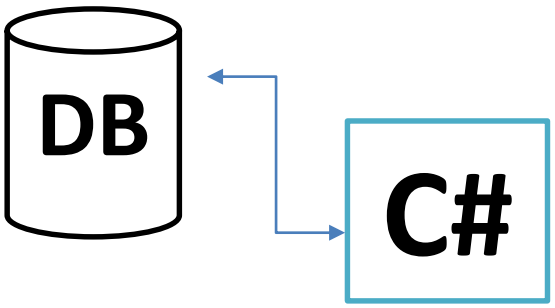
المعهد المتوسط لتقنيات الحاسوب

قسم البرمجة

مقرر برمجة متقدمة

السنة الثانية / الفصل الدراسي الأول

إعداد المهندسة: نجوى ناصرآغا



العام الدراسي2014 – 2015

الفهرس

الاتصال مع قواعد البيانات

محاور الجلسة الأولى:

- مقدمة.
- إنشاء قاعدة البيانات
- الإجرائيات المخزنة Stored Procedure
- بناء الاتصال بقاعدة البيانات
- إرسال الأوامر إلى قاعدة البيانات
- نسخ قاعدة البيانات من جهاز لآخر.

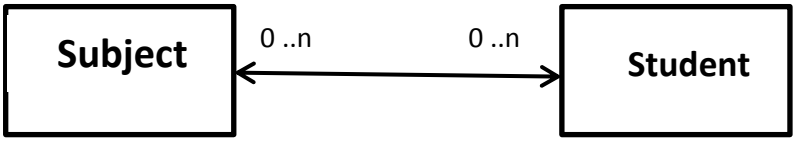
مقدمة:

تحتاج أغلب تطبيقات الحاسب إلى مخزن يحوي جميع البيانات اللازمة لعمل التطبيق، لحفظ بيانات المستخدمين والبيانات اللازمة لعمل التطبيق يسمى هذا المخزن بقاعدة البيانات.
في هذه الجلسة سنتعلم كيفية بناء الاتصال بين التطبيقات و قواعد البيانات وكيفية التعامل مع هذا الاتصال حيث سيتم توضيح الخطوات من خلال مثال عملي

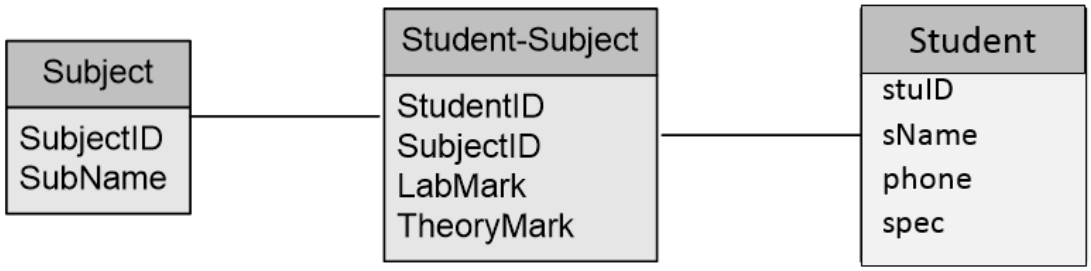
مثال:

لنفترض أننا نريد بناء نظام بسيط لإدارة بيانات الطلاب في المعهد، حيث نحتاج لتخزين بيانات الطالب والمواد التي يقوم بدراستها، إضافة لدرجاته في هذه المواد.

أي أن النظام مكون من الكائنات الأساسية التالية:



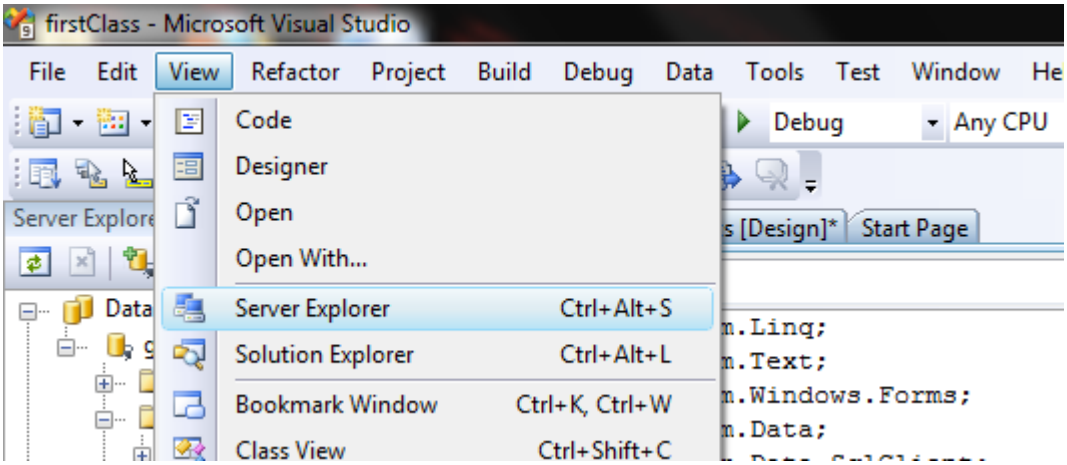
وبما أن الطالب يدرس أكثر من مادة، والمادة يمكن أن يدرسها أكثر من طالب، فإنه يجب كسر العلاقة بين الكائنين، وبالتالي يصبح الشكل النهائي كالتالي:



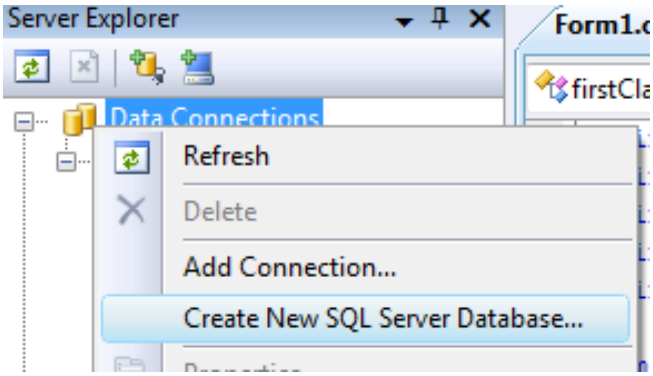
وبعد القيام بتحليل النظام سنقوم بإنشاء قاعدة البيانات.

أولاً: إنشاء قاعدة البيانات:

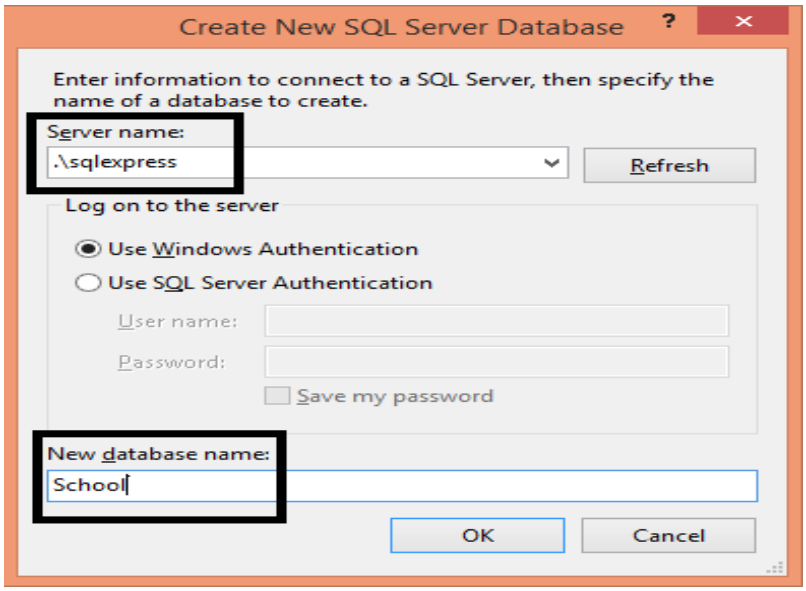
1 - من القائمة View نختار الأمر Server Explorer .



2 - نقوم بإنشاء قاعدة بيانات جديدة وذلك من خلال النقر بزر الفأرة اليميني على خيار Data Connections ثم اختيار Create New SQL Server Database.

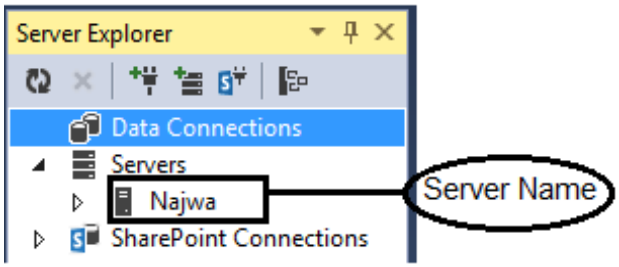


فتظهر لدينا الواجهة التالية لتحديد السيرفر الذي نريد إنشاء قاعدة البيانات فيه.

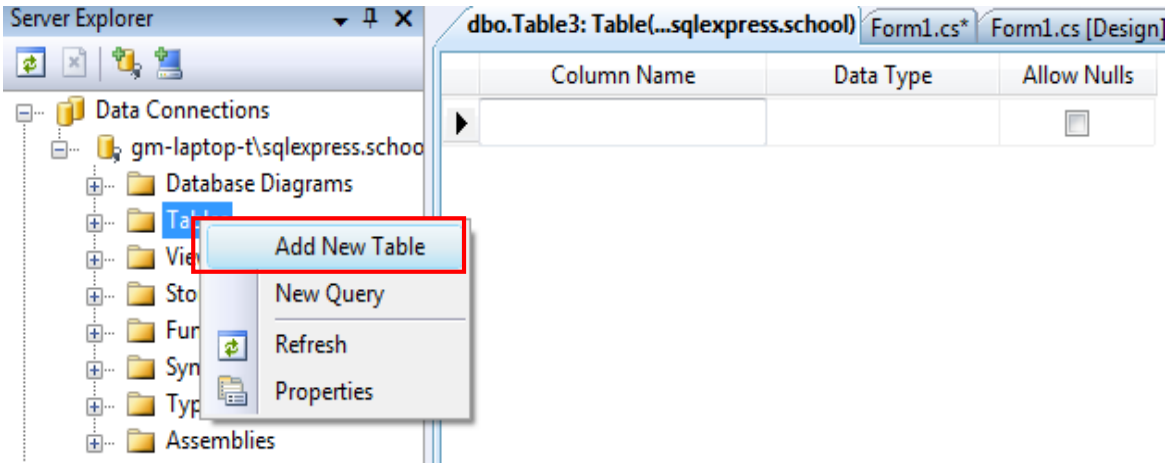


حيث نكتب ما يلي :" sqlexpress\. " ونحدد اسم قاعدة البيانات.

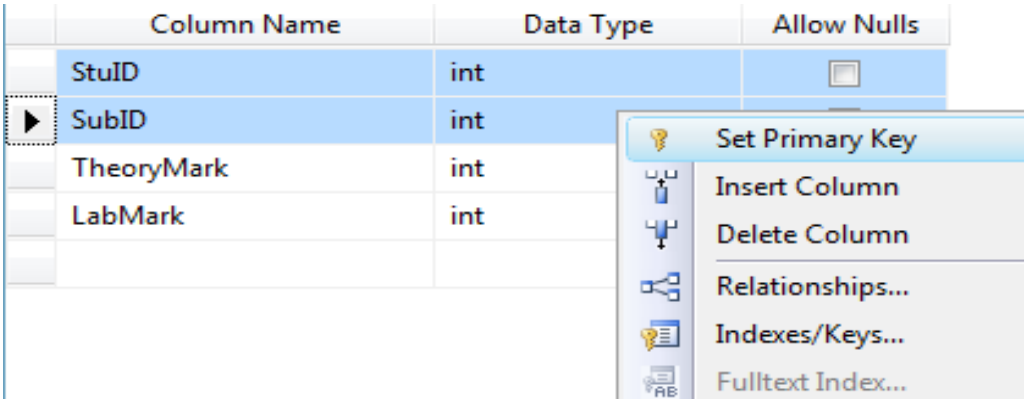
ملاحظة: في حال إنشاء قاعدة البيانات على الـ Sql Server بدل الـ SqlExpress، نقوم بكتابة اسم السيرفر في خانة Server name وذلك بدل كتابة sqlexpress\.
حيث يمكن معرفة اسم السيرفر من لائحة Server Explorer ضمن الـ Servers كما في الشكل التالي:



3 - يظهر لنا محرر Sqlserver حيث نقوم بإنشاء جدول جديد من خلال الضغط على مجلد Tables ضمن قاعدة البيانات التي قمنا بإنشائها بالزر الأيمن واختيار الأمر Add new table :



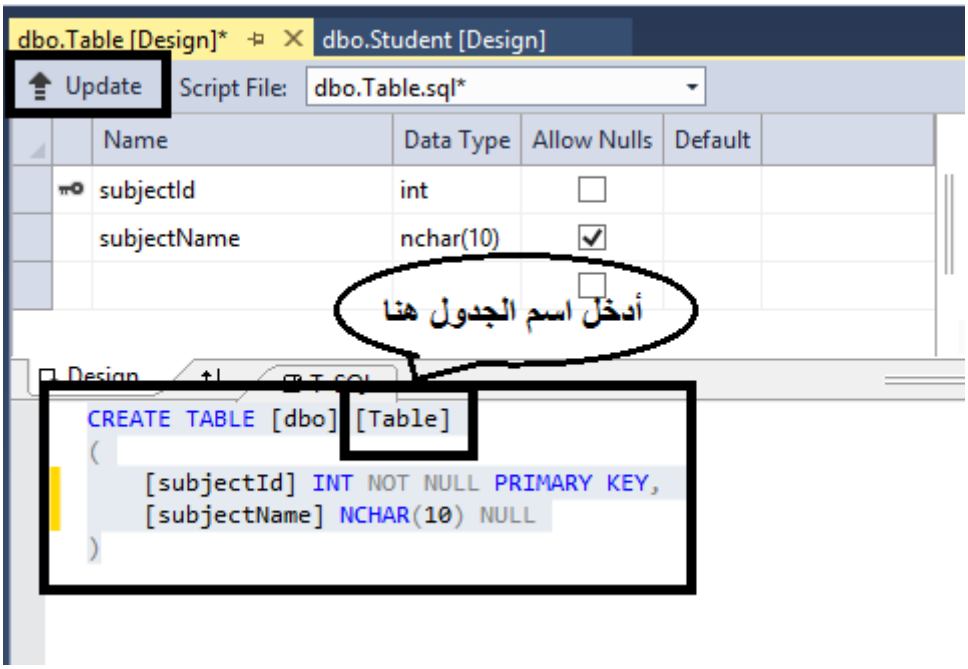
نقوم بإدخال اسم الحقل و نوعه، لتحديد المفتاح الرئيسي ننقر على الحقل المراد تحديده باليمين ثم نختار set as primary key. وفي حال أردنا تحديد أكثر من حقل كمفتاح رئيسي نحدد الحقل الأول ثم نضغط ctrl ونحدد الحقل الآخر ثم ننقر باليمين ونختار Set as Primary key.



لحفظ الجدول بعد الانتهاء من بنائه:

- بالنسبة للـ Visual Studio 2010 وماقبل نكتفي بضغط ctrl+s وإدخال اسم الجدول.
- بالنسبة للـ Visual Studio 2012 والنسخ الأحدث:

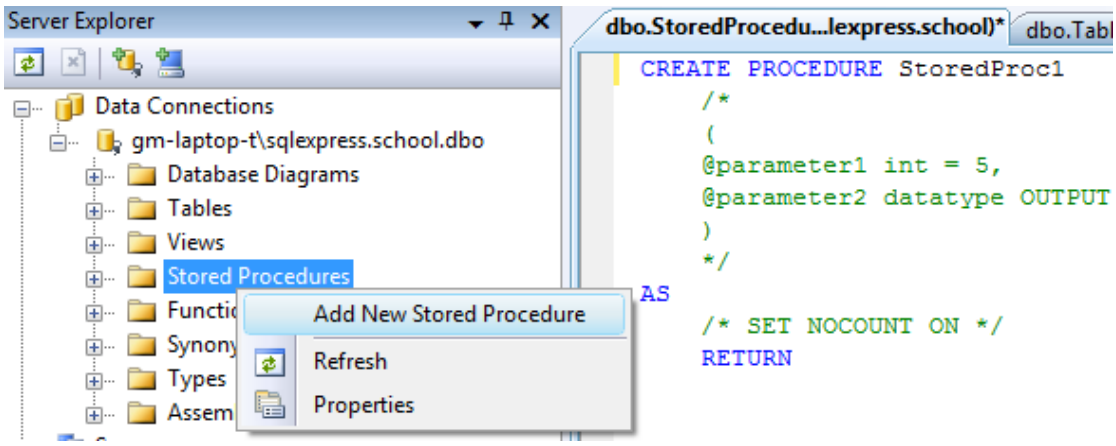
نلاحظ في واجهة إنشاء الجدول ظهور نص Sql في اسفل الشاشة هو عبارة عن شيفرة إنشاء الجدول، نعدل في النص كما هو مبين في الشكل التالي، حيث نستبدل كلمة Table باسم الجدول (Subject) مثلا، ثم نضغط على زر Update الموجود فوق الجدول (وهو محدد في الشكل التالي)، وفي الواجهة التي تظهر بعدها نختار Update Database، فيقوم البرنامج بحفظ الجدول.



ثانياً: بناء الإجرائيات المخزنة (Stored Procedure):

الإجراءات المخزنة هي مجموعة عبارات مكتوبة بلغة TSQL تقوم بتنفيذ عمليات متكررة على قاعدة البيانات ، وتتمثل أهميتها في أن تنفيذ العمليات المتضمنة فيها أسرع ، لأنها compiled ، وتقلل من الضغط على موارد الحاسب ، لأنها ترسل ال parameters فقط ، ويتم تنفيذ العمليات في السيرفر، وترجع النتائج النهائية فقط،

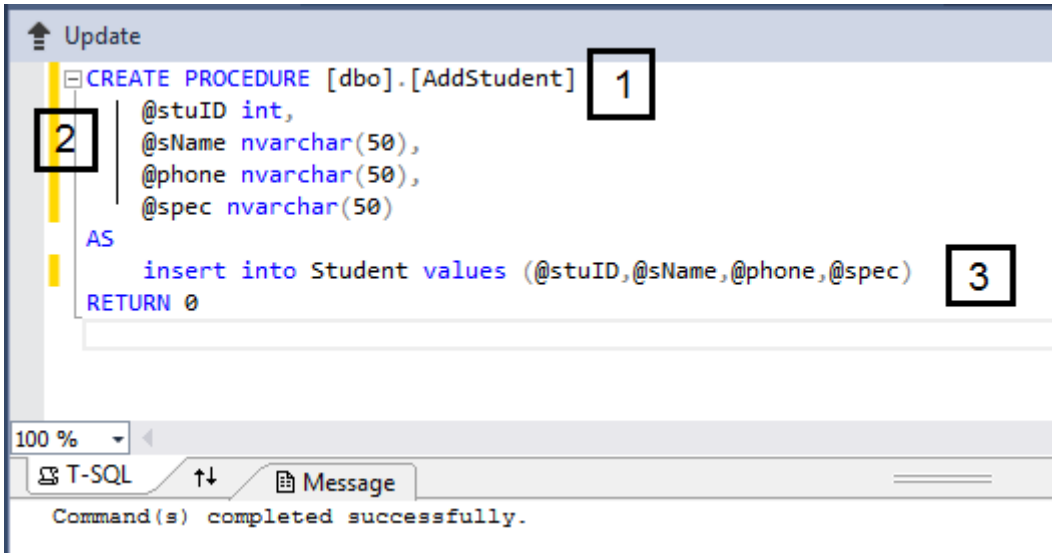
بدلا من إرجاع نتيجة كل عملية بحث لكل مستخدم للسيرفر، إضافة لذلك تدعم مبدأ إعادة استخدام البرمجيات Software Reusabilityحيث يمكن الاستعانة بها دائما دون الحاجة لإعادة كتابتها. في أي وقت.



