

GRAPHICAL USER INTERFACE DESIGN

Chapter 7

Software Engineering
Computer Science School
DSIC – UPV

Goal

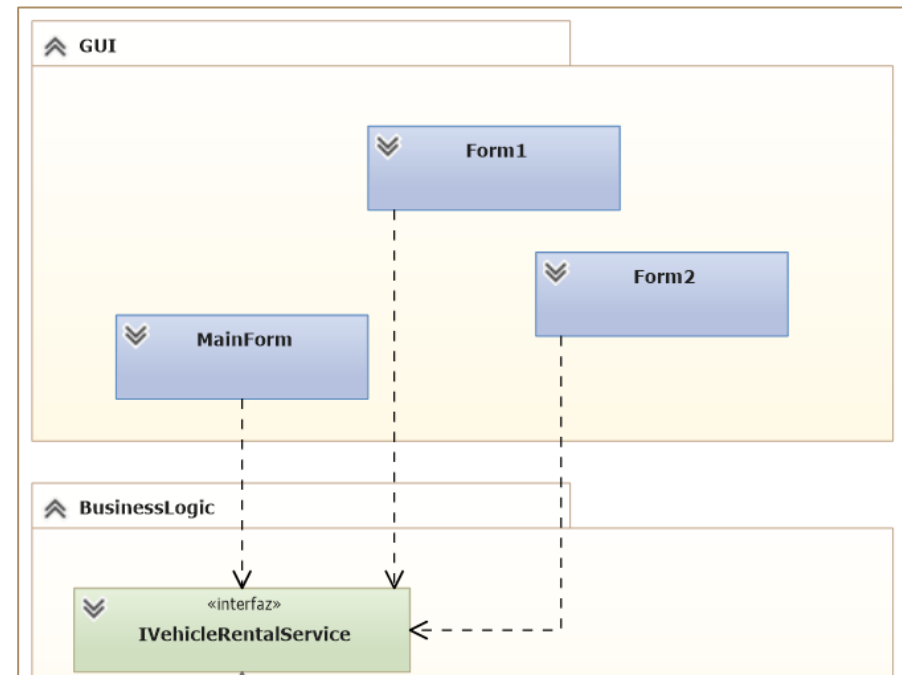
- Understand the principles of visual applications.
- Understand the design of the graphical user interface (use of controls and events).
- Understand the communication between the presentation and the business logic layers.

Contents

1. Creating a Basic Windows Application
2. Forms with controls
3. Events in forms
4. Designing and using menus
5. Apps with several forms
 1. Designed by the coder
 2. Dialog forms
6. Displaying data sets
7. Advanced operations: Visual Inheritance

Architectural Design. Presentation

- Collection of forms (one of them MainForm)
- All forms will access the services provided by the Business Logic Layer by means of an interface (e.g. `IVehicleRentalService`)
- All forms need a reference to an object of type `IVehicleRentalService`, passed as a parameter in the constructor.

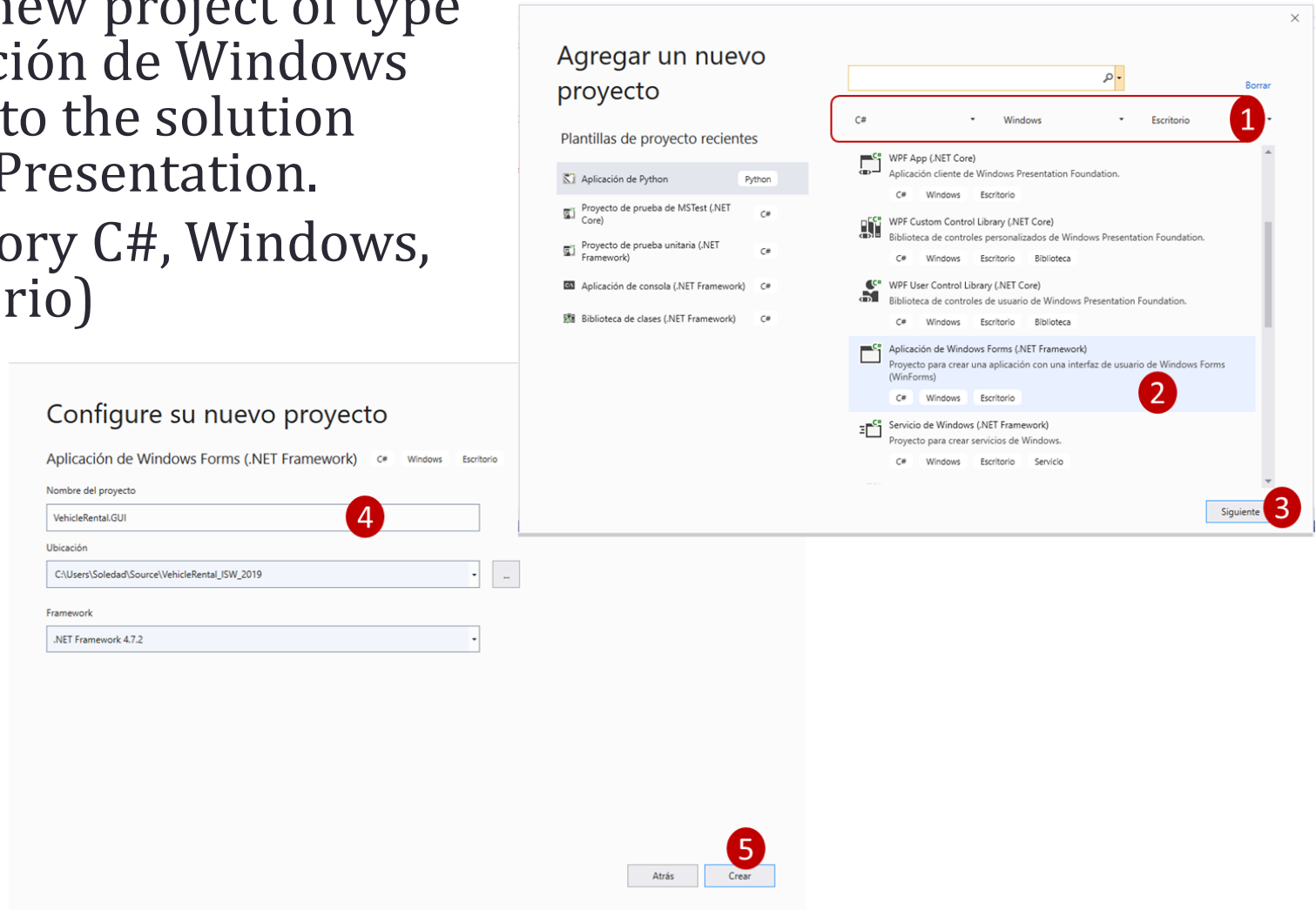


Introduction

- 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 destroyed.
 - **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.
- (category C#, Windows, Escritorio)



Creating a Windows Application

- If the app is run, a Windows with the standard basic features is created.
- The files in this Project are:

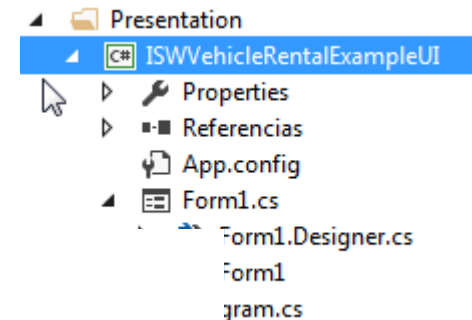
- Form1.cs: contains the design of the form. If opened the form may be modified in a visual designer.

