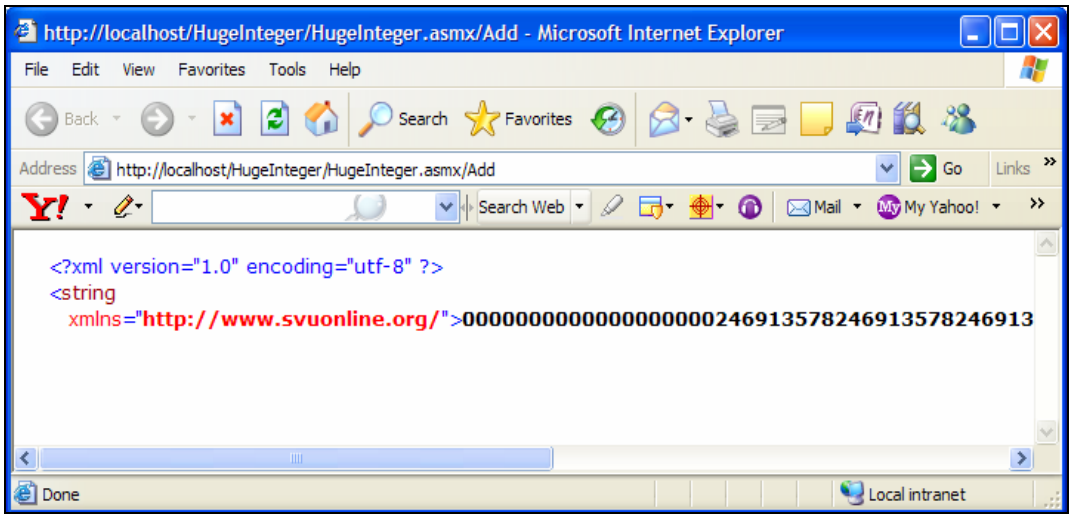
يمكن الآن طلب الخدمة **HugeInteger** من المستعرض على العنوان **http://localhost/HugeInteger/HugeInteger.asmx** كما يبين الشكل:



لنطلب مثلاً الطريقة **Add**:


HugeInteger Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/HugeInteger/HugeInteger.asmx?op=Add
Go Links
Search Web
HugeInteger
Click here for a complete list of operations.
Add
Adds two huge integers.
Test
To test the operation using the HTTP POST protocol, click the 'Invoke' button.
Parameter Value
first:
second:
Invoke
Done
Local intranet

نجد:


http://localhost/HugeInteger/HugeInteger.asmx/Add - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/HugeInteger/HugeInteger.asmx/Add
Go Links
Search Web Mail My Yahoo!
<?xml version="1.0" encoding="utf-8" ?>
<string
xmlns="http://www.svuonline.org/">00000000000000000002469135782469135782469
Done
Local intranet

نشر خدمة الويب

تكون خدمة الويب السابقة **HugeInteger** منشورة **deployed** تلقائياً لأننا أنشئناها على **IIS**مباشرة. أي يمكن فتح خدمة الويب من المستعرض كما وضحنا سابقاً مع العنوان:

```
http://localhost/HugeInteger/HugeInteger.asmx
```

يمكن في **Visual Studio** اختيار **Build→ Build Web Site** للتأكد من خلو المشروع من الأخطاء.

كما يمكن تجريب خدمة الويب ضمن **Visual Studio** باختيار **Debug→ Start Debugging** مما سيؤدي لفتح المستعرض للملف **asmx**.

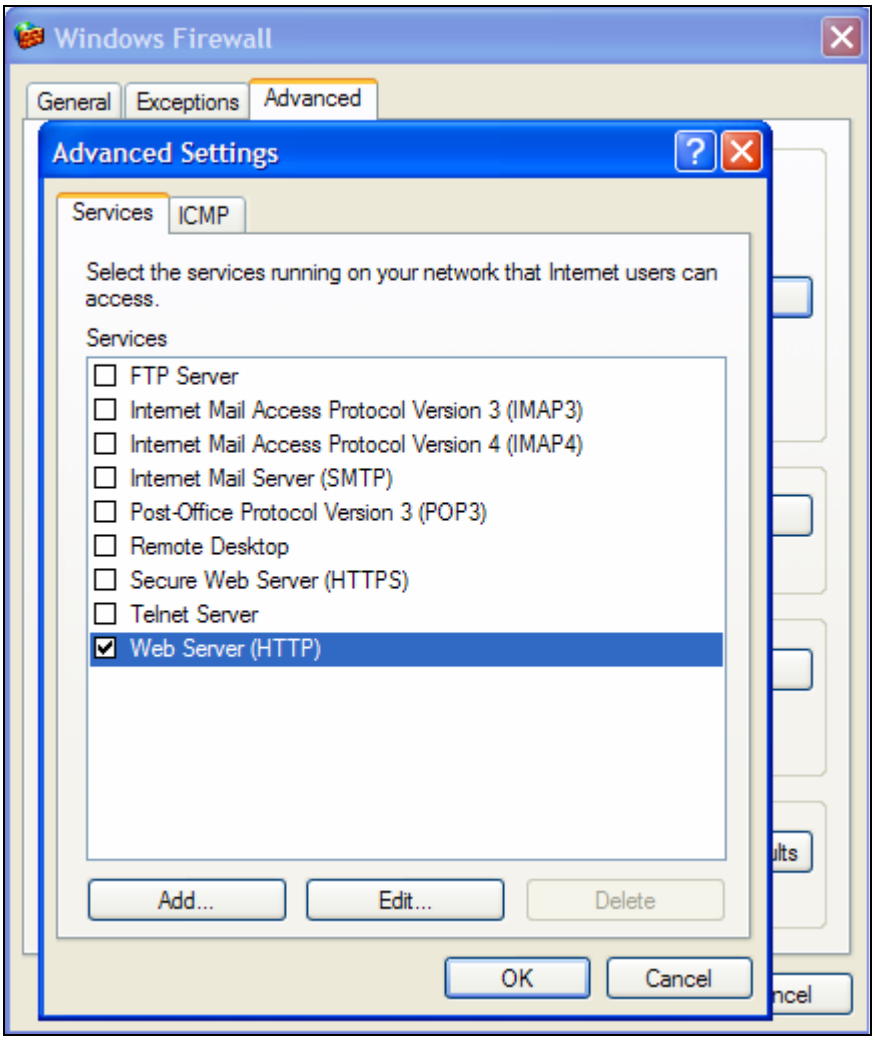
يمكن للآخرين الموصولين عبر الشبكة من استخدام خدمة الويب المنشورة على **IIS** على حاسبك بكتابة العنوان التالي في المستعرض:

```
http://host/HugeInteger/HugeInteger.asmx
```

حيث **host** هو اسم حاسبك أو عنوانك **IP address** على الشبكة.

يجب القيام بالخطوات التالية للسماح للآخرين باستخدام خدمات الويب من حاسبك:

1. اختر **Start→Control Panel** لفتح نافذة لوحة التحكم للحاسب، ثم اختر **Windows Firewall**.
2. افتح الصفحة **Advanced** في نافذة**Windows Firewall**.
3. انقر الزر **Setting** في الصفحة **Advanced**.
4. اختر صندوق التحقق **Web Server (HTTP)**.


Windows Firewall
General
Exceptions
Advanced
Advanced Settings
Services
ICMP
Select the services running on your network that Internet users can access.
Services
FTP Server
Internet Mail Access Protocol Version 3 (IMAP3)
Internet Mail Access Protocol Version 4 (IMAP4)
Internet Mail Server (SMTP)
Post-Office Protocol Version 3 (POP3)
Remote Desktop
Secure Web Server (HTTPS)
Telnet Server
Web Server (HTTP)
Add...
Edit...
Delete
OK
Cancel

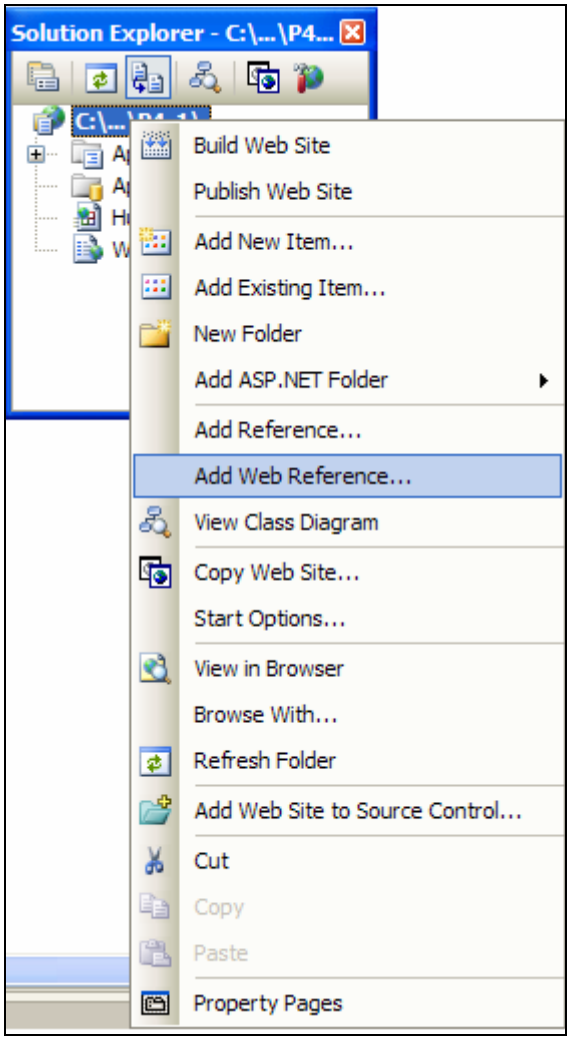
إنشاء زبون لاستهلاك خدمة الويب

يمكن الآن، بعد القيام بتعريف ونشر خدمة الويب، إنشاء زبون يستهلك هذه الخدمة.

