

INTERFAZ GRÁFICA CON WINDOWS FORMS

DOCENCIA VIRTUAL

Finalidad:

Prestación del servicio Público de educación superior (art. 1 LOU)

Responsable:

Universitat Politècnica de València.

Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento conforme a políticas de privacidad:

<http://www.upv.es/contenidos/DPD/>

Propiedad intelectual:

Uso exclusivo en el entorno de aula virtual.

Queda prohibida la difusión, distribución o divulgación de la grabación de las clases y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes.

La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa o civil



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Tema 7 – Seminario – Desarrollo de SW en Visual Studio

Ingeniería del Software DSIC-UPV

Objetivos

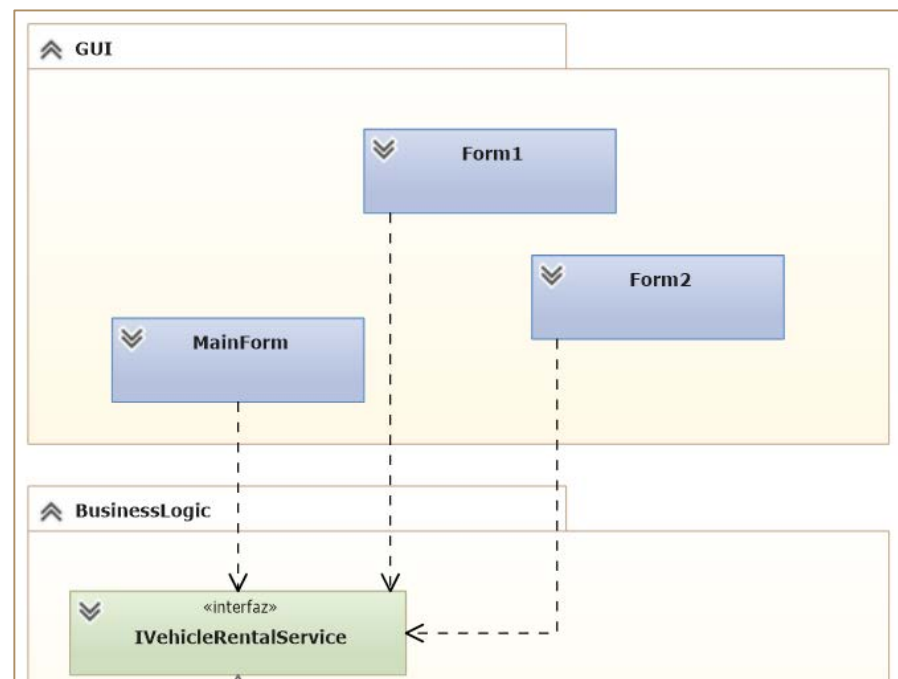
- Comprender los principios de las aplicaciones visuales.
- Comprender el diseño de la interfaz gráfica de usuario, uso de controles y eventos.
- Comprender la comunicación de la capa de presentación con la capa de negocio tal y como está concebida en el proyecto de prácticas.
- Conocer las características generales de la tecnología Winforms de Microsoft

Contenidos

1. Creación de una Aplicación Winforms Básica
2. Controles en los formularios
3. Eventos en los formularios
4. Diseño y uso de menús
5. Aplicaciones con varios formularios
 1. Diseñado por el programador
 2. Cuadros de diálogo
6. Visualización de colección de datos
7. Herencia Visual

Diseño arquitectónico. Presentación

- Conjunto de formularios (uno de ellos el MainForm)
- Todos los formularios accederán a los servicios que ofrece la lógica de negocio a través de una fachada, la interfaz `IVehicleRentalService`
- Por lo tanto, los formularios necesitan una referencia a un objeto de tipo `IVehicleRentalService`, que se puede pasar vía constructor.



