**a) Comenta sobre lo que es y cómo se utiliza una librería (archivo .dll) en .net.**

¿Qué es un archivo DLL?

Una biblioteca de vínculos dinámicos (DLL) es un archivo ejecutable que actúa como una biblioteca compartida de funciones. La vinculación dinámica proporciona una forma para que un proceso para llamar a una función que no es parte de su código ejecutable. El código ejecutable para la función se encuentra en una DLL, que contiene una o más funciones que se compilan, vinculados entre sí, y se almacenan por separado de los procesos que los utilizan. DLL también facilitar el intercambio de datos y recursos. Múltiples aplicaciones pueden acceder simultáneamente a los contenidos de una única copia de un archivo DLL en la memoria.

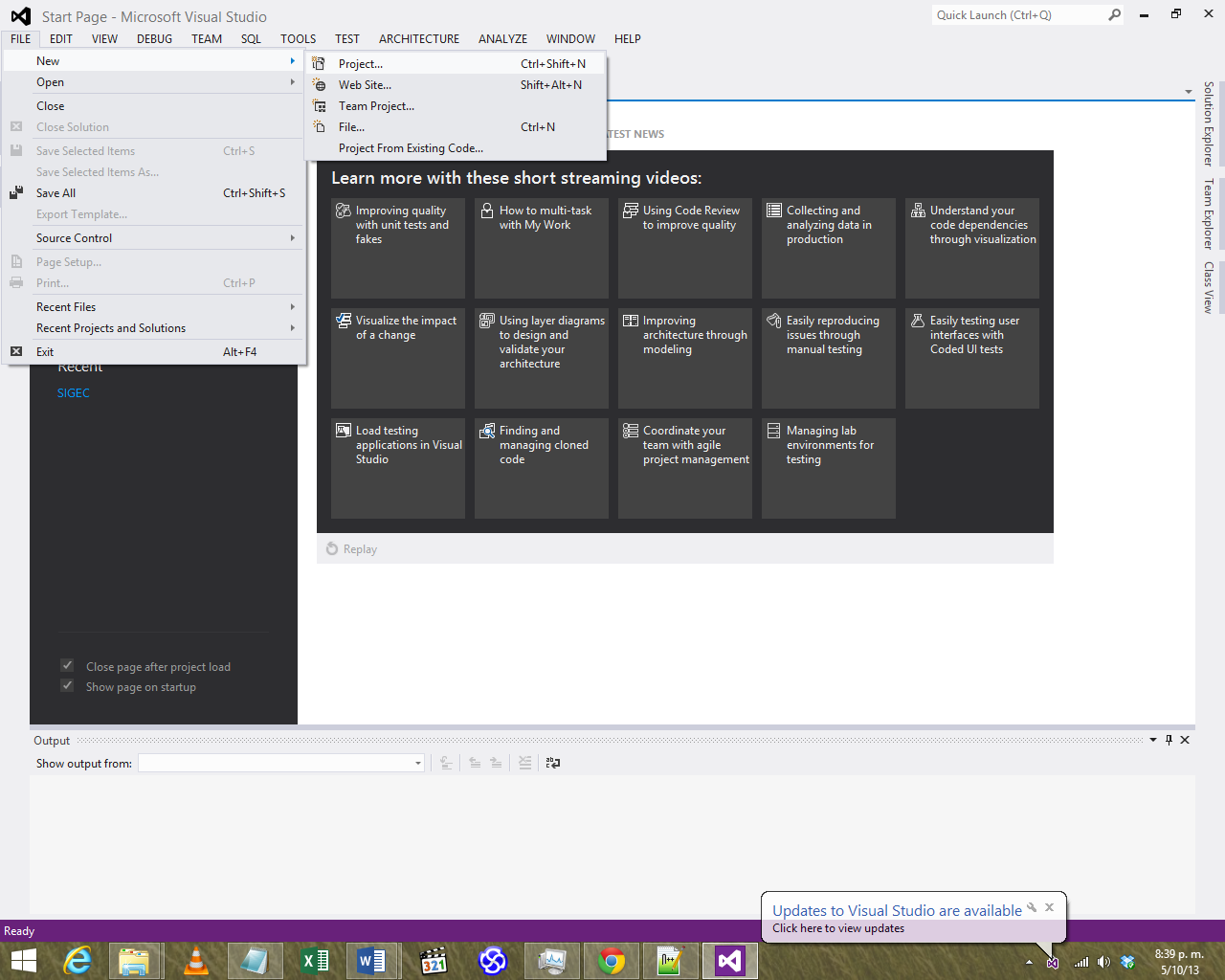
Un archivo DLL es una biblioteca que contiene el código y los datos que se pueden utilizar por más de un programa al mismo tiempo. Por ejemplo, en sistemas operativos Windows, el archivo Comdlg32 DLL realiza común funciones relacionadas con el cuadro de diálogo. Por lo tanto, cada programa puede usar la funcionalidad contenida en este archivo DLL para implementar un cuadro de diálogo abrir. Esto ayuda a promover la reutilización de código y el uso eficaz de la memoria.Mediante el uso de un archivo DLL, un programa puede ser modular en componentes separados. Por ejemplo, un programa de contabilidad puede venderse por módulo. Cada módulo puede cargarse en el programa principal en tiempo de ejecución, si está instalado el módulo. Dado que los módulos son independientes, el tiempo de carga del programa es más rápido y un módulo sólo se carga cuando se solicita esa funcionalidad.

