GRAPHICAL USER INTERFACE DESIGN

Chapter 7

Software Engineering
Computer Science School
DSIC – UPV

Goal

- Understand the principles of visual applications.
- Understand the design of the graphical user interface (use of controls and events).
- Understand the communication between the presentation and the business logic layers.

Contents

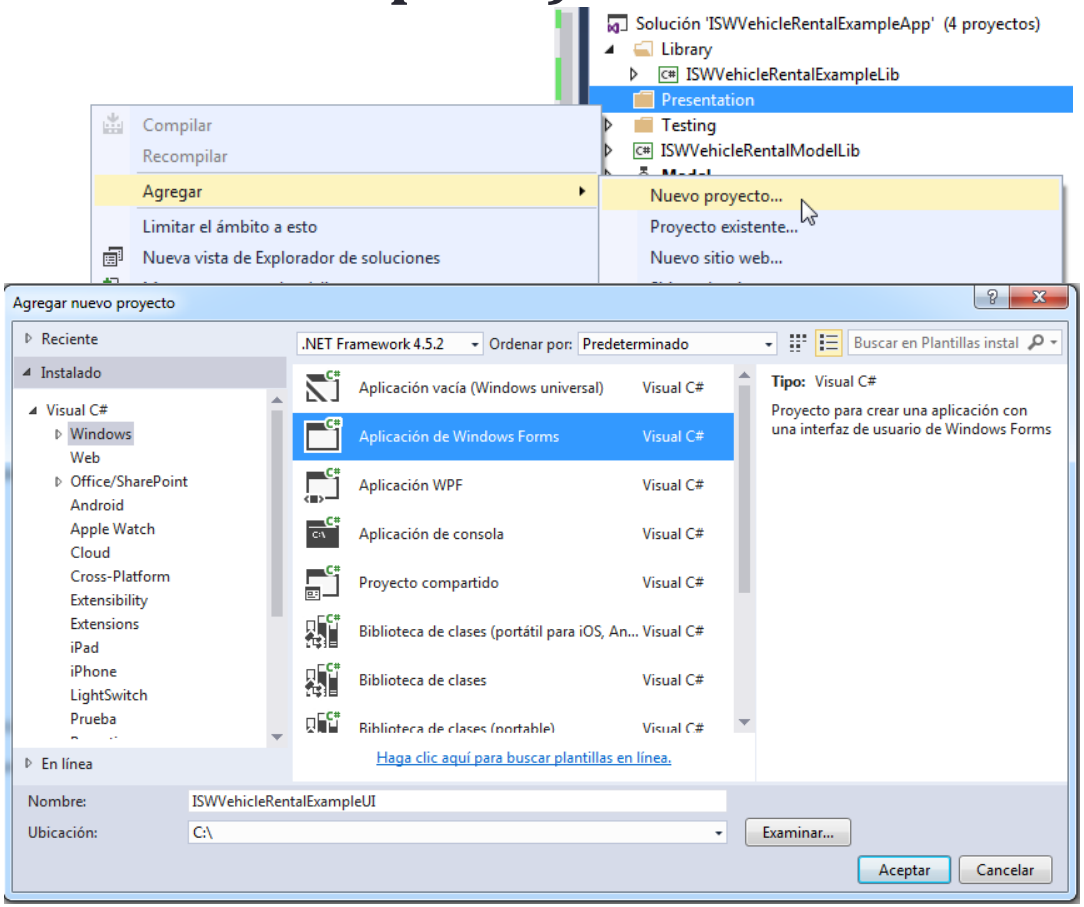
Introduction

- The creation of **Visual Apps for Windows** may be done, among others with the namespace `System.Windows.Forms` which includes classes, structures, interfaces, etc. to develop these types of applications.

- The namespace `System.Windows.Forms` includes the following classes:
  - **`Application`**: The core of a Windows app. Its methods are used to process Windows messages and visual apps are created and destruyed.
  - **`Form`**: Represents a window or a dialog box in a visual application.
  - **`Button`**, **`ListBox`**, **`TextBox`**, **`PictureBox`**, **`Label`**,...: Providing the functionality of common Windows controls.
  - **`StatusBar`**, **`ToolBar`**,...:Windows utilities.
  - **`ColorDialog`**, **`FileDialog`**,...: Standard dialog boxes.
  - **`StripMenu`**, **`StripMenuItem`**,...: Use to create different types of menus.
  - **`ToolTip`**, **`Timer`**,...: To ease the interactivity of applications.

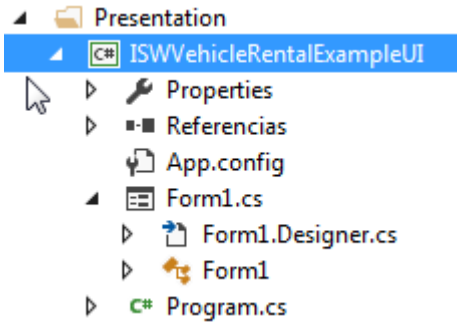
Creating a Windows Application

- Add a new project of type **Aplicación de Windows Forms** to the solution folder **Presentation** (e.g. named **ISWVehicleRentalExampleUI)**.



Creating a Windows Application

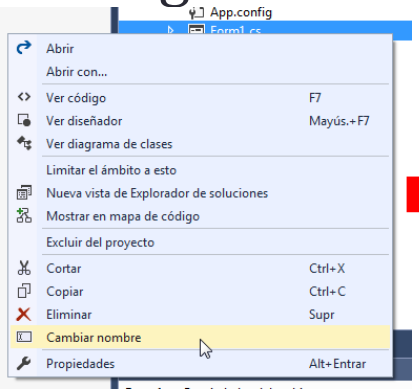
- If the app is run, a Windows with the standard basic features is created.
- The files in this Project are:
  - Form1.cs: contains the design of the form. If opened the form may be modified in a visual designer.
    - Form1 has the definition of the class Form1 with its constructor and a call to the method InicializeComponent().
    - Form1.Designer.cs has the Dispose() method and includes generated code for the method InicializeComponent()
  - Program.cs: contains the definition of the Mai method().