- Form1 has constructor InitializeC

- Form1.Des generated

- Program.cs method().

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

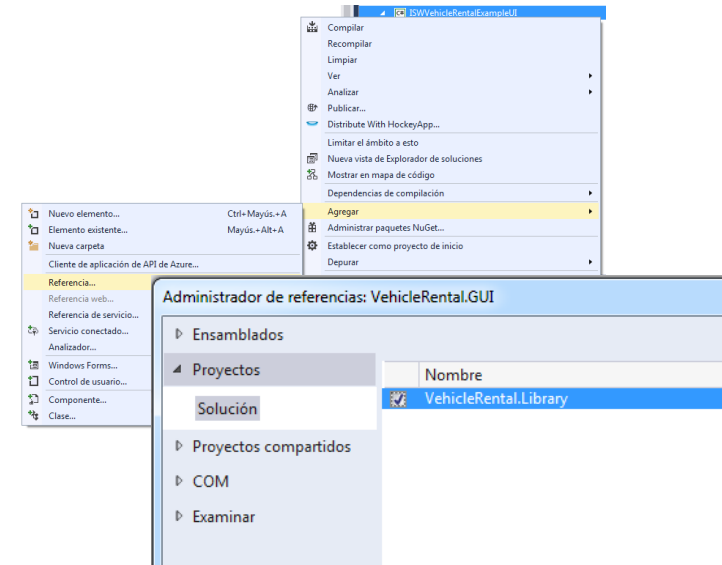


Dependencies Management

- This Project will depend on `IVehicleRentalService` and on the domain classes located at `VehicleRental.Services`. Thus, a reference has to be added

```
using VehicleRental.Services;

namespace VehicleRental.Presentation
{
    2 referencias
    public partial class VehicleRentalApps : Form
    {
        private IVehicleRentalService service;
        0 referencias
        public VehicleRentalApps(IVehicleRentalService service)
        {
            InitializeComponent();
            this.service = service;
        }
    }
}
```



Dependencies Management

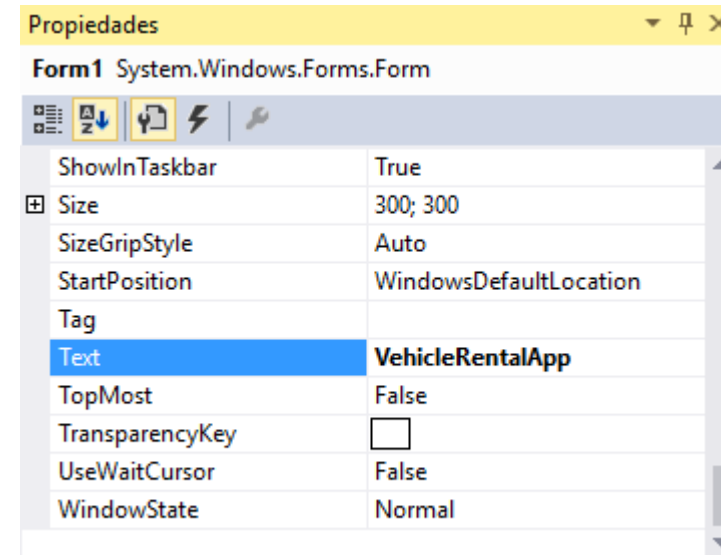
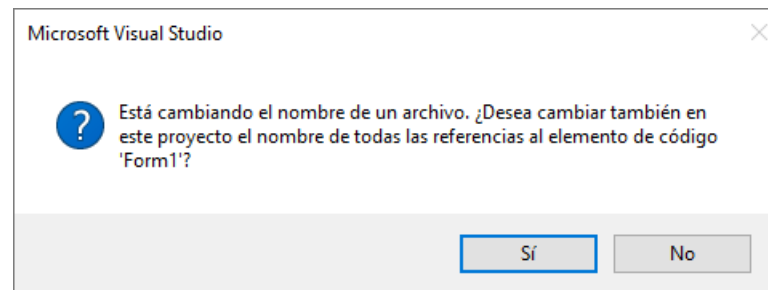
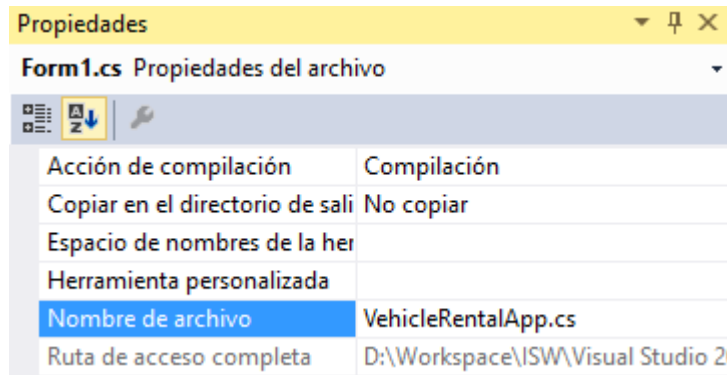
- This app will use Entity Framework, and the corresponding NuGet package must be added:

The image illustrates the steps to add the EntityFramework NuGet package to a project. It is divided into three numbered steps:

- 1**: In the Visual Studio Solution Explorer, right-click the **Referencias** (References) folder and select **Administrar paquetes NuGet...**.
- 2**: The **NuGet Package Manager** window is open, showing a list of packages. The **EntityFramework** package by Microsoft (version 6.3.0) is selected.
- 3**: The details for the **EntityFramework** package are shown. The **Versión** (Version) is set to **Versión estable más reciente 6.3.0**, and the **Instalar** (Install) button is visible.

First steps...

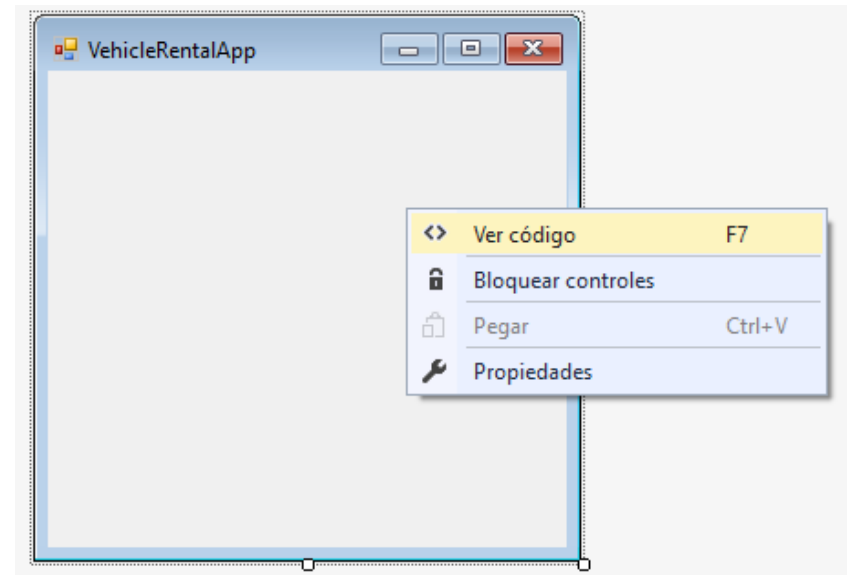
- Give an appropriate name to the elements in the Project (e.g. change the name of the file **Form1.cs** to **VehicleRentalApp**).