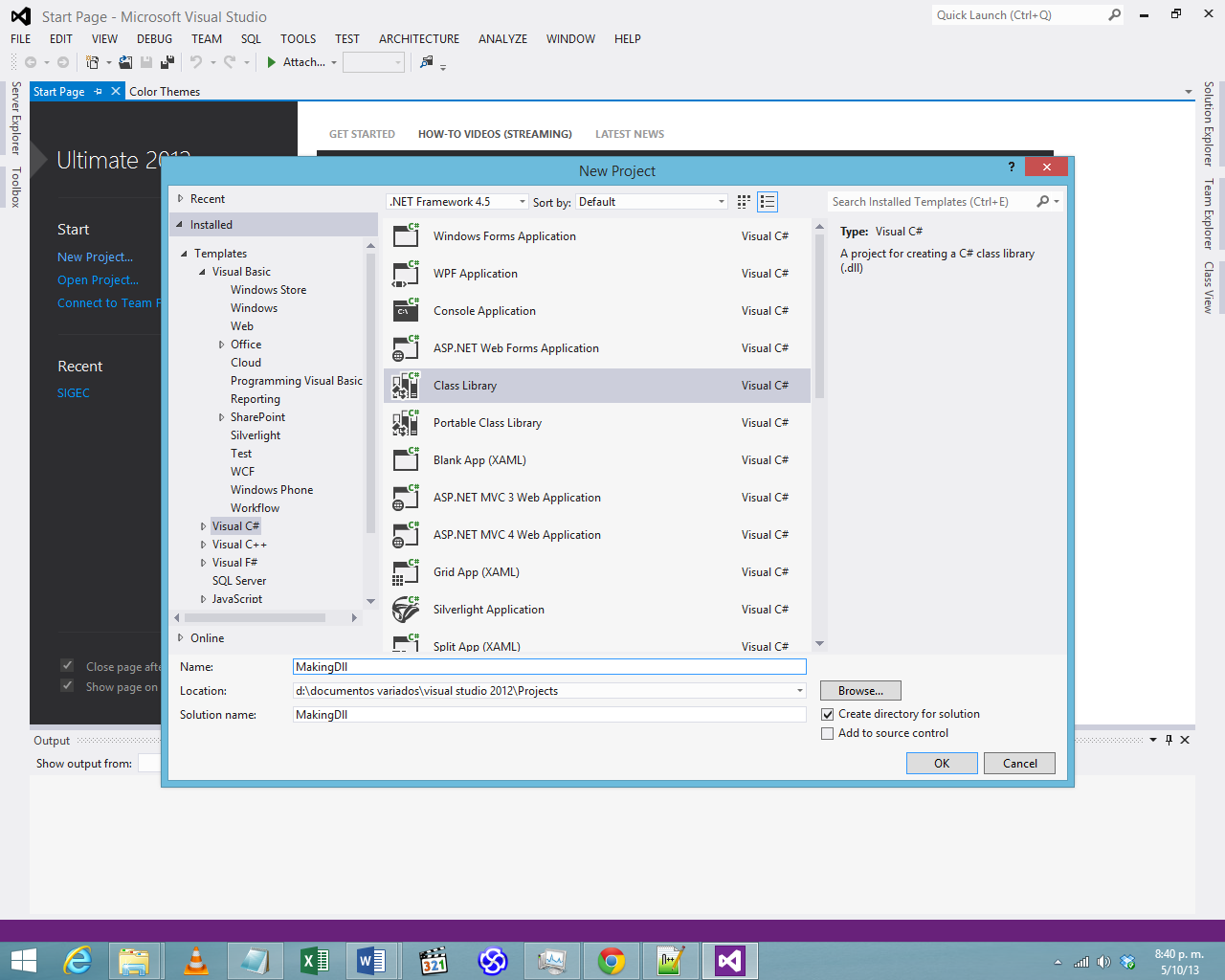
Además, las actualizaciones son más fáciles de aplicar a cada módulo sin afectar a otras partes del programa. Por ejemplo, puede que un programa de nóminas y los tipos impositivos cambian cada año. Cuando estos cambios se aíslan en un archivo DLL, puede aplicar una actualización sin necesidad de generar o volver a instalar el programa.