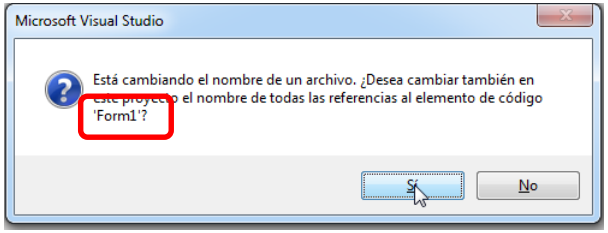



```
namespace ISWVehicleRentalExampleUI
{
    0 referencias
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        0 referencias
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

First steps…

- Give an appropriate name to the elements in the Project (e.g. change the name of the file **Form1.cs** to **ISWVehicleRentalApp**).

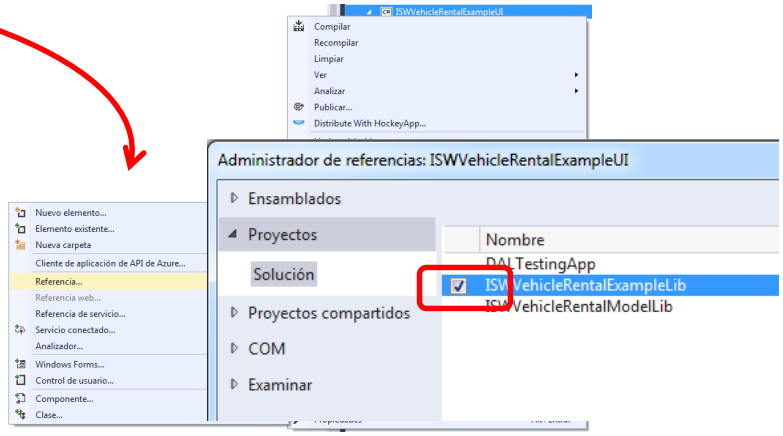