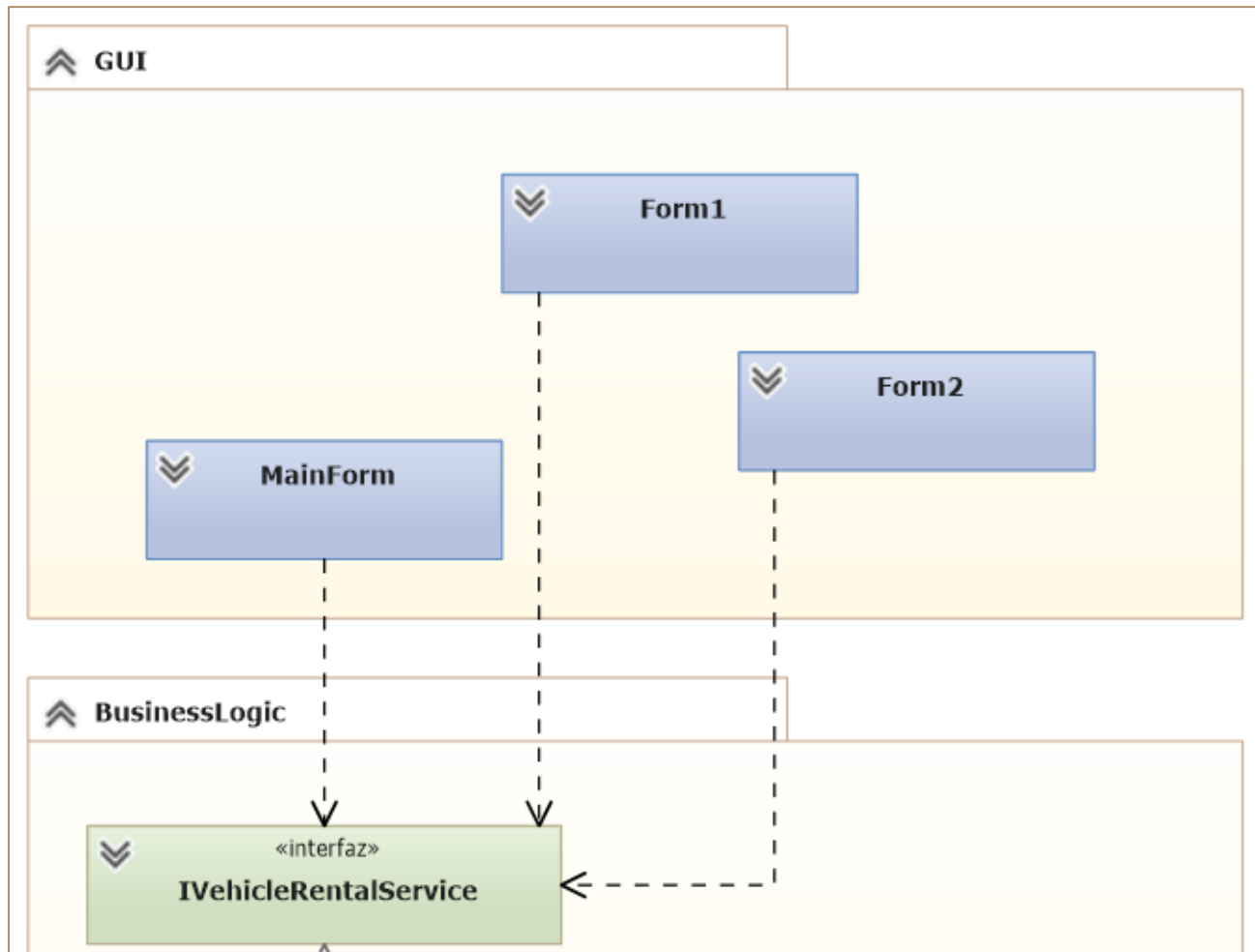
Text

Code Inspection...

- Two ways to Access C# editable code of the form:
 - Double click on
 - Select the form *right button click* > **Ver código**, or **F7**



Connect with Business Logic Layer



Connect with Business Logic Layer

Modify class **VehicleRentalApp** to have an attribute of type **IVehicleRentalService**, which is passed as a parameter in the constructor.

```
using VehicleRental.Services;

namespace VehicleRentalUI

public partial class VehicleRentalApp:Form
{
    private IVehicleRentalService service;

    public VehicleRentalApp(IVehicleRentalService service)
    {
        InitializeComponent();
        this.service = service;
    }
}
```

Connect with Business Logic Layer

Modify the Main method (Program class) to create an object **IVehicleRentalService** and pass it to the main form.

Where dbcontext and dal instantiated? In program class, in the main method

```
static void Main()
{
    // We only have 1 dbcontext object and 1 dal object
    IVehicleRentalService service = new VehicleRentalService(new
    EntityFrameworkDAL(new VehicleRentalDbContext()));

    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new VehicleRentalApp(service));
}
```

We create a dal object to provide an abstraction:
Idal implements repository pattern
The dbcontext implements the unit of work pattern