يجب أولاً إنشاء مشروع من النمط **Windows Application** ثم القيام بالخطوات التالية:

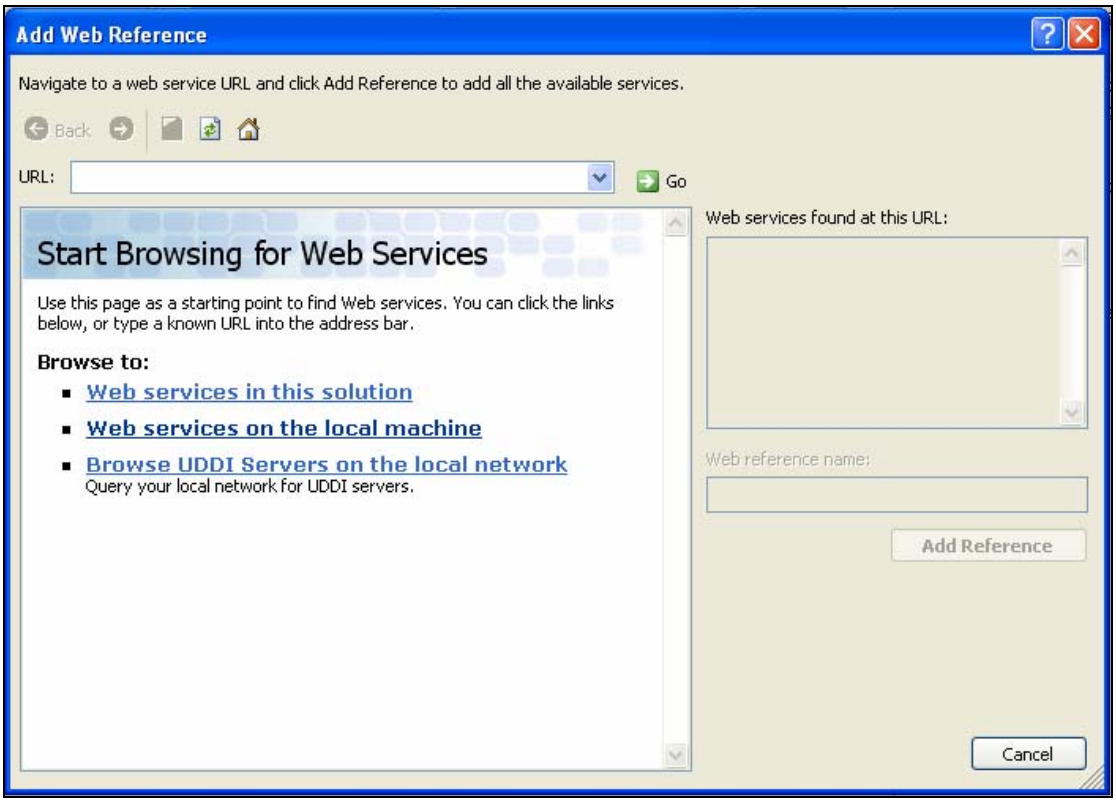
1. الخطوة الأولى: فتح صندوق الحوار **Add Web Reference**

يتمّ ذلك بفتح **Solution Explorer** ومن ثم **Add Web Reference** من قائمة الزر الأيمن:



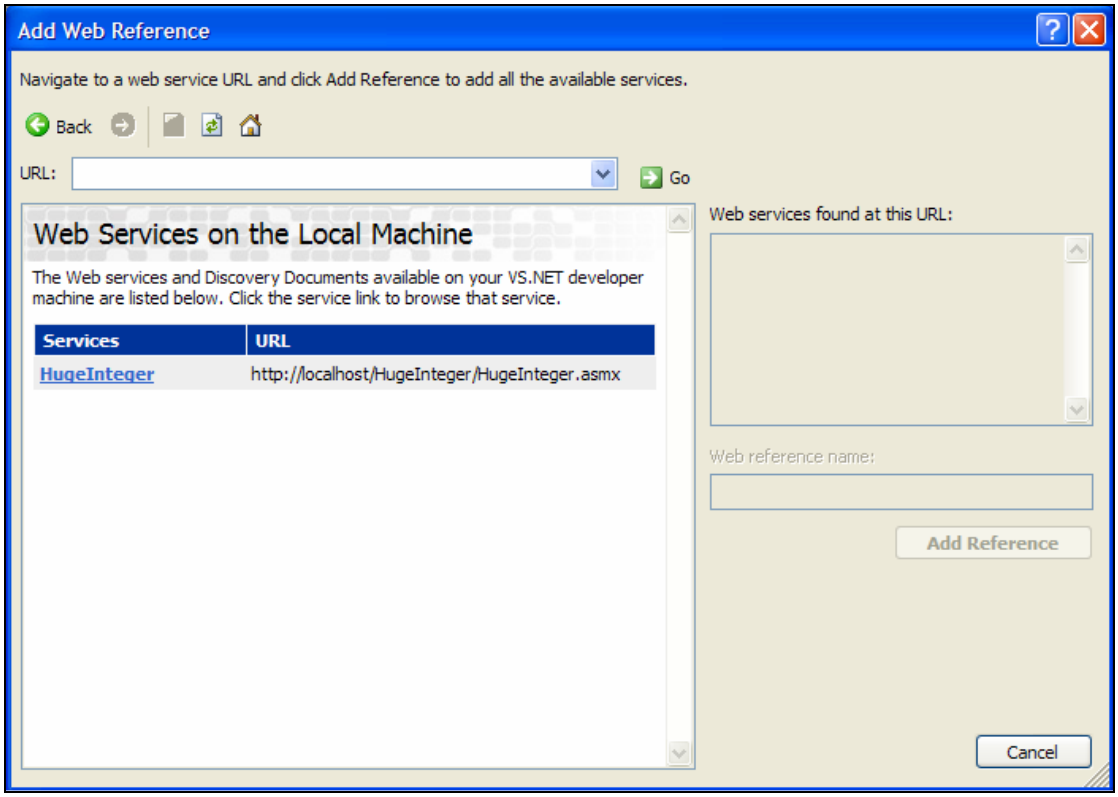
2. الخطوة الثانية: تحديد خدمات الويب المتوفرة على الحاسب

اختر **Web sevices on the local machine** لإظهار مراجع الويب المستضافة على الحاسب المحلي في **IIS** (http://localhost).



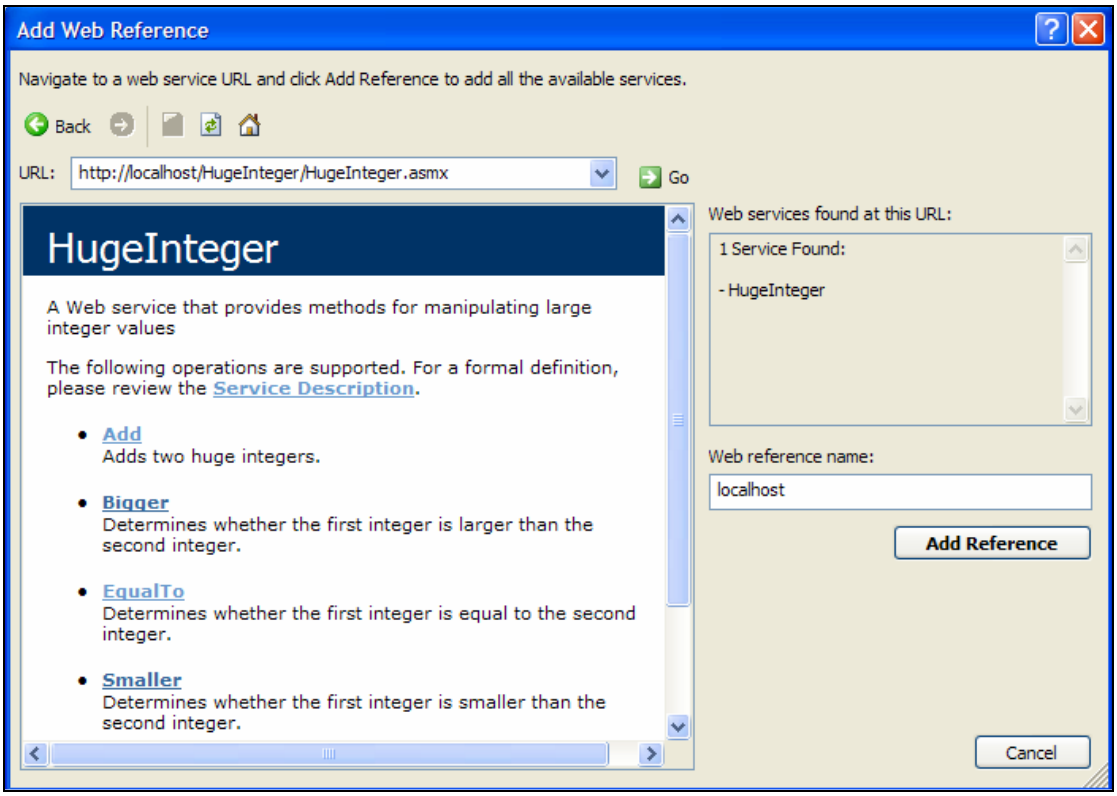
3. الخطوة الثالثة: تحديد الخدمة المطلوبة

اختر الخدمة **HugeInteger**



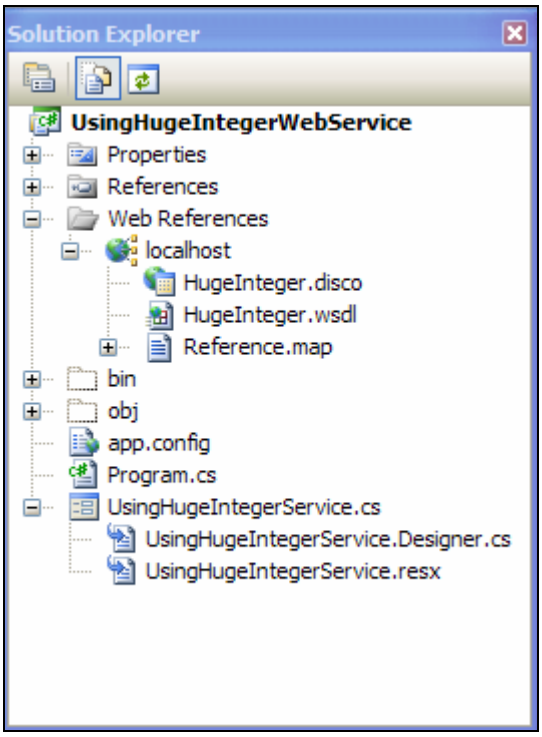
4. الخطوة الرابعة: إضافة مرجع الويب

اختر **Add Reference**



5. الخطوة الخامسة: معاينة مراجع الويب

افتح الآن المجلد **Web References** في **Solution Explorer** وتأكد من وجود ملفات **HugeInteger**.