```
namespace ISWVehicleRentalExampleUI
{
    3 referencias
    public partial class ISWVehicleRentalApp : Form
    {
        1 referencia
        public ISWVehicleRentalApp()
        {
            InitializeComponent();
        }
    }
}
```

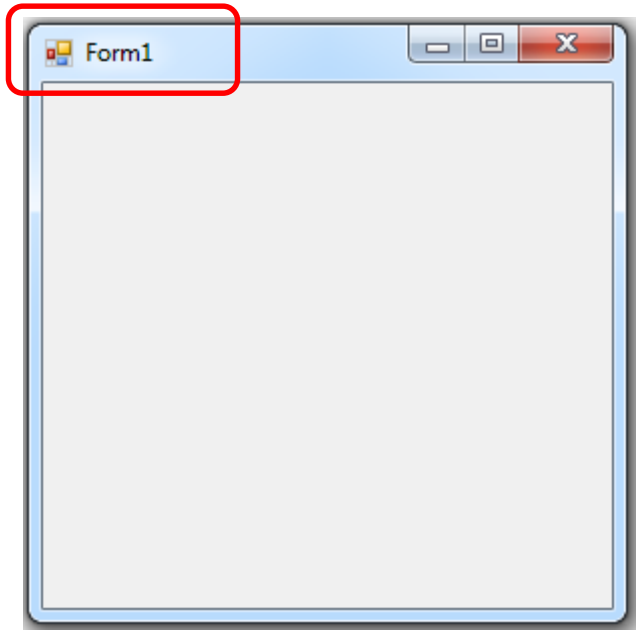
- Obtain an instance of the **BusinessController** class.

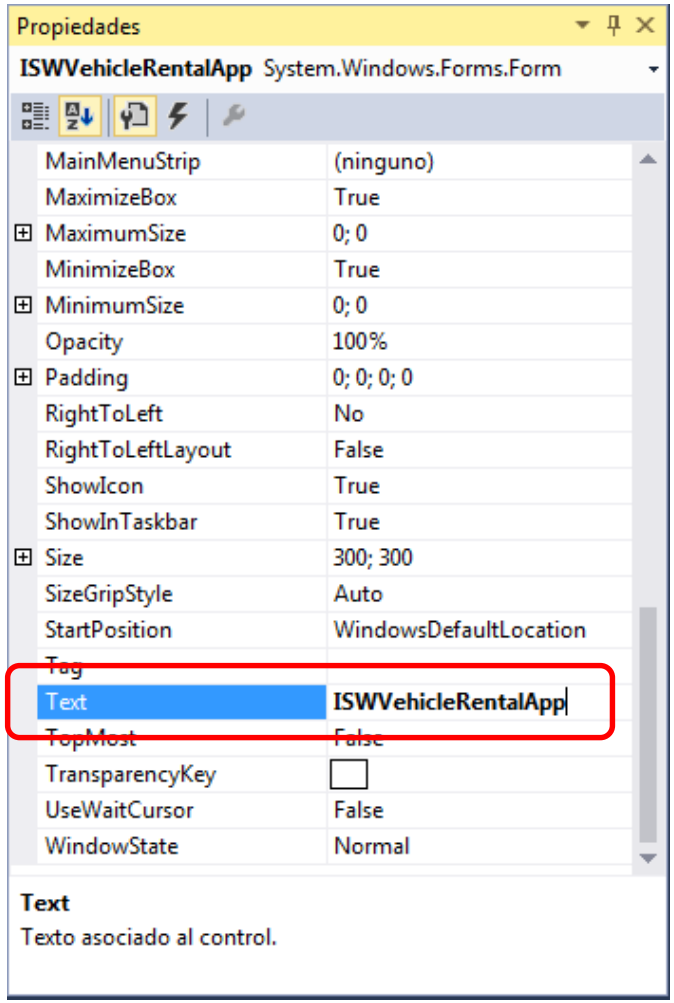
```
namespace ISWVehicleRentalExample.Presentation
{
    3 referencias
    public partial class IswVehicleRentalApp : Form
    {
        private BusinessController businessControl;
        1 referencia
        public IswVehicleRentalApp()
        {
            InitializeComponent();
            businessControl = BusinessController.getBusinessController();
        }
    }
}
```



First steps…

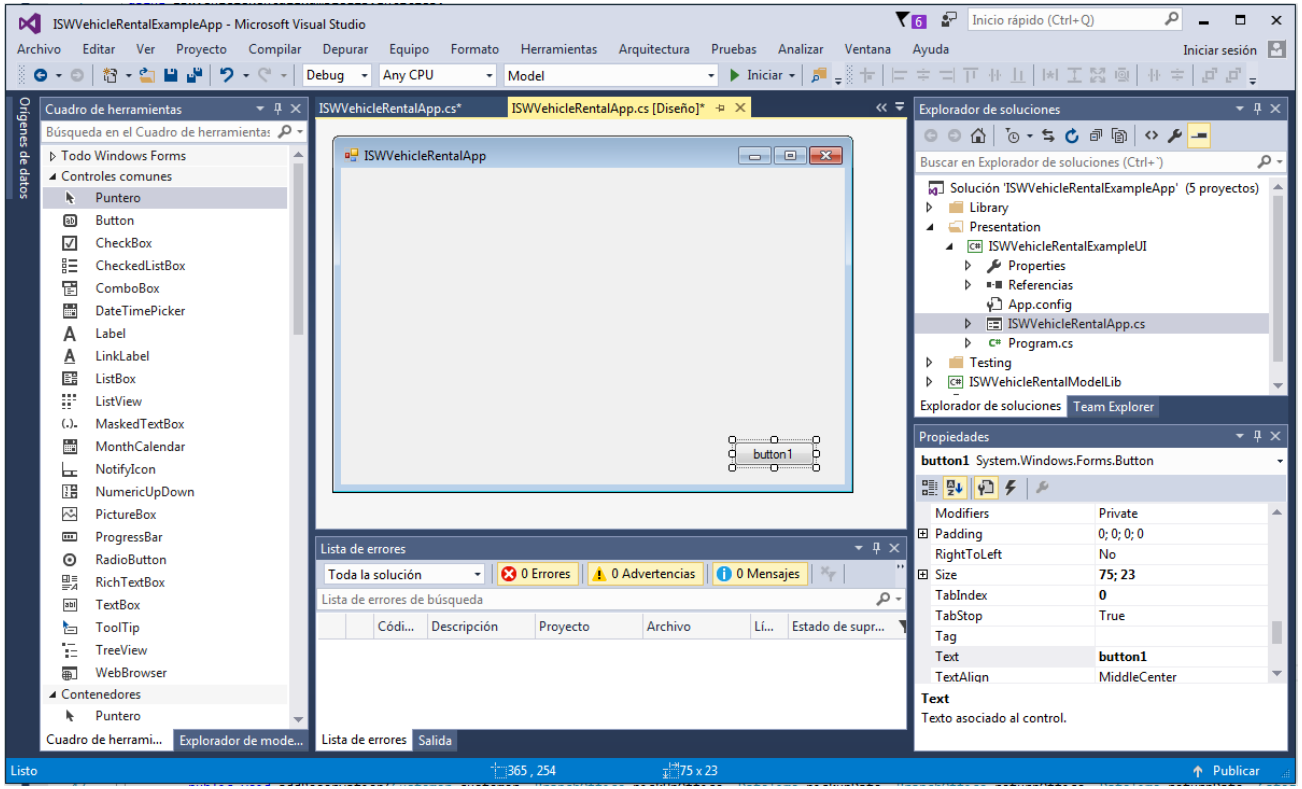
- Modify the properties of the form elements:





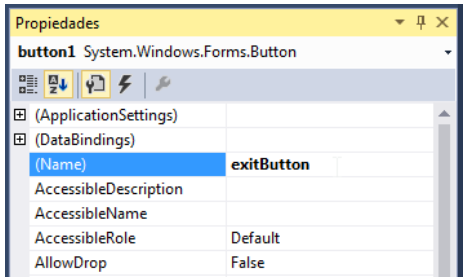
Forms with controls

- Controls are objects of the Control class: buttons, textboxes, …
- Can be added at design time (visual editor and toolbox) or at execution time.

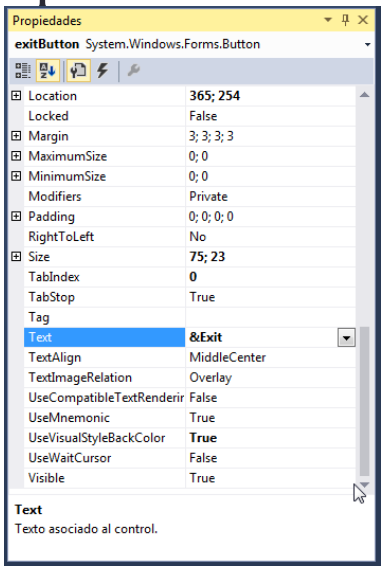


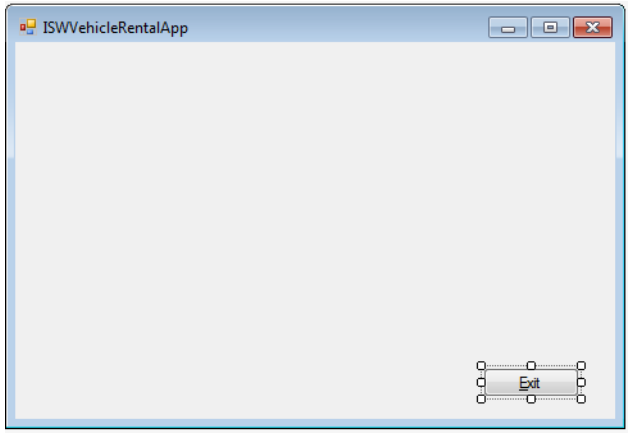
Controls: Properties