WHERE ARE WE GOING TO INSTANTIATE THE FORM? In the main

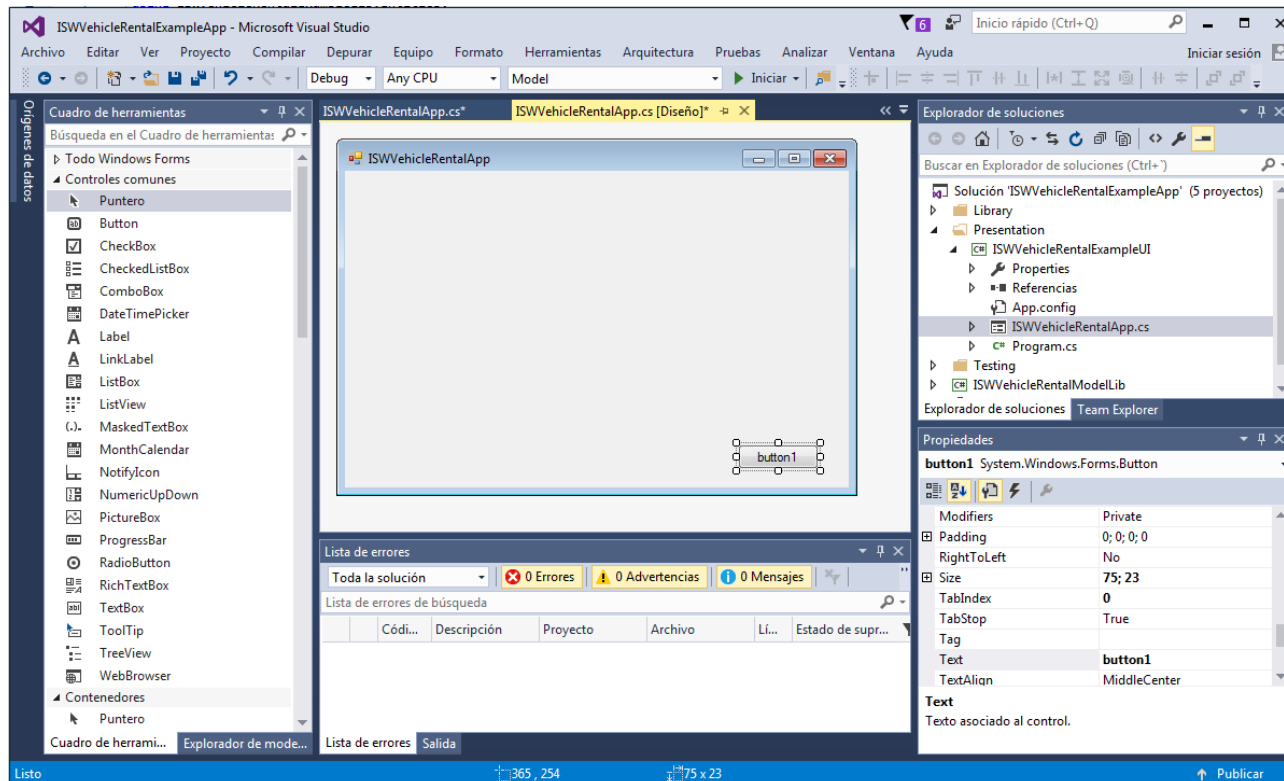
Connect with Persistence Layer

Modify App.config to add the configuration of the connection to the database:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
  </startup>
  <connectionStrings>
    <clear />
    <add name="VehicleRentalDbConnection"
          connectionString="Server=(localdb)\mssqllocaldb;Database=VehicleRentalDemo;Trusted_Connection=True;MultipleActiveResultSets=true"
          providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

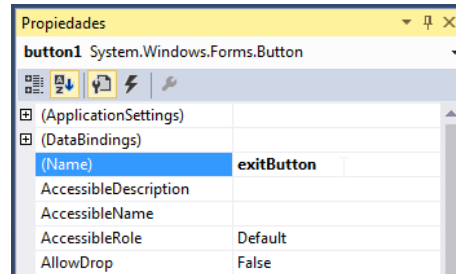
Forms with controls

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

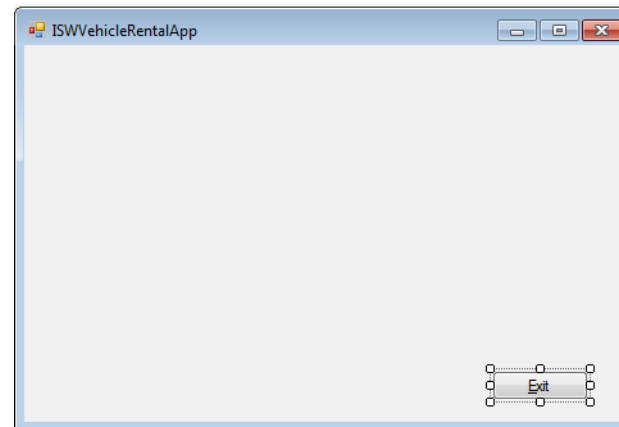
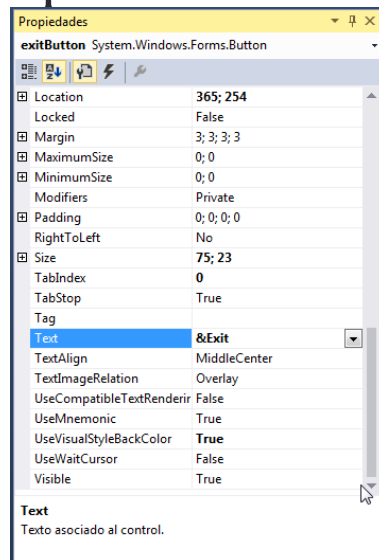


Controls: Properties

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

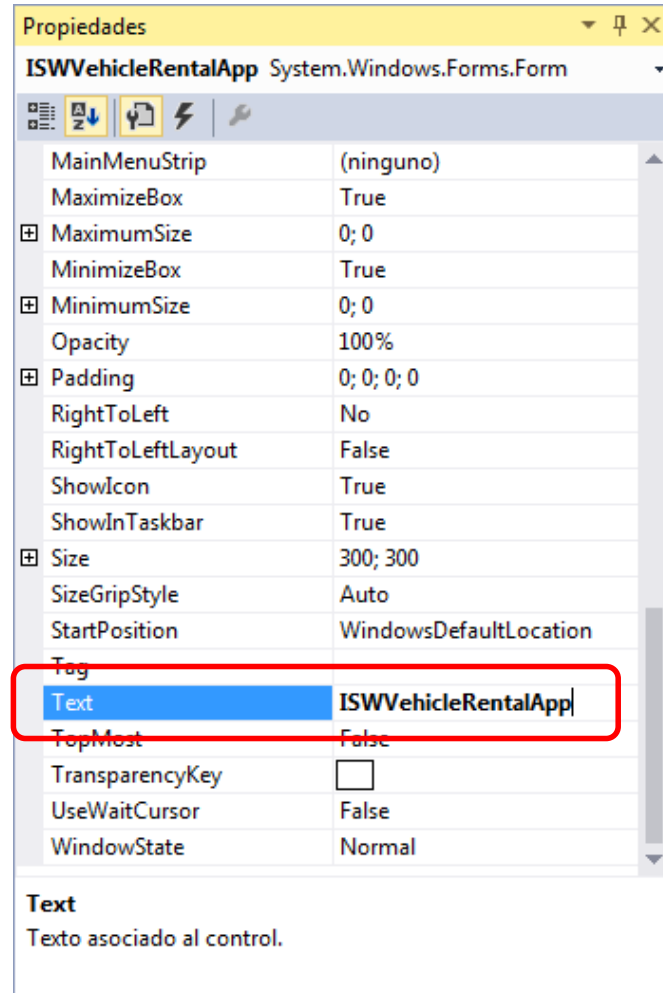
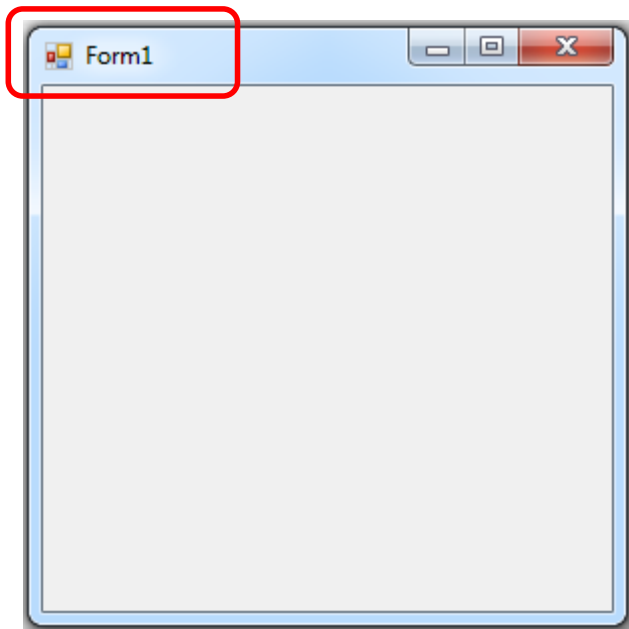


- **Text:** Represents the title of the control



First steps...