عند كتابة الإجرائية نركز على التفاصيل التالية:

1- اسم الإجرائية.
2- المتحولات: حيث أن المتحولات تبدأ برمز @ دوماً، كما أننا نكتب اسم المتحول ثم نوعه، كما نستطيع تحديد قيمته الافتراضية، كما نقوم بوضع فاصلة للفصل بين المتحول والآخر.
3- جسم الإجرائية حيث يتم كتابة الكود بعبارات الـ SQL.

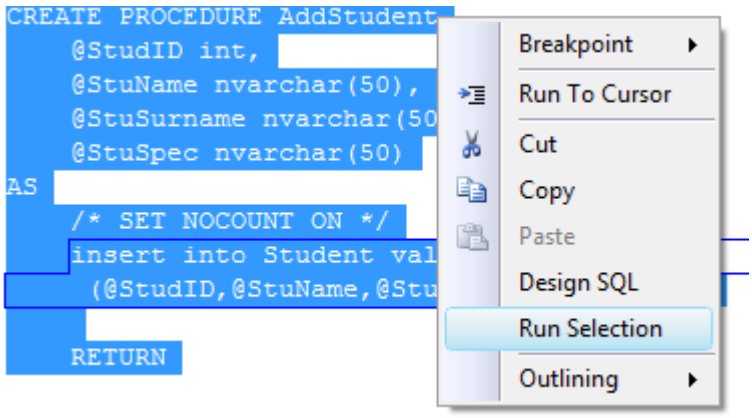
مثلاً يمكننا كتابة إجرائية إضافة طالب AddStudent كالتالي:



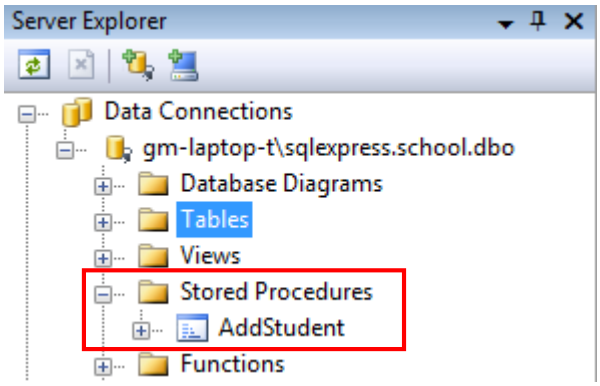
وبعد الانتهاء من كتابة الإجرائية نقوم بتحديد كامل الإجرائية وننقر باليمين ثم نختار Run Selection.

في نسخة VS2012 والنسخ الأحدث نختار Execute.

وفي حال صحة الكود يجب أن تظهر عبارة .Command(s) completed successfully.



حيث أنه يقوم بتنفيذ الكود، وفي حال نجاح التنفيذ فيجب أن تظهر الإجرائية ضمن مسار ال Stored Procedure



ثالثاً: بناء التطبيق:

نقوم بإنشاء الواجهة التالية:

وفي حدث الضغط على زر إضافة طالب نقوم بكتابة الكود التالي:

```
private void AddBtn_Click(object sender, EventArgs e)
{
    int stuID = Convert.ToInt32(IDTxt.Text);
    string name = NameTxt.Text;
    string phone = phoneTxt.Text;
    string spec = specTxt.Text;
    resultLbl.Text =
              AddStudent(stuID,name,phone,spec);
}
```

حيث تعيد إجرائية AddStudent قيمة نصية توضح نجاح الإجرائية أو فشلها و تحتوي على الخطوات التالية:

1- نكتب صيغة الإجرائية AddStudent في نفس صفحة الكود الخاصة بالـ Form1 كالتالي:

```
public string AddStudent(int stuID, string name, string
phone, string spec)
    { return "";}
```

ونجعلها تعيد قيمة نصية فارغة (مؤقتا حتى نفرغ من كتابة الشيفرة).

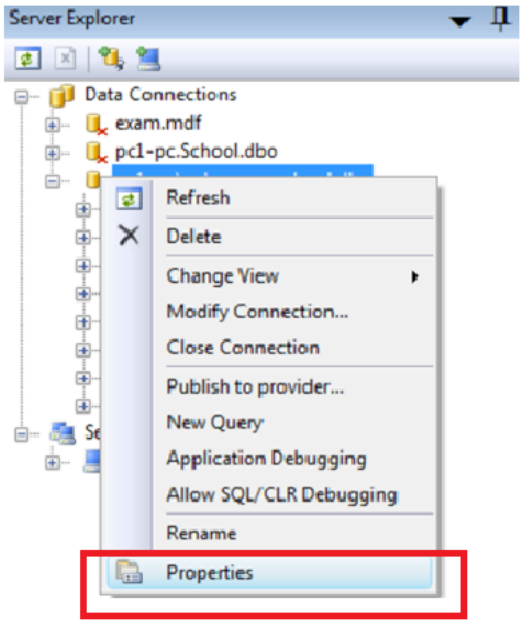
2- قبل البدء بالاتصال بقاعدة البيانات يجب استدعاء المكتبتين التاليتين المخصصتين للتعامل مع قواعد البيانات:

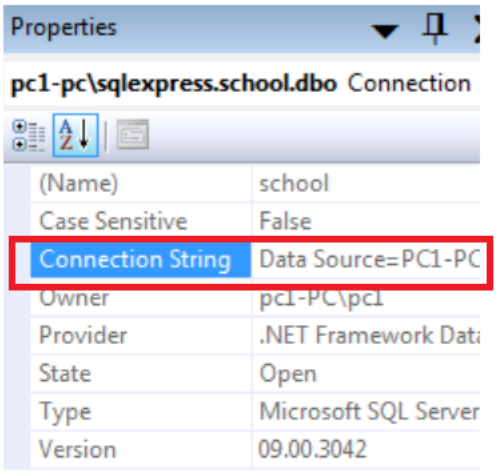
```
using System.Data;
using System.Data.SqlClient;
```

3- إنشاء الاتصال

```
//1- بناء الاتصال
SqlConnection myConn = new SqlConnection();
myConn.ConnectionString = @"Data Source=GM-LAPTOP-T\SQLEXPRESS;Initial
        Catalog=school;Integrated Security=True;Pooling=False";
```

حيث يتطلب الاتصا ل معرفة ال connection string الخاص بقاعدة البيانات والذي يمكن معرفته كالتالي: من لوحة ال Server Explorer ننقر باليمين على قاعدة البيانات ونختار البند Properties





فتظهر لدينا اللائحة Properties ونقوم بنسخ ال Connection string منها، ونلصقه كما يلي مع الانتباه لوضع إشارة @ قبله، لتفدي مشاكل الرموز الغريبة.

```
//1- بناء الاتصال
SqlConnection myConn = new SqlConnection();
myConn.ConnectionString = @"Data Source=GM-LAPTOP-T\SQLEXPRESS;Initial
        Catalog=school;Integrated Security=True;Pooling=False";
```

4- إنشاء الأمر SQLCommand

```
//2- بناء الأمر
SqlCommand com1 = new SqlCommand();
com1.CommandType = CommandType.StoredProcedure;
com1.CommandText = "AddStudent";//اسم الإجرائية المخزنة
com1.Connection = myConn;
```

إنشاء البارامترات التي سيتم إرسالها لقاعدة البيانات

```
SqlParameter[] myArr = new SqlParameter[4];
myArr[0] = new SqlParameter("@stuID", stuID);
myArr[1] = new SqlParameter("@sName", name);
myArr[2] = new SqlParameter("@phone", phone);
myArr[3] = new SqlParameter("@spec", spec);
com1.Parameters.AddRange(myArr);
```

5- الاتصال بقاعدة البيانات، تنفيذ الأمر، قطع الاتصال.

```
//4- فتح الاتصال وتنفيذ الأمر
try
{
    myConn.Open();
    com1.ExecuteNonQuery();
    myConn.Close();
    return "تمت عملية الإضافة بنجاح";

}
catch (Exception e1)
{
    myConn.Close();
    return e1.Message;//في حال حدوث خطأ تظهر لدينا رسالة الخطأ

}
```

حيث أنه لفتح الاتصال نكتب ;()myConn.Open

لتنفيذ الأمر نكتب ;()com1.ExecuteNonQuery

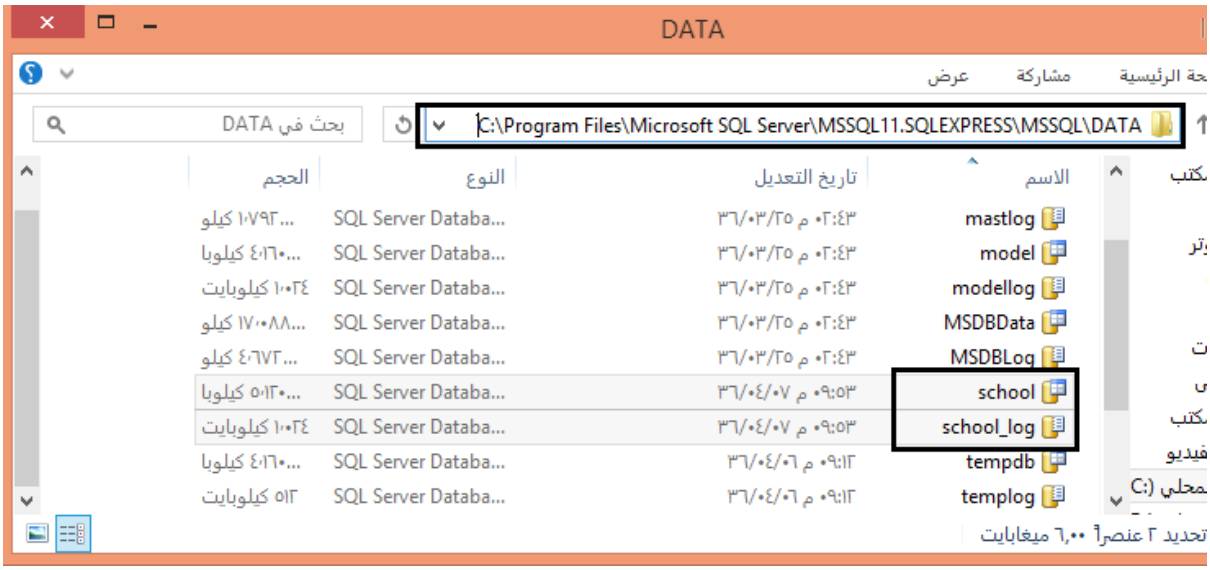
لإغلاق الاتصال ;()myConn.Close

ثم نقوم بتنفيذ البرنامج.

نسخ قاعدة البيانات من جهاز لآخر:

هناك أكثر من طريقة لنسخ قاعدة بيانات مشروعك من حاسب لآخر:

1- في حال كانت قاعدة البيانات مبنية باستخدام الـ SqlExpress:
كل قاعدة بيانات تكون مكونة من ملفين الملف الأساسي وملف الـ log.
نقوم بنسخ ملفي قاعدة البيانات من المسار التالي:
C:\Program Files\Microsoft SQL Server\MSSQL11.SQLEXPRESS\MSSQL\DATA

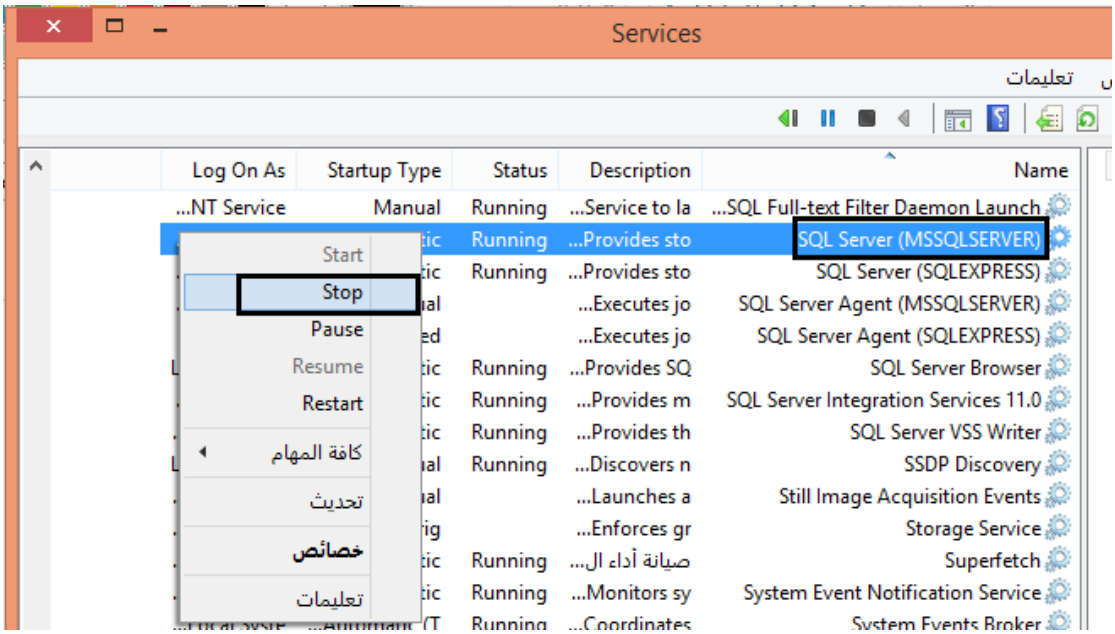


2- في حال كانت قاعدة البيانات مبنية باستخدام الـ SqlServer هنا لايمكن نسخ الملف من مساره مباشرة بل يجب إيقاف السيرفر أو فصل قاعدة البيانات، هناك أكثر من طريقة ولكن الطريقة الأفضل هي كالتالي:

أولا : من لوحة التحكم – الأدوات الإدارية – الخدمات

(ControlPanel→ Administartive Tools →Services)

نبحث عن خدمة اسمها Sql Server (MSSQLSERVER) وننقر عليها باليمين ونختار الخيار إيقاف Stop.



ثانيا : نذهب إلى المجلد الذي يحوي ملفي قاعدة البيانات ونقوم بنسخهما:

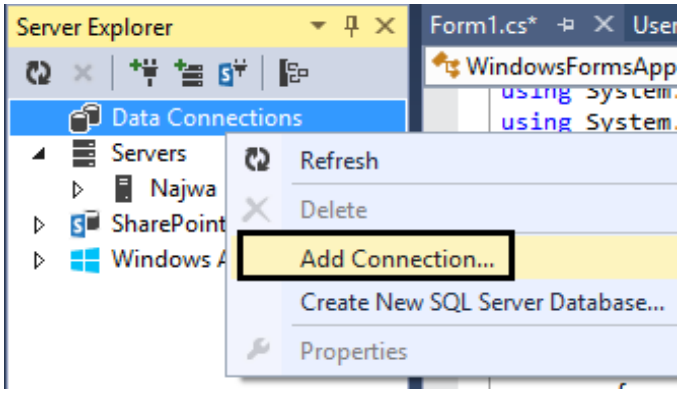
C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA

ثالثا: بعد الانتهاء من النسخ نقوم بإعادة تشغيل الخدمة.

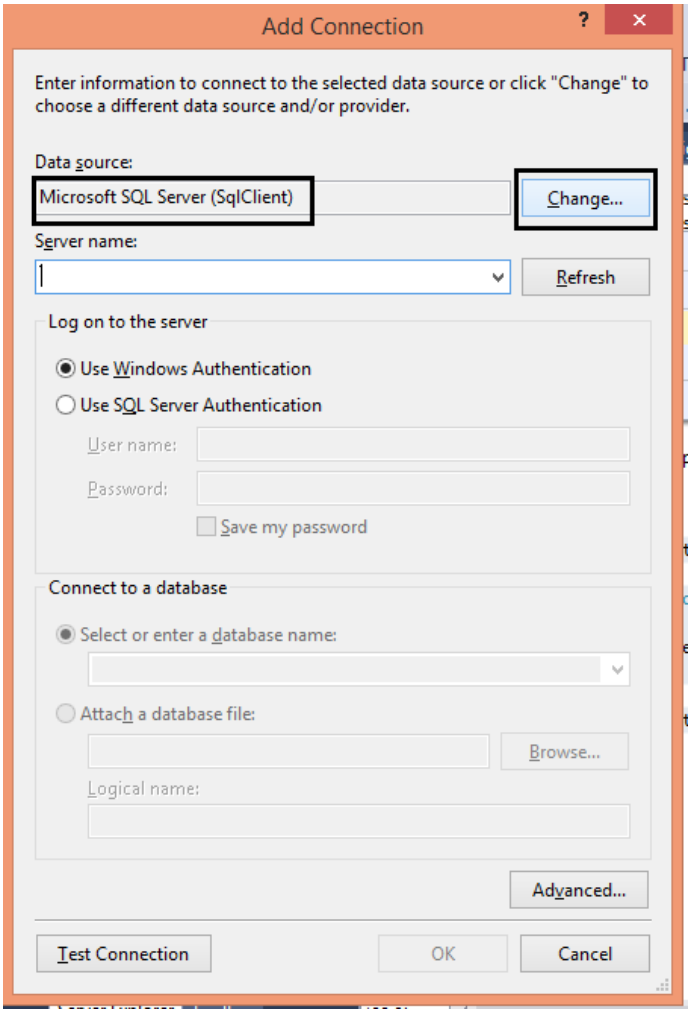
أما لوصل قاعدة البيانات بعد نسخها على جهاز آخر نقوم بمايلي:

أولاً : في حال كانت قاعدة البيانات مبنية باستخدام SqlExpress نقوم بوصلها كما يلي:

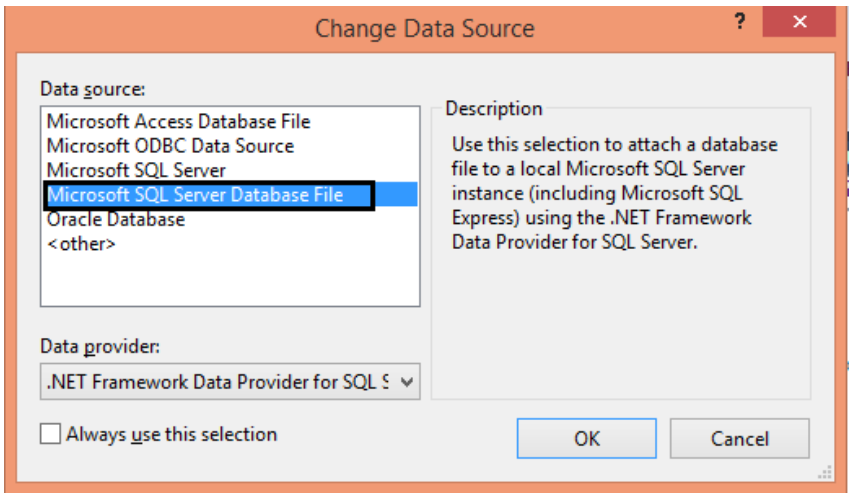
1- من واجهة الVisual Studio ومن قائمة Server Explorer ننقر باليمين على Data Connections ونختار AddConnection



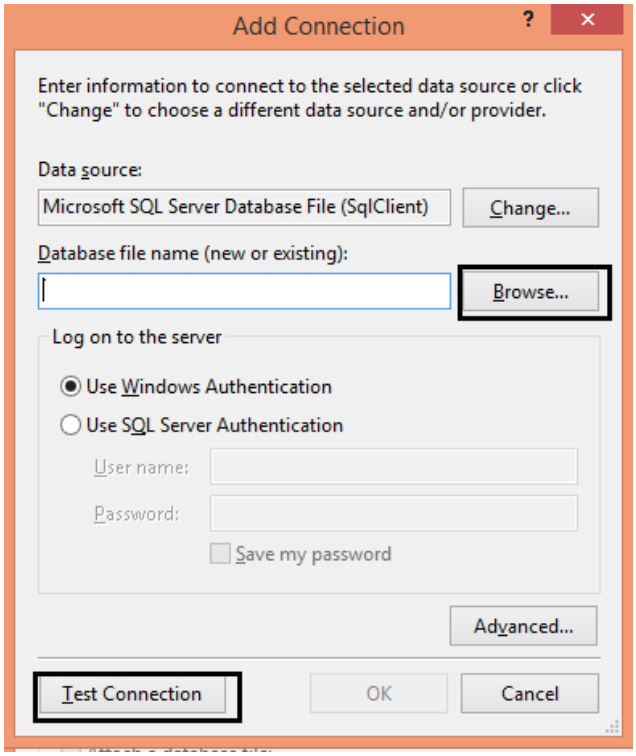
وتظهر لدينا الواجهة التالية:



2- نضغط على الزر Change ثم نختار خيار Microsoft Sql Server Database File

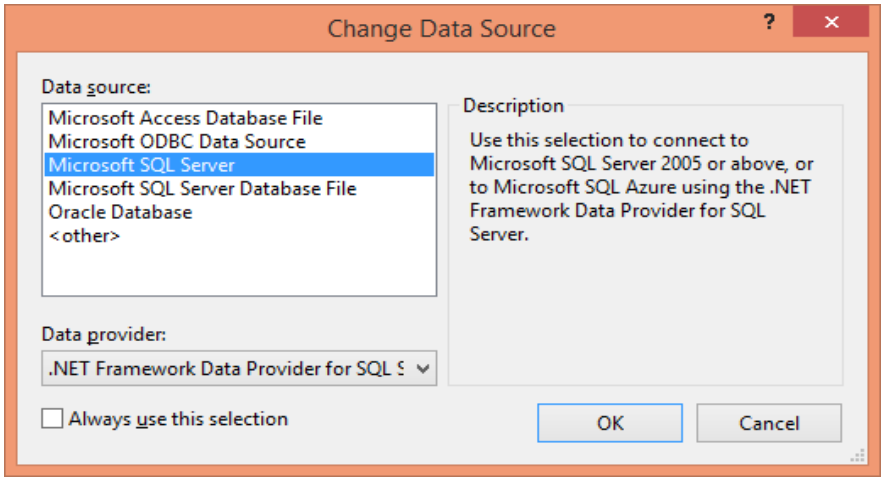


ثم نقوم بتحديد مسار قاعدة البيانات من خلال الزر Browse ثم نضغط TestConnection

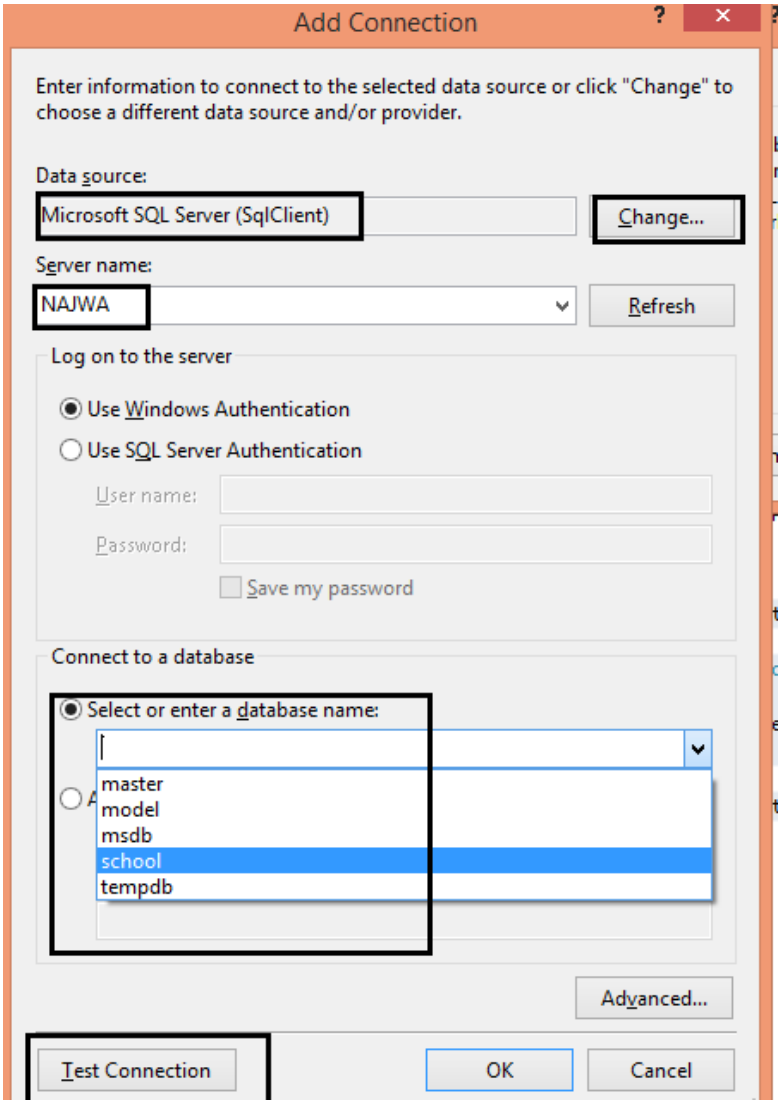


ثانياً: في حال كانت البيانات مبنية باستخدام الـ Sql Server ونريد ربطها مع برنامج الـ Sql Server ثم ربطها مع الـ Visual Studio:

1- نقوم توصيل قاعدة البيانات في برنامج الSql Server باستخدام خيار Attach Database.
2- من واجهة الـ Visual Studio ، من لائحة الـ Server Explorer ننقر باليمين على Data Connections ونختار AddConnection
3- نختار change من الواجهة التي تظهر ونختار Microsoft SQL Server :



4- نقوم بإدخال اسم السيرفر في الواجهة التالية، مع اسم المستخدم وكلمة السر (في حال الحاجة اليهما)
5- ثم نقوم بتحديد قاعدة البيانات التي سنتعامل معها من القائمة المنسدلة وبعدها نضغط Test Connection للتأكد من صحة الاتصال.



تمرين:

المطلوب إنشاء قاعدة البيانات school.

إنشاء تطبيق بواجهة واحدة لإضافة بيانات مادة.

نهاية الجلسة

البرمجة بطريقة الطبقات

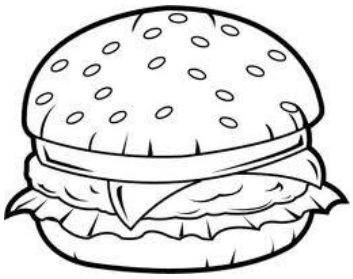
محاور الجلسة الثانية:

- مفهوم البرمجة بطريقة الطبقات.
- مزايا وعيوب التصميم متعدد الطبقات.
- بناء طبقة قاعدة البيانات.
- بناء طبقة المكتبة.
- بناء طبقة الواجهات.

مفهوم البرمجة بطريقة الطبقات

هي عبارة عن أسلوب لتصميم معماري Architecture Design يهدف الى تقسيم التطبيق الى طبقات مستقلة، تقوم كل طبقة بأداء مهمة ووظيفة ((محددة)) و ((واضحة)). مع التنويه بأن هذه الطبقات يمكن ان تعمل في نفس منصة العمل Platformاو منصات عمل مختلفة.

لتوضيح فكرة الطبقات سنستعين بالمثال التالي:



في الصورة المغرية السابقة، تبدأ الطبقة العلوية Upper Tier بقطعة من الخبز التي تمثل الواجهة العلوية والمغلفة للساندويتش وهي نفس الطبقة السفلية، ومن ثم طبقة تحتوي على تشكيلة متنوعة من الخضرواتVegetable Tier ، وبعدها الطبقة الرئيسية (طبقة شريحة اللحم(Meat Tier)

كيف يمكن أن تفيدنا طبقات البرغر

إن عملية بناء البرغر بهذا الشكل لها فوائد عديدة، أول فائدة أراها من منظور التخصص Specialization لزيادة الجودة Quality، فهناك خباز له خبرة عشرات السنين في التعامل مع الدقيق لصناعة الخبز، ويمكننا ان نعتمد عليه في بناء خبز وجبتنا، ومن ناحية الخضروات، فقد نعتمد على أحد المراكز التي تعرض أنقى وأفضل أنواع الخضراوات الطازجة، كذلك الحال مع طبقة اللحم، يمكننا أن نتعامل مع مطبخ يأتي يوميا بلحوم طازجة ذبحت للتو، وبذلك تزيد الجودة Quality للمنتج بشكل خرافي بسبب تخصص كل شخص في بناء الطبقة المناسبة، ويبقى علينا تجميع الطبقات ومن ثم بيعها للمستخدم النهائي (أقصد الزبون).

من ناحية التنقيح Debugging او التعديل، فعندما نريد (مثلا) تطوير نسخة أخرى من الهامبرجر (نسخة خفيفة Light Edition)لأشخاص يتبعون حمية غذائية خاصة، فكل ما سنفعله تغيير طبقة الخبز Bread Tier لتعتمد على خبز بر (خبز أسمر Brown Bread) ، دون تحمل أي تكلفة لتعديل الطبقات الأخرى .

وعندما نفكر في التدويل Internationalization وأخذ الإعدادات الإقليمية Localizations بعين الاعتبار، فالمسألة لن تتعدى طبقة اللحمMeat Tier ، فاليهود لا يأكلون لحوم الخنزير أو زيوت من مشتقاته، ويزيدهم على ذلك المسلمون أيضا، فلابد أن تكون طريقة ذبح اللحم حلالHalal ، وعند الهندوس فلا يمكن أن يكون لحم بقري (فهو محرم أكلها بعقيدتهم)، كل ما نحتاجه تعديل في طبقة اللحم فقط، وتبقى عقود الموردين الذين نتعامل معهم للطبقات الأخرى كما هي دون مشاكل.

ميزة أخرى قد نحتاجها بعد فترة من الإنتاج، وهي عند ظهور شوائب Bugs او أخطاء في هذا المنتج، فيمكن إصلاح المشكلة في الطبقة التي ينتمي لها ((فقط)) دون ان تتأثر ودون أن نحتاج الى تعديل الطبقات الاخرى.

أما في حال كان إعداد البرغر بطريقة مختلطة (مثل السلطة) فهنا أي تعديل او تطوير او صيانة ستستهلك جهد ووقت اضافي (إن لم تكن مستحيلة)، بل سنحتاج الى اعادة البناء من جديد عند التعديلات الجوهرية، وكل المزايا التي عرضناها بالفقرات السابقة واختصار الوقت وتوفير الاموال (خاصة التي مع تعاقدات الموردين Vendors لا يمكن تطبيقها هنا -أي كل تغيير بسيط بحاجة الى تعديل جديد للمنتج.

مزايا/عيوب التصميم متعدد الطبقات

مثل هذه الفقرة يصعب جدا التعميم معها، ولكن سأحاول عرض ابرز المزايا والعيوب بشكل مبسط ومختصر:

المزايا:

- تجعل البرنامج اكثر تنظيما وتخفض تكلفة تعقيدات البرامج Software Complexity مما تسهل عملية بنائها وتقل مخاطر Risks فشل المشروع.
- يمكنك تعديل أي طبقة وتصحيح مشاكلها او تطويرها دون تغيير او تعديل الطبقات الاخرى.
- يمكنك اضافة طبقات جديدة دون مجهود.
- التطبيقات المعدة بهذا التصميم دائما ما تكون أكثر أمانا (فالثغرات الأمنية بها قليلة جدا).
- إعادة الاستخدامCode Reusability ، يمكنك تصميم طبقاتك بطرق احترافية حتى تتمكن من إعادة استخدامها في مشاريع اخرى.
- مناسبة جدا للعمل الجماعي وفرق التطوير، حيث توضح مهام كل عضو بدقة ووضوح.

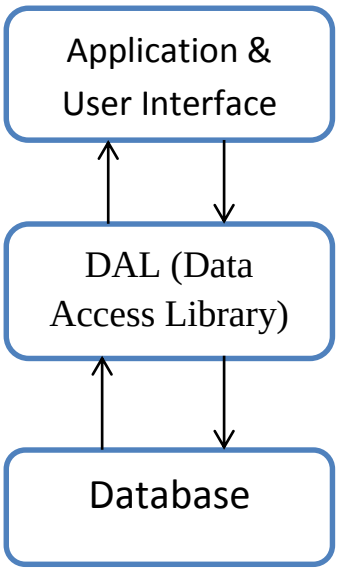
العيوب:

- تتطلب كتابة الكثير من الاكواد.
- التوثيق Documentation يصبح أمرا إلزاميا خاصة إن كبر حجم وكثر عدد الطبقات.
- بحاجة الى مجهود لضمان نجاح الاتصالات بين الطبقات المختلفة.
- التكرار في كتابة بعض الاكواد خاصة اكواد القوانين (Rules) في اكثر من طبقة.

مفهوم الطبقات بلغة السي شارب:

يمكن تقسيم أي تطبيق تتم كتابته بلغة ال#C إلى طبقتين أو أكثر، كل طبقة مسؤولة عن مهمة معينة.

والشكل التالي يوضح الطبقات الأساسية التي يجب أن يتكون منها أي تطبيق يتعامل مع قواعد البيانات:



وسيتم توضيح طريقة بناء التطبيقات من هذا النوع من خلال المثال التالي:

مثال توضيحي:

1- بناء طبقة قاعدة البيانات:

قمنا في الجلسة السابقة بإنشاء قاعدة البيانات باسم School إضافة الجداول الثلاثة لها و هي :

Students, Subjects, Student_Subjects

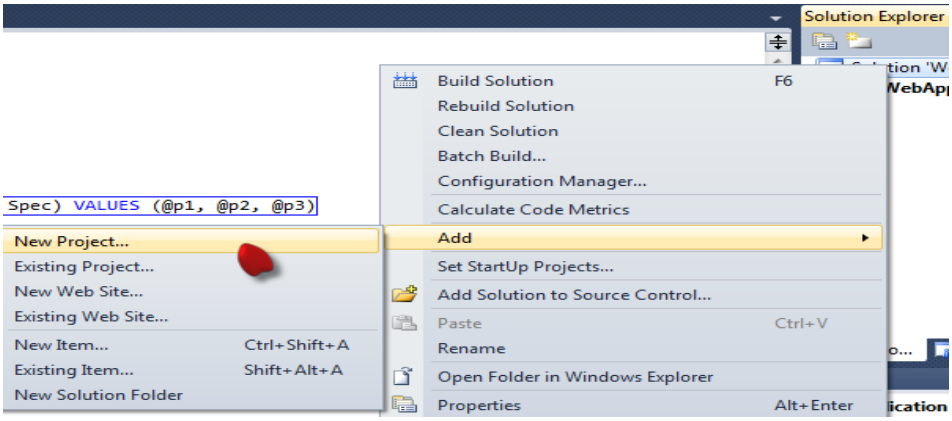
وإضافة التوابع المناسبة ضمن قاعدة البيانات Stored Procedure لعمليات الإضافة و التعديل و الحذف و الاسترجاع.

أي أن طبقة قاعدة البيانات باتت جاهزة.

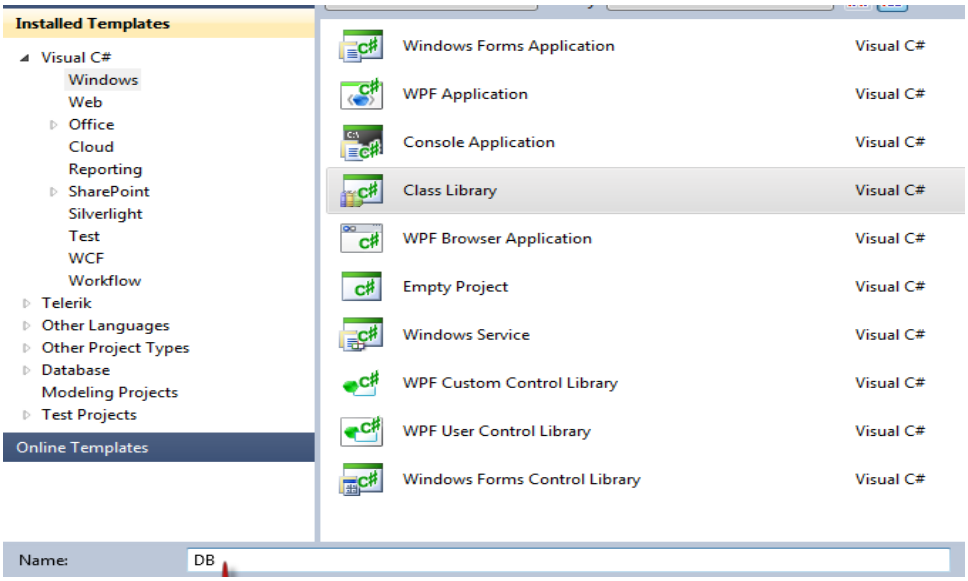
2- بناء طبقة المكتبة:

وهي طبقة تحتوي على مجموعة من الصفوف classes حيث أنه لكل جدول في قاعدة البيانات يجب بناء class مقابل له في طبقة المكتبة، بالإضافة إلى class خاص بالاتصال بقاعدة البيانات يحتوي على كافة الكود اللازم للاتصال.