استهلاك خدمة ويب

نعرض فيما يلي الكود الذي يستخدم خدمة الويب **HugeInteger** وذلك بهدف القيام بحسابات على أعداد صحيحة تصل إلى **100** خانة.

يصرح السطر **22** عن المتحول **remoteInteger** من النمط **localhost.HugeInteger**. يستخدم هذا المتحول في كل معالجات أحداث التطبيق لاستدعاء طرق **HugeInteger**. يتمّ إنشاء غرض وكيل **proxy** وإسناده إلى هذا المتحول في السطر **31** من حدث تحميل النموذج **Form_Load**.

تقوم الأسطر **52-53**، **66-67**، **95-96**، **116-117**، **135-136** باستدعاء طرق خدمة الويب في مختلف معالجات أحداث الأزرار الموضوعة على النموذج:

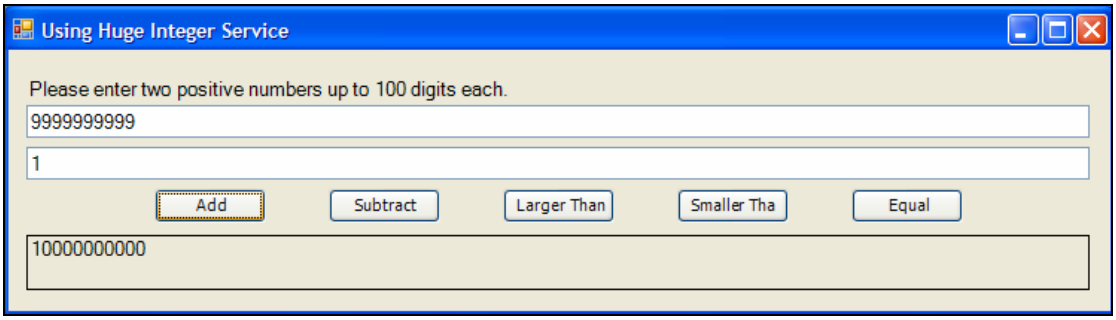
```
1  // UsingHugeIntegerService.cs
2  // Using the HugeInteger Web Service.
3  using System;
4  using System.Collections.Generic;
5  using System.ComponentModel;
6  using System.Data;
7  using System.Drawing;
8  using System.Text;
9  using System.Windows.Forms;
10 using System.Web.Services.Protocols;
11
12  namespace UsingHugeIntegerWebService
13  {
14   public partial class UsingHugeIntegerServiceForm : Form
15   {
```

```
16    public UsingHugeIntegerServiceForm()
17    {
18       InitializeComponent();
19    } // end constructor
20
21    // declare a reference to Web service
22    private localhost.HugeInteger remoteInteger;
23
24    private char[] zeros={'0'};//character to trim from strings
25
26     // instantiates object to interact with Web service
27    private void UsingHugeIntegerServiceForm_Load( object sender,
28             EventArgs e )
29          {
30             // instantiate remoteInteger
31             remoteInteger = new localhost.HugeInteger();
32          } // end method UsingHugeIntegerServiceForm_Load
33
34          // adds two numbers input by user
35    private void addButton_Click( object sender, EventArgs e )
36    {
37     // make sure numbers do not exceed 100 digits and that both
38     // are not 100 digits long, which would result in overflow
39             if ( firstTextBox.Text.Length > 100 ||
40                secondTextBox.Text.Length > 100 ||
41                ( firstTextBox.Text.Length == 100 &&
42                secondTextBox.Text.Length == 100) )
43             {
44                MessageBox.Show( "HugeIntegers must not be more " +
45                "than 100 digits\r\nBoth integers cannot be " +
46                "of length 100: this causes an overflow", "Error",
47                MessageBoxButtons.OK, MessageBoxIcon.Information );
48                return;
49             } // end if
50
51             // perform addition
52          resultLabel.Text = remoteInteger.Add(
53          firstTextBox.Text, secondTextBox.Text ).TrimStart( zeros );
54     } // end method addButton_Click
55
56           // subtracts two numbers input by user
57    private void subtractButton_Click( object sender, EventArgs e )
58    {
59             // make sure HugeIntegers do not exceed 100 digits
60             if ( SizeCheck( firstTextBox, secondTextBox ) )
61                return;
62
63          // perform subtraction
64          try
65          {
66          string result = remoteInteger.Subtract(
67          firstTextBox.Text, secondTextBox.Text ).TrimStart( zeros );
68
69                if ( result == "" )
70                   resultLabel.Text = "0";
71                else
72                   resultLabel.Text = result;
73
```

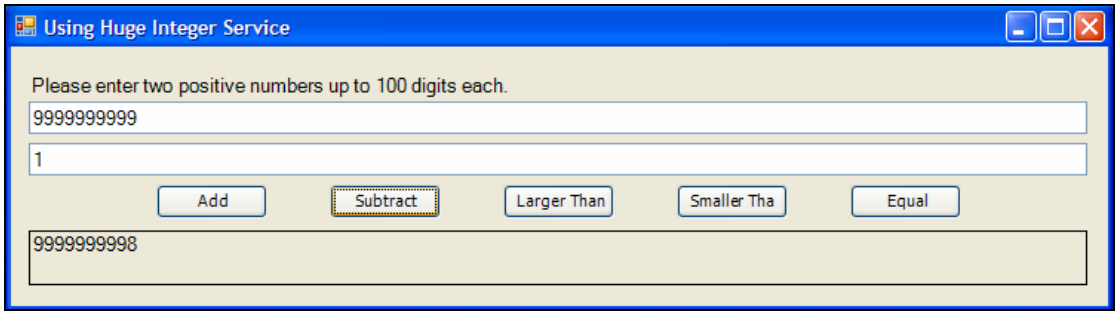
```
74         } // end try
75
76         // if WebMethod throws an exception,
77         // then first argument was smaller than second
78         catch ( SoapException exception )
79              {
80                 MessageBox.Show(
81                    "First argument was smaller than the second" );
82              } // end catch
83      } // end method subtractButton_Click
84
85      // determines whether first number
86      // input by user is larger than second
87      private void largerButton_Click( object sender, EventArgs e )
88      {
89       // make sure HugeIntegers do not exceed 100 digits
90       if ( SizeCheck( firstTextBox, secondTextBox ) )
91       return;
92
93       // call Web-service method to determine if
94       // first integer is larger than the second
95       if ( remoteInteger.Bigger( firstTextBox.Text,
96                 secondTextBox.Text ) )
97       resultLabel.Text = firstTextBox.Text.TrimStart( zeros ) +
98                    " is larger than " +
99                    secondTextBox.Text.TrimStart( zeros );
100      else
101       resultLabel.Text = firstTextBox.Text.TrimStart( zeros ) +
102                    " is not larger than " +
103                    secondTextBox.Text.TrimStart( zeros );
104         } // end method largerButton_Click
105
106         // determines whether first number
107         // input by user is smaller than second
108   private void smallerButton_Click( object sender, EventArgs e )
109   {
110    // make sure HugeIntegers do not exceed 100 digits
111    if ( SizeCheck( firstTextBox, secondTextBox ) )
112                 return;
113
114    // call Web-service method to determine if
115    // first integer is smaller than second
116    if ( remoteInteger.Smaller( firstTextBox.Text,
117                 secondTextBox.Text ) )
118         resultLabel.Text = firstTextBox.Text.TrimStart( zeros ) +
119                    " is smaller than " +
120                    secondTextBox.Text.TrimStart( zeros );
121    else
122        resultLabel.Text = firstTextBox.Text.TrimStart( zeros ) +
123                    " is not smaller than " +
124                    secondTextBox.Text.TrimStart( zeros );
125    } // end method smallerButton_Click
126
127    // determines whether two numbers input by user are equal
128    private void equalButton_Click( object sender, EventArgs e )
129    {
130     // make sure HugeIntegers do not exceed 100 digits
131     if ( SizeCheck( firstTextBox, secondTextBox ) )
```

```
132                return;
133
134     // call Web-service method to determine if integers are equal
135     if ( remoteInteger.EqualTo( firstTextBox.Text,
136                secondTextBox.Text ) )
137     resultLabel.Text = firstTextBox.Text.TrimStart( zeros ) +
138      " is equal to " + secondTextBox.Text.TrimStart( zeros );
139    else
140      resultLabel.Text = firstTextBox.Text.TrimStart( zeros ) +
141                  " is not equal to " +
142                  secondTextBox.Text.TrimStart( zeros );
143     } // end method equalButton_Click
144
145         // determines whether numbers input by user are too big
146    private bool SizeCheck( TextBox first, TextBox second )
147    {
148      // error message if either number has too many digits
149    if ( ( first.Text.Length > 100 ) ||
150                ( second.Text.Length > 100 ) )
151    {
152     MessageBox.Show("HugeIntegers must be less than 100 digits" ,
153       "Error", MessageBoxButtons.OK, MessageBoxIcon.Information);
154        return true;
155    } // end if
156
157            return false;
158         } // end method SizeCheck
159      } // end class UsingHugeIntegerServiceForm
160  } // end namespace UsingHugeIntegerWebService
```

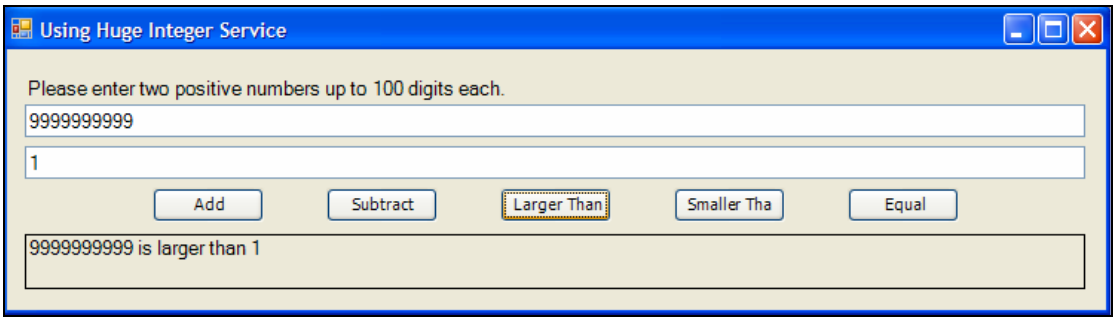
يبين الشكل التالي إضافة عددين:



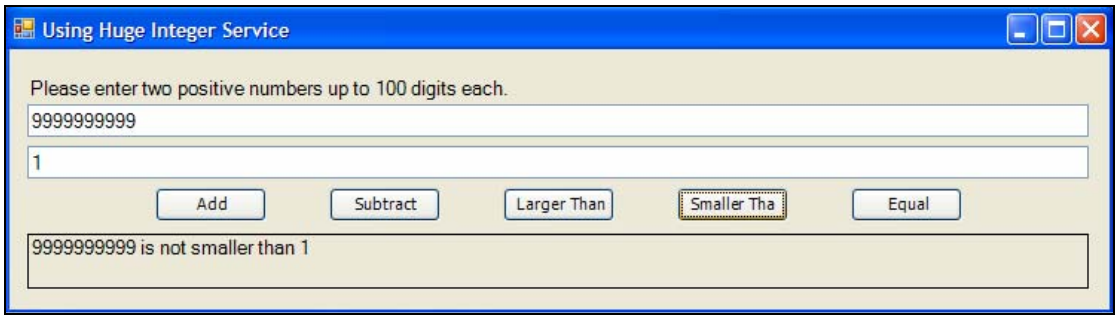
يبين الشكل التالي طرح عددين:



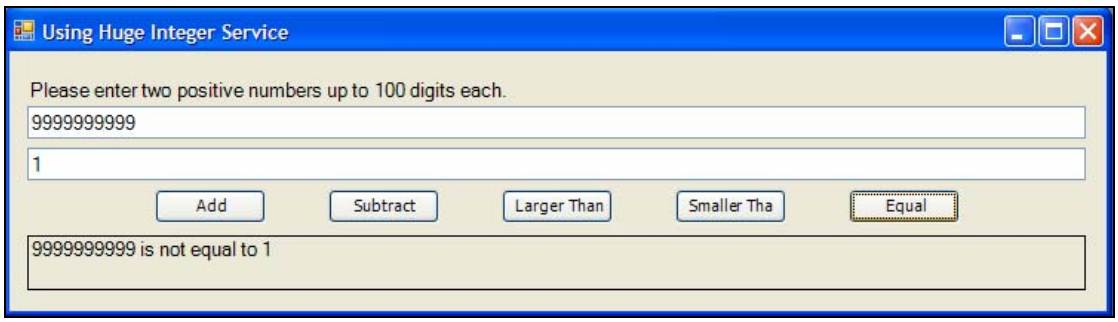
يبين الشكل التالي اختبار أكبر من:



يبين الشكل التالي اختبار أصغر من:



يبين الشكل التالي اختبار المساواة:



5. استخدام خدمات الويب في نماذج الويب

استعرضنا في الأمثلة السابقة من هذا الفصل استهلاك خدمات الويب من قبل نماذج ويندوز **Windows Forms**. سنرى في هذه الفقرة كيفية استهلاك خدمات الويب من قبل نماذج الويب **Web Forms**. لاسيما مع التزايد المطرد لاستخدام نماذج الويب في الأعمال.

سوف نقوم أولاً بإنشاء خدمة ويب تسمح لزبون بإدخال المعلومات عن المقعد الذي يريد حجزه ومن ثم تقوم بحجز المقعد إن كان متاحاً.

ثم سنقوم فيما بعد ببناء تطبيق ويب يستخدم هذه الخدمة من خلال صفحة ويب تسمح بإجراء عملية الحجز.

تملك خدمة الويب **ReservationService** التالية طريقة واحدة **Reserve** (الأسطر **24-42**) تقوم بالبحث في مقاعد قاعدة البيانات **Tickets.Mdf** لتحديد مقعد يطابق رغبة الزبون. إذا وجدت الطريقة مقعد تقوم بتعديل قاعدة البيانات لحجزه و تُعيد **True**. أما إذا لم يتمّ الحجز فتُعيد **False**.

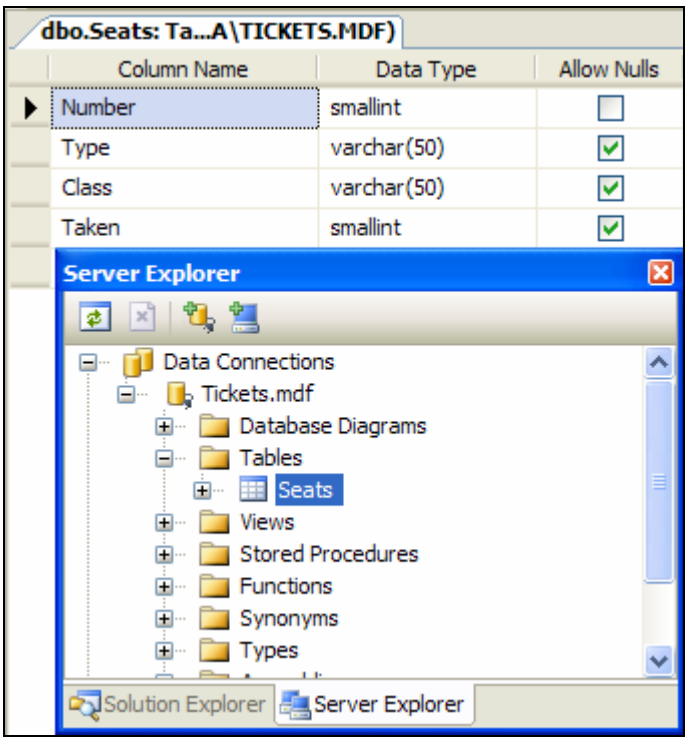
```
1 // ReservationService.cs
2 // Airline reservation Web Service.
3 using System;
4 using System.Web;
5 using System.Web.Services;
6 using System.Web.Services.Protocols;
7
8 [WebService(Namespace = "http://www.svuonline.org/", Description =
9  "Service that enables a user to reserve a seat on a plane." ) ]
10 [ WebServiceBinding( ConformsTo = WsiProfiles.BasicProfile1_1 ) ]
11 public class ReservationService : System.Web.Services.WebService
12    {
13       // create TicketsDataSet object for caching data
14       // from the Tickets database
15  private TicketsDataSet ticketsDataSet = new TicketsDataSet();
16
17  // create SeatsTableAdapter for interacting with the database
18  private TicketsDataSetTableAdapters.SeatsTableAdapter
19  SeatsTableAdapter =
20          new TicketsDataSetTableAdapters.SeatsTableAdapter();
21
22 // checks database to determine whether matching seat is available
23 [ WebMethod( Description = "Method to reserve a seat." ) ]
24 public bool Reserve( string seatType, string classType )
25    {
26      // fill TicketsDataSet.Seats with rows that represent untaken
27      // seats that match the specified seatType and classType
28        SeatsTableAdapter.FillByTypeAndClass(
29             ticketsDataSet.Seats, seatType, classType );
30
```

```
31      // if the number of seats returned is nonzero,
32      // obtain the first matching seat number and mark it as taken
33          if ( ticketsDataSet.Seats.Count != 0 )
34          {
35             string seatNumber = ticketsDataSet.Seats[ 0 ].Number;
36
37             SeatsTableAdapter.UpdateSeatAsTaken( seatNumber );
38             return true; // seat was reserved
39          } // end if
40
41          return false; // no seat was reserved
42       } // end method Reserve
43 } // end class ReservationService
```

تمتلك الطريقة **Reserve** معاملي دخل، الأول هو سلسلة نصية **string** تُحدد نوع المقعد المطلوب:**(Window, Middle, Aisle)**. والثاني سلسلة نصية تُحدد درجة المقعد: **(Economy, First)**.

يحوي الجدول **Seats** في قاعدة البيانات **Tickets.mdf** أربعة أعمدة:

- **Number** رقم المقعد.
- **Type** نوع المقعد **(Window, Middle, Aisle)**.
- **Class** درجة المقعد **(Economy, First)**.
- **Taken** محجوز **(1)** أو متاح **(0)**.



تقوم الأسطر **29-28** بالاستعلام عن أرقام المقاعد المتاحة والتي توافق نوع و درجة مقعد محددين، تقوم هذه التعليمة بملء الجدول **Seat** في مجموعة البيانات **ticketsDataSet** بنتائج الاستعلام التالي:

```
SELECT Number
FROM Seats
WHERE (Taken = 0 ) AND (Type=@type) AND (Class=@class)
```

يتمّ استبدال معاملات الاستعلام السابق **@type** و **@class** بقيم معاملي دخل الطريقة **seatType** و **classType**.

يختبر السطر **33** فيما إذا كان عدد الأسطر في الجدول **Seats** الناتج عن تنفيذ الاستعلام لايساوي الصفر (مما يعني وجود مقعد متاح) وفي هذه الحالة تقوم الطريقة بحجز أول مقعد متاح. حيث يتمّ الحصول على رقم أول مقعد متاح في السطر **35** بالوصول لأول عنصر في الجدول (أي **Seats[0]**). يقوم بعدها السطر **37** باستدعاء الطريقة **UpdateSeatsAsTaken** للموائم **SeatsTableAdapter** ممرراً لها رقم المقعد والتي تقوم بتنفيذ عبارة **SQL** التالية:

```
UPDATE Seats
SET Taken=1
WHERE (Number=@number)
```

تُعيد الطريقة **Reserve** القيمة **true** السطر **(38)** للدلالة على نجاح الحجز. أما إذا لم يكن هناك مقعداً متاحاً فإنها تُرجع **false** السطر **41**.

إضافة مكونة بيانات إلى خدمة ويب

نستعرض فيما يلي الخطوات الواجب إجرائها في **Visaul Studio** لتهيئة مجموعة بيانات **DataSet** تسمح لخدمة الويب السابقة بالتفاعل مع قاعدة البيانات **Tickets.mdf**.

1. الخطوة الأولى: إنشاء خدمة الويب **ReservationService** وإضافة **DataSet** إلى المشروع:
   - ابدأ بإنشاء مشروع من النمط **Web Service** وسمّه **ReservationService**.
   - أعد تسمية الملف **Service.cs** بالاسم **ReservationService.cs**.
   - ضع في الملف **ReservationService.cs** الكود من الفقرة السابقة.
   - أضف مجموعة بيانات **DataSet** للمشروع وسمّها **TicketsDataSet**. سينشئ تلقائياً

موائم جدول **TableAdapter** ويفتح المعالج **Table Adapter Configuration Wizard**.