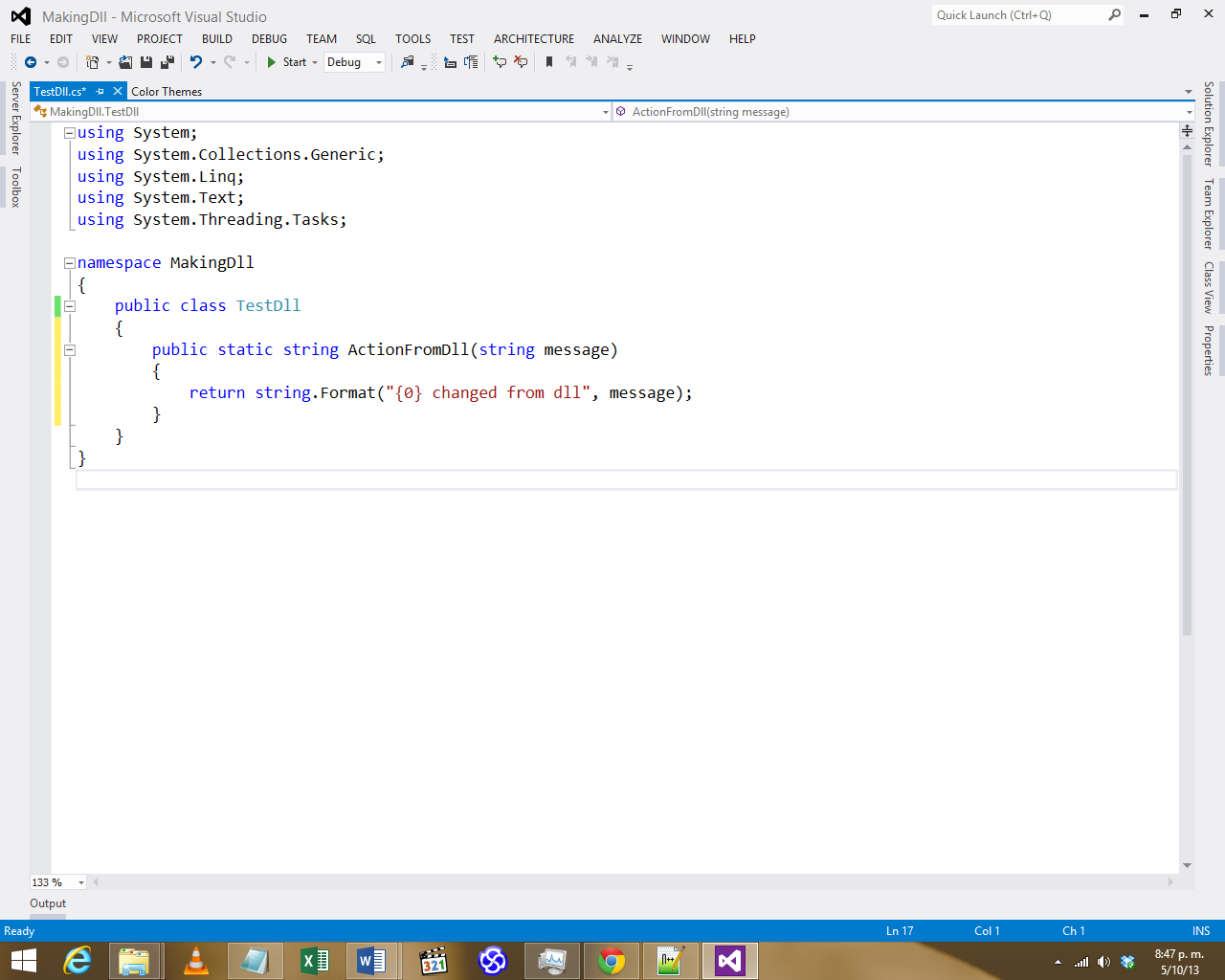
El uso de archivos DLL ayuda a promover el diseño modular de código, la reutilización de código, uso eficaz de la memoria y espacio en disco reducido. Por lo tanto, el sistema operativo y los programas se cargan más rápido, se ejecuten más rápidamente y necesitan menos espacio de disco en el equipo.Cuando un programa utiliza un archivo DLL, un problema que se denomina dependencia puede provocar que el programa no se ejecute. Cuando un programa utiliza un archivo DLL, se crea una dependencia. Si otro programa se sobrescribe y saltos de esta dependencia, no puede ejecutar correctamente el programa original.