Introducción a WinForms

- Espacio de nombre `System.Windows.Forms` ofrece componentes para el desarrollo de aplicaciones de escritorio visuales (basadas en ventanas)
 - `Application`: Es el corazón de una aplicación Windows. Utilizando sus métodos se procesan los mensajes de Windows y se crean y destruyen las aplicaciones visuales.
 - `Form`: Representa a una ventana o cuadro de diálogo de una aplicación visual.
 - `Button`, `ListBox`, `TextBox`, `PictureBox`, `Label`,...: los controles habituales de Windows.
 - `StatusBar`, `ToolBar`,...: Barras de estado, de tareas, etc.
 - `ColorDialog`, `FileDialog`,...: Cuadros de diálogo estándar.
 - `StripMenu`, `StripMenuItem`,...: Menús
 - `ToolTip`, `Timer`,...: Utilidades varias

Creación de una Aplicación Windows

- Agregar proyecto de tipo **Aplicación de Windows Forms .NET Framework** (categoría C#, Windows, Escritorio)

The image shows two screenshots from Visual Studio illustrating the steps to create a new Windows Forms application. Red circles with numbers 1 through 5 highlight specific elements.

Top Screenshot: "Agregar un nuevo proyecto"

- 1**: The "C#" language filter is selected in the top right.
- 2**: The "Aplicación de Windows Forms (.NET Framework)" template is selected in the list.
- 3**: The "Siguiente" (Next) button is highlighted at the bottom right.

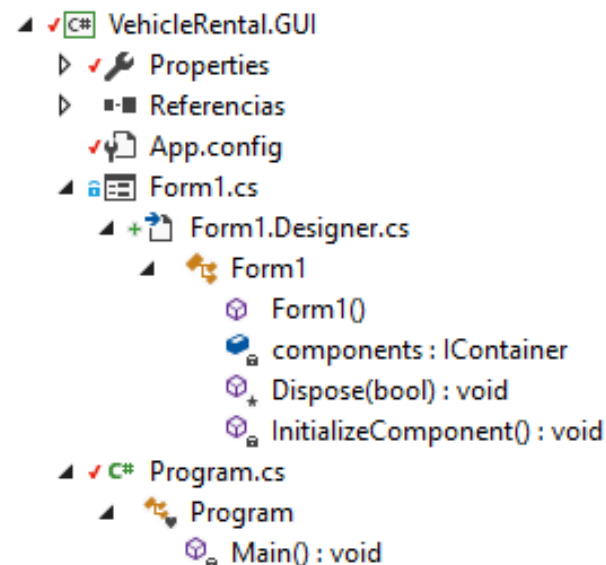
Bottom Screenshot: "Configure su nuevo proyecto"

- 4**: The "Nombre del proyecto" (Project Name) field is highlighted, containing the text "VehicleRental.GUI".
- 5**: The "Crear" (Create) button is highlighted at the bottom right.