2. الخطوة الثانية: تحديد مصدر البيانات وإنشاء الاتصال

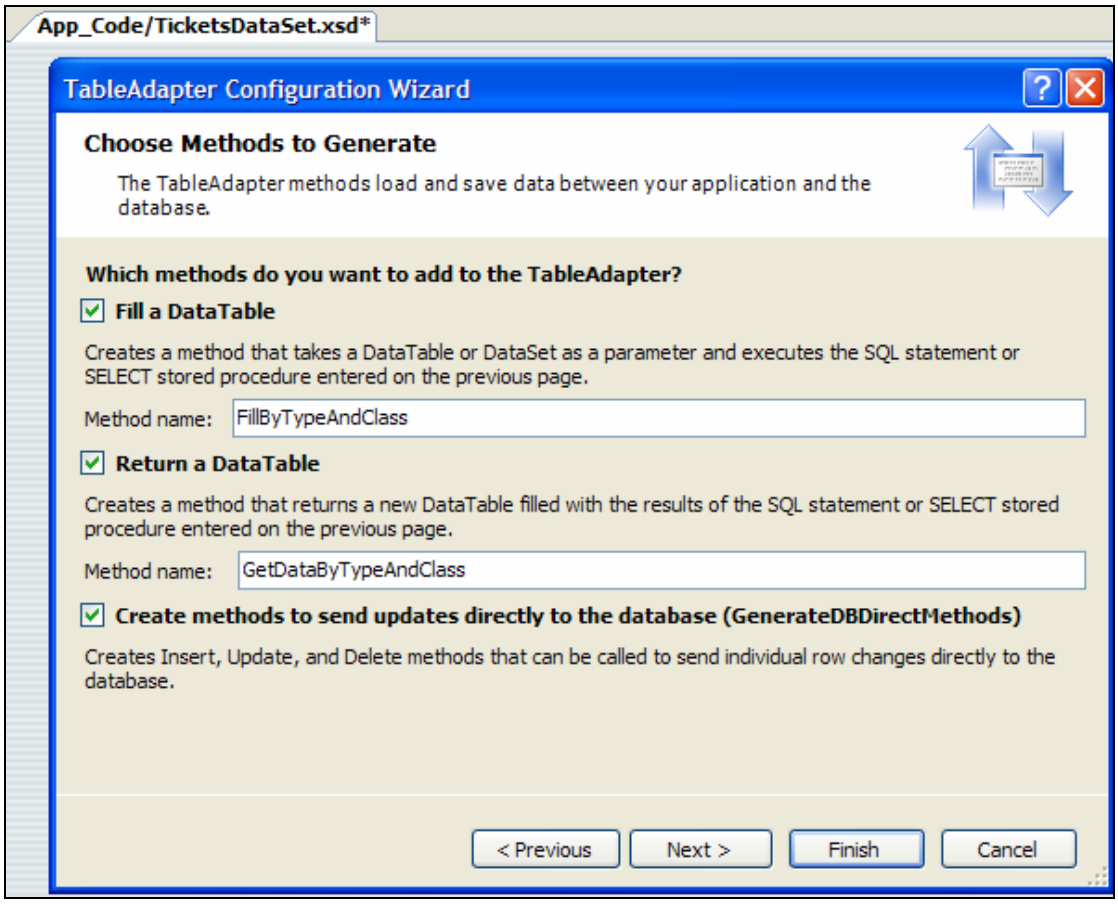
- حدد في نافذة **Add Connection** من المعالج الخيار **Microsoft SQL Server Database File** كمصدر البيانات **Data Source** .
- حدد مسار قاعدة البيانات **Tickets.mdf**.

3. الخطوة الثالثة: تهيئة استعلام للحصول على المقاعد المتاحة

- حدد الاستعلام التالي كمصدر لبيانات موائم الجدول:

```
SELECT Number
FROM Seats
WHERE (Taken = 0 ) AND (Type=@type) AND (Class=@class)
```

أعد تسمية الطرق.



4. الخطوة الرابعة: تهيئة عبارة **Update** لحجز مقعد

- أضف استعلام التحديث التالي إلى موائم الجدول:

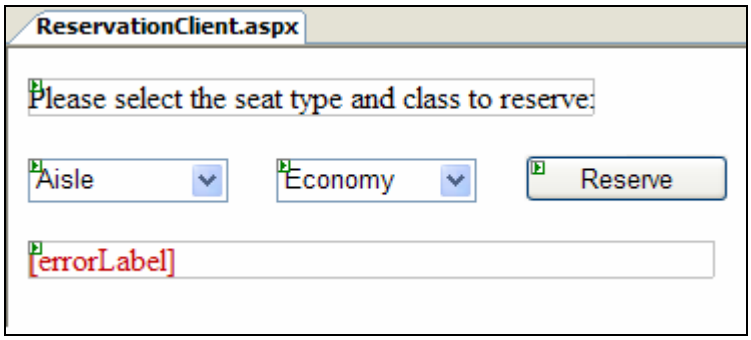
```
UPDATE Seats
SET Taken=1
WHERE (Number =@number)
```

- أعط الاسم **UpdateSeatAsTaken** للطريقة الناتجة.

إنشاء نموذج ويب يستهلك خدمة الويب

يبين الشكل التالي تصميم صفحة الويب والتي تسمح للمستخدم بتحديد كل من درجة المقعد **(Economy, First)** ونوع المقعد **(Aisle, Middle, Window)**.

تستخدم هذه الصفحة خدمة الويب السابقة لتأمين حجز المقعد.



يبين الشكل التالي كود **XML** للصفحة **ReservationClient.aspx** السابقة:

```
1  <%-- ReservationClient.aspx                          --%>
2  <%-- Web Form that allows users to reserve seats on a plane. --%>
3  <%@ Page Language="C#" AutoEventWireup="true"
4    CodeFile="ReservationClient.aspx.cs"
5    Inherits="ReservationClient" %>
6
7  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
8    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
9
10 <html xmlns="http://www.w3.org/1999/xhtml" >
11   <head id="Head1" runat="server">
12      <title>Ticket Reservation</title>
13   </head>
14
15   <body>
16      <form id="form1" runat="server">
17         <div>
18            <asp:Label ID="instructionsLabel" runat="server"
19           Text="Please select the seat type and class to reserve:">
20            </asp:Label><br /><br />
```

```
21
22            <asp:DropDownList ID="seatList" runat="server"
23               Height="22px" Width="100px">
24               <asp:ListItem>Aisle</asp:ListItem>
25               <asp:ListItem>Middle</asp:ListItem>
26               <asp:ListItem>Window</asp:ListItem>
27            </asp:DropDownList>     
28
29     <asp:DropDownList ID="classList" runat="server" Width="100px">
30               <asp:ListItem>Economy</asp:ListItem>
31               <asp:ListItem>First</asp:ListItem>
32            </asp:DropDownList>     
33
34       <asp:Button ID="reserveButton" runat="server" Height="24px"
35               OnClick="reserveButton_Click"
36               Text="Reserve" Width="102px" /><br /><br />
37
38      <asp:Label ID="errorLabel" runat="server" ForeColor="#C00000"
39               Height="19px" Width="343px"></asp:Label>
40         </div>
41      </form>
42   </body>
43 </html>
```

تُعرّف هذه الصفحة قائمتين منسدلتين **DropDownList** وزر أمر **Button** .

تُعرّف الأسطر **22-27** قائمة أنواع المقاعد، بينما تُعرّف الأسطر **29-32** قائمة درجة المقعد.

ينقر المستخدم زر الأمر **reserveButton** (الأسطر **34-36**) لإرسال طلبه للمخدم ليقوم بتنفيذ الإجرائية المرتبطة بالزر **reserveButton_Click**.

تُعرّف الأسطر **38-39** التسمية **errorLabel** والتي تُستخدم لإظهار الرسائل الموافقة للمستخدم.

يبين الشكل التالي الكود الخلفي **ReservationClient.aspx.cs** للصفحة السابقة:

```
1 // ReservationClient.aspx.cs
2 // ReservationClient code behind file.
3 using System;
4 using System.Data;
5 using System.Configuration;
6 using System.Web;
7 using System.Web.Security;
8 using System.Web.UI;
9 using System.Web.UI.WebControls;
10 using System.Web.UI.WebControls.WebParts;
11 using System.Web.UI.HtmlControls;
12
13 public partial class ReservationClient : System.Web.UI.Page
14 {
15 // object of proxy type used to connect to Reservation Web service
16   private localhost.ReservationService ticketAgent =
17      new localhost.ReservationService();
18
```

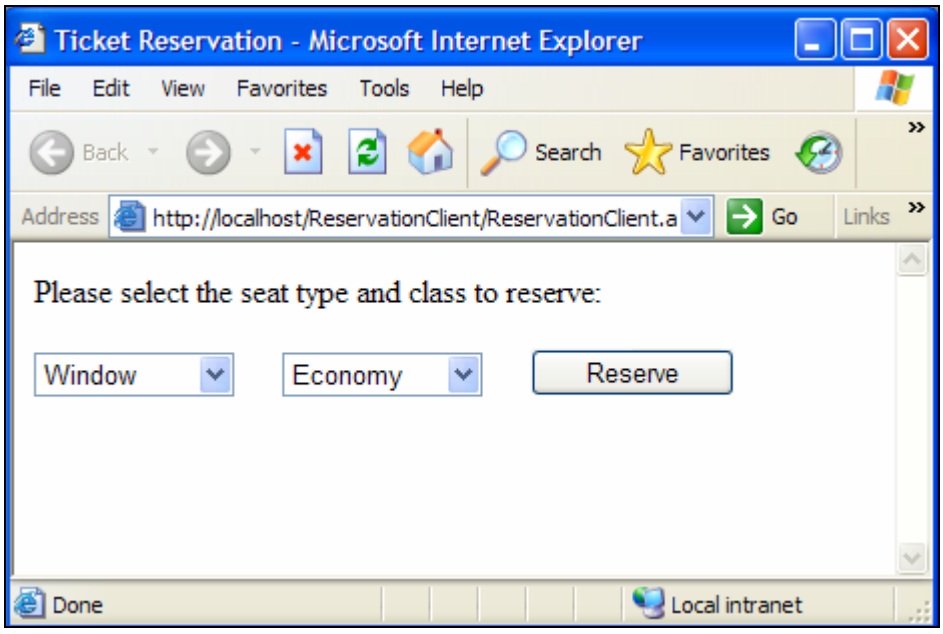
```
19    // attempt to reserve the selected type of seat
20    protected void reserveButton_Click( object sender, EventArgs e )
21    {
22       // if WebMethod returned true, signal success
23       if ( ticketAgent.Reserve( seatList.SelectedItem.Text,
24          classList.SelectedItem.Text.ToString() ) )
25       {
26          // hide other controls
27          instructionsLabel.Visible = false;
28          seatList.Visible = false;
29          classList.Visible = false;
30          reserveButton.Visible = false;
31          errorLabel.Visible = false;
32
33          // display message indicating success
34     Response.Write( "Your reservation has been made. Thank you." );
35       } // end if
36       else // WebMethod returned false, so signal failure
37       {
38          // display message in the initially blank errorLabel
39          errorLabel.Text = "This type of seat is not available. " +
40             "Please modify your request and try again.";
41       } // end else
42    } // end method reserveButton_Click
43 } // end class ReservationClient
```

تُنشئ الأسطر **(17-16)** غرض من النمط **ReservationService**. عندما ينقر المستخدم على الزر **Reserve**، فإن معالج الحدث **reserveButton_Click** الأسطر **(42-20)** ينفذ.