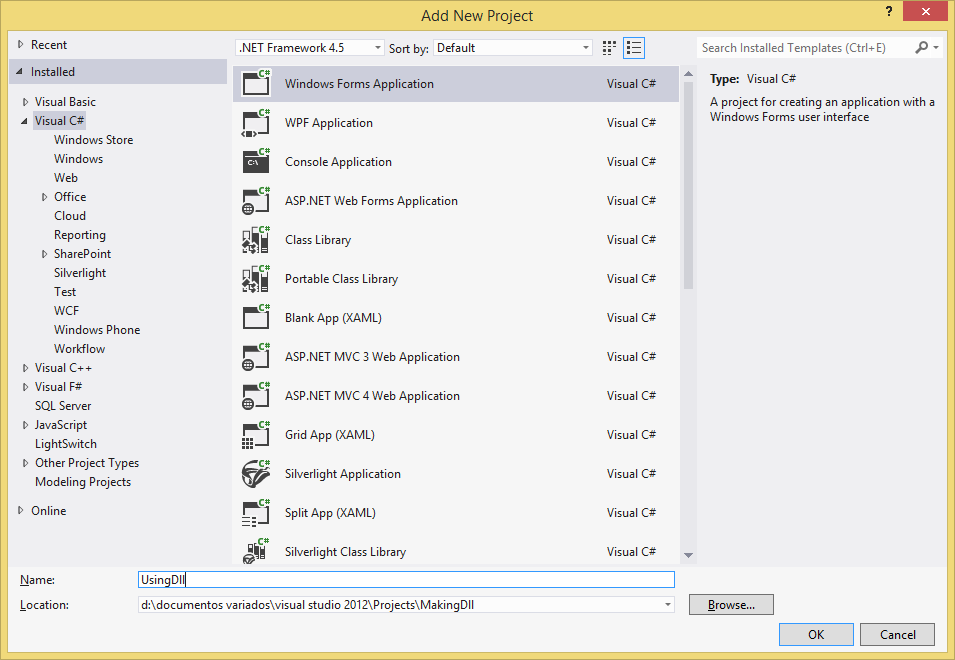
**Crear un archivo dll**Para crear un archivo dll usando visual studio se pueden seguir las siguientes instrucciones:

Abrir 'Visual Studio' y seleccionar las siguientes opciones: File -> Open -> New ->Project  para crear un nuevo proyecto. Luego presionar ok.