- **Name**: The name of the control. It is important to select a meaningful name.



- **Text**: Represents the title of the control



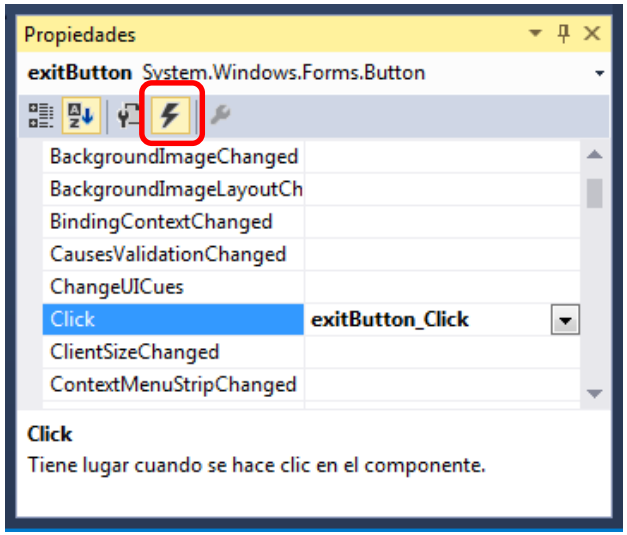


Events in forms

- An event describes a situation to which the application must respond.
- Events are generated by:
  - A user action (click a mouse button, hit a key, etc.)
  - The app code.
  - The operating system.
- Windows apps are event-driven:
  - When an event occurs the app may specify methods (event handlers) to process the event and execute the corresponding actions
- Every control exhibits events to which a handler can be associated.

Events: handlers

- When an event occurs the associated handler is executed
- The events that may be raised by a control appear in the properties window.
- A handler may be associated as follows:
  - Writing the name of the of the handler method.
  - Selecting a handler method from the dropdown list.
  - double *click*, and Visual Studio creates a default handler definition.



Objeto which raised the event

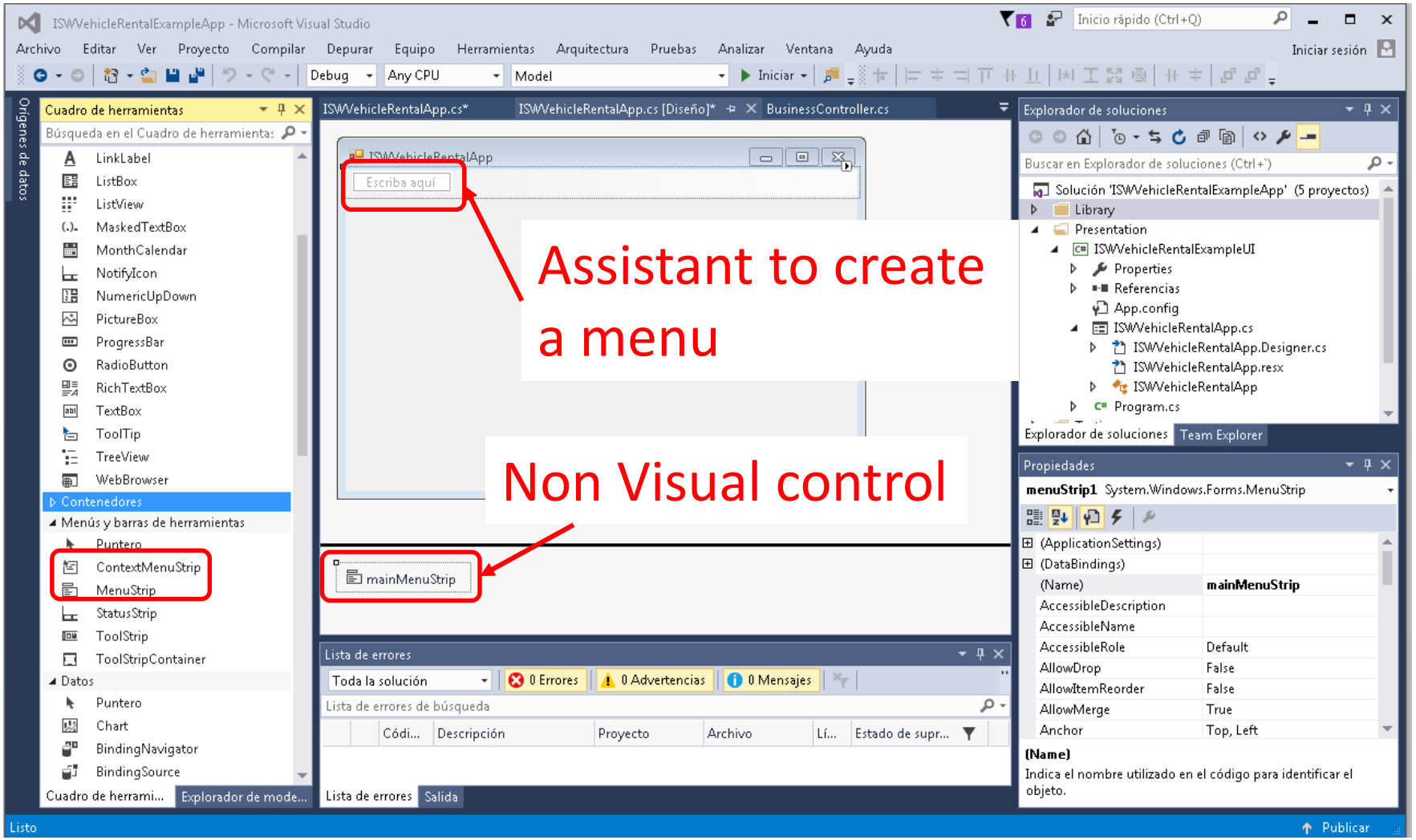
```
1 referencia
private void exitButton_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