لإضافة طبقة المكتبة على الشكل التالي:

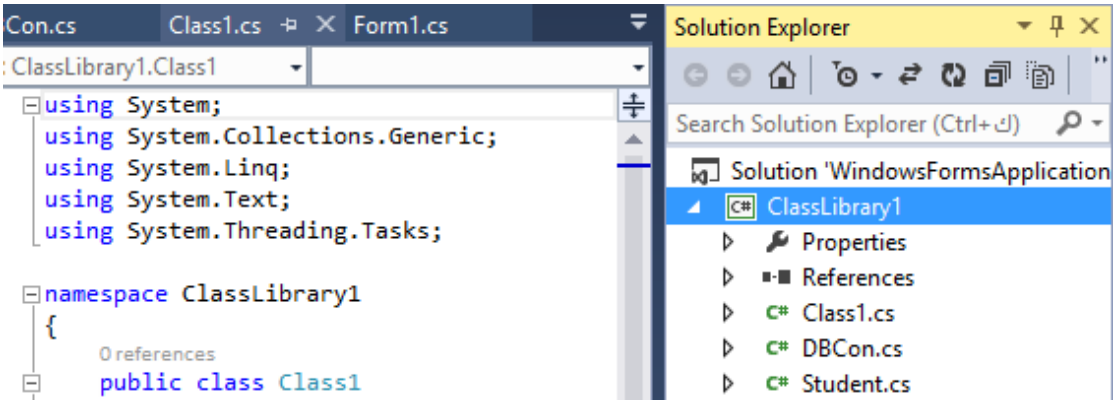


- نضيف مشروع جديد من نوع مكتبة (class Library)



- نقوم بتسميتها مثلاً: classLibrary1

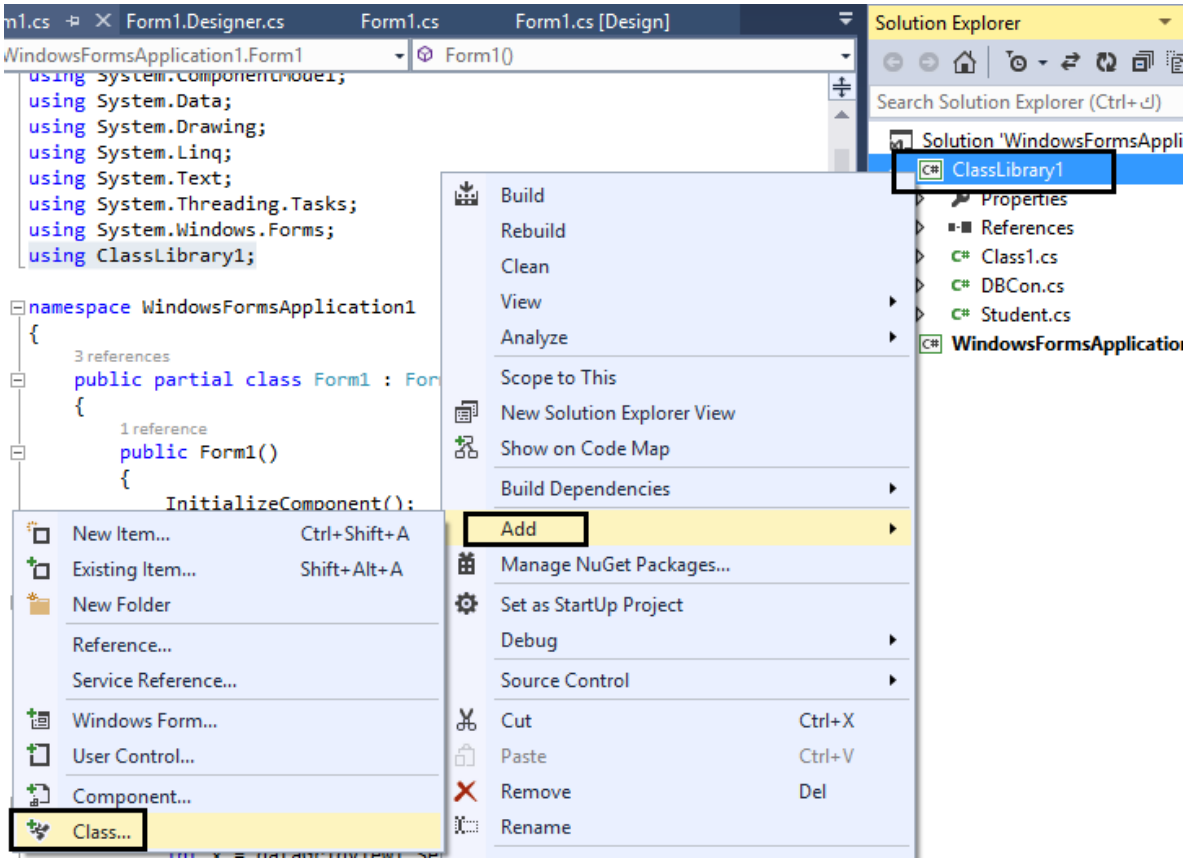
فتظهر ضمن Solution Explorer على الشكل التالي :



نلاحظ أن هذا المشروع يحتوي بشكل افتراضي على class واحد يدعى (Class1) ، ويمكننا أن نضيف ضمن المكتبة classesالتي يحتاجها مشروعنا.

ملاحظة هامة: يجب أن تكون جميع ال classes التي سنقوم بإضافتها هنا من نوع Public

طريقة إضافة class للمكتبة على الشكل التالي عند ضغط على المكتبة برز الماوس الأيمن :



ثم نقوم بتسميته الاسم الذي نريده DBcon ثم نضغط على الزر Add

ملاحظة هامة: يجب أن تكون جميع ال classes التي سنقوم بإضافتها هنا من نوع Public

نضيف class رئيسي نكتب ضمنه التعليمات المتكررة والخاصة بالاتصال بقاعدة البيانات وتنفيذ الأوامر، وبهذه الطريقة نكون قد عزلنا جميع أوامر الاتصال ب class واحد نستطيع تغييره متى أردنا. نسميه مثلا DBcon

DBCon.cs* | Class1.cs | Form1.cs | Student.cs* | Form1.Designer.cs

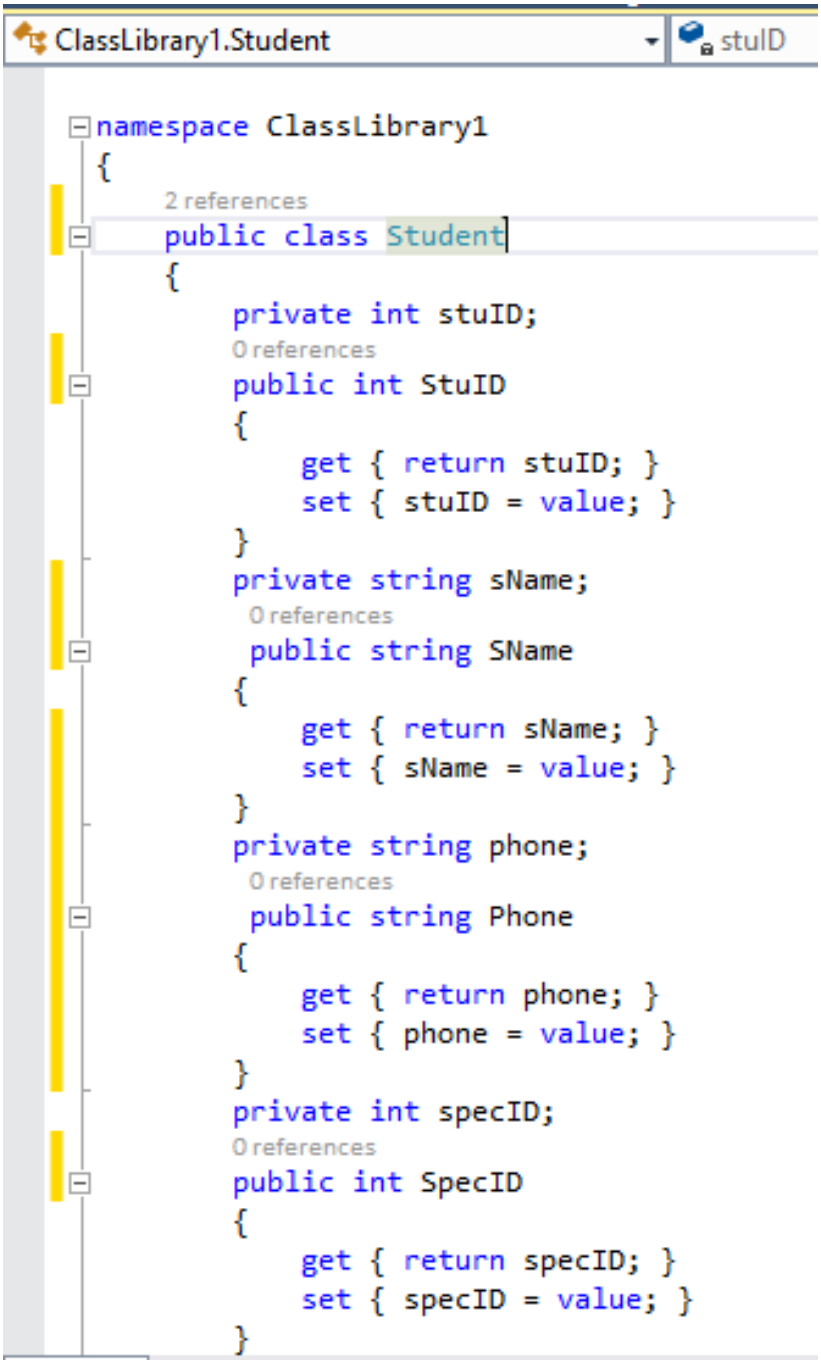
ClassLibrary1.DBCon | DBCon()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace ClassLibrary1
{
    2 references
    public class DBCon
    {
        protected SqlConnection cn;
        protected SqlCommand cmd;
        protected SqlDataReader rdr;
        string constring = @"Data Source=NAJWA;Initial Catalog=school;Integra

        0 references
        public DBCon()
        {
            cn = new SqlConnection(constring);
            cmd = new SqlCommand();
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Connection = cn;
        }
    }
}
```

مقابل كل جدول في قاعدة البيانات نقوم بإنشاء Class بنفس الاسم ضمن طبقة المكتبة، في مثالنا نقوم بإضافة ثلاثة classes حسب قاعدة البيانات الموجودة لدينا و هي :

Student, Subject, and Student_Subject

ثم نقوم بكتابة المتحولات و الخصائص لكل class على الشكل التالي:



هذه ال classes يجب أن تقوم بعملية الوراثة من صف الاتصال الذي قمن بإنشائه DBConعلى الشكل التالي:

```
public class Student:DBCon
{
```

ثم نقوم بإضافة التوابع الخاصة لكل class التي تمثل جدول في قاعدة البيانات و كل تابع يمثل عملية على البيانات الموجودة كالإضافة و التعديل و الحذف.

فمثلاً نقوم بإضافة أربع توابع أساسية ضمن class (Student) وهي :

AddStudent() , UpdateStudent() , DeleteStudent(), and SelectAll()

كما يلي:

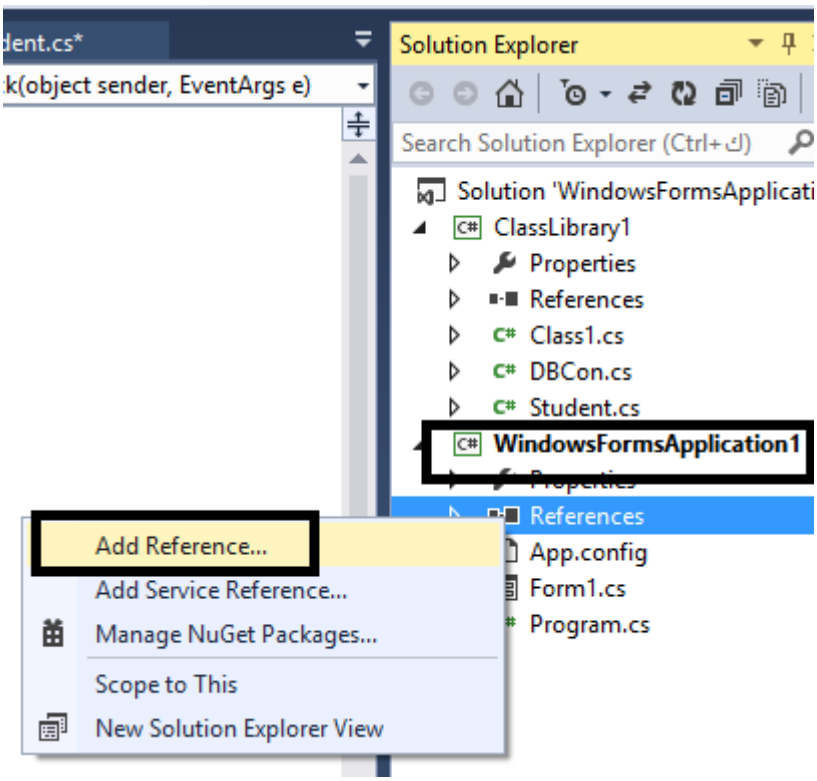
```
public string AddStudent()
{
    cmd.Parameters.Clear();
    cmd.CommandText = "AddStudent";
    cmd.Parameters.AddWithValue("@stuID",stuID);
    cmd.Parameters.AddWithValue("@sName", sName);
    cmd.Parameters.AddWithValue("@phone", phone);
    cmd.Parameters.AddWithValue("@spec", spec);

    try
    {
        cn.Open();
        cmd.ExecuteNonQuery();
        cn.Close();
        return "تمت العملية بنجاح";


    }
    catch (Exception e1)
    {

        if (cn.State == ConnectionState.Open)
            cn.Close();
        return e1.Message;
    }

}
```

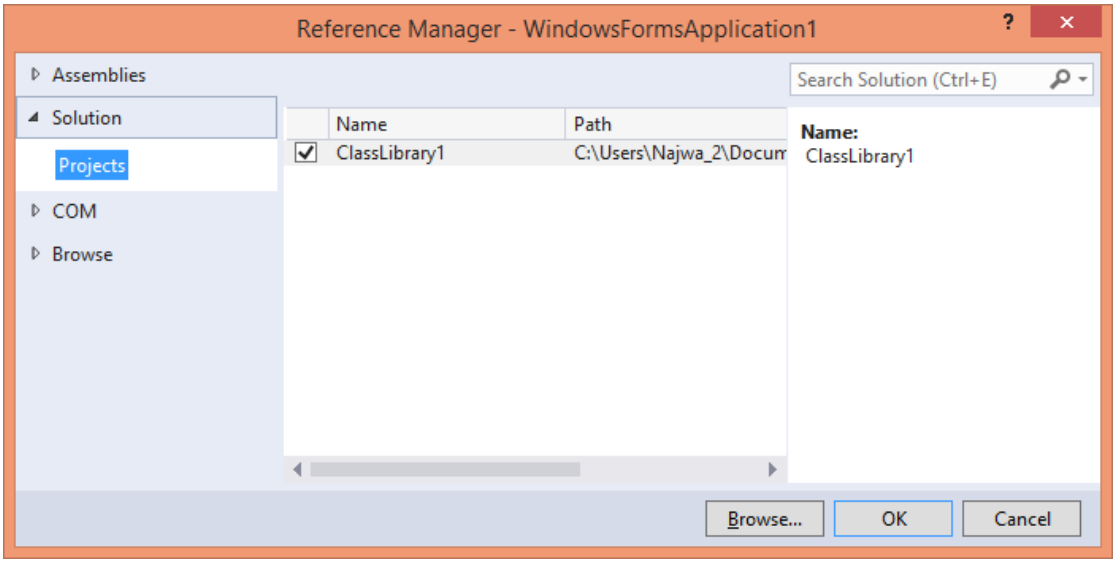
نلاحظ أن الكود ضمن تابع اضافة تابع أصبح مختصر جدا و يحتوي فقط على المعلومات الخاصة باضافة الطالب حصرا

3- إنشاء طبقة الواجهات:

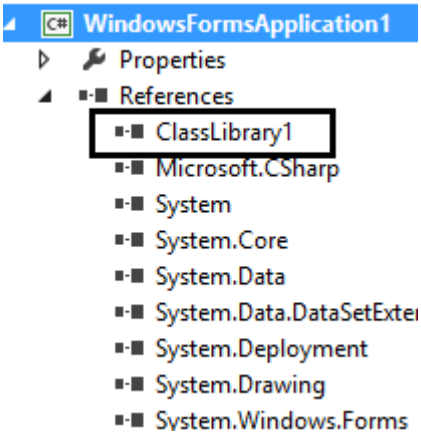
لإضافة الطبقة النهائية نقوم باضافة مشروع من نوع Windows Application و نقوم بتصميم ال Form على شكل واجهة لاضافة معلومات الطالب على الشكل التالي:

ثم نقوم باضافة المكتبة classLibrary1 لطبقة الـــ User Interface نضغط بزر الماوس الأيمن على مجلد referencesكما في الشكل التالي:

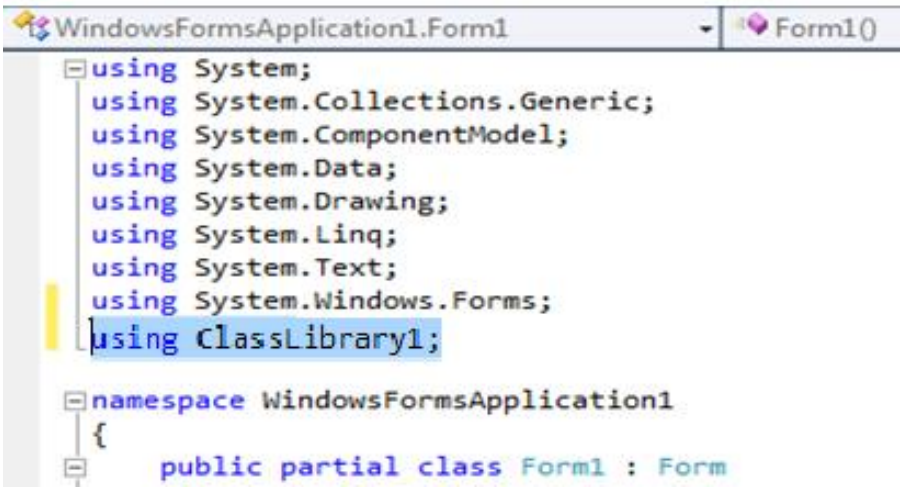




نلاحظ أن المكتبة ظهرت ضمن المشروع داخل مجلد references



وكي نتمكن من استدعاء التوابع المكتوبة ضمن هذه المكتبة يجب علينا استدعاء هذه المكتبة باستخدام تعليمة using في بداية الملف على الشكل التالي:

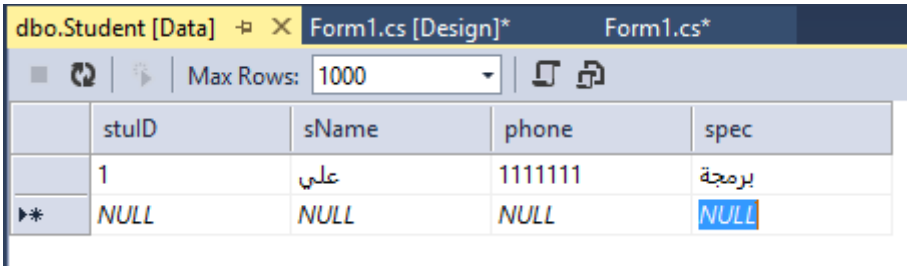


ثم نقوم باستدعاء تابع اضافة طالب لقاعدة البيانات ضمن حدث الزر الخاص بإضافة طالب على الشكل التالي:

```
private void AddBtn_Click(object sender, EventArgs e)
{
    Student s1 = new Student();
    s1.stuID = Convert.ToInt32(IDTxt.Text);
    s1.sname = NameTxt.Text;
   s1.phone = phoneTxt.Text;
   s1.spec = specTxt.Text;
    resultLbl.Text =s1.AddStudent();
}
```

عند تنفيذ المشروع تظهر لنا نافذة الإضافة نقوم بادخال الاسم و الكنية و الاختصاص للطالب:

نلاحظ أن عملية الإضافة تمت بنجاح ضمن قاعدة البيانات :



تمرين وظيفة:

القيام بإنشاء تطبيق بطريقة الطبقات وكتابة الكود اللازم لإضافة مادة .

انتهت الجلسة

البرمجة بطريقة الطبقات2

محاور الجلسة الثالثة:

- أنواع الأوامر المستخدمة للتعامل مع قاعدة البيانات
- استعلامات عرض البيانات.
- تعديل البيانات المخزنة في قاعدة البيانات.
- حذف البيانات المخزنة في قاعدة البيانات.
- الاستعلامات التي تعيد قيمة واحدة فقط.

مقدمة:

في المحاضرة السابقة تعلمنا كيفية إنشاء تطبيق متعدد الطبقات، كمثال تطبيقي تعلمنا كيفية إتمام عملية الإضافة إلى قاعدة البيانات بهذه الطريقة.

في هذه المحاضرة سنتعلم كيفية الاستعلام عن البيانات والتعديل عليها وحذف البيانات وبهذا نكون قد تعلمنا الخطوات الأساسية اللازمة لبناء تطبيق متعدد الطبقات قادر على التعامل مع قاعدة البيانات.

أنواع الأوامر المستخدمة للتعامل مع قاعدة البيانات:

لدينا ثلاثة أنواع للأوامر التي يتم تنفيذها على قاعدة البيانات وهي:

1- الأوامر التي لاتحتوي على استعلام، أي أن تنفيذ الأمر لن يعيد أي قيمة (مثل عملية إضافة أو تعديل بيانات الطالب).
ولتنفيذ هذا الأمر تستخدم تعليمة EcexuteNonQuery.

2- الاستعلامات، والتي تعيد مجموعة من البيانات وفقاً للاستعلام الذي نقوم بكتابته (مثل الاستعلام عن أسماء جميع الطلاب) ولتنفيذ هذا الاستعلام نحتاج لكائن SqlDataReader لقراءة البيانات على شكل جدول.

3- الاستعلامات التي تعيد قيمة واحدة فقط (مثل الاستعلام عن رقم طالب محدد). ولتنفيذ هذا الاستعلام تستخدم تعليمة ExecuteScalar.

رأينا في الجلسة السابقة كيفية التعامل مع النوع الاول من الأوامر. وفي هذه الجلسة سنوضح كيفية التعامل مع الاستعلامات التي تعيد قيمة واحدة أو لائحة من البيانات.

استعلامات عرض البيانات:

لنفترض أننا نريد كتابة استعلام لعرض لائحة بأسماء الطلاب، في برنامج الطبقات الذي بنيناه في الجلسة السابقة.

1- كتابة الإجرائية المخزنة:

في طبقة قاعدة البيانات نقوم بكتابة إجرائية مخزنة لعرض بيانات جميع الطلاب كالتالي:

```
create PROCEDURE ViewAllStudent

AS
    select * from Student
    RETURN
```

2- كتابة إجرائية عرض البيانات في طبقة المكتبة:

بما أن الإجرائية هي لعرض بيانات جميع الطلاب Student فإن التعديل في طبقة المكتبة يكون على Class Student ، وذلك بإضافة إجرائية جديدة تعيد DataTable:

```
public DataTable ViewAllStudent()
{
    //1- بناء الجدول الذي ستخزن فيه بيانات الاستعلام
    DataTable dt = new DataTable();
    cmd.CommandText = "ViewAllStudent";
    cmd.Parameters.Clear();
    try
    {
        cn.Open();
        //2-بناء كائن مخصص لقراءة البيانات من القاعدة
        SqlDataReader myRdr = myRdr = cmd.ExecuteReader();
        // وضع البيانات في الجدول المخصص
        dt.Load(myRdr);
        myRdr.Close();
        cn.Close();
        return dt;
    }
    catch (Exception e1)
    {
        if (cn.State == ConnectionState.Open)
            cn.Close();
        dt.Columns.Add(e1.Message);
        return dt;

    }
```

حيث أنه في الكود السابق قمنا بما يلي

- إنشاء كائن DataTable مهمته التعامل مع البيانات على شكل جدول.
- تعريف كائن لقراءة البيانات:

```
SqlDataReader myRdr = cmd.ExecuteReader();
```

حيث أن هذه التعليمة هي المسؤولة عن إنشاء قارئ البيانات وتنفيذ الإجرائية المخزنة.

- نقل البيانات من الDataReader إلى ال DataTable وذلك من خلال التعليمة:

```
dt.Load(myRdr);
```

- وفي النهاية نعرض الDataTable في جدول البيانات ومن ثم نقوم بإغلاق قارئ البيانات وإغلاق الاتصال.
- في حال حدوث خطأ أثناء تنفيذ الأمر فإن ال DataTable ستحوي عمود واحد هو رسالة الخطأ.

3- عرض البيانات في طبقة الواجهة:

سيتم عرض البيانات باستخدام كائن DataGridView وذلك عند النقر على زر العرض ما يلي:

نقوم بإضافة زر لعرض البيانات وليكن اسمه ViewData كما نقوم بإضافة عنصر DataGridView لعرض البيانات ونحدد اسمه بـ dgv1، وفي حدث النقر على الزر نكتب الكود التالي:

```
Student s1 = new Student();
```

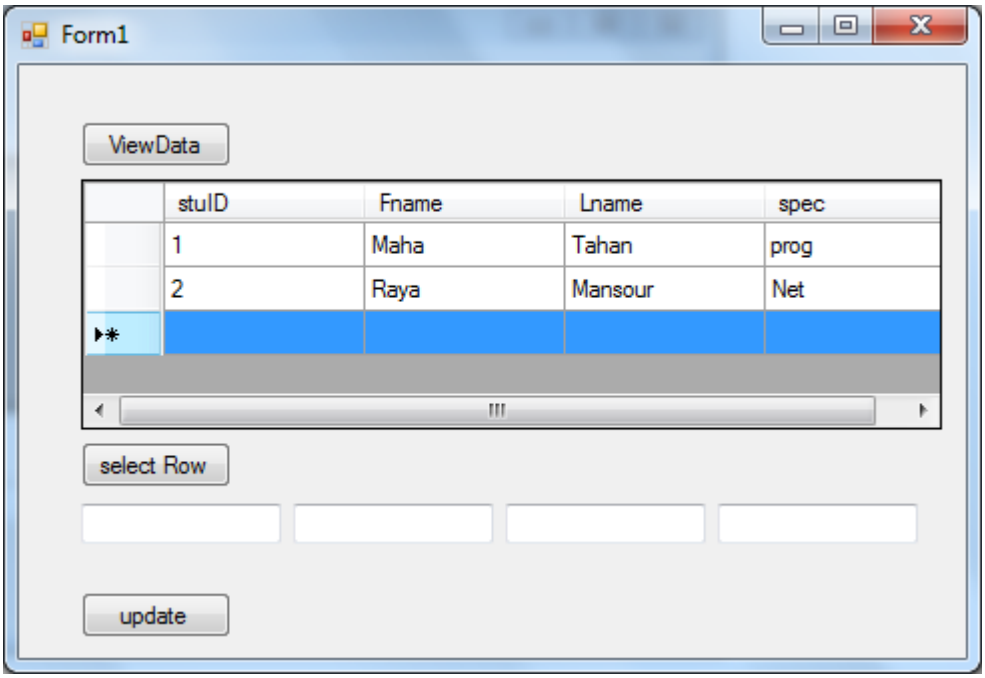
```
dgv1.DataSource = s1.ViewAllStudent();
```

وهو استدعاء لإجرائية العرض الذي كتبناه في طبقة المكتبة.

ملاحظة:

يفضل التعديل في خصائص الDataGridView حيث يصبح الـ Selection Mode: FullRowSelect.

وذلك ليتم تحديد السطر بكامله عند النقر على أي خلية في هذا السطر.



تعديل البيانات المخزنة في قاعدة البيانات:

لنفترض أننا نريد تعديل بيانات أحد الطلاب، طبعا للقيام بهذه العملية نحتاج للتعديل على الطبقات الثلاث:

1- في طبقة قاعدة البيانات

كتابة إجرائية مخزنة لتعديل بيانات طالب معين:

```
CREATE PROCEDURE updateStudent
    @stuID int,
    @sName nvarchar(50),
    @phone nvarchar(50),
    @spec nvarchar(50)
AS
    update student set sName = @sName, phone = @phone,spec = @spec
    where stuID = @stuID
RETURN 0
```

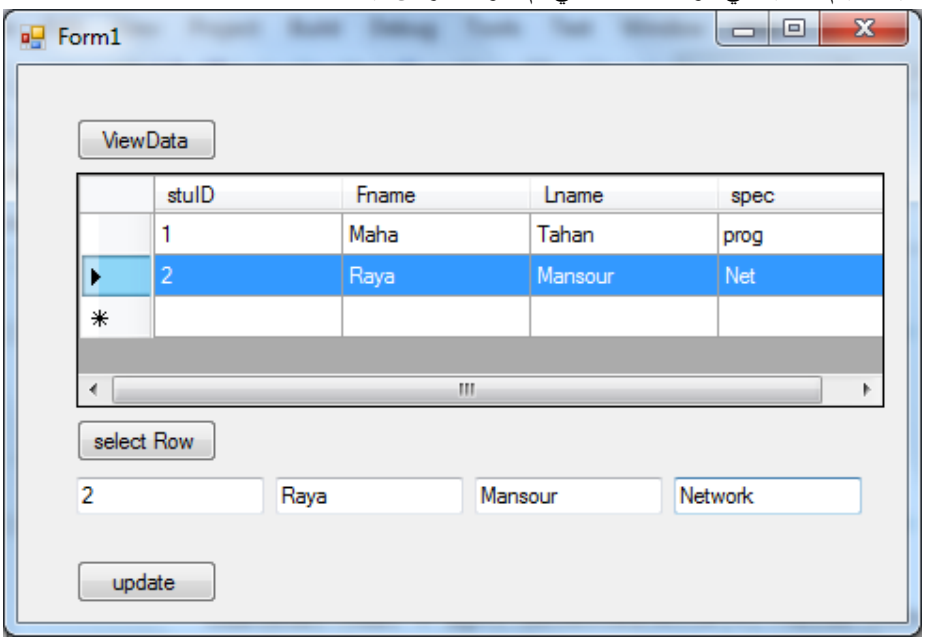
2- في طبقة المكتبة:

نقوم بكتابة إجرائية لتعديل بيانات الطالب حيث أن الإجرائية تشابه عملية إضافة الطالب إلى حد كبير وذلك في class Student طبعا كما يلي:

```
public string updateStudent()
{
    cmd.CommandText = "updateStudent";
    cmd.Parameters.Clear();
    //إضافة البارامترات
    cmd.Parameters.AddWithValue("@sID",StuID);
    cmd.Parameters.AddWithValue("@Fname", Fname);
    cmd.Parameters.AddWithValue("@phone", Lname);
    cmd.Parameters.AddWithValue("@spec", Spec);

    try
    {
        cn.Open();
        cmd.ExecuteNonQuery();
        cn.Close();
        return "succeed";
    }
    catch (Exception e1)
    {
        if (cn.State == ConnectionState.Open)
            cn.Close();
        return e1.Message;

    }
```

3- في طبقة الواجهات:

حيث سيتم التعديل في الواجهة ذاتها التي تم بناؤها لعرض بيانات الطلاب.



حيث يستطيع المستخدم تحديد سطر الطالب من الـ DataGridView الذي يريد تعديل بياناته وبالضغط على زر Select Row سيتم نقل بيانات الطالب إلى الـ TextBoxes الموجودة في الأسفل، وذلك باستخدام التعليمات التالية:

```
textBox1.Text = dgv1.SelectedCells[0].Value.ToString();
textBox2.Text = dgv1.SelectedCells[1].Value.ToString();
textBox3.Text = dgv1.SelectedCells[2].Value.ToString();
textBox4.Text = dgv1.SelectedCells[3].Value.ToString();
```

وبعد أن يقوم المستخدم بتعديل البيانات يضغط على زر update يتم إرسال التعديلات إلى قاعدة البيانات حيث أنه في حدث النقر على زر update يوجد الكود التالي:

```
Student s1 = new Student();
s1.StuID = Convert.ToInt32(textBox1.Text);
s1. Name = textBox2.Text;
s1. phone =textBox3.Text;
s1.Spec = textBox4.Text;

resultLbl.Text = s1.updateStudent();
//لتحديث لائحة البيانات المعروضة
dgv1.DataSource = s1.ViewAllStudent();
```

حذف البيانات المخزنة في قاعدة البيانات:

لنفترض أننا نريد حذف بيانات أحد الطلاب، طبعا للقيام بهذه العملية نحتاج للتعديل على الطبقات الثلاث:

1- في طبقة قاعدة البيانات

كتابة إجرائية مخزنة لحذف بيانات طالب معين:

```
create PROCEDURE DeleteStudent
@sID int

AS
    delete from student
    where stuID = @sID

    RETURN
```

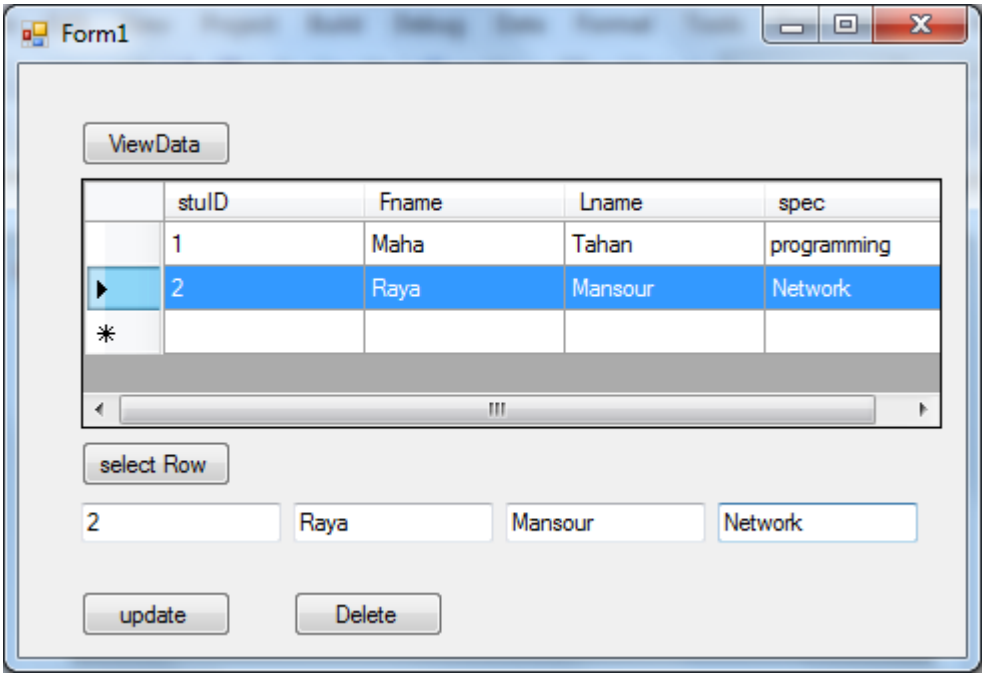
2- في طبقة المكتبة:

نقوم بكتابة إجرائية لحذف بيانات الطالب حيث أن الإجرائية تشابه عملية إضافة الطالب إلى حد كبير وذلك في class Student طبعا كما يلي:

```
public string DeleteStudent()
{
    cmd.CommandText = "deleteStudent";
    cmd.Parameters.Clear();
    //إضافة البارامترات
    cmd.Parameters.AddWithValue("@sID", StuID);
    try
    {
        cn.Open();
        cmd.ExecuteNonQuery();
        cn.Close();
        return "succeed";
    }
    catch (Exception e1)
    {
        if (cn.State == ConnectionState.Open)
            cn.Close();
        return e1.Message;

    }
}
```

3- في طبقة الواجهات:

حيث سيتم التعديل في الواجهة ذاتها التي تم بناؤها لعرض بيانات الطلاب وتعديلها.



حيث يستطيع المستخدم تحديد سطر الطالب من الـ DataGridView الذي يريد تعديل بياناته وبالضغط على زر Select Row سيتم نقل بيانات الطالب إلى الـ TextBoxes الموجودة في الأسفل، كما ورد سابقا

وهنا يستطيع المستخدم حذف البيانات يضغط على زر Delete حيث يتم إرسال الأمر إلى قاعدة البيانات
وفي حدث النقر على زر Delete يوجد الكود التالي:

```
Student s1 = new Student();
s1.StuID = Convert.ToInt32(textBox1.Text);
resultLbl.Text = s1.DeleteStudent();
//لتحديث لائحة البيانات المعروضة
dgv1.DataSource = s1.ViewAllStudent();
```

الاستعلامات التي تعيد قيمة واحدة فقط:

مثلا إذا أردنا الاستعلام عن اسم الطالب من خلال رقمه نقوم بما يلي:

1- في طبقة قاعدة البيانات
نكتب الإجرائية المخزنة التالية:

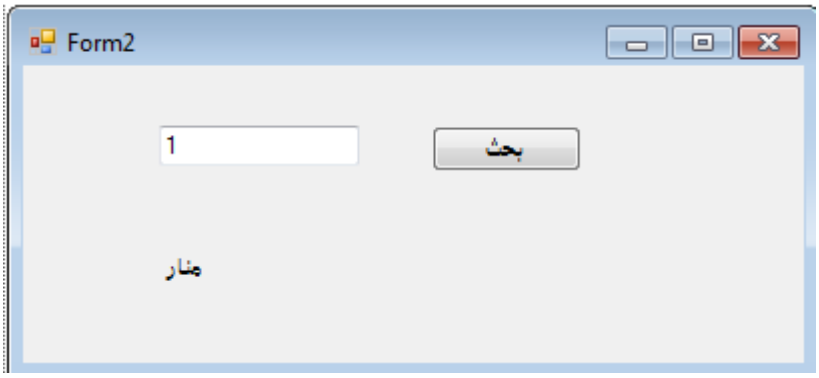
```
CREATE PROCEDURE getStudentName
    @sID int
AS
    select  Name from Student
    where stuID = @sID
    RETURN
```

2- في طبقة المكتبة:
نكتب الإجرائية التالية التي تعيد قيمة واحدة هي اسم الطالب، وفي حال حدوث خطأ يعيد رسالة الخطأ.