- Modify the properties of the form elements:



Events in forms

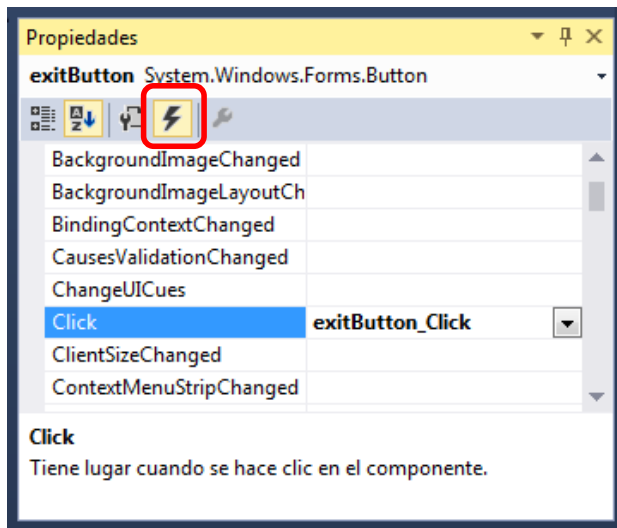
Everything is based on events:
implement handlers

- 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 handler method.
 - Selecting a handler method from the dropdown list.
 - double *click*, and Visual Studio creates a default handler definition.

In the file we dont
have tontouch is
placed the subscription



Object that raised the event

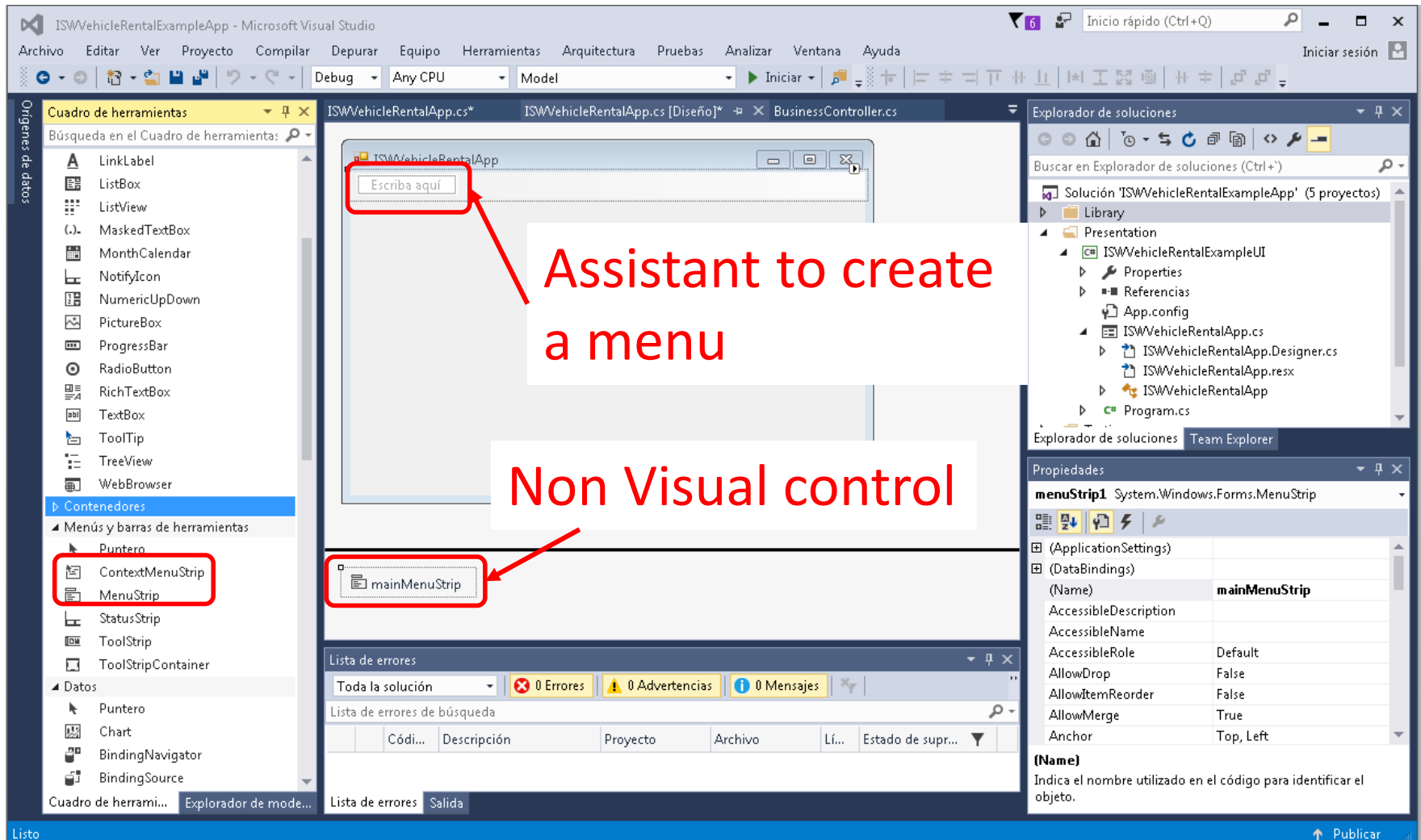
```
1 referencia
private void exitButton_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

Event information

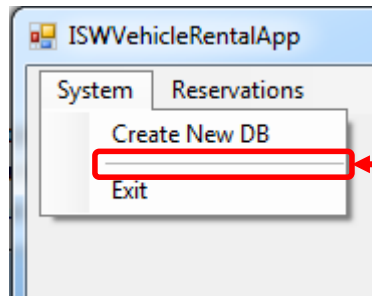
Designing and using menus

- 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.

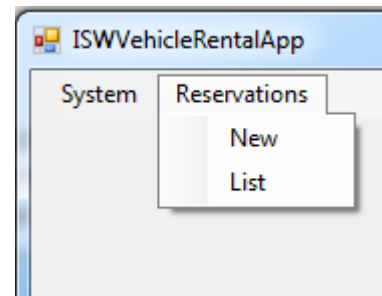
Designing and using menus



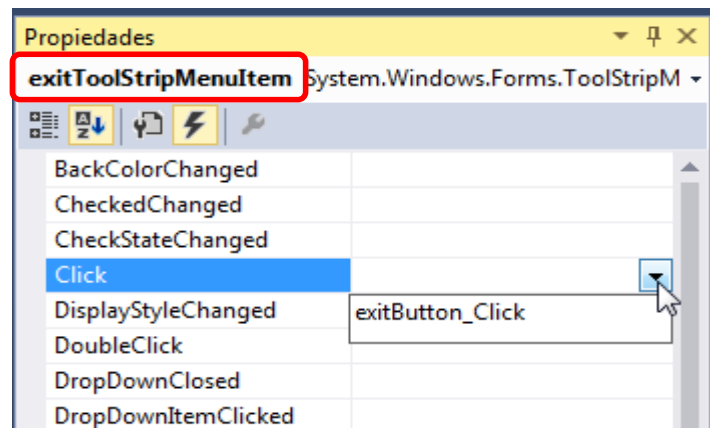
Example Menu



Text: -



- 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();
```