Event information

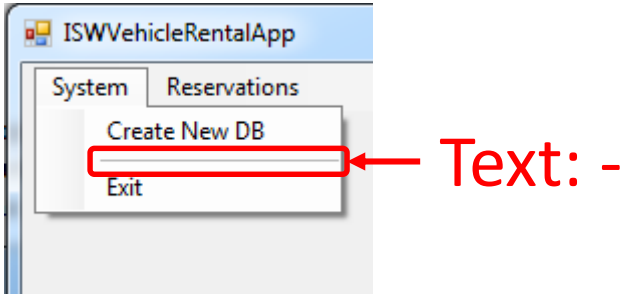
Designing and using menus

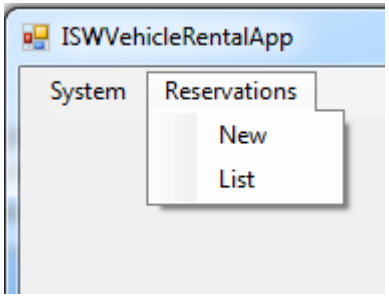
- Most Windows applications have menus
- There are two types of menus:
  - MenuStrip: a main menu
  - ContextMenuStrip: a contextual menu
- All the elements of a menu are stored in the Item property which is a collection of objects belonging to the class ToolStripMenuItem. These elements may contain other submenus.

Designing and using menus

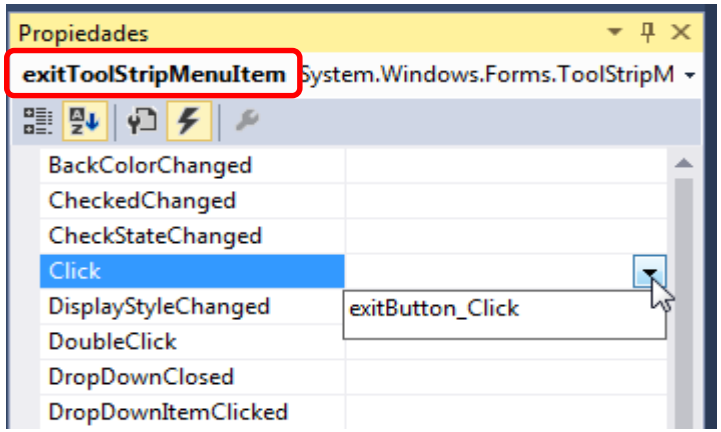


Example Menu





- Assigning a handler is done in the same way as with other controls.



Applications with several forms

- Usually several forms are used.
- The predefined aspect of a form is defined by the property FormBorderStyle.
- There are several types of forms:
  - User designed: added to the Project with Proyecto|Agregar Windows Forms.
  - Predefined in the environment: dialog box.

User Defined Forms

- Modal: It must be closed to return to the main form. It is shown using the method ShowDialog().
- Non Modal: several forms may be used simultaneously. Shown using the method Show().

Creating an object of the class ExampleFOrm

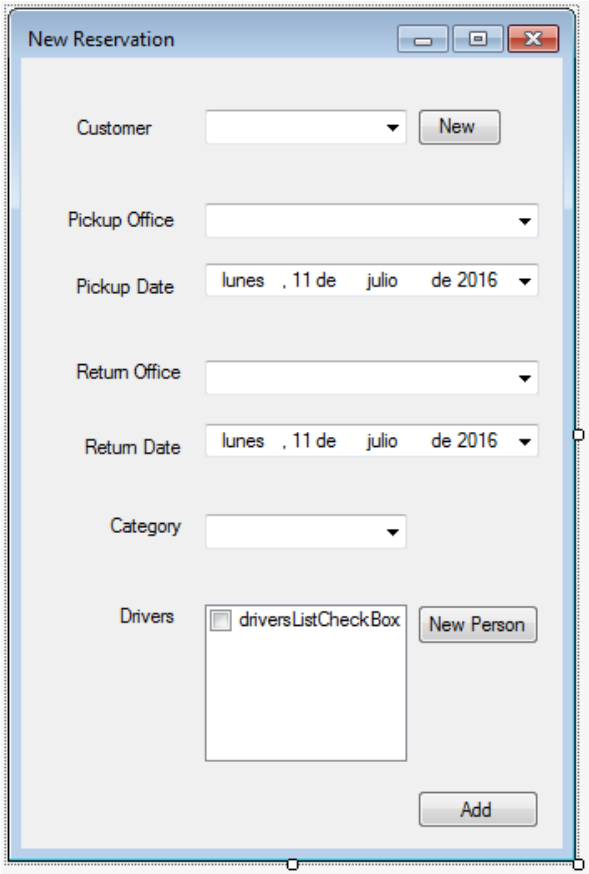
```
ExampleForm myForm = new ExampleForm();
myForm.ShowDialog();
```

```
ExampleForm myForm = new ExampleForm();
myForm.Show();
```

Shown in a modal way

Shown in a non modal way

Forms: Example



Forms: Example

- Passing the BusinessController object between forms

Creating the form NewReservation

Controller is passed

```
namespace ISWVehicleRentalExampleUI
{
    3 referencias
    public partial class ISWVehicleRentalApp :
    {
        private BusinessController businessControl;

        private NewReservationForm newReservationForm;
        1 referencia
        public ISWVehicleRentalApp()
        {
            InitializeComponent();
            businessControl = new BusinessController();
            newReservationForm = new NewReservationForm(businessControl);
        }

        1 referencia
        private void newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.newReservationForm.ShowDialog();
        }
```