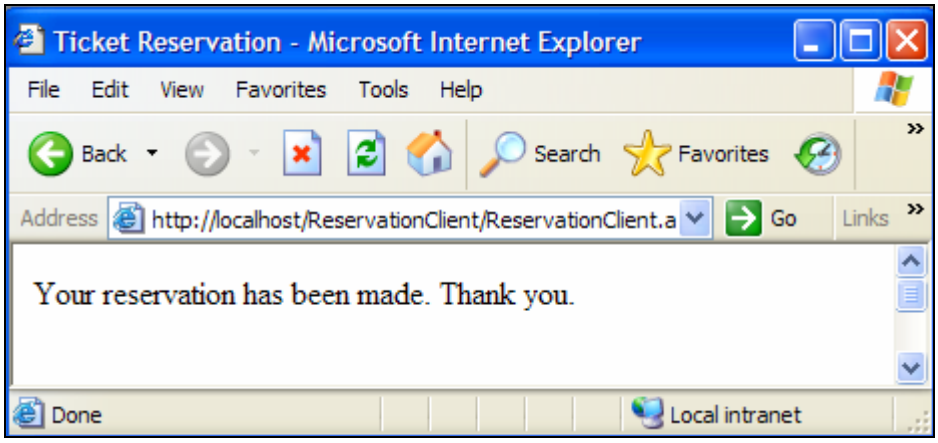
يستدعي معالج الحدث طريقة خدمة الويب **Reserve** ويمرر لها كمعاملي دخل نوع المقعد ودرجته الأسطر **(24-23)**.

إذا أعادت الطريقة **Reserve** القيمة **True**، تظهر رسالة تُعلم المستخدم بإتمام عملية الحجز السطر **(34)**. وإلا، توضع رسالة موافقة في التسمية **errorLabel** لإعلام المستخدم بعدم توفر مقعد من المواصفات المحددة.

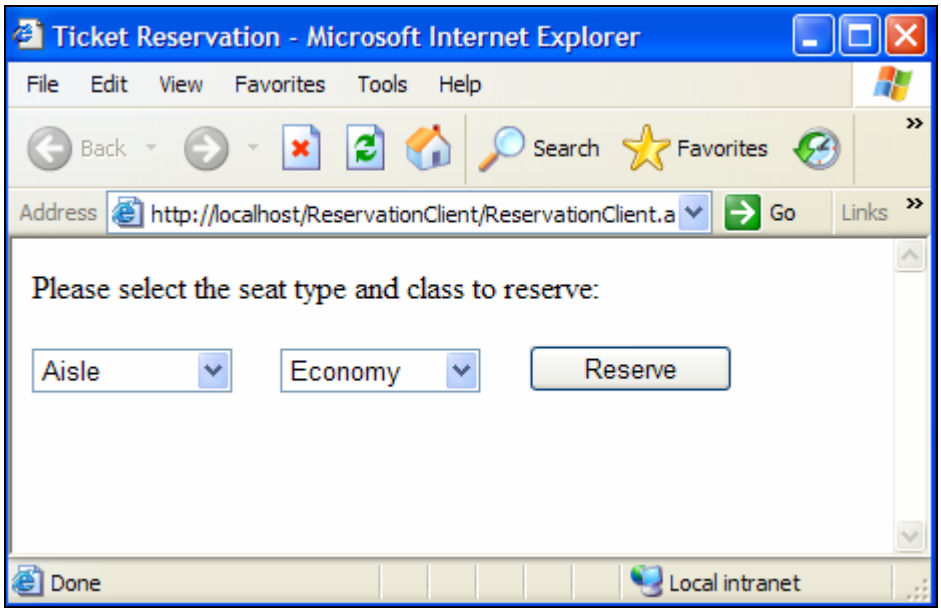
يبين الإيضاح استعراض صفحة الويب.



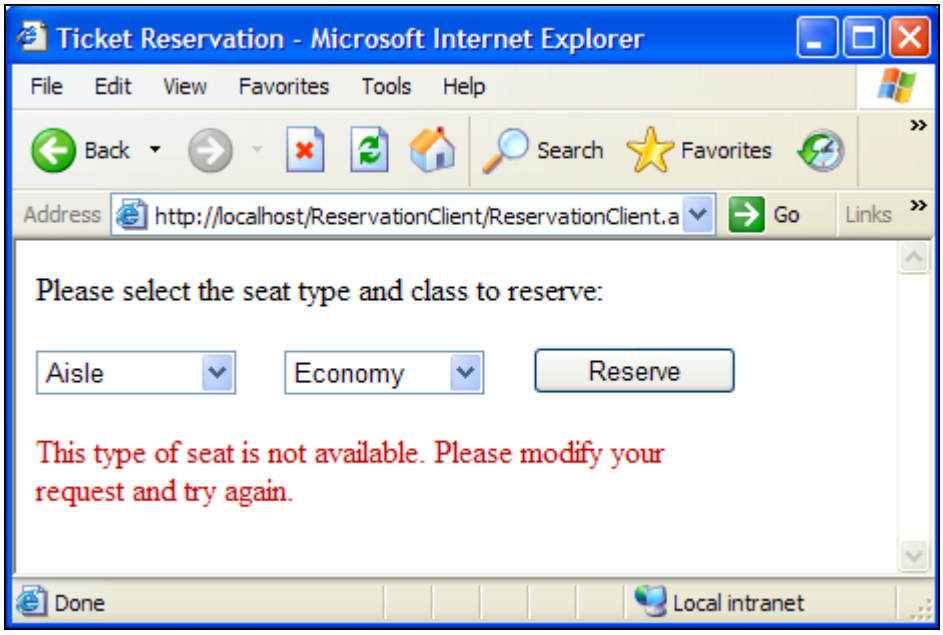
عند اختيار المستخدم لمقعد متاح تظهر الرسالة:



أما عند اختياره لمقعد غير متاح:



فتظهر الرسالة:



6. الأنماط المعرفة في خدمات الويب

تستقبل و تُعيد جميع طرق خدمات الويب التي استعرضناها سابقاً أنماط بيانات بسيطة.

يمكن أيضاً معالجة الأنماط المعرّفة من قبل المبرمج **User-Defined Types**. حيث يمكن تمرير وإعادة هذه الأنماط من طرق خدمة ويب. كذلك، يمكن للزبائن مستهلكي خدمة الويب من استخدام هذه الأنماط لأن الصف **proxy** المنشئ يحوي تعريف هذه الأنماط.

نعرض في هذه الفقرة خدمة ويب تُدعى **EquationGenerator** تقوم بتوليد معادلات جبرية عشوائية من النمط **Equation**.

يكون الزبون في هذه الحالة برنامج رياضيات تعليمي يقوم بسؤال المستخدم عن السؤال المرغوب (جمع، طرح، ضرب) وعن المستوى (أرقام ذات خانة واحدة، خانتين، ثلاث خانات). يمرر هذا البرنامج الزبون رغبات المستخدم لخدمة الويب والتي تُعيد معادلة مع أرقام عشوائية حسب عدد الخانات المطلوب. يستقبل البرنامج الزبون هذه المعادلة ويقوم بعرضها للمستخدم.

وتشمل:

1. النشر المتسلسل للأنماط المعرفة.
2. إنشاء النمط المعرف.

3. إنشاء خدمة الويب.
4. اختبار خدمة الويب.
5. إنشاء الزبون مستهلك خدمة الويب.

النشر المتسلسل للأنماط المعرفة

أشرنا سابقاً بأن جميع الأنماط الممررة من وإلى خدمة ويب يجب أن تُدعم من قبل البروتوكول **SOAP**. إلا أنه كيف يمكن للبروتوكول **SOAP** أن يدعم نمط لم ينشئ بعد؟. يتمّ من أجل ذلك عملية نشر متسلسل للأنماط المعرفة والمرسلة من وإلى خدمة ويب مما يمكّن من تمريرهم بتنسيق **XML**. يدعى هذا الإجراء بالسلسلة **XML Serialization**.

يجب أن تُحقق الصفوف المستخدمة لتحديد النمط المرجع لطريقة ويب أو لمعاملات طريقة ويب المتطلبات التالية:

1. يجب أن تحوي باني عام **public constructor** بدون معاملات.
2. يجب أن تكون الخصائص عامة **public properties**.
3. يجب أن تحوي الخصائص كل من الطريقتين **get** و **set**.