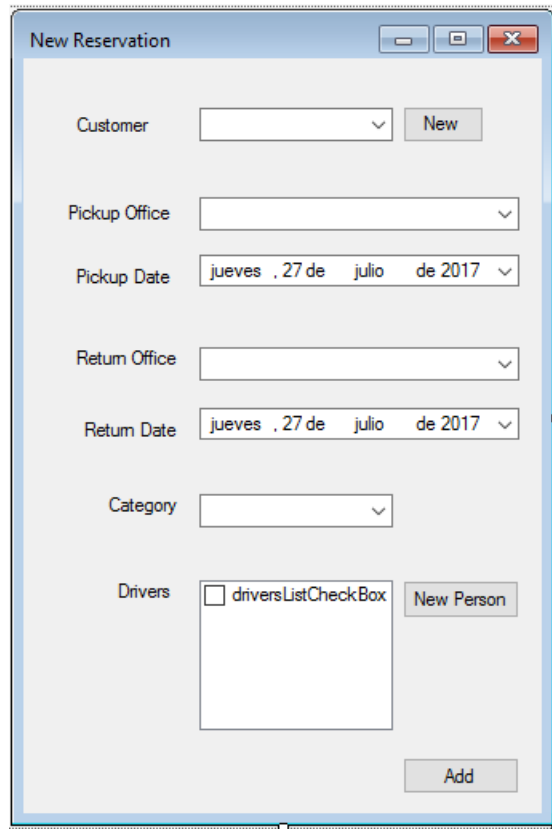
```
ExampleForm myForm = new ExampleForm();  
myForm.Show();
```

Shown in a modal way

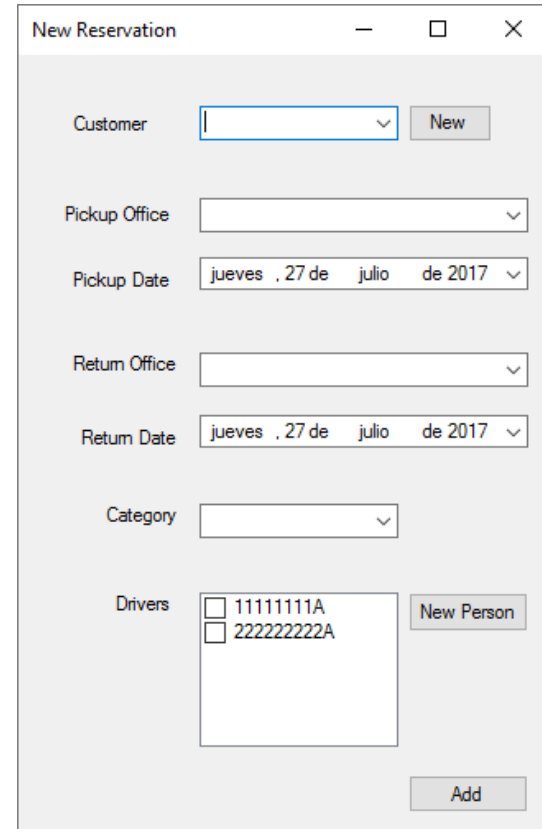
Shown in a non modal way

Forms: Example

- Design view
- Run time view



The image shows the design view of a 'New Reservation' form. The form is titled 'New Reservation' and has a standard Windows window border. It contains several input fields: 'Customer' (a dropdown menu with a 'New' button next to it), 'Pickup Office' (a dropdown menu), 'Pickup Date' (a date picker showing 'jueves , 27 de julio de 2017'), 'Return Office' (a dropdown menu), 'Return Date' (a date picker showing 'jueves , 27 de julio de 2017'), 'Category' (a dropdown menu), and 'Drivers' (a checkbox labeled 'driversListCheckBox' next to a 'New Person' button). At the bottom right, there is an 'Add' button.



The image shows the run time view of the 'New Reservation' form. The form is titled 'New Reservation' and has a standard Windows window border. It contains several input fields: 'Customer' (a dropdown menu with a 'New' button next to it), 'Pickup Office' (a dropdown menu), 'Pickup Date' (a date picker showing 'jueves , 27 de julio de 2017'), 'Return Office' (a dropdown menu), 'Return Date' (a date picker showing 'jueves , 27 de julio de 2017'), 'Category' (a dropdown menu), and 'Drivers' (a list box with two items: '11111111A' and '22222222A', each with a checkbox next to it, and a 'New Person' button). At the bottom right, there is an 'Add' button.

Forms: Example of Main Form

Every form will depend on the main form

```
public partial class VehicleRentalApp : Form
```

```
{
```

```
    private IVehicleRentalService service;
```

```
    private NewReservationForm newReservationForm;
```

```
    private ListReservationsForm listReservationForm;
```

```
    public VehicleRentalApp(IVehicleRentalService service)
```

```
{
```

```
        InitializeComponent();
```

```
        listReservationForm = new ListReservationsForm(service);
```

```
        newReservationForm = new NewReservationForm(service);
```

```
}
```

Passing parameters in constructor

```
    private void newToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
```

```
        newReservationForm.ShowDialog();
```

```
}
```

```
...
```

New form is shown "Modal"

The service was created in the main

Forms: Example of Secondary Form

```
public partial class NewReservationForm : Form
{
```

```
    private IVehicleRentalService service;
    private NewPersonForm newPersonForm;
    private NewCustomerForm newCustomerForm;
    private Customer previousCustomerAdded;
    private string previousSelectedCustomerDNI;
```

```
    public NewReservationForm(IVehicleRentalService service)
```

```
{
```

```
    InitializeComponent();
```

```
    this.service = service;
```

```
    newPersonForm = new NewPersonForm(service);
```

```
    newCustomerForm = new NewCustomerForm(service);
```

```
    LoadData();
```

```
}...
```