Creación de una Aplicación Windows

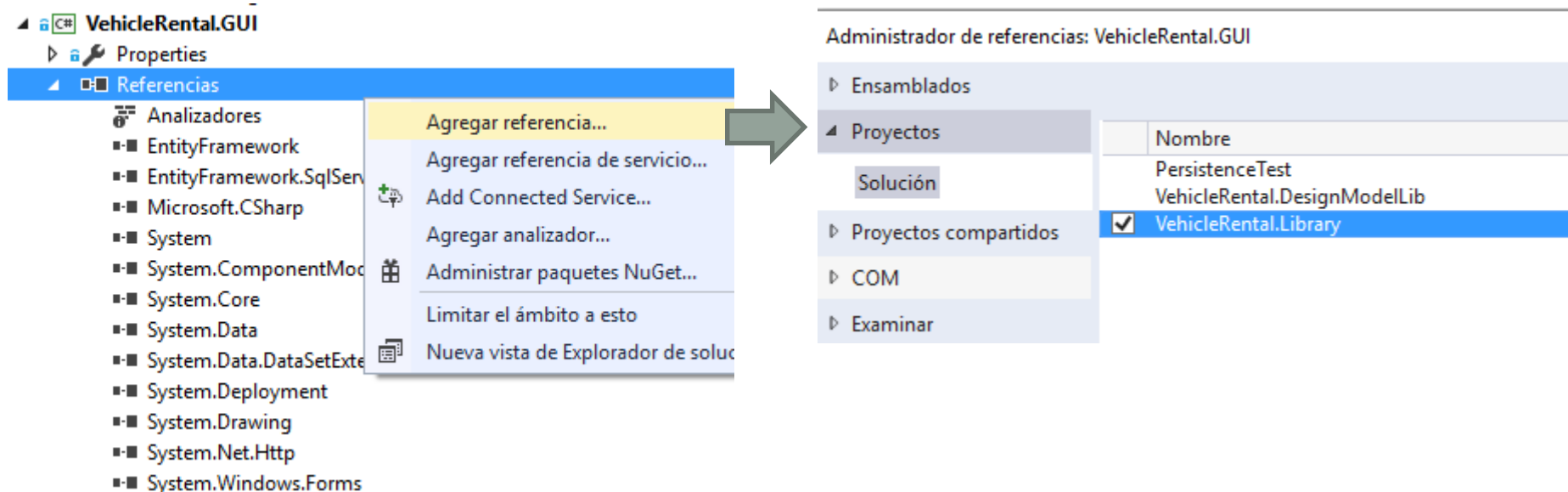
- Los ficheros que forman parte de este proyecto son:
 - `Form1.cs`: diseño visual del formulario, permite añadir/eliminar/editar controles y cambiar su apariencia
 - `Form1.Designer.cs` contiene la **definición parcial** de la clase `Form1`, generada automáticamente por el diseñador (contiene métodos `Dispose` e `InitializeComponent`). No debe ser modificada manualmente.
 - `Program.cs`: contiene la definición del método `Main()`.

```
static class Program
{
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
```



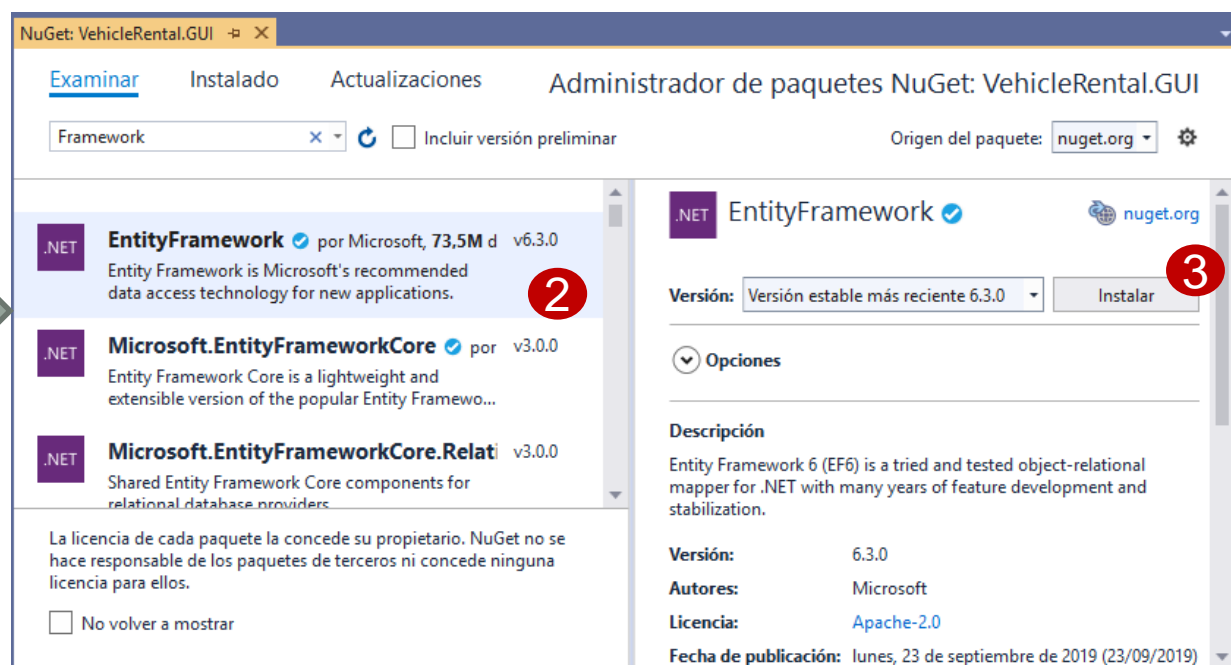
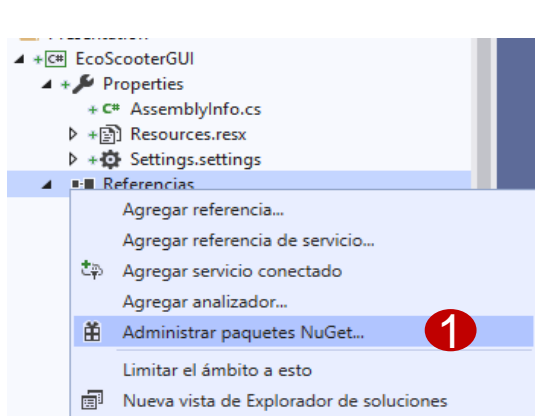
Gestión de dependencias