Shown in a modal way

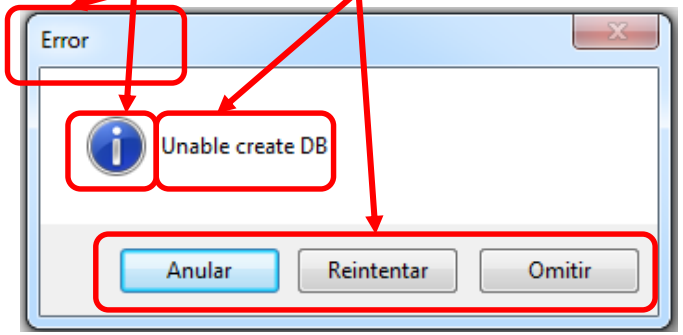
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using ISWVehicleRentalExampleLib.BusinessLogic;
using ISWVehicleRentalExampleLib.Entities;

namespace ISWVehicleRentalExampleUI
{
    4 referencias
    public partial class NewReservationForm : Form
    {
        private BusinessController businesscontrol;

        1 referencia
        public NewReservationForm(BusinessController control)
        {
            InitializeComponent();
            this.businesscontrol = control;
        }
```

Constructor is modified

Dialog boxes

- The class **MessageBox** provides simple dialog boxes and modal behavior.
- The title, the descriptive message and the icon may be customized using the Show method