إنشاء النمط المعرف

يعرّف الصف التالي النمط **Eqution**. تُعرّف الأسطر **28-46** باني يأخذ ثلاثة معاملات دخل. رقمين صحيحين يمثلان المعامل اليساري واليميني، وسلسلة نصية تُمثل العملية الجبرية المطلوب القيام بها. يقوم هذا الباني بإسناد معاملات دخله إلى حقول الصف الموافقة **leftOperand, rightOperand, resultValue**. ومن ثم يقوم بحساب نتيجة المعادلة الجبرية وإسنادها إلى حقل الصف **resultValue**. تُعرّف الأسطر **21-25** باني ليس له معاملات، يقوم هذا الباني باستدعاء الباني السابق مع قيم ابتدائية. يعرّف الصف **Equation** خمسة خصائص:

- **LeftHandSide** الأسطر **(56-68)** تُعيد سلسلة نصية تُمثل كل ما يوجد على يسار إشارة المساواة.
- **RightHandSide** الأسطر **(71-82)** تُعيد سلسلة نصية تُمثل كل ما يوجد على يمين إشارة المساواة.
- **Left** الأسطر **(85-96)** تُعيد المعامل اليساري (رقم).
- **Right** الأسطر **(99-110)** تُعيد المعامل الأيمن (رقم).
- **Result** الأسطر **(114-125)** تُعيد الناتج (رقم).

- **Operation** الأسطر **(139-128)** تُعيد العملية (سلسلة نصية).

لايحتاج زبون خدمة الويب إلى تغيير قيم كل من الخصائص **LeftHandSide** و **RightHandSide**. إلا أنه كما أشرنا سابقاً يجب تعريف طريقتي **set** و **get** لكل خاصية.

```
1 // Equation.cs
2 // Class Equation that contains information about an equation.
3 using System;
4 using System.Data;
5 using System.Configuration;
6 using System.Web;
7 using System.Web.Security;
8 using System.Web.UI;
9 using System.Web.UI.WebControls;
10 using System.Web.UI.WebControls.WebParts;
11 using System.Web.UI.HtmlControls;
12
13 public class Equation
14 {
15   private int leftOperand; // number to the left of the operator
16   private int rightOperand; // number to the right of the operator
17   private int resultValue; // result of the operation
18   private string operationType; // type of the operation
19
20   // required default constructor
21   public Equation()
22      : this( 0, 0, "+" )
23   {
24      // empty body
25   } // end default constructor
26
27   // three-argument constructor for class Equation
28   public Equation( int leftValue, int rightValue, string type )
29   {
30      leftOperand = leftValue;
31      rightOperand = rightValue;
32      operationType = type;
33
34      switch ( operationType ) // perform appropriate operation
35      {
36           case "+": // addition
37            resultValue = leftOperand + rightOperand;
38            break;
39         case "-": // subtraction
40            resultValue = leftOperand - rightOperand;
41            break;
42         case "*": // multiplication
43            resultValue = leftOperand * rightOperand;
44            break;
45      } // end switch
46   } // end three-argument constructor
47
48   // return string representation of the Equation object
49   public override string ToString()
50   {
```

```
51       return leftOperand.ToString() + " " + operationType + " " +
52          rightOperand.ToString() + " = " + resultValue.ToString();
53    } // end method ToString
54
55    // property that returns a string representing left-hand side
56    public string LeftHandSide
57    {
58       get
59       {
60         return leftOperand.ToString() + " " + operationType + " " +
61             rightOperand.ToString();
62       } // end get
63
64       set // required set accessor
65       {
66          // empty body
67       } // end set
68    } // end property LeftHandSide
69
70    // property that returns a string representing right-hand side
71    public string RightHandSide
72    {
73       get
74       {
75          return resultValue.ToString();
76       } // end get
77
78       set // required set accessor
79       {
80          // empty body
81       } // end set
82    } // end property RightHandSide
83
84    // property to access the left operand
85    public int Left
86    {
87       get
88{
89          return leftOperand;
90       } // end get
91
92       set
93       {
94          leftOperand = value;
95       } // end set
96    } // end property Left
97
98    // property to access the right operand
99    public int Right
100    {
101       get
102       {
103          return rightOperand;
104       } // end get
105
106       set
107       {
108          rightOperand = value;
109       } // end set
```

```
110   } // end property Right
111
112   // property to access the result of applying
113   // an operation to the left and right operands
114   public int Result
115   {
116      get
117      {
118         return resultValue;
119      } // end get
120
121      set
122      {
123         resultValue = value;
124      } // end set
125   } // end property Result
126
127   // property to access the operation
128   public string Operation
129   {
130      get
131      {
132         return operationType;
133      } // end get
134
135      set
136      {
137         operationType = value;
138      } // end set
139   } // end property Operation
140} // end class Equation
```

إنشاء خدمة الويب

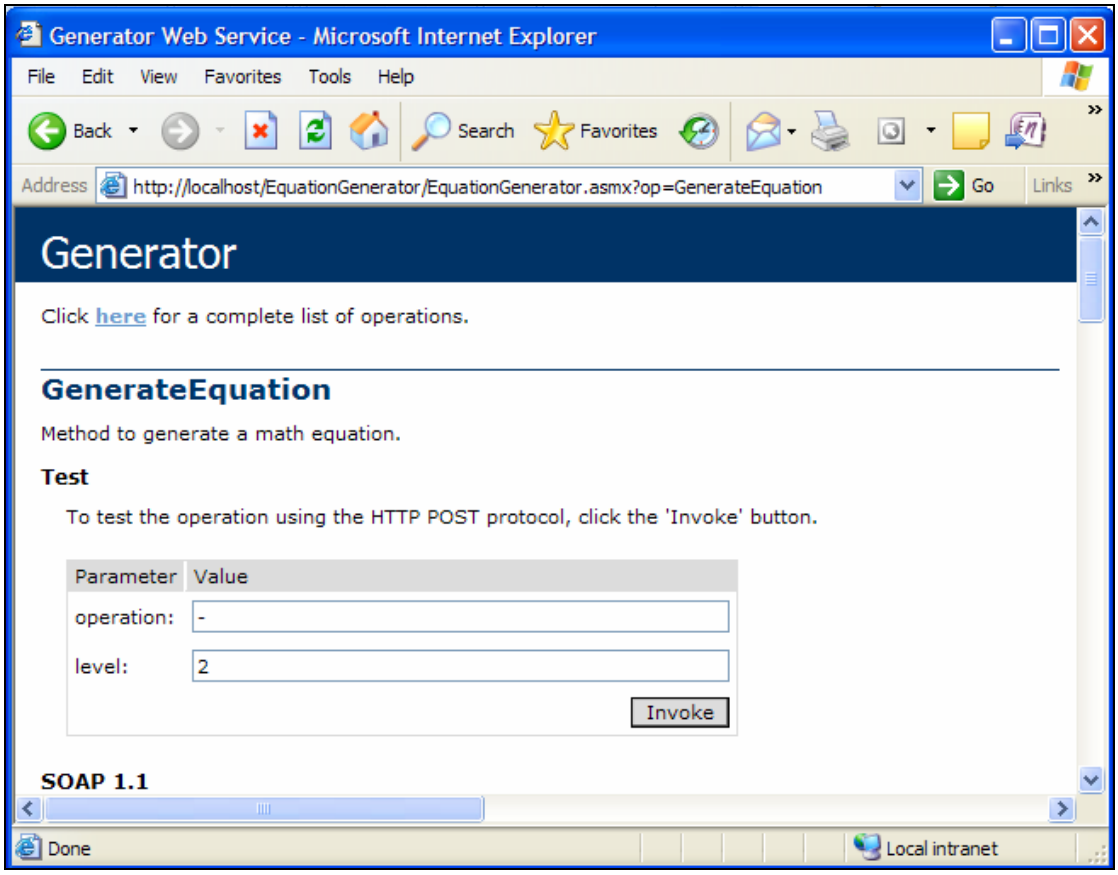
تقوم خدمة الويب التالية **EquationGenerator** بإنشاء معادلات جبرية عشوائية. تحوي هذه الخدمة فقط طريقة الويب **GenerateEquation** الأسطر **(32-16)** والتي تأخذ معاملين الأول سلسلة نصية تُمثل العملية الجبرية والثاني رقم يمثل صعوبة المعادلة حيث يأخذ القيم **(1، 2، 3)** والتي تكون عدد خانات المعاملات.

```
1 // Generator.cs
2 // Web Service to generate random equations based on a specified
3 // operation and difficulty level.
4 using System;
5 using System.Web;
6 using System.Web.Services;
7 using System.Web.Services.Protocols;
8
9 [ WebService( Namespace = "http://www.deitel.com/", Description =
10   "Web service that generates a math equation." ) ]
11 [ WebServiceBinding( ConformsTo = WsiProfiles.BasicProfile1_1 ) ]
12 public class Generator : System.Web.Services.WebService
13 {
```

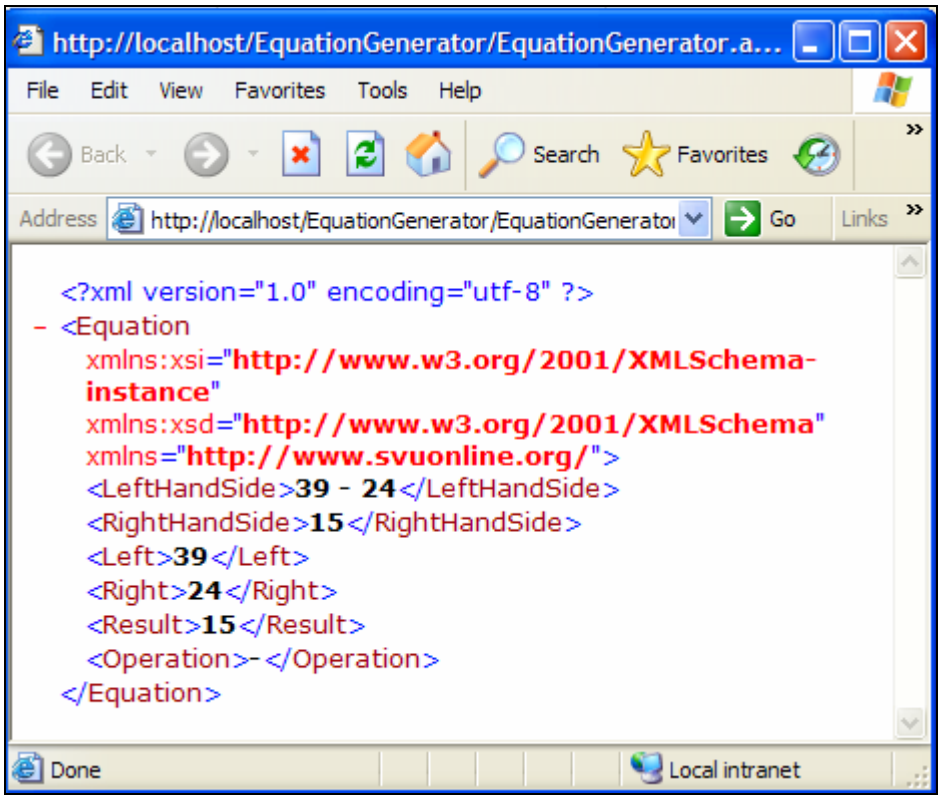
```
14    // Method to generate a math equation
15 [WebMethod(Description = "Method to generate a math equation." ) ]
16    public Equation GenerateEquation( string operation, int level )
17    {
18       // find maximum and minimum number to be used
19       int maximum = Convert.ToInt32( Math.Pow( 10, level ) );
20       int minimum = Convert.ToInt32( Math.Pow( 10, level - 1 ) );
21
22       // object to generate random numbers
23       Random randomObject = new Random();
24
25       // create equation consisting of two random
26       // numbers between minimum and maximum parameters
27       Equation equation = new Equation(
28          randomObject.Next( minimum, maximum ),
29          randomObject.Next( minimum, maximum ), operation );
30
31    return equation;
32    } // end method GenerateEquation
33 } // end class Generator
```