- Este proyecto dependerá de `IVehicleRentalService` y las clases del dominio, ubicadas en el proyecto `VehicleRental.Library`, por lo que hay que agregar una referencia.
- Desplegar proyecto y en la sección Referencias, click derecho para abrir menú contextual, pulsar Agregar referencia.
- En el Administrador de referencias marcar `VehicleRental.Library`



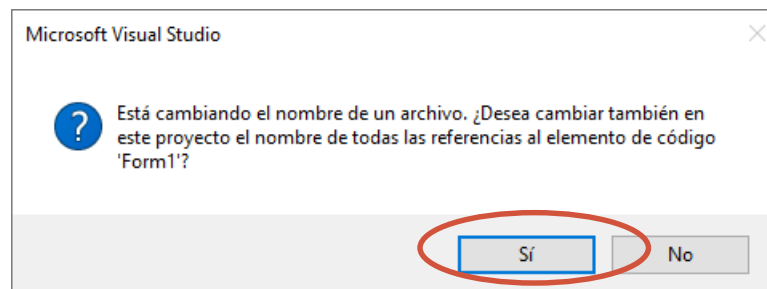
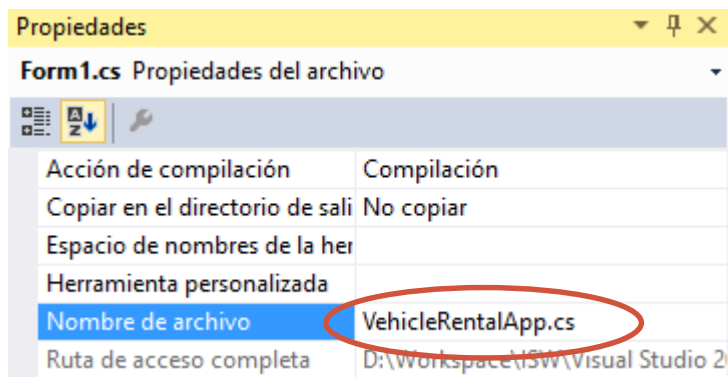
Gestión de dependencias

- Además, hará uso de Entity Framework, por lo que hay que añadir el paquete NuGet correspondiente:
 - Desplegar proyecto y en la sección Referencias, click derecho para abrir menú contextual, pulsar Administrar Paquetes NuGet
 - Buscar Entity Framework y pulsar instalar