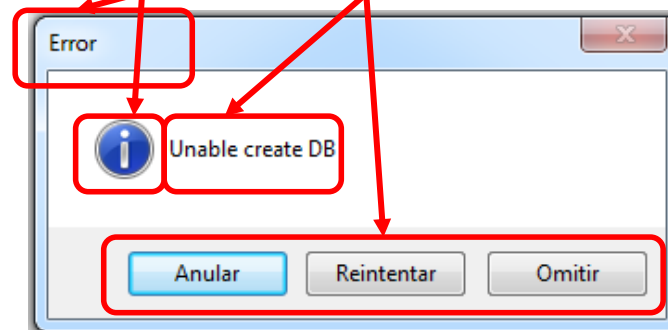
Receives parameters in constructor

Method to implement to load all data in this form

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  
{ }
```



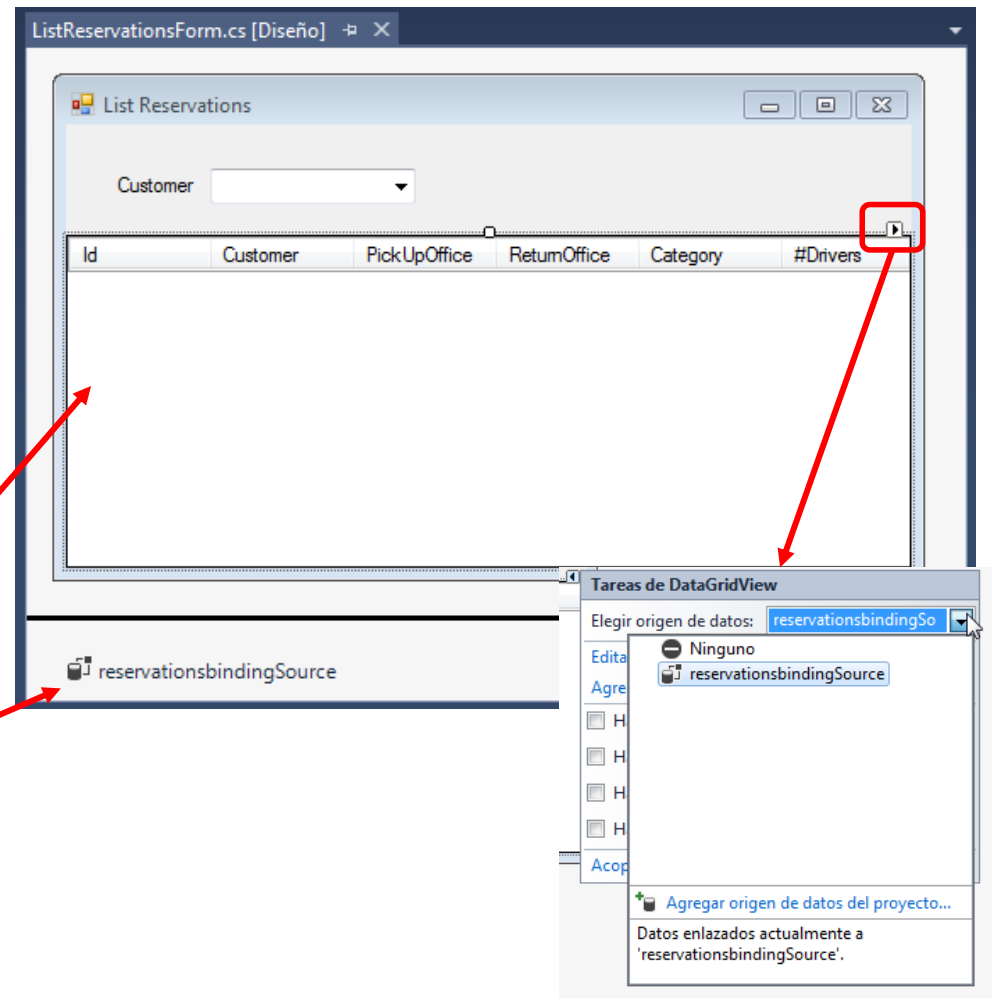
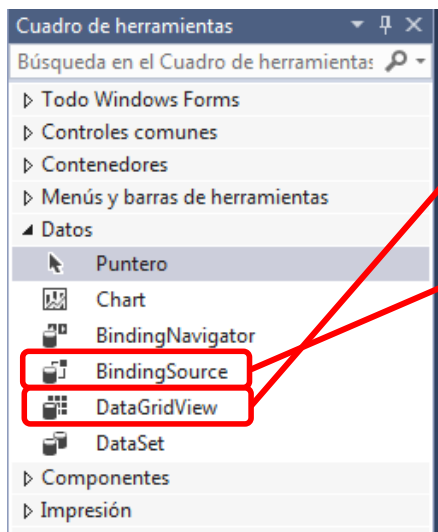
Dialog Boxes

- **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 important 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



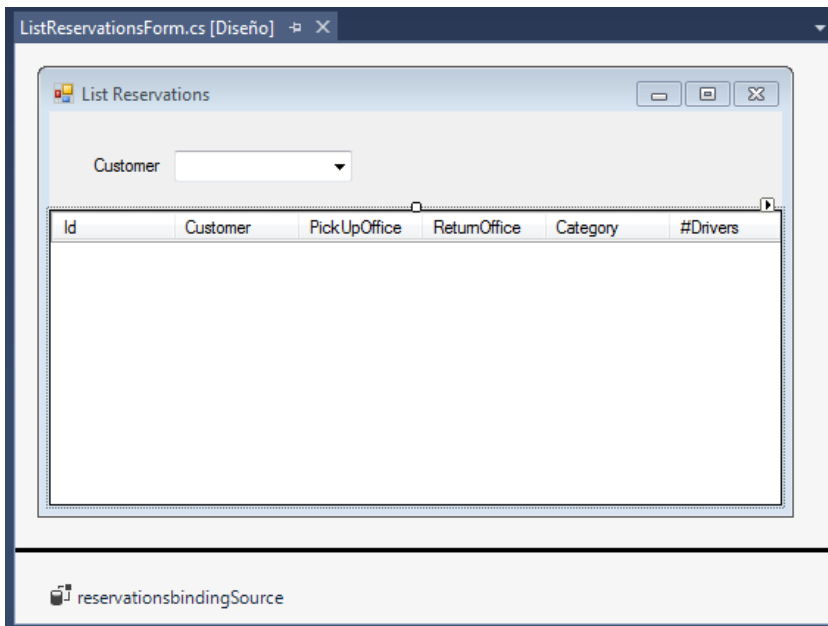
Displaying data sets

In the

The image shows a Visual Studio IDE with a Windows Form titled 'List Reservations'. The form has a 'Customer' dropdown and a 'reservationsbindingSource' data source. A context menu is open over the DataGridView, with 'Agregar columna...' highlighted. A red arrow points from this menu item to the 'Agregar columna' dialog box. This dialog shows 'Columna sin enlazar' selected, with 'Nombre' set to 'Id' and 'Tipo' set to 'DataGridViewTextBoxColumn'. Below this, the 'Editar columnas' dialog is shown. It lists selected columns: 'Id', 'Customer', 'PickUpOffice', 'ReturnOffice', 'Category', and '#Drivers'. The 'Propiedades de columnas enlazadas' section shows 'DataPropertyName' set to 'ds_Id' for the 'Id' column, which is highlighted with a red box. The 'DataPropertyName' property is described as 'El nombre de la propiedad de origen de datos o la columna de base de datos a la que DataGridViewColumn...'. Buttons for 'Agregar..', 'Quitar', 'Aceptar', and 'Cancelar' are visible at the bottom of the dialog.

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*.

The interface asks the service, and the service asks the DAL

Displaying data sets

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

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