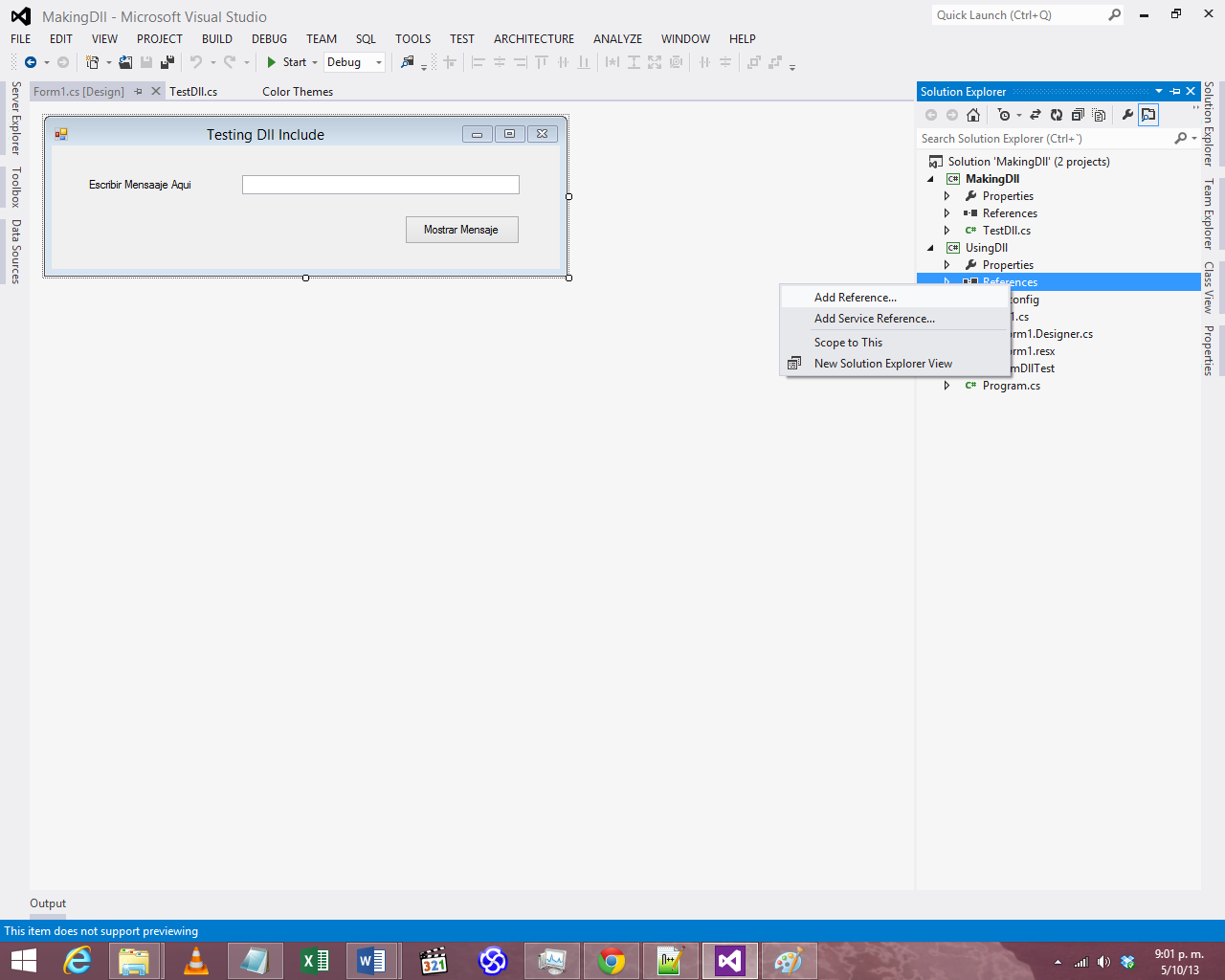


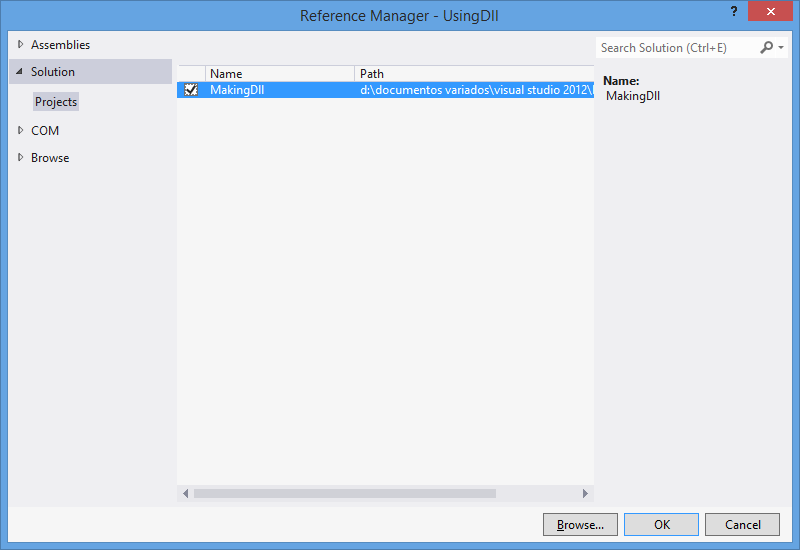


Luego de creado el proyecto podemos agregar cualquiera que sea la funcionalidad que deseamos agregar en nuestra dll. Para el ejemplo usaremos una simple función que devuelve un string nuevo a partir de otro recibido por parámetros alterándolo usando string.format.

Luego de compilar y generar nuestra librería, se creará un archivo .dll que podrá ser referenciado desde cualquier otro proyecto. En nuestro ejemplo lo referenciaremos desde una nueva aplicación de Windows forms. 

Luego, procedemos a incluir la referencia a la dll que previamente creamos al geenerar nuestra biblioteca de clases.





Luego de añadir la referencia podemos usar las funciones de la dll agregada de esta manera:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MakingDll;

namespace UsingDll

{

public partial class FrmDllTest : Form

{

public FrmDllTest()

{

InitializeComponent();

}

private void btnMostrar\_Click(object sender, EventArgs e)

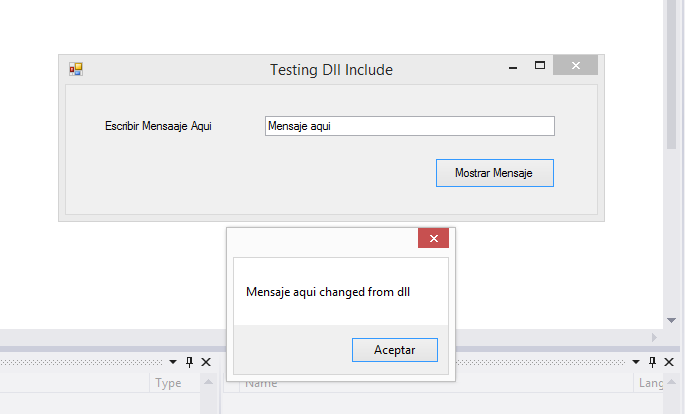
{

MessageBox.Show(TestDll.ActionFromDll(txtMensaje.Text));

}

}

}

Cuyo resultado es el siguiente:

**b) Investiga sobre el propósito de la clase SQLConnection, muestra un ejemplo.