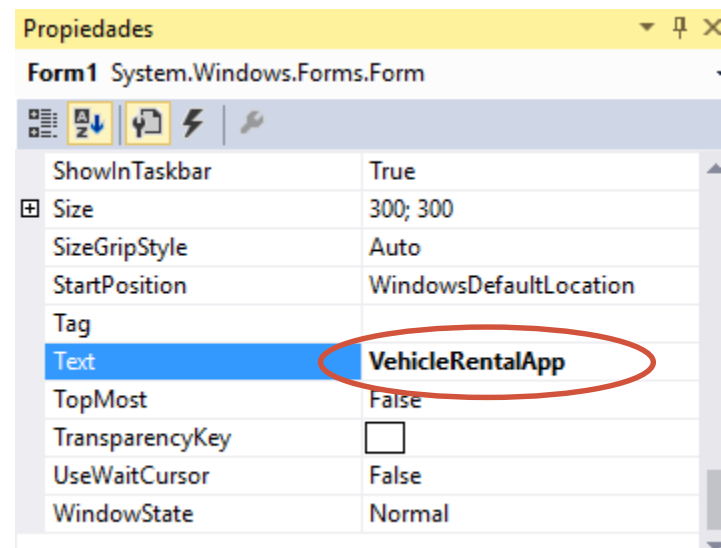


Renombrar formulario y darle título


- En propiedades del archivo Form1.cs cambiar su nombre (será el nombre de esa clase formulario)



- En propiedades del formulario cambiar la propiedad Text, que será el título de la ventana



Abrir código

- Acceder al código C# editable del formulario, hay dos formas:
 - Hacer doble click sobre  VehicleRentalApp
 - Seleccionar el formulario en el diseñador o en el explorador de soluciones y hacer *click derecho* > **Ver código**, o pulsar tecla **F7**



Conectar con capa de negocio

Modificar la clase **VehicleRentalApp** para que tenga un atributo de tipo **IVehicleRentalService**, el cual habrá que pasar como parámetro en el constructor, o mediante un método.

```
using VehicleRental.Services;

namespace VehicleRentalUI

public partial class VehicleRentalApp
{
    private IVehicleRentalService service; // también podría ser protected

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

Conectar con capa de negocio...

Modificar el método Main (en la clase Program) para crear el objeto de tipo **IVehicleRentalService** y pasarlo al formulario principal.

```
static void Main()
{
    IVehicleRentalService service = new VehicleRentalService(new
    EntityFrameworkDAL(new VehicleRentalDbContext()));

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

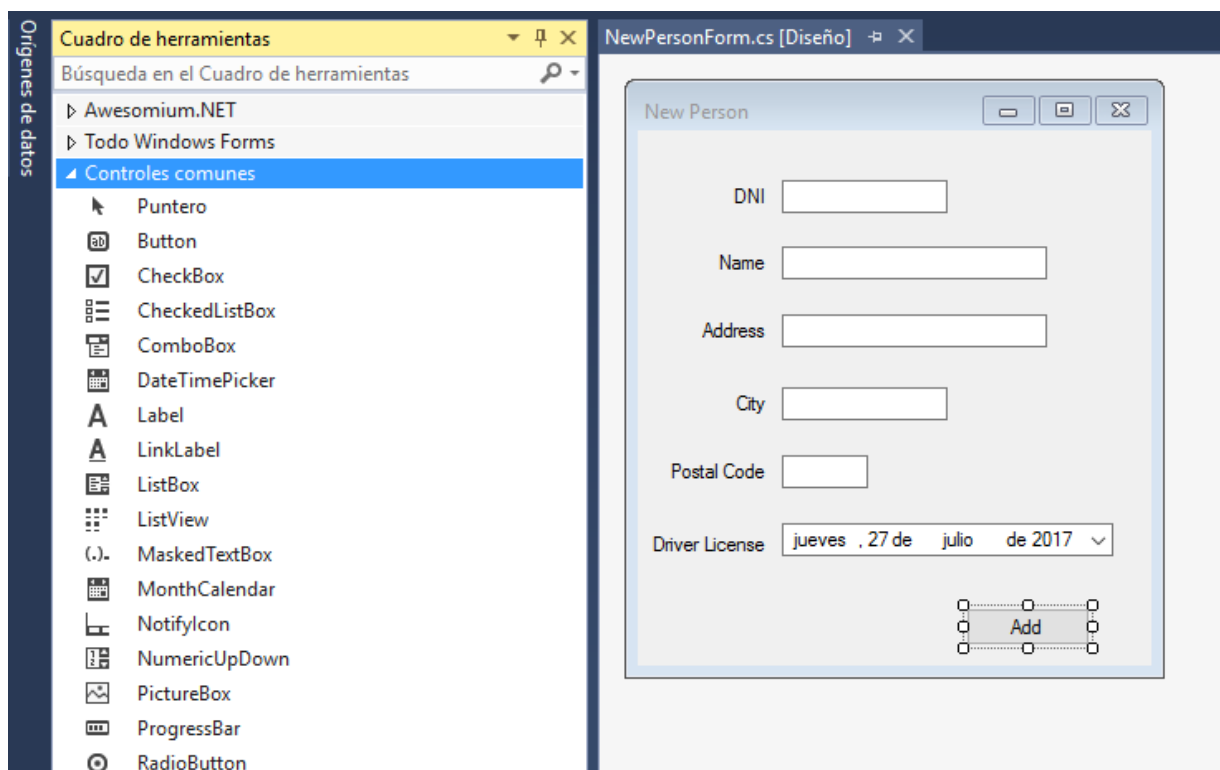
Conectar con capa de persistencia...