```
public ListReservationsForm(IVehicleRentalService service) : base(service)
{
    InitializeComponent();
    LoadData();
}

public void LoadData()
{
    ICollection<string> customersDNIs = service.findAllCustomers();
    customersComboBox.Items.Clear();
    if (customersDNIs!=null)
        foreach (string Dni in customersDNIs)
            customersComboBox.Items.Add(Dni);
    customersComboBox.SelectedIndex = -1;
    customersComboBox.ResetText();
    reservationsbindingSource.DataSource = null;
}
```

Displaying data sets

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 = service.findReservationsbyCustomerID(dni);

    //A BindingList of anonymous objects is used to provide the data model to the DataGridView
```

```
    BindingList<object> bindinglist = new BindingList<object>();
    foreach (Reservation r in reservations)
```

//Adding one anonymous object for each reservation obtained

bindinglist.Add(new ? We are define anonymous objects → created at runtime

```
{
```

//ds_... are DataPropertyNames defined in the DataGridView object

//see DataGridView column definitions in Visual Studio Designer

ds_Id = r.Id, Reservation → Customer → Name

ds_Customer = r.Customer.Name, { Making work the Db Context!

ds_PickUpOffice = r.PickUpOffice.Address,

ds_ReturnOffice = r.ReturnOffice.Address,

ds_Category = r.Category.Name,

ds_NumDrivers = r.Drivers.Count

```
});
```

```
reservationsbindingSource.DataSource = bindinglist;
```

```
}
```

Binding property

↓
A column of the table

Data source

Access database
Create object
Loaded in DbSet of DbContext

We create as many anonymous objects as reservations

With this statement we populate the dataview

Advanced Operations: Visual Inheritance

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

The image illustrates the process of Visual Inheritance in Windows Forms. It shows three windows:

- New Person**: A form with fields for DNI, Name, Address, City, Postal Code, and Driver License. An **Add** button is at the bottom.
- Agregar nuevo elemento - ISWVehicleRentalApp**: A dialog box showing the process of adding a new element. The **Formulario heredado** (Inherited Form) option is selected. The **Selector de herencia** (Inheritance Selector) dialog box is also shown, listing the components to inherit from, with **NewPersonForm** selected.
- New Customer**: A form that inherits from **New Person**. It has fields for Name, Address, City, Postal Code, Driver License, and Credit Card. The **Add** button is highlighted with a red box.

Red arrows indicate the flow of the process: from the **New Person** form to the **Agregar nuevo elemento** dialog, and from the **Selector de herencia** dialog to the **New Customer** form.

After compiling a form

Visual inheritance. Reusing behaviour

All forms use `IVehicleRentalService`. Therefore, we may use a base form `VehicleRentalFormBase` with this reference and all forms will inherit from it

Every form → *Accepts service as parameter*
 ↓ *the service attribute* } *We create a parent class and let every form extend from it*

// Visual Studio no lo permite, pero VehicleRentalFormBase sería una clase abstracta

```
public partial class VehicleRentalFormBase : Form
{
    private IVehicleRentalService service; // también podría ser atributo protected

    public VehicleRentalFormBase()
    {
        InitializeComponent();
    }

    public VehicleRentalFormBase(IVehicleRentalService service) : this()
    {
        this.service = service;
    }
}
```

WATCH OUT : We need a fresh compilation of library & project to generate the form, even if the code is correct.

Visual inheritance. Reusing behaviour

For instance the VehicleRentalApp form

```
public partial class VehicleRentalApp : VehicleRentalFormBase
{
    private ListReservationsForm listReservationForm;
    private NewReservationForm newReservationForm;

    public VehicleRentalApp(IVehicleRentalService service) : base(service)
    {
        InitializeComponent();
        listReservationForm = new ListReservationsForm(service);
        newReservationForm = new NewReservationForm(service);
    }
    ...

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

Visual inheritance. Appearance reuse

New Person

DNI

Name

Address

City

Postal Code

Driver License

lunes . 20 de junio de 2016 ▾

Add

Inherits

New Customer

Customer Information

Name

Address

City

Postal Code

Driver License

Credit Card

Number

Exp. Date

CVC

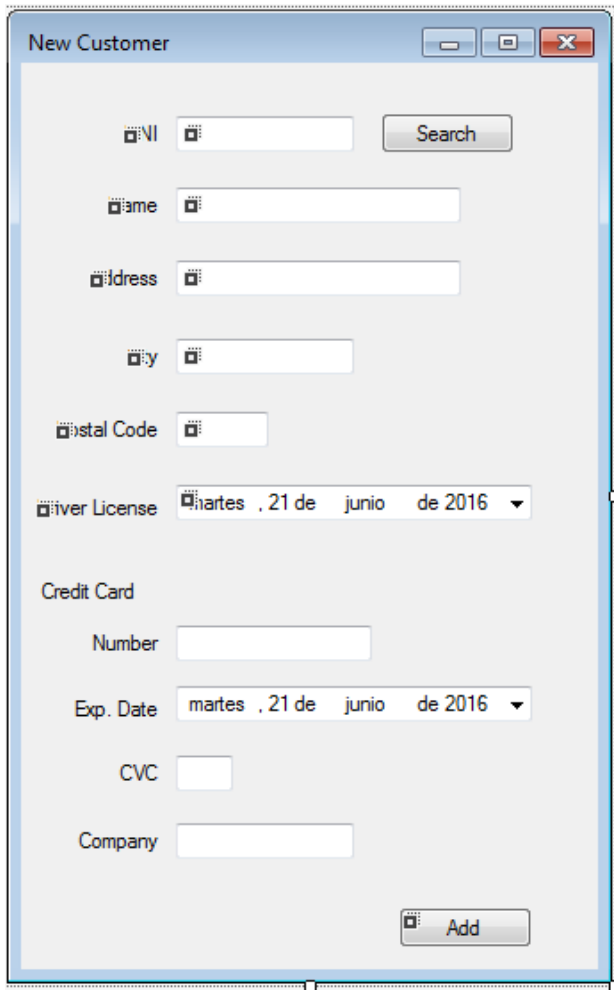
Company

Search

Add

New

Visual inheritance. Appearance reuse



The screenshot shows a Windows-style dialog box titled "New Customer". It contains several input fields with small square icons to their left: "NI" (with a search icon), "Name", "Address", "City", "Postal Code", "Driver License" (with a date dropdown showing "martes , 21 de junio de 2016"), "Credit Card Number", "Exp. Date" (with a date dropdown showing "martes , 21 de junio de 2016"), "CVC", and "Company". There is a "Search" button at the top right and an "Add" button at the bottom right.

```
public partial class NewCustomerForm : NewPersonForm
{
    public NewCustomerForm() : base()
    {
        InitializeComponent();
    }

    public NewCustomerForm(IVehicleRentalService service)
    : base(service)
    {
        InitializeComponent();
    }
}
```


Bibliography

- D. Stone, C. Jarrett, M. Woodroffe. User Interface Design and Evaluation. Morgan Kaufmann, 2005
- S. Lauesen. User Interface Design. A Software Engineering Perspective. Addison Wesley, 2005
- Shneiderman, B. y Plaisant, C. Designing the User Interface. Pearson 5th ed., 2010

Resources

- Qwindows Forms tutorials
[https://msdn.microsoft.com/es-es/library/zftbwa2b\(v=vs.110\).aspx](https://msdn.microsoft.com/es-es/library/zftbwa2b(v=vs.110).aspx)
- Tutorial 1: Create an image viewer
<https://msdn.microsoft.com/es-es/library/dd492135.aspx>