**

La clase SQLConnection es la clase que se encarga de gestionar y manejar las conexiones hacia la base de datos. Utilizando una instancia de este tipo de objetivo, se inicia una conexión con la base de datos SQL. Esta clase se utiliza mejor dentro de una sentencia using, para obtener mejores resultados en la utilización y liberación de recursos utilizados. Cuando trabajamos con esta clase debemos hacer ciertas operaciones básicas, las cuales son:

1. Se Crea una instancia del objeto SqlConnection.
2. Se Abre la conexión para poder realizar consultas a la base de datos con otros objetos como SQLCommand.
3. Realizar operaciones de base de datos con el resto de los objetos de ADO.NET.
4. Cerrar la conexión (este paso no es obligatorio, sí los recursos de SqlConnection se adquieren dentro de una sentencia using).

Para instanciar un objeto SqlConnection y usarlo apropiadamente se necesita una cadena de conexión a nuestra Base de datos, la cual usualmente está compuesta de los siguientes parámetros de conexión:

|  |  |
| --- | --- |
| **Cadena de conexión Nombre del parámetro** | **Descripción** |
| **Fuente de datos** | Identifica el servidor. Podría ser equipo local, el nombre de dominio de la máquina o la dirección IP. |
| **Initial Catalog** | Nombre de la base base de datos a conectar por defecto dentro de la instancia. |
| **Integrated Security** | Ajuste a SSPI para realizar la conexión con Windows con la cuenta de inicio de sesión del usuario |
| **ID de usuario** | Nombre de usuario configurado en SQL Server. |
| **Contraseña** | Contraseña de SQL Server, para el ID de usuario. |