Modificar App.config para añadir la configuración de conexión a la base de datos que crea la capa de persistencia:

```
<?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>
```

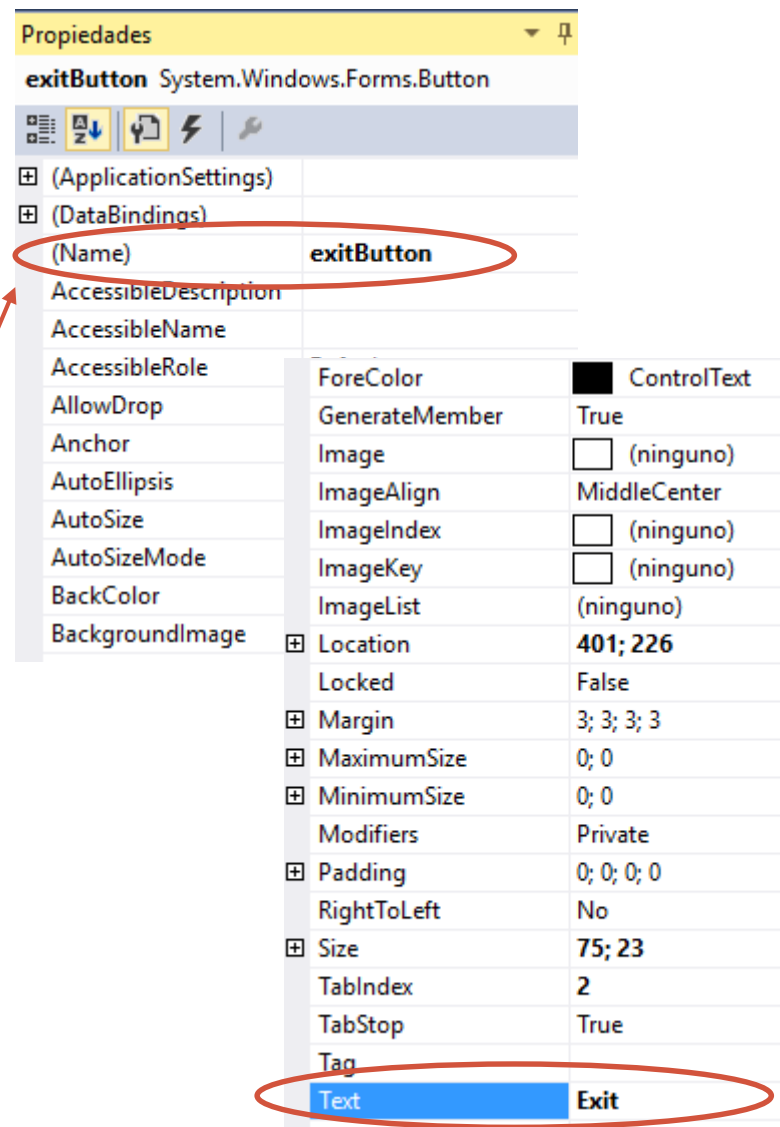
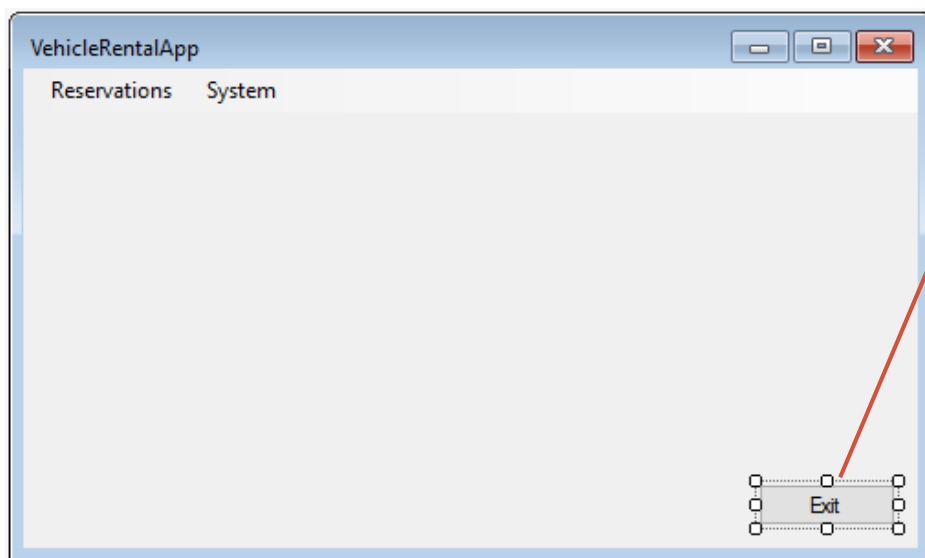
Controles en los formularios