اختبار خدمة الويب

يبين الشكل التالي استدعاء طريقة الويب **EquationGenerator**:



كما يبين الشكل التالي ناتج الاستدعاء والذي هو عبارة عن ملف **XML**. يحدد هذا الملف جميع قيم الخصائص العامة **public** وبيانات الغرض المرجع



إنشاء الزبون مستهلك خدمة الويب

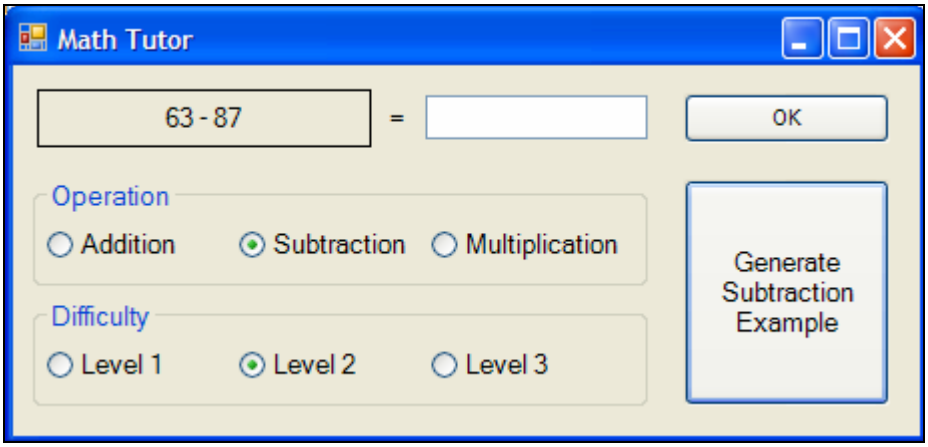
يستخدم التطبيق **MathTutor** التالي خدمة الويب **EquationGenerator**. يستدعي هذا التطبيق طريقة خدمة الويب **GenerateEquation** لإنشاء غرض من النمط **Equation**. ثم يقوم بعرض الطرف الأيسر من المعادلة وينتظر جواب المستخدم.

```
1 // MathTutor.cs
2 //Math tutoring program using WS to generate random equations.
3 using System;
4 using System.Collections.Generic;
5 using System.ComponentModel;
6 using System.Data;
7 using System.Drawing;
8 using System.Text;
9 using System.Windows.Forms;
10
11 namespace MathTutor
12 {
13   public partial class MathTutorForm : Form
14   {
15      public MathTutorForm()
16     {
```

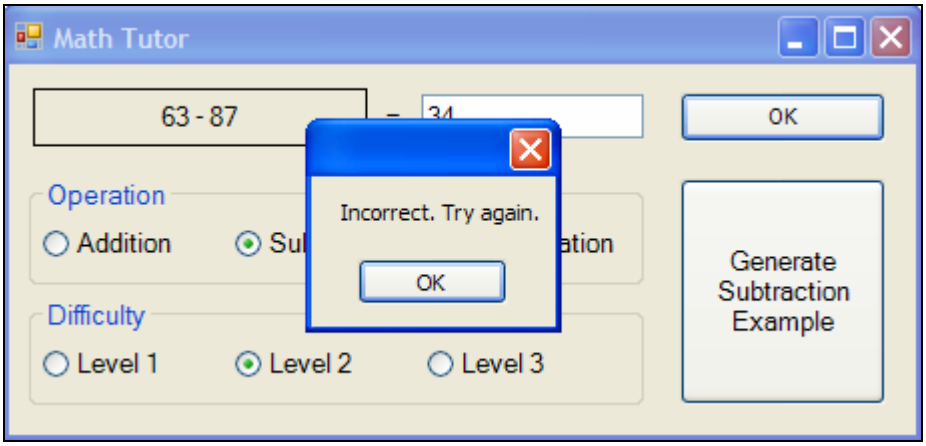
```
17          InitializeComponent();
18       } // end constructor
19
20       private string operation = "+";
21       private int level = 1;
22       private localhost.Equation equation;
23 private localhost.Generator generator = new localhost.Generator();
24
25       // generates new equation when user clicks button
26    private void generateButton_Click( object sender, EventArgs e )
27       {
28          // generate equation using current operation and level
29          equation = generator.GenerateEquation( operation, level );
30
31          // display left-hand side of equation
32          questionLabel.Text = equation.LeftHandSide;
33
34          okButton.Enabled = true;
35          answerTextBox.Enabled = true;
36       } // end method generateButton_Click
37
38       // check user's answer
39       private void okButton_Click( object sender, EventArgs e )
40       {
41          // determine correct result from Equation object
42          int answer = equation.Result;
43
44          if ( answerTextBox.Text == "" )
45             return;
46
47          // get user's answer
48          int userAnswer = Int32.Parse( answerTextBox.Text );
49
50          // determine whether user's answer is correct
51          if ( answer == userAnswer )
52          {
53             questionLabel.Text = ""; // clear question
54             answerTextBox.Text = ""; // clear answer
55             okButton.Enabled = false; // disable OK button
56             MessageBox.Show( "Correct! Good job!" );
57          } // end if
58          else
59             MessageBox.Show( "Incorrect. Try again." );
60       } // end method okButton_Click
61
62       // set difficulty level to 1
63    private void levelOneRadioButton_CheckedChanged( object sender,
64          EventArgs e )
65       {
66          level = 1;
67       } // end method levelOneRadioButton_CheckedChanged
68
69    // set difficulty level to 2
70    private void levelTwoRadioButton_CheckedChanged( object sender,
71          EventArgs e )
72       {
73          level = 2;
74       } // end method levelTwoRadioButton_CheckedChanged
75
```

```
76       // set difficulty level to 3
77    private void levelThreeRadioButton_CheckedChanged( object sender,
78          EventArgs e )
79       {
80          level = 3;
81       } // end method levelThreeRadioButton_CheckedChanged
82
83       // set the operation to addition
84      private void additionRadioButton_CheckedChanged( object sender,
85          EventArgs e )
86       {
87          operation = "+";
88          generateButton.Text =
89             "Generate " + additionRadioButton.Text + " Example";
90       } // end method additionRadioButton_CheckedChanged
91
92       // set the operation to subtraction
93 private void subtractionRadioButton_CheckedChanged( object sender,
94          EventArgs e )
95       {
96          operation = "-";
97          generateButton.Text = "Generate " +
98             subtractionRadioButton.Text + " Example";
99       } // end method subtractionRadioButton_CheckedChanged
100
101       // set the operation to multiplication
102       private void multiplicationRadioButton_CheckedChanged(
103          object sender, EventArgs e )
104       {
105          operation = "*";
106          generateButton.Text = "Generate " +
107             multiplicationRadioButton.Text + " Example";
108       } // end method multiplicationRadioButton_CheckedChanged
109    } // end class MathTutorForm
110 } // end namespace MathTutor
```

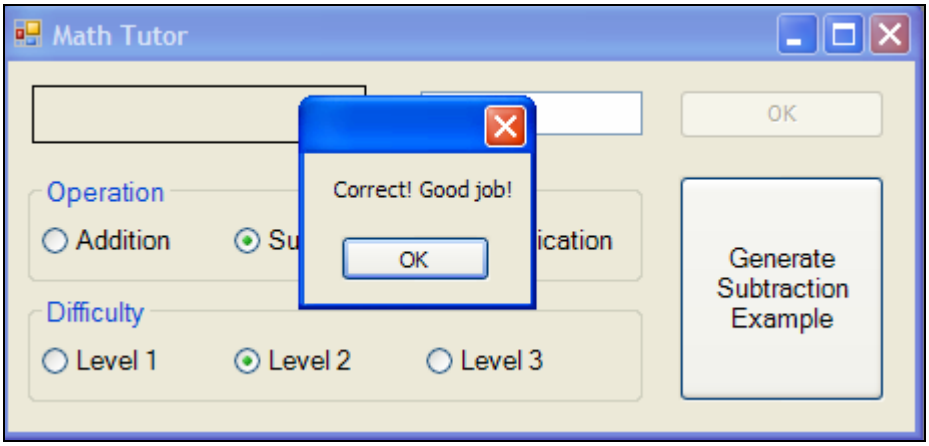
يبين الشكل التالي تنفيذ البرنامج:



في حال إدخال قيمة خاطئة تظهر رسالة خطأ موافقة:



وفي حال إدخال قيمة الجواب الصحيح:



7. الخلاصة

استعرضنا في هذا الفصل خدمات الويب كتقانة تسمح للمبرمجين بطلب واستقبال نتائج تنفيذ طرق عبر الشبكة مما يسمح باستخدام أمثلي للطرق في النظم الموزعة (إعادة استخدام، توزيع المهام، ...).

كما بيّنا بأن خدمة الويب هي صف يسمح لمختلف الزبائن باستدعاء طرقه عبر الشبكة باستخدام قوالب وبروتوكولات مثل **XML** و**HTTP** و **SOAP**.

أوضحنا الميزات العديدة لهذا المنهج في الحسابات الموزعة، كوصول الزبائن إلى بيانات محددة على حواسب بعيدة وتمتعهم بمقدرات هذه الحواسب البعيدة.