Ejemplo de uso de la clase SqlConnection:

using System;

using System.Data.SqlClient;

class Program

{

static void Main()

{

// Obtener Connection string,

//en este caso desde un archivo de configuración

string connectionString =

ConsoleApplication1.Properties.Settings.Default.ConnectionString;

//instanciar el objeto SqlConnection desde una sentencia using

using (SqlConnection con = new SqlConnection(connectionString))

{

//Abrir la conexión

con.Open();

// Usar el objeto SqlConnection para las operaciones,

//en este caso con un objeto SqlCommand

using (

SqlCommand command =

new SqlCommand("SELECT TOP 2 \* FROM Dogs1", con))

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Console.WriteLine("{0} {1} {2}",

reader.GetInt32(0), reader.GetString(1),

reader.GetString(2));

}

}

}

}

}

**c) Investiga sobre el propósito de la clase SQLCommand, muestra un ejemplo.**

Cuando trabajamos con una base de datos desde nuestra aplicación, un objeto SqlCommand nos permite especificar qué tipo de interacción se desea llevar a cabo con una base de datos. Por ejemplo, se pueden hacer operaciones de selección, inserción, modificación y eliminación de filas de datos en una tabla de base de datos. El objeto SqlCommand también puede ser utilizado para escenarios de gestión de datos sin conexión usando otros objetos como SqlDataAdapter.

El objeto SqlCommand tiene un constructor con parámetros y que puede ser utilizado con uno o más argumentos.

-El primer parámetro especifica la sentencia SQL que se desea ejecutar en la base de datos.

-El segundo parámetro es el SqlConnection para especificar el objeto de conexión a la base de datos.

-El tercer parámetro es la SqlTransaction para definir una transacción asociada para las operaciones a ejecutar.

Ej:

using System;

using System.Data.SqlClient;

class Program

{

static void Main()

{

// Obtener la cadena de conexion

string conString = ConsoleApplication1.Properties.Settings.Default.ConnectionString;

using (SqlConnection connection = new SqlConnection(conString))

{

connection.Open();

// Crear el objeto SqlCommand, recomendable dentro de using

using (SqlCommand command = new SqlCommand(

"SELECT TOP 3 \* FROM Dogs1 ORDER BY Weight",

connection))

{

//Obtener y procesar los datos de la base de datos

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

for (int i = 0; i < reader.FieldCount; i++)

{

Console.WriteLine(reader.GetValue(i));

}

Console.WriteLine();

}

}

}

}

}

}

**d) Investiga sobre el propósito de la clase DataTable, muestra un ejemplo.**

Los datos se leen a partir de una base de datos. Se genera en la memoria de la entrada. DataTable es una clase que se utiliza para el almacenamiento de datos de cualquier fuente de datos. Con él tomamos objetos de la memoria y mostramos los resultados en controles, como el DataGridView.

A la hora de trabajar con este objeto, debemos tener en cuenta el nombre con el cuál definamos una determinada tabla, ya que los objetos declarados en en el DataTable tienen nombres sensitivos a mayúsculas y minúsculas, al igual que las reglas aplicables a las variables y objetos creados.

Ej:

using System;

using System.Data;

class Program

{

static void Main()

{

// Crear instancia de datatable usando uan funcion

DataTable table = GetTable();

//Luego se puede usar el datatable como fuente de datos.

dataGridView1.DataSource = table;

}

/// Este método crea el DataTable.

static DataTable GetTable()

{

// Aquí creamos un Datatable de 4 columnas

DataTable table = new DataTable();

table.Columns.Add("Dosage", typeof(int));

table.Columns.Add("Drug", typeof(string));

table.Columns.Add("Patient", typeof(string));

table.Columns.Add("Date", typeof(DateTime));

// Luego añadimos algunos records al datatable

table.Rows.Add(25, "Indocin", "David", DateTime.Now);

table.Rows.Add(50, "Enebrel", "Sam", DateTime.Now);

table.Rows.Add(10, "Hydralazine", "Christoff", DateTime.Now);

table.Rows.Add(21, "Combivent", "Janet", DateTime.Now);

table.Rows.Add(100, "Dilantin", "Melanie", DateTime.Now);

return table;

}

}