- Son objetos de la clase Control: botones, cuadros de texto, botones de selección, ...
- Se pueden añadir en **tiempo de diseño** (mediante el editor y la paleta de componentes) o en **tiempo de ejecución**.



Controles: Propiedades

- **Name:** Representa el nombre del control. Es muy conveniente usar nombres significativos
- **Text:** Texto del control

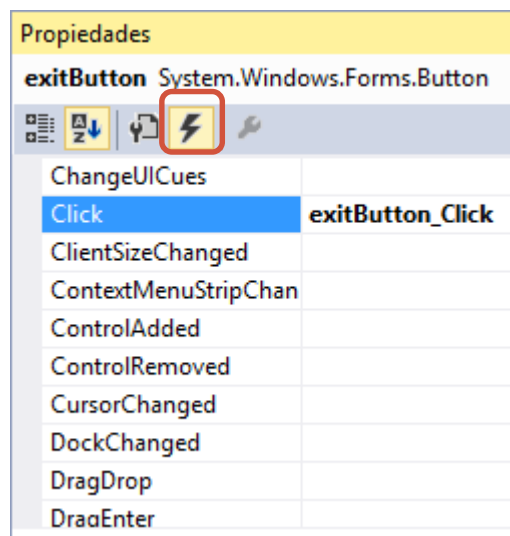


Eventos en los formularios

- Un **evento** es una acción a la cual se puede responder desde el código.
- Los eventos pueden ser **generados** por:
 - Una acción del usuario (pulsar una tecla, un botón del ratón, etc.)
 - El código del programa.
 - El sistema operativo.
- Las aplicaciones Windows suele ser aplicaciones **controladas** por eventos:
 - La ocurrencia de un evento provoca la ejecución de código.
 - Esto se conoce como respuesta a un evento.
 - El código debe estar en métodos “especiales” denominados “**manejadores de eventos**”.
- Todo control expone un conjunto de eventos a los cuales el programador puede asociar un manejador/controlador

Eventos: manejadores

- Cuando ocurre un **evento** y su método **manejador** tiene código, se **ejecuta** dicho código. En caso contrario no ocurre nada.
- Los eventos publicados aparecen en la ventana de propiedades.
- Podemos asociar un manejador a un evento de las siguientes formas:
 - Escribir el nombre del método manejador.
 - Seleccionar un método manejador del desplegable.
 - Hacer doble *click*, con lo que el entorno nos creará la declaración del manejador por defecto.



Objeto generador
del evento