```
public string getStudentName()
{
    cmd.CommandText = "getStudentName";
    cmd.Parameters.Clear();
    //إضافة البارامترات
    cmd.Parameters.AddWithValue("@sID", StuID);
    string result;
    try
    {
        cn.Open();
        result =  cmd.ExecuteScalar().ToString();
        cn.Close();
        return result;
    }
    catch (Exception e1)
    {
        if (cn.State == ConnectionState.Open)
            cn.Close();
        return e1.Message;

}   }
```

3- في طبقة الواجهة:

نقوم ببناء الواجهة التالية :



حيث أنه في حدث الضغط على زر بحث نكتب الكود التالي:

```
Student s1 = new Student();
s1.StuID = Convert.ToInt32(textBox1.Text);
resultLbl.Text = s1.getStudentName();
```

تمرين وظيفة:

القيام ببناء تطبيق بطريقة الطبقات لقاعدة البيانات school، وكتابة عمليات عرض البيانات وتعديلها وحذفها وذلك بالنسبة لجدول subject.

نهاية الجلسة

برمجة الـ User Control

محاور الجلسة الرابعة:

- تعريف الـ User Control.
- مزايا الـ User Control.
- مثال1: بناء User Control لعرض الوقت.
- إضافة معالجات الأحداث للـ UserControl.
- مثال2: بناء User Control لعرض جزء من قاعدة البيانات.
- استخدام مفهوم الـ User Control لبناء واجهة كاملة.
- مثال3: بناء برنامج للتعامل مع ملفات الفيديو والصوت والصور.
- التحميل الديناميكي للـ User Control.

تعريف الـ User Control:

هو عبارة عن جزء من واجهة رسومية، يمكن برمجتها مرة واحدة والاستفادة منها في عدة أماكن. يمكن إنشائها داخل مشروع محدد Windows Forms Application، أو إنشائها داخل مشروع منفصل من نوع Windows Forms Control Library، وفي كلا الحالتين يتم إضافتها إلى صندوق الأدوات حيث تصبح جاهزة للاستخدام.

مزايا الـ User Control:

- توفير في استخدام الذاكرة.
- توفير الوقت والجهد.
- سهولة تصحيح الأخطاء.
- إمكانية إعادة استخدام الكود في أكثر من مكان Code Reusability.

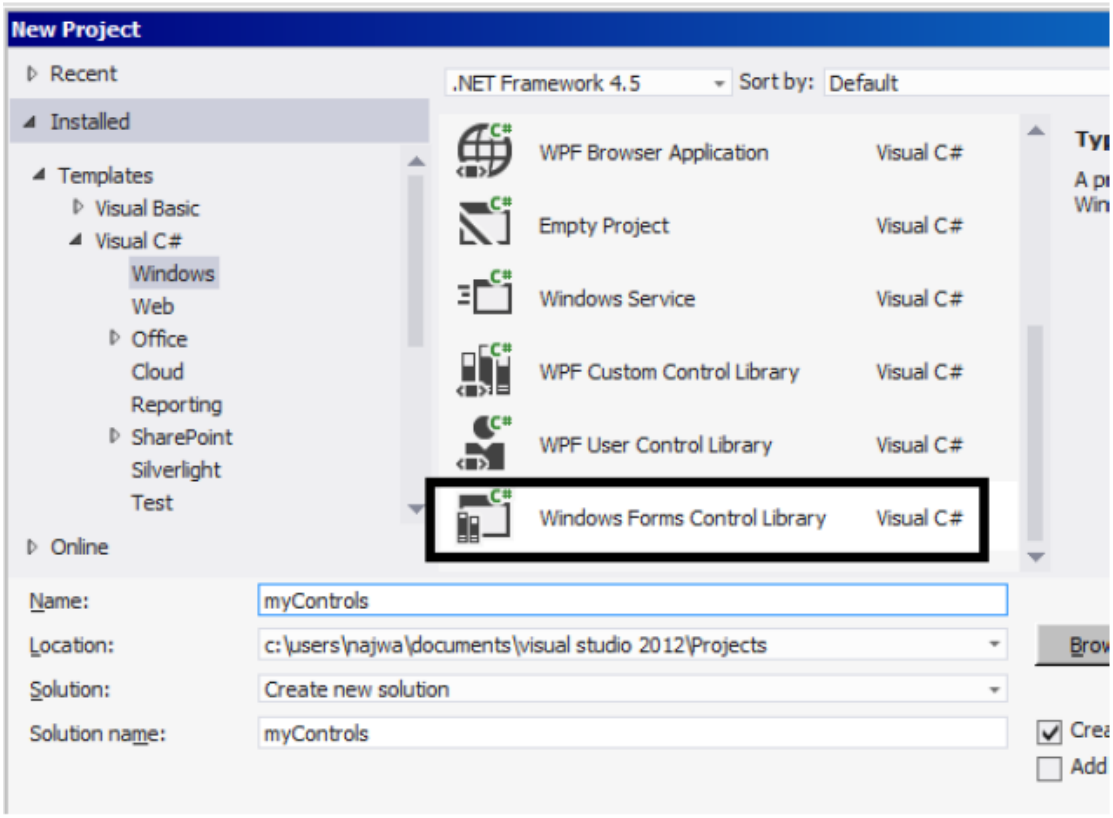
بناء واستخدام الـ User Control:

سنتعرف على طريقة بناء واستخدام الـ User Control من خلال الأمثلة:

مثال1: بناء User Control لعرض الوقت:

نريد بناء Control يعرض الوقت مثل الساعة الالكترونية بحيث نستطيع استخدامه في أكثر من مشروع:

1- نقوم بإنشاء مشروع جديد من نمط Windows Forms Control Library ولنفترض اسمه myControls.



2- في الـ Solution Explorer ننقر باليمين على المشروع ونختار Add → User Control ونقوم بتسميتها Clock.

3- تظهر لدينا لوحة رمادية اللون يمكن التعامل معها على أنها واجهة، نقوم بإضافة Label كما نضيف عنصر مؤقت Timer من قسم الـ Components في صندوق الأدوات.

4- نقوم ببرمجة الـ Label بحيث يعرض التوقيت الحالي، وباستخدام الـ Timer يتم تحديث الـ Label في كل ثانية:

ننقر نقرة مزدوجة على الـ Timer فيظهر لدينا حدث Timer-Tick نكتب الكود التالي داخل الحدث:

```
Label1.Text = DateTime.Now.ToLongTimeString();
```

في الباني الخاص بالـ Clock والذي يكون مكتوبا بشكل تلقائي نكتب:

```
timer1.Start();
```

وذلك لبدء عمل الـ Timer عند بدء تشغيل الـ Control.

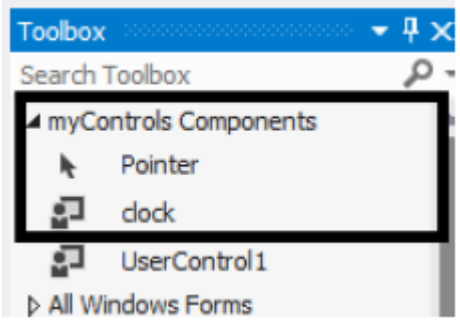
5- نقوم بعمل Build للمشروع.

6- نقوم بإضافة مشروع جديد، وذلك بالنقر باليمين على الـ Soloution:

Add -> new project -> windows forms application

وطبعا ننقر على المشروع الجديد باليمين ونختار set as startup project ليبدأ التنفيذ منه.

7- تظهر لدينا ضمن صندوق الأدوات قسم خاص بالأدوات التي قمنا بإنشائها وله نفس عنوان مكتبة الأدوات myControls.



8- هنا نستطيع استخدام الـ Clock مثل أي أداة أخرى من صندوق الأدوات.

إضافة معالجات الأحداث للـ UserControl:

عندما نقوم بإضافة UserControl إلى الـ Form فإن جميع العناصر التي بداخلها تكون مغلفة، ولا نستطيع الوصول إلى خصائص أو أحداث هذه العناصر. مثلا إذا كانت الـ UserControl تحتوي على عنصر ComboBox فلن نستطيع من خلال الـ Form الوصول إلى حدث SelectedIndexChanged الخاص بالـ CombBox، وهكذا لن نستطيع الاستفادة من هذا الحدث.

في هذه الحالة نستطيع فقط الوصول لأحداث الـ UserControl، وبالتالي الحل هو أن نقوم بصناعة أحداث للـUserControl تطابق الأحداث المطلوبة (مثلا حدث SelectedIndexChanged للـ ComboBox).

وسيتم توضيح الفكرة عمليا من خلال المثال التالي:

مثال2: بناء UserControl لعرض جزء من قاعدة بيانات:

لنفترض لدينا في مشروع بيانات طلاب المعهد جدول Department يحدد من خلاله القسم الذي ينتمي إليه الطالب، والقسم الذي تتبع له المادة، ونحتاج لوضع عنصر قائمة منسدلة ComboBox تحتوي على جميع أسماء الأقسام في المعهد.

برمجة مثل هذا العنصر في بضعة أسطر من الكود قد تستغرق مدة 10 دقائق، وهي ليست بالمهمة الصعبة، لكن مثل هذه القائمة نحتاج إليها في أكثر من واجهة ضمن نفس المشروع، مثل واجهة إضافة مادة وواجهة البحث عن طالب و ... بدل أن نقوم بتكرار برمجة هذا العنصر في كل واجهة، نستطيع برمجته في عنصر UserControl وإضافته للـ ToolBox وهكذا يصبح هذا العنصر جاهزا للاستخدام في أي وقت.

طريقة البناء:

يمكن الرجوع إلى محاضرة البرمجة بطريقة الطبقات لمعرفة طريقة البناء، هنا فقط سأكتفي بذكر الكود.

1- في جدول قاعدة البيانات يجب إنشاء الجدول Departments:

	DeptID	deptName
	1	برمجة
	2	شبكات
	3	صيانة
	4	تقنيات

2- نقوم بكتابة إجرائية مخزنة لجلب اسماءو أرقام جميع الأقسام:

```
CREATE PROCEDURE getAllDepts AS
       SELECT * from Departments
```

3- في طبقة المكتبة وبعد كتابة كود Department Class نكتب إجرائية GetAllDepts التي تعيد DataTable تحوي أرقام وأسماء جميع الأقسام:

```
public DataTable getAllDepts()
        {
            DataTable dt = new DataTable();
            cmd.CommandText = "getAllDepts";
           try
            {
                cn.Open();
                rdr = cmd.ExecuteReader();
                dt.Load(rdr);
                rdr.Close();
                cn.Close();
                return dt;
            }

            catch (Exception e1)
            {
                cn.Close();
                return dt;        }}
```

4- نقوم بإنشاء مشروع طبقة الواجهة من نوع Windows Forms Application، ونقوم بإضافة المكتبة كمرجع Reference.

5- ننقر على مشروع طبقة الواجهة باليمين User Control→ new item → Add ونسميها deptComboUC .

- تظهر لدينا لوحة رمادية اللون نضيف عليها: ComboBox
- في صفحة كود ال Control نكتب:

using ClassLibrary1;

- في حدث الـ Load للكونترول: DeptComboUc_Load نكتب الكود التالي:

```
Department d1 = new Department();
DataTable dt = d1.getAllDepts();
comboBox1.DataSource = dt;
comboBox1.DisplayMember = "DeptID";
comboBox1.ValueMember = "DeptName";
```

6- نقوم بتعريف المتحولات التالية لمعرفة القيمة المحددة في الComboBox :

```
public string SelectedText
{
    get { return comboBox1.Text; }

}
0 references
public string SelectedValue
{
    get { return comboBox1.SelectedValue.ToString(); }

}
```

حيث يعيد المتحول الأول القيمة النصية المحددة، بينما يعيد المتحول الثاني القيمة الضمنية للعنصر.

7- نقوم بتعريف الحدث SelectedIndexChanged والذي سيقابل حدث comboBox1_SelectedIndexChanged ولكن هذا الحدث سيكون تابعا للـ UserControl وليس للـcomboBox الذي بداخلها:

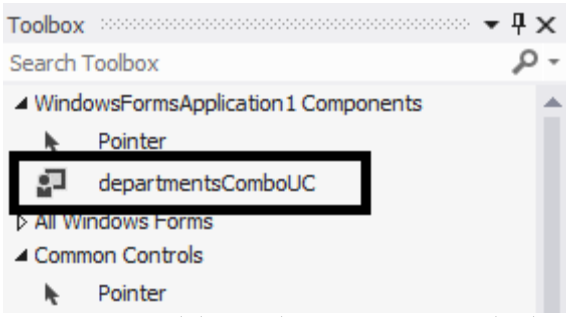
```
public event EventHandler SelectedIdexChanged;
```

8- نقوم بتفعيل حدث comboBox1_SelectedIndexChanged عند تحميل الـ UserControl وذلك من خلال كتابة السطر التالي داخل حدث الـ Load للـ UserControl:

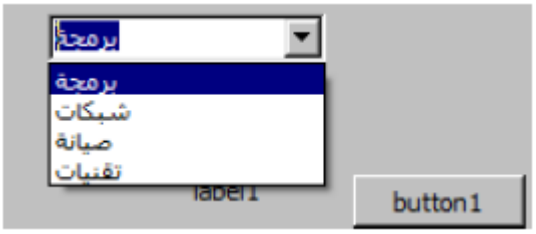
```
private void deptComboUC_Load(object sender, EventArgs e)
{
    this.comboBox1.SelectedIndexChanged += selectedIndexChanged;

}
```

وهكذا نكون قد انتهينا من برمجة الحدث داخل الـ UserControl.

9- بعد عمل Build للمشروع تظهر هذه الكونترول كأداة في صندوق الأدوات، لكن في هذه الحالة تكون خاصة بهذا المشروع فقط، ويمكن استخدامها ضمن أي واجهة من واجهات المشروع أو حتى ضمن UserControl أخرى.



10- للتجربة قم بسحب DeptComboUC ووضعها على الـ Form1 وستظهر النتيجة كالتالي:



مثال3: بناء برنامج للتعامل مع ملفات الفيديو والصوت والصور:

في هذا المثال سنقوم ببناء تطبيق يحتوي على عنصرين من نوع UserControl الأول لعرض ملفات الصور والآخر لتشغيل ملفات الصوت والفيديو:

نقوم بإنشاء مشروع من نوع Windows Forms Application ونضيف عليه عنصرين من نوع UserControl كما يلي:

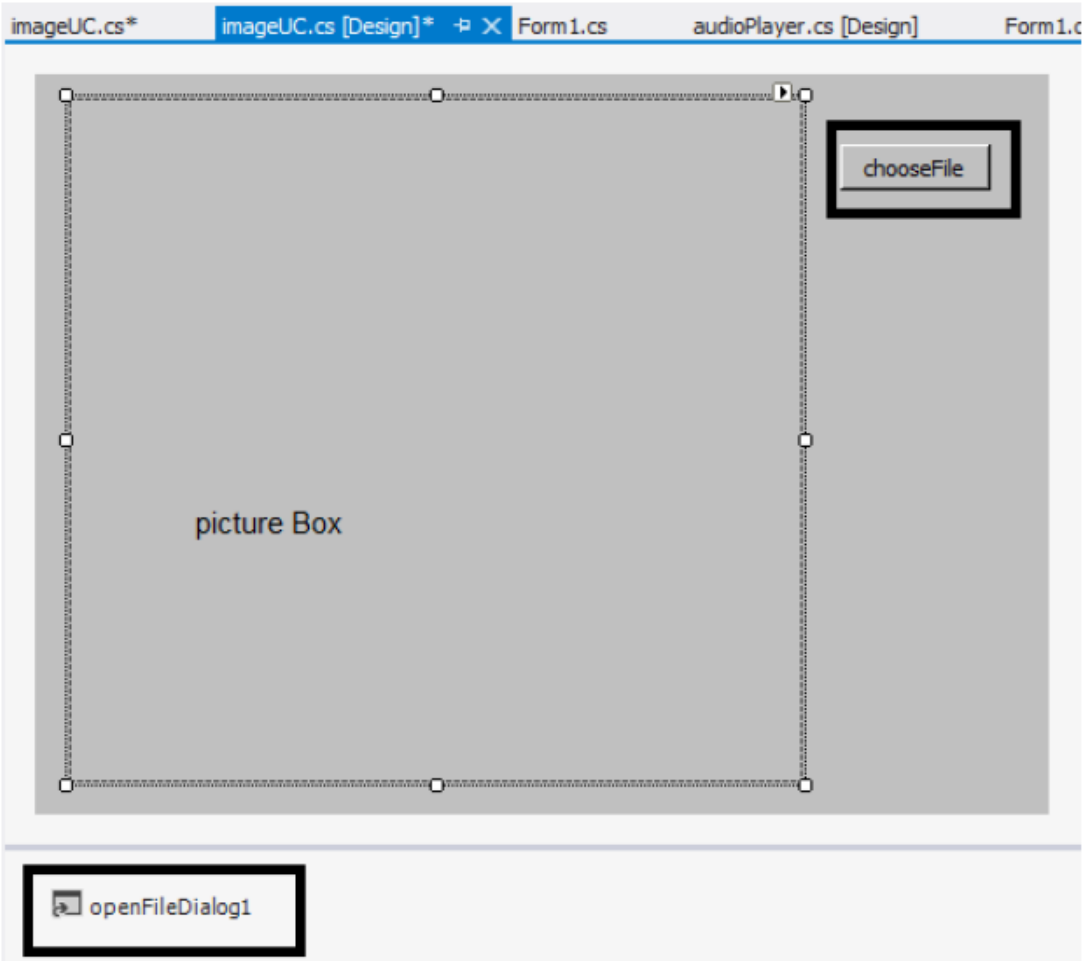
ننقر باليمين على المشروع User Control ← New Item ← Add

نسمي الـ UserControl الأولى بـ ImageUC وهي التي ستكون مخصصة لعرض الصور.

نسمي الـUserControl الثانية بـ mediaPlayerUC وهي التي ستكون مخصصة لتشغيل ملفات الصوت والفيديو.

أولاً: برمجة الـImageUC:

نضيف على هذه الواجهة زر وعنصر PictureBox لعرض الصور وعنصر OpenFileDialog لتحديد مسار الصورة التي سيقوم بعرضها، و Label مخفي لعرض النتائج نسميه resultLbl.



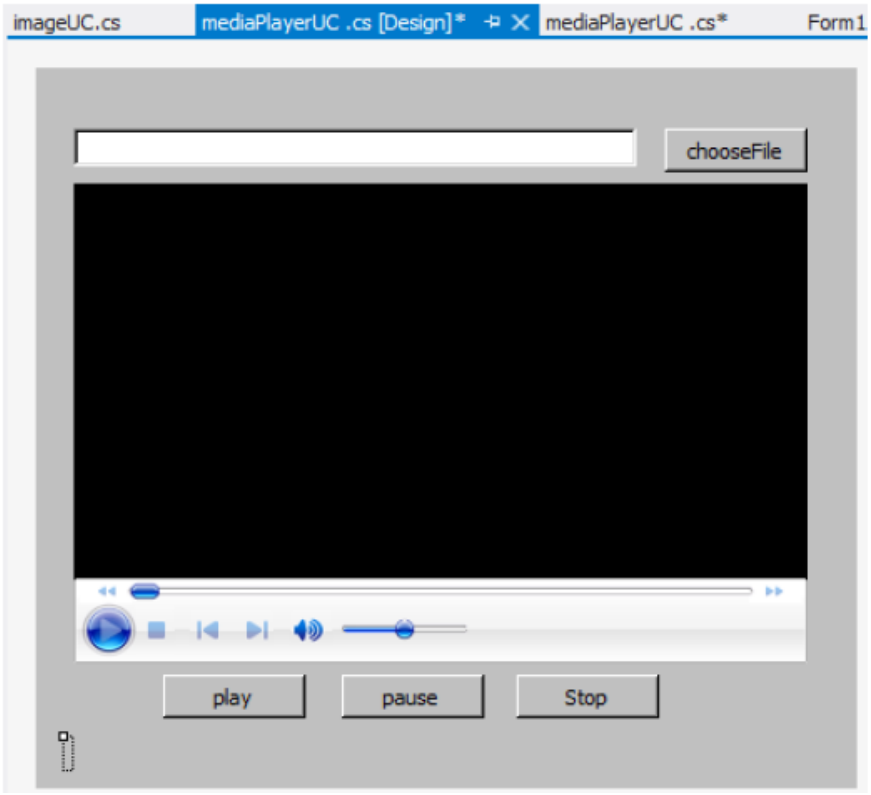
في حدث النقر على الزر سنكتب الكود التالي:

```
try
      {
          if (openFileDialog1.ShowDialog() == DialogResult.OK)
          {
              pictureBox1.ImageLocation = openFileDialog1.FileName;
              pictureBox1.Load();
              resultLbl.Text = openFileDialog1.FileName;
          }
      }
      catch (Exception e1)
      {
          resultLbl.Text = e1.Message;
      }
```

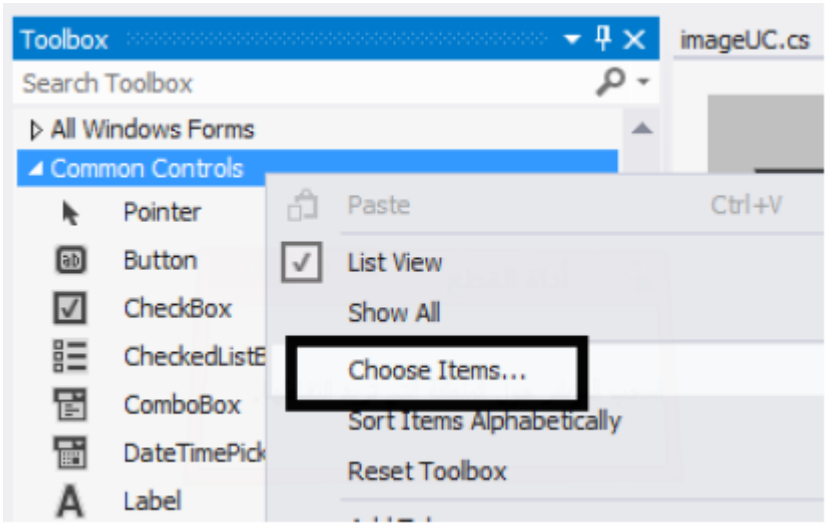
طبعا يمكننا التعديل بخاصية ال Filter للـ OpenFileDialog1 بحيث لا تظهر للمستخدم إلا ملفات الصور فقط، مثلا لجعله يعرض فقط ملفات من نمط jpeg نكتب ضمن خاصة ال filter عبارة. jpeg|*.jpeg

ثانياً: برمجة الـ MediaPlayerUC والتي سيتم من خلالها تشغيل ملفات الصوت والفيديو:

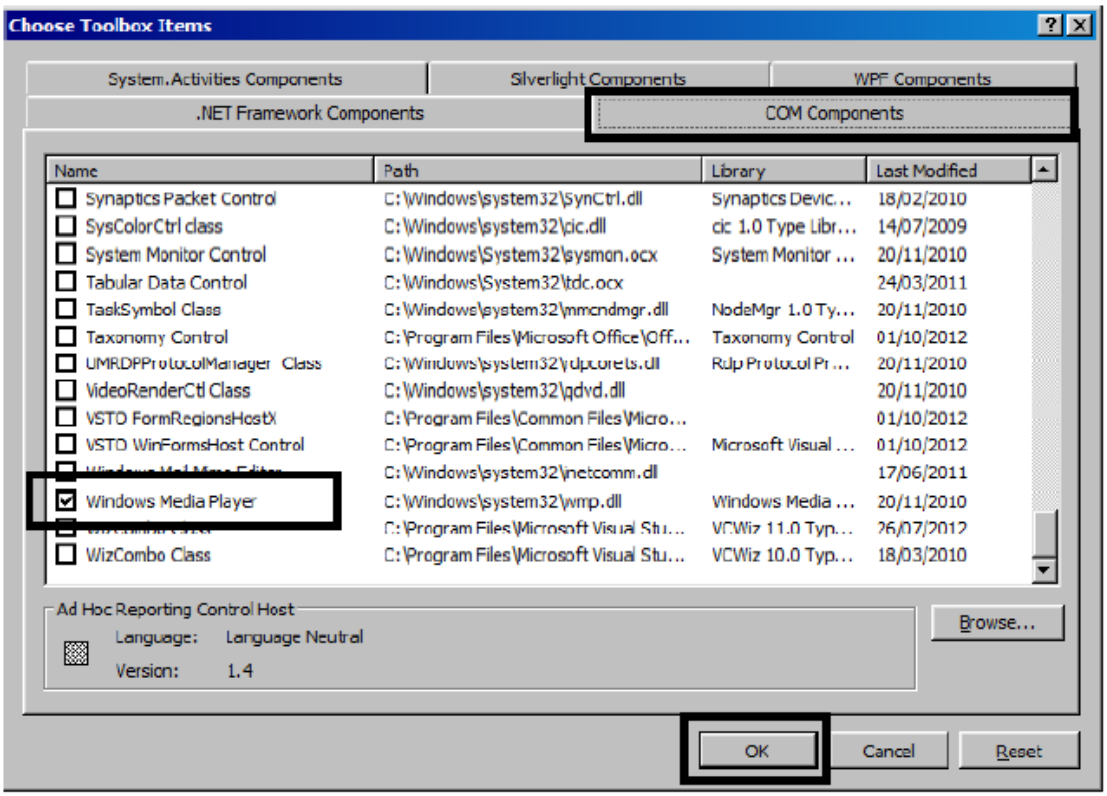
نحتاج في هذه الواجهة إلى أربعة أزرار، و `TextBox` و `Label` لعرض النتائج، وعنصر`OpenFileDialog` لتحديد مسار الملف الذي نود تشغيله.



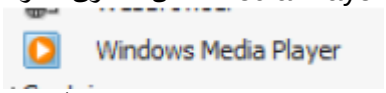
كما نحتاج إلى عنصر خاص هو الـ Windows Media Player والذي لا يكون موجودا بشكل افتراضي ضمن صندوق الأدوات، بل نحتاج إلى إضافته يدوياً :
ننقر باليمين على صندوق الأدوات ونختار Choose Items



تظهر لدينا واجهة إضافة العناصر الخاصة إلى الـ ToolBox حيث تكون هذه العناصر من مكتبات مختلفة:



من صفحة Com Componenets نختار Windows Media Player ثم نضغط ok، فتظهر أداة Windows Media Player ضمن صندوق الأدوات وهنا نستطيع إضافتها لواجهتنا وتعديل حجمها بالشكل الملائم.



نقوم بتعديل الاسم البرمجي لهذه الأداة ليصبح mp.

في حدث النقر على زر Choose File نكتب الكود التالي:

```
try
            {
                if (openFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    mp.URL = openFileDialog1.FileName;
                    textBox1.Text = openFileDialog1.FileName;
                }


            }
            catch (Exception e1)
            {
                resultLbl.Text = e1.Message;
            }
```

حيث يتم من خلال هذه التعليمة تحديد مسار الملف الصوتي أو الفيديو الذي يريد المستخدم تشغيله.

في حدث النقر على زر play نكتب الكود التالي:

```
mp.Ctlcontrols.play();
```

في حدث النقر على زر pause نكتب الكود التالي:

```
mp.Ctlcontrols.pause();
```

في حدث النقر على زر Stop نكتب الكود التالي:

```
mp.Ctlcontrols.stop();
```

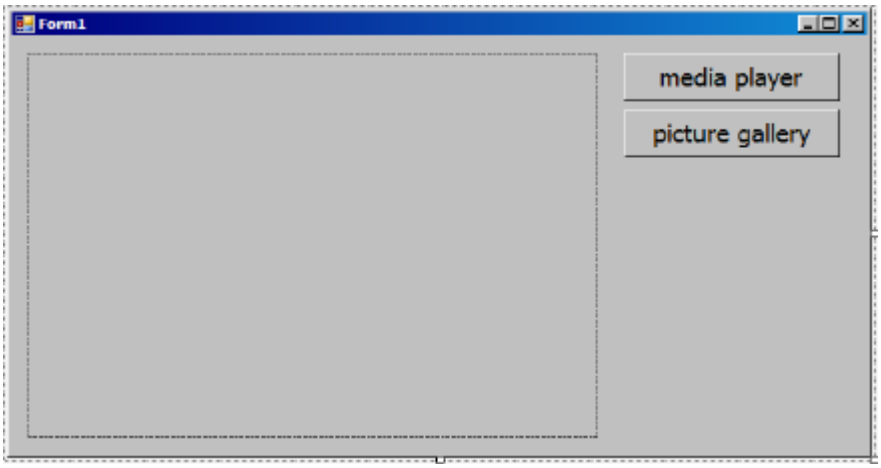
كما يمكن التعديل على خاصية ال Filter لل OpenFileDialog1 وذلك لجعله يعرض فقط ملفات الصوت والفيديو.
بعد الانتهاء من العمل نقوم ببناء المشروع للتأكد من خلوه من الأخطاء:

Build → rebuild soloution

بقيت لدينا الخطوة الأخيرة وهي عرض الـ UserControl التي قمنا بإنشائها على الواجهة الأساسية في المشروع، حيث سيتم تحميلها على الواجهة أثناء التنفيذ .وسيتم ذلك باستخدام التحميل الديناميكي للـ UserControl .

التحميل الديناميكي للـ UserControl:

في الواجهة الرئيسية نقوم بإضافة زرين وعنصر Panel:



بحدث النقر على زر media Player نكتب الكود التالي:

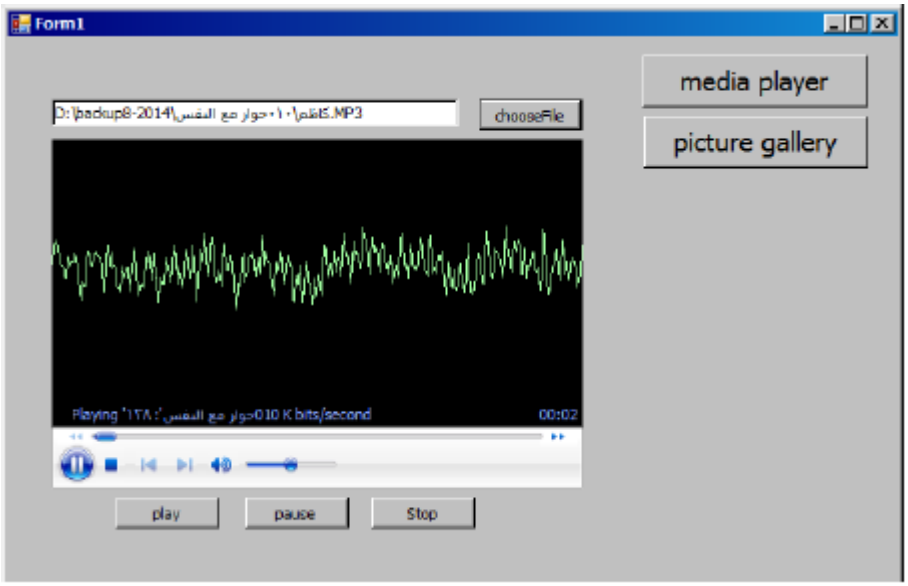
```
mediaPlayerUC a = new mediaPlayerUC();
panel1.Controls.Clear();
panel1.Controls.Add(a);
```

بحدث النقر على زر picture Gallery نكتب الكود التالي:

```
imageUC i = new imageUC();
panel1.Controls.Clear();
panel1.Controls.Add(i);
```

شرح الشيفرة:
في السطر الأول قمنا بتعريف غرض من الـImageUC User Control
في السطر الثاني قمنا بتنظيف الـ Panel1 من أية عناصر قد تكون موجودة عليها وذلك لنتمكن من عرض العنصر

الجديد.
في السطر الثالث نقوم بإضافة العنصر الجديد للـPanel1 .
عند تنفيذ المشروع سترى أن الواجهات يتم بناؤها وتحميلها ضمن حدث النقر على الأزرار الموجودة على الواجهة الرئيسية:



نهاية الجلسة

التعابير النظامية
Regular Expressions

محاور الجلسة الخامسة:

- مقدمة
- أمثلة بسيطة.
- مثال: التحقق من أرقام الهواتف.
- مبادئ أساسية في التعابير النظامية.
- التكرارات.
- مجموعات المحارف.

مقدمة:

عند كتابة برامج أو صفحات ويب التي تتعامل مع القيم النصية، غالبا ما نحتاج لتحديد جزء من النص الذي يطابق نمطا معينا، تستخدم التعابير النظامية Regular Expressions لوصف هذا النمط، مثلاً الرمز "\w+" هو طريقة ترميز نص مكون من محرف واحد أو أكثر وحيث تكون المحارف أرقاما أو أحرفا بالتحديد.
تحتوي بيئة الـ net. على مكتبة قوية للتعامل مع التعابير النظامية، يمكن من خلال هذه المكتبة البحث عن نص معين واستبداله أومعالجته.
سنتعلم كيفية التعامل مع هذه المكتبة من خلال الأمثلة.

أمثلة بسيطة:

لنفترض أننا نريد البحث في الملفات عن معلومات عن بيل جيتس وأي معلومات عن حياته، يمكنك البحث عن هذه البيانات كما يلي:

1- bil: عند كتابة هذه العبارة سيبحث عن جميع الكلمات التي تحتوي على bil بغض النظر عن حالة الحرف، Bil , bIL ، إلا أنه سيحض جميع الكلمات التي تحتوي على تسلسل الأحرف السابق مثل Bill و Isabil.
2- \bbil\b: حيث أن الرمز \b هو رمز خاص يعني بداية أو نهاية الكلمة، وفي هذه الحالة سيبحث عن الكلمة bil فقط، مع عدم التركيز على الأحرف الكبيرة أو الصغيرة.
3- لنفترض أننا نريد البحث عن أي معلومة عن طبيعة حياة بيل جيتس، أي كلمة bil تليها بعد عدد غير محدد من المحارف كلمة Life. الرمز "." يعني أي محرف من المحارف ماعدا محرف السطر الجديد (NewLine)، أما الرمز "*" يعني تكرار المحرف الذي قبله عدة مرات (صفر مرة أو أكثر).
وهكذا لنبحث عن معلومات حياة بيل جيتس نكتب التعبير:
\bbil\b.*\blife\b وهي تعني أوجد نص يحوي كلمة bil تليها كلمة life.

مثال التحقق من أرقام الهواتف:

لنفترض أننا نريد التحقق من أن النص الذي يقوم المستخدم بإدخاله مكون 7 أرقام وبالشكل "xxx-xxxx" نكتب التعبير التالي، الذي سيبحث في النص عن صيغة مشابهة مكونة من 7 أرقام:

4- \b\d\d\d-\d\d\d\d\b وهو يعني أوجد رقم هاتف مكون من 7 أرقام.

الرمز \d يعني رقم مفرد بين ال0 وال9 ، الرمز "-" ليس له أي معنى خاص وبالتالي بحث عنه حرفيا ضمن النص.

5- \b\d{3}-\d{4}\b وهو يعني أوجد رقم هاتف مكون من 7 أرقام.

بدل تكرار الرمز "\d" بعدد المرات المطلوبة نكتفي بكتابة "\d{3}" حيث أن هذا الرمز"{3}" يعني تكرار الرمز السابق ثلاث مرات.

مبادئ أساسية في التعابير النظامية:

أوجد كلمة تبدأ بالحرف a.

\ba\w*\b

حيث أنه يبحث عن بداية كلمة \b، ثم الحرف a ، ثم عدد غير محدد من الرموز الرقمية أو الحرفية، ثم نهاية الكلمة \b.

\d+ إيجاد نص يحوي أرقاما (رقم واحد أو أكثر).

\b\w\{6}\b إيجاد كلمة مكونة من 6 أحرف.

والجدول التالي يوضح الرموز الشائعة الاستخدام في التعابير النظامية:

.	Match any character except newline
\w	Match any alphanumeric character
\s	Match any whitespace character
\d	Match any digit
\b	Match the beginning or end of a word
^	Match the beginning of the string
$	Match the end of the string

الرمز "^" يجب أن يتم استخدامه للدلالة عن بحث عن كلمة تقع في أول النص.

الرمز "$" يجب أن يتم استخدامه للدلالة عن بحث عن كلمة تقع في آخر النص.

9 - ^\d{3}-\d{4}$ البحث عن نص مكون من 7 أرقام فقط، ولا يحتوي أي سلاسل نصية أخرى.

إذا أردنا أن يظهر الرمز على حاله ضمن النص (مثلا البحث عن رمز $ ضمن النص) نستخدم قبله الإشارة "\" مثلا:

10 – كل من ".\" ,"\$" ,"\\" تكافئ "." ,"$" ,"\" بالترتيب.

التكرارت:

الجدول التالي يحدد الرموز الخاصة بالتكرار:

*	Repeat any number of times
+	Repeat one or more times
?	Repeat zero or one time
{n}	Repeat *n* times
{n,m}	Repeat at least *n*, but no more than *m* times
{n,}	Repeat at least *n* times

أمثلة على التكرار:

10. `\b\w{5,6}\b` *Find all five and six letter words*

11. `\b\d{3}\s\d{3}-\d{4}` *Find ten digit phone numbers*

12. `\d{3}-\d{2}-\d{4}` *Social security number*

13. `^\w*` *The first word in the line or in the text*

مجموعات المحارف:

إذا أردنا البحث عن مجموعة محارف معينة مثلا عن إذا أردنا البحث عن أحد الأحرف الصوتية التالية نكتب [aeiou] ، إذا أردنا البحث عن أحد علامات الترقيم نكتب [!?.] ونلاحظ هنا أن الرموز (. و ! و ؟) عندما تم وضعها داخل أقواس مربعة يتم تفسيرها حرفياً.

للبحث عن رمز رقم أو حرف صغير نكتب [a-z,0-9] إذا أردنا البحث عن أحرف كبيرة فقط

.[A-Z]

نفي المحارف:

\W	Match any character that is NOT alphanumeric
\S	Match any character that is NOT whitespace
\D	Match any character that is NOT a digit
\B	Match a position that is NOT the beginning or end of a word
[^x]	Match any character that is NOT *x*
[^aeiou]	Match any character that is NOT one of the characters *aeiou*

تمت الجلسة

تدقيق القيم المدخلة
Input Validation

محاور الجلسة السادسة:

- مقدمة
- العنصر Error Provider.
- مثال: إنشاء صف لتدقيق القيم المدخلة في الـ C#.

مقدمة:

تحتاج أغلب المشاريع إلى تدقيق البيانات التي يقوم المستخدم بإدخالها، وذلك للتأكد من صحة هذه البيانات. سنتعرف في هذه الجلسة على عدة طرق لتدقيق البيانات المدخلة، ثم سنقوم ببناء class يكون مسؤول عن أغلب عمليات التحقق من المدخلات Input Validation.

العنصر Error Provider:

هو أداة مخصصة لعرض إشارة للمستخدم تفيد بحدوث خطأ أثناء إدخاله للبيانات، ويمكن استخدامها للإشارة إلى أن البيانات التي تم إدخالها صحيحة، لاستخدام الـ ErrorProvider نقوم بسحبها من قسم الـ Components من الـ Toolbox ووضعها على الـ Form وتكفي نسخة واحدة من هذه الأداة لتدقيق الأخطاء لكافة عناصر الـ Form.

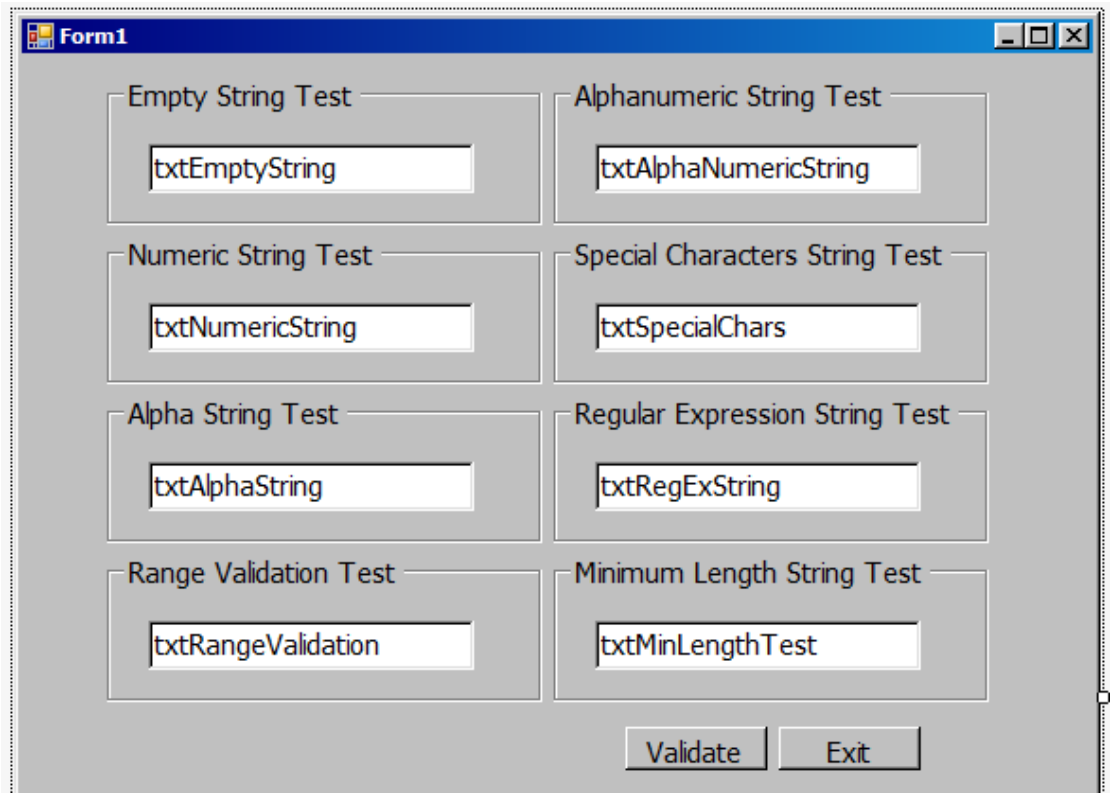
مثال: تدقيق إدخالات المستخدم:

في هذا المثال سنقوم بإنشاء واجهة لإدخال البيانات وتدقيقها حيث يشمل هذا المثال تدقيق البيانات وفق المعايير التالية كل على حدة:

1. التحقق من أن مربع النص غير فارغ.
2. التحقق من أن النص يحوي أحرفا فقط.
3. التحقق من أن النص يحوي أرقاما فقط.
4. التحقق من أن النص مكون من أحرف وأرقام فقط (لا يحوي رموزا خاصة).
5. التحقق من النص يحوي فقط رموزا خاصة (#$%&@...).
6. التحقق من أن العدد يقع ضمن مجال محدد (بين الـ 50 و الـ 100).
7. التحقق من أن النص يطابق تعبيرا نظاميا محددا (Regular Expression).
8. التحقق من عدد محارف النص.

نقوم بإنشاء مشروع جديد من نمط WindowsFormsApplication ونبني الـ Form1 كما هو موضح بالشكل التالي، وطبعا لا ننسى إضافة عنصرين من نوع ErrorProvider.

ملاحظة : النص المكتوب داخل الـ TextBox في الشكل التالي هو فقط اسم العنصر البرمجي Name ولكن تمت كتابتها ضمن الـ Textbox للتوضيح فقط، أما خاصة الـ Text يجب أن تكون فارغة لجميع الـ TextBoxes.



نقوم بإنشاء class نسميه Validation ويكون من نوع public static سيحتوي على جميع اسطر الكود المخصص للتحقق من الإدخالات، وسنقوم ببرمجة البنود السابقة واحداً تلو الآخر:

سنعتمد في كامل هذا الصف على طريقة محددة وهي أن نقوم بكتابة تابع لتدقيق القيمة المدخلة وتكون بارامترات الدخل لهذا التابع هي مربع النص، وعنصر الErrorProvider و قد نحتاج لتمرير عناصر أخرى.

1. **<u>التحقق من أن مربع النص غير فارغ:</u>**

نكتب في صف Validation الكود التالي:

```
public static void validateEmptyString(object sender,ErrorProvider
ep)
        {
 objectيتم إرسال مربع النص من الواجهة الرئيسية على شكل غرض من نوع          //

            TextBox tb = (TextBox)sender;//Converting object into a
textbox
            if (tb.Text == string.Empty)
            {
مربع النص فارغ ويجب أن تظهر فيه رسالة خطأ                    //
                ep.SetError(tb, "you need to enter text here");
            }
            else
            {
في حال تم إدخال نص ضمن مربع النص يجب إزالة رسالة الخطأ                  //
```

```
            ep.SetError(tb,"");
        } }
```

يجب أن تظهر لدينا إشارة خطأ في حال مغادرة مربع النص (إزالة التحديد عنه) قبل إدخال قيمة نصية فيه.
وبالتالي نقوم بتحديد مربع النص الذي نريد تدقيقه ونختار الحدث Leave الذي يعمل عند انتقال التحديد إلى عنصر آخر من الـ Form ونستدعي بداخله الإجرائية السابقة:

```
private void txtEmptyString_Leave(object sender, EventArgs e)
        {
            validation.validateEmptyString(sender, errorProvider1);}
```

ملاحظات:

- الإجرائية تم استدعاؤها دون عمل غرض من صف Validation والسبب أننا قمنا ببناء الصف على أنه static والإجراءيات بداخله من نوع staticأيضا، أي يمكن استدعاءها دون إنشاء غرض من الصف.

- sender هو اسم العنصر الذي حدث عليه الحدث Leave وفي مثالنا هو الـ txtEmptyString.

في حال قمنا بإدخال نص في مربع النص يجب أن تختفي إشارة الخطأ، لذا نستدعي الإجرائية نفسها في حدث الـ TextChanged لمربع النص نفسه، حيث يعمل هذا الحدث في حال كان مربع النص محددا وتم الضغط على اي زر من لوحة المفاتيح:

```
private void txtEmptyString_TextChanged(object sender,EventArgs e)
        {
            validation.validateEmptyString(sender, errorProvider1);}
```

يمكنك الآن تنفيذ المشروع ومحاولة ترك مربع النص الذي قمنا بتدقيقه فارغا.

2- التحقق من أن النص يحوي أحرفاً فقط:

نكتب في صف الـ Validation الكود التالي:

```
public static void validateLettersOnly(object sender, ErrorProvider
ep, KeyPressEventArgs e)
        {
            TextBox tb = (TextBox)sender;
            //إذا لم يكن الزر المضغوط حرفا، ولم يكن من زر أزرار التحكم
            if (!char.IsLetter(e.KeyChar) &&
!char.IsControl(e.KeyChar))
            {
                e.Handled = true; //تجاهل حدث ضغط الزر
                ep.SetError(tb, "letters only");
            tb.Focus();

            }
            else
            {
                e.Handled = false;//لا تتجاهل حدث ضغط الزر
                ep.SetError(tb, "");        }}
```

في هذه الحالة يفضل منع المستخدم من إدخال الأرقام أو الرموز داخل مربع النص، وذلك بتجاهل حدث ضغط الزر (أو اعتبار أن معالجة الحدث قد انتهت).
مثل هذا التدقيق يجب أن يتم عند كل ضغطة زر من لوحة المفاتيح تتم أثناء تحديد مربع النص الذي نقوم بتدقيقه، لذلك نقوم باستدعاء إجرائية ValidateLettersOnly داخل الحدث KeyPress لمربع النص:

```
private void txtAlphaString_KeyPress(object sender,
KeyPressEventArgs e)
        {
        validation.validateLettersOnly(sender, errorProvider1, e); }
```

حيث أن الـ KeyPressEventArgs هو عبارة عن غرض يحوي تفاصيل حدث ضغط الزر.

3- التحقق من أن النص يحوي أرقاما فقط:

نكتب في صف الـ Validation الكود التالي:

```
public static void validateNumbersOnly(object sender, ErrorProvider
ep, KeyPressEventArgs e)
        {
            TextBox tb = (TextBox)sender;
            //التحكم أزرار من زر يكن ولم رقما، المضغوط الزر يكن لم إذا
            if (!char.IsDigit(e.KeyChar) &&
!char.IsControl(e.KeyChar))
            {
                e.Handled = true;
                ep.SetError(tb, "digits only");
                  tb.Focus();

            }
            else
            {
                e.Handled = false;
                ep.SetError(tb,"");    }}
```

في هذه الحالة يفضل منع المستخدم من إدخال الأحرف أو الرموز داخل مربع النص، وذلك بتجاهل حدث ضغط الزر (أو اعتبار أن معالجة الحدث قد انتهت).
مثل هذا التدقيق يجب أن يتم عند كل ضغطة زر من لوحة المفاتيح تتم أثناء تحديد مربع النص الذي نقوم بتدقيقه، لذلك نقوم باستدعاء إجرائية ValidateNumbersOnly داخل الحدث KeyPress لمربع النص:

```
private void txtNumericString_KeyPress(object sender,
KeyPressEventArgs e)
        {
       validation.validateNumbersOnly(sender,errorProvider1,e);    }
```

وفي حدث مغادرة مربع النص Leave نكتب الكود التالي:

```
private void txtNumericString_Leave(object sender, EventArgs e)
        {
            errorProvider1.SetError(txtNumericString,""); }
```

4 - التحقق من أن النص مكون من أحرف وأرقام فقط (لا يحوي رموزا خاصة).

نكتب في صف الـ Validation الكود التالي:

```
public static void validateAlphaNumericOnly(object sender,
ErrorProvider ep, KeyPressEventArgs e)
        {
            TextBox tb = (TextBox)sender;
            //التحكم أزرار من زر يكن ولم رقما، أو حرفا المضغوط الزر يكن لم إذا
            if (!char.IsLetter(e.KeyChar) &&
!char.IsDigit(e.KeyChar)&&!char.IsControl(e.KeyChar))
            {
                e.Handled = true;//الزر ضغط حدث تجاهل
                ep.SetError(tb, "letters and digits only");
                  tb.Focus();
            }
            else
            {
                e.Handled = false;//لا تتجاهل ضغط حدث الزر
                ep.SetError(tb, "");      }}
```

في هذه الحالة نقوم بمنع المستخدم من إدخال الرموز داخل مربع النص، وذلك بتجاهل حدث ضغط الزر (أو اعتبار أن معالجة الحدث قد انتهت).
مثل هذا التدقيق يجب أن يتم عند كل ضغطة زر من لوحة المفاتيح تتم أثناء تحديد مربع النص الذي نقوم بتدقيقه، لذلك نقوم باستدعاء إجرائية ValidateAlphaNumericOnly داخل الحدث KeyPress لمربع النص:

```
private void txtAlphaNumericString_KeyPress(object sender,
KeyPressEventArgs e)
{
   validation.validateAlphaNumericOnly(sender, errorProvider1, e); }
```

وفي حدث مغادرة مربع النص Leave نكتب الكود التالي:

```
private void txtAlphaNumericString_Leave(object sender, EventArgs e)
        {
            errorProvider1.SetError(txtAlphaNumericString, ""); }
```

5- التحقق من النص يحوي فقط رموزا خاصة (#$%&@...):

نكتب في صف الـ Validation الكود التالي:

```
public static void validateSymbolOnly(object sender, ErrorProvider
ep, KeyPressEventArgs e)
        {
            TextBox tb = (TextBox)sender;
            //التحكم أزرار من زر يكن ولم رمزاً، المضغوط الزر يكن لم إذا
            if (!char.IsSymbol(e.KeyChar)  &&
!char.IsControl(e.KeyChar))
            {
                e.Handled = true;//الزر ضغط حدث تجاهل
                ep.SetError(tb, "symbols only");
            tb.Focus();
            }
            else
            {
                e.Handled = false;//الزر ضغط حدث تتجاهل لا

                ep.SetError(tb, "");          }}
```

في هذه الحالة نقوم بمنع المستخدم من إدخال نصا غير الرموز داخل مربع النص، وذلك بتجاهل حدث ضغط الزر (أو اعتبار أن معالجة الحدث قد انتهت).
مثل هذا التدقيق يجب أن يتم عند كل ضغطة زر من لوحة المفاتيح تتم أثناء تحديد مربع النص الذي نقوم بتدقيقه، لذلك نقوم باستدعاء إجرائية validateSymbolOnly داخل الحدث KeyPress لمربع النص:

```
private void txtSpecialChars_KeyPress(object sender,
KeyPressEventArgs e)
        {
         validation.validateSymbolOnly(sender, errorProvider1, e); }
```

6- التحقق من أن العدد يقع ضمن مجال محدد (بين الـ 50 و الـ 100):

هنا يجب التحقق أولا من أن النص المدخل مكون من أرقام فقط، أي نستدعي إجرائية ValidateNumbersOnly - التي كتبناها سابقا- ضمن حدث keyPress لمربع النص:

```
private void txtRangeValidation_KeyPress(object sender,
KeyPressEventArgs e)
   {
        validation.validateNumbersOnly(sender, errorProvider1, e); }
```

في صف الـ Validation نكتب إجرائية لمقارنة القيمة المدخلة مع المجال المحدد من قبل المستخدم:

```
public static void validateRange(int mivValue, int maxValue, object
sender, ErrorProvider ep)
        {
            TextBox tb = (TextBox)sender;
            int txtValue;
            try
            {
                txtValue = Convert.ToInt32(tb.Text);
                if (txtValue >= mivValue && txtValue <= maxValue)
                {
                    ep.SetError(tb, "");
                }
                else
                {  ep.SetError(tb,"the value must be between "+
mivValue.ToString()+" and "+ maxValue.ToString());

                  tb.Focus();

                }
            }
            catch { }}
```

بعد أن يقوم المستخدم بإدخال القيمة الرقمية ومحاولته الانتقال إلى عنصر آخر يجب أن يتم فحص القيمة المدخلة لمعرفة إذا كانت تنتمي للمجال المحدد، أي أن إجرائية ValidateRange سيتم استدعاؤها ضمن حدث Leave لمربع النص:

```
private void txtRangeValidation_Leave(object sender, EventArgs e)
 {
    validation.validateRange(50, 100, sender, errorProvider1); }
```

7- التحقق من أن النص يطابق تعبيرا نظاميا محددا (Regular Expression):

في هذا المثال سنقوم بجعل التعبير النظامي مطابقا لرقم الهاتف : 000 000-0000
في صف الـ Validation نكتب الكود التالي:

أولا نقوم باستدعاء المكتبة

```
using System.Text.RegularExpressions;
```

ثم نقوم بكتابة الإجرائية التالية:

```
public static void validatePhoneRegEx(object sender, ErrorProvider
ep){
            TextBox tb = (TextBox)sender;//Converting object into a
textbox

            string TheRegExTest = @"\d{3}-\d{3}-\d{4}";
            Regex TheRegExpression = new Regex(TheRegExTest);
            // test text with expression
            if (TheRegExpression.IsMatch(tb.Text))
```

```
            {
                ep.SetError(tb, "");
            }
            else
            {
                ep.SetError(tb, "input string was not in the correct
format");
                tb.Focus();}}
```

وفي حدث مغادرة مربع النصLeave نكتب:

```
private void txtRegExString_Leave(object sender, EventArgs e)
        {
            validation.validatePhoneRegEx(sender, errorProvider1); }
```

8- التحقق من عدد محارف النص:

نقوم بكتابة هذا الكود ضمن صف الـ Validation:

```
public static void validateMinLength(object sender, ErrorProvider
ep, int minLength)
        {
            TextBox tb = (TextBox)sender;//Converting object into a
textbox
            char[] testArr = tb.Text.ToCharArray();

            if (testArr.Length < minLength)
            {
                ep.SetError(tb, "text must be 3 characters at
least");
                tb.Focus();
            }

            else { ep.SetError(tb, ""); }
       }
```

وفي حدث مغادرة مربع النصLeave نكتب:

```
private void txtMinLengthTest_Leave(object sender, EventArgs e)
        {
            validation.validateMinLength(sender,errorProvider1,3); }
```

نهاية الجلسة