```
DialogResult answer = MessageBox.Show(this, "Unable create DB", "Error",
    MessageBoxButtons.AbortRetryIgnore,
    MessageBoxIcon.Asterisk);
if (answer == DialogResult.Retry) //retry operation
{ }
```

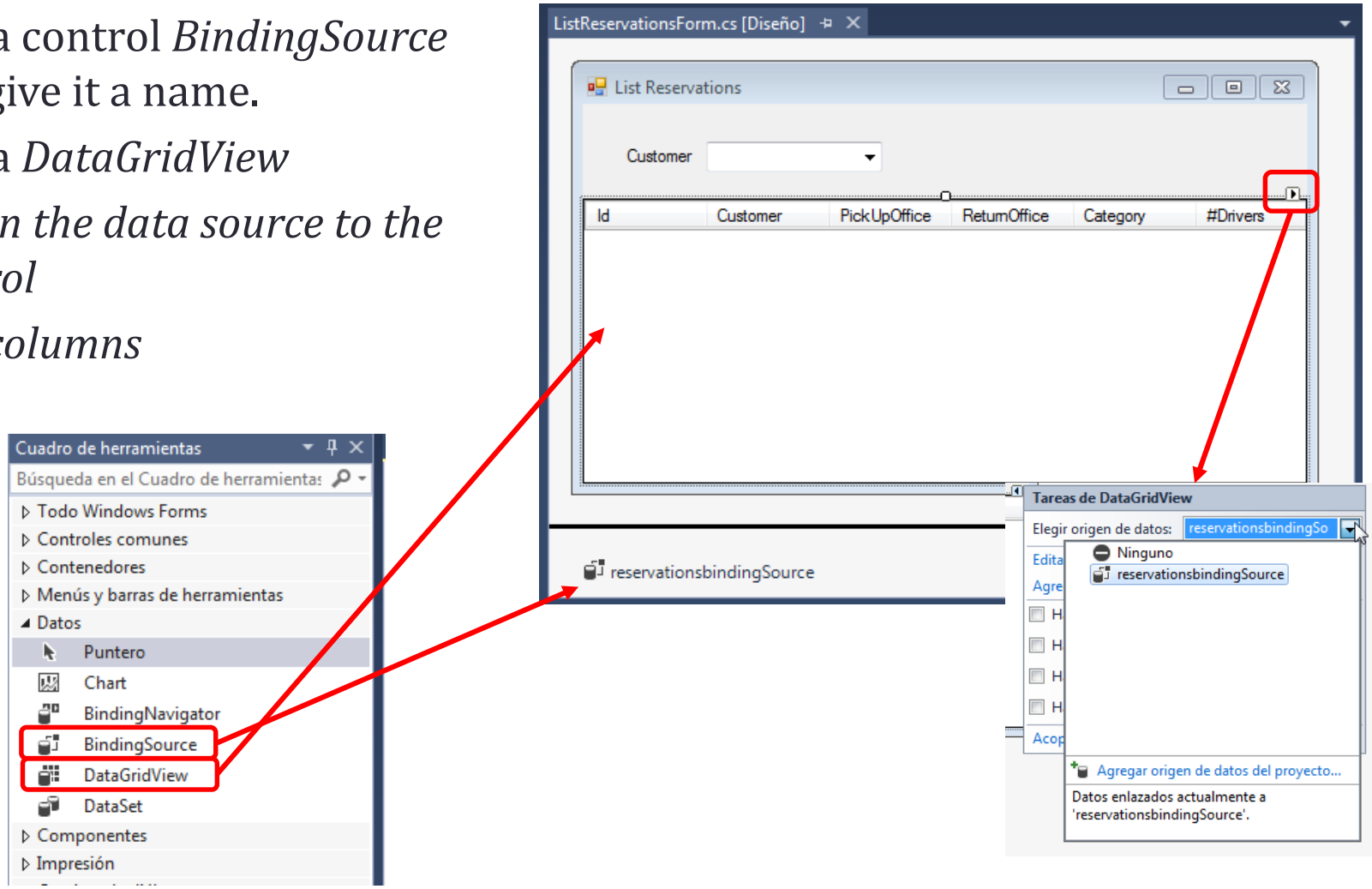


Dialog Boxes

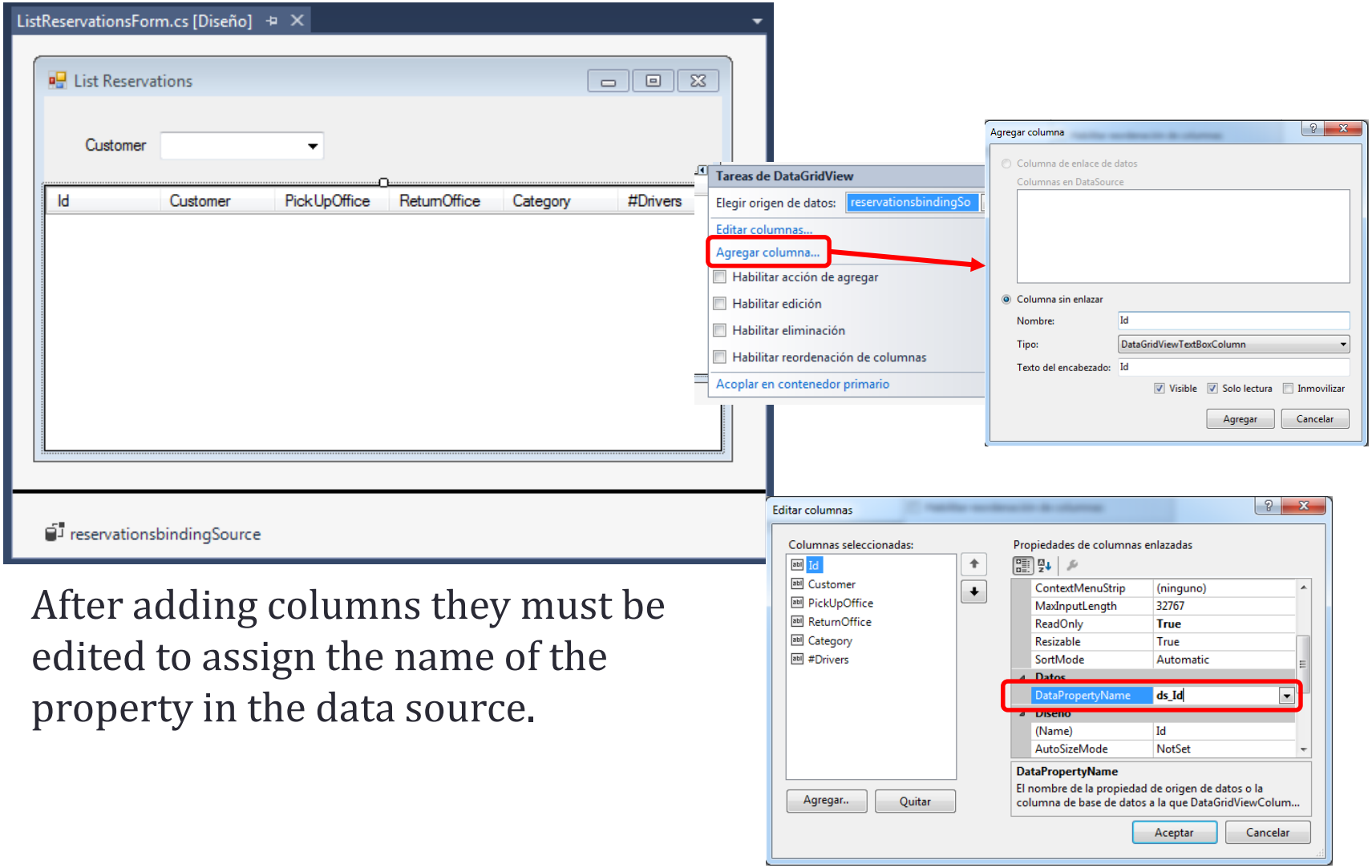
- **Standard Dialog Boxes**
  - These allow carrying out operations such as opening and storing files, printing, selecting colors, etc: ***OpenFileDialog, SaveFileDialog, FolderBrowserDialog, ColorDialog, FontDialog, PageSetupDialog*** and ***PrintDialog***.
  - Inherit from the class CommonDialog. The most imporant method is ShowDialog(), that shows the form and returns an object DialogResult :
    - DialogResult.OK if the user clicks the OK button
    - DialogResult.CANCEL otherwise.

Displaying Data Sets

1. Add a control *BindingSource* and give it a name.
2. Add a *DataGridView*
3. *Assign the data source to the control*
4. *Add columns*

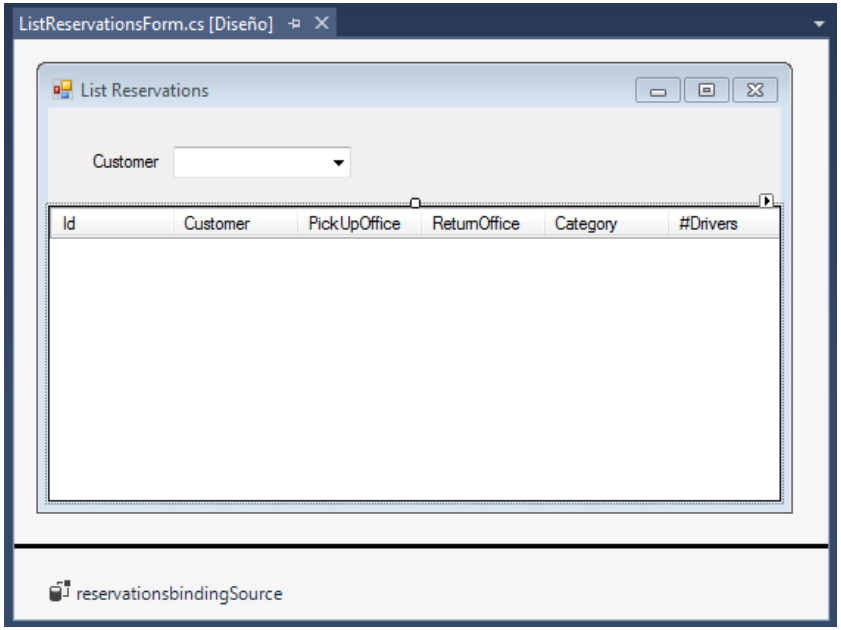


Displaying data sets



After adding columns they must be edited to assign the name of the property in the data source.

Displaying data sets



Functionality

1. When the form is shown a Customer may be selected.
2. After selecting the customer the information is displayed in the *DataGridView.*

Displaying data sets

1. When the form is created the *ComboBox* is populated.

The method *Initialize* populates the ComboBox *customersComboBox*:

```
public void Initialize()
{
    ICollection<Customer> customers;
    customersComboBox.Items.Clear();
    customers = businesscontrol.findAllCustomers();
    if (customers!=null)
        foreach (Customer c in businesscontrol.findAllCustomers())
            customersComboBox.Items.Add(c.dni);
    customersComboBox.SelectedIndex = -1;
    customersComboBox.ResetText();
    reservationsbindingSource.DataSource = null;
}
```

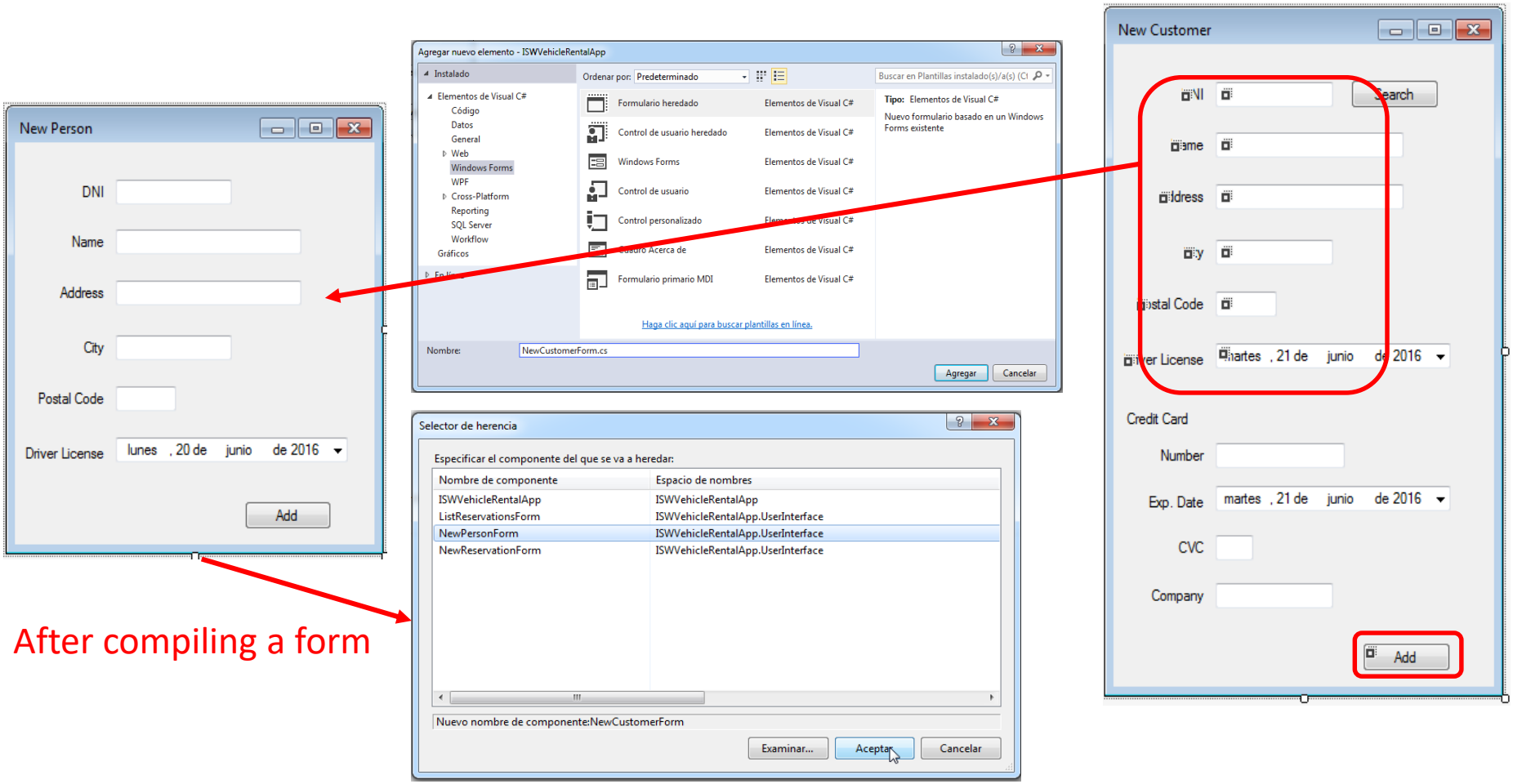
2. When an element is selected in the *ComboBox* the *DataGridView* is populated.

The event handler *SelectedIndexChanged* of the *ComboBox* object is executed.

```
private void customersComboBox_SelectedIndexChanged(object sender, EventArgs e)
{
    string dni = (string) customersComboBox.SelectedItem;
    ICollection<Reservation> reservations = businesscontrol.findReservationsbyCustomerID(dni);

    //A list of anonymous objects is created to display
    //DataGrid with the info that is needed
    BindingList<object> bindinglist = new BindingList<object>();
    foreach (Reservation r in reservations)

        //Adding one anonymous object for each reservation obtained
        bindinglist.Add(new
        {
            //ds_... are DataPropertyNames defined in the DataGridView object
            //see DataGridView column definitions in Visual Studio Designer
            ds_Id = r.Id,
            ds_Customer = r.Customer.name,
            ds_PickUpOffice = r.PickupBranchOffice.address,
            ds_ReturnOffice = r.ReturnBranchOffice.address,
            ds_Category = r.Category.name,
            ds_NumDrivers = r.Drivers.Count
        });

    reservationsbindingSource.DataSource = bindinglist;
}
```

Advanced Operations: Visual Inheritance

Forms may inherit from other forms so that the behavior and visual appeareance is reused



After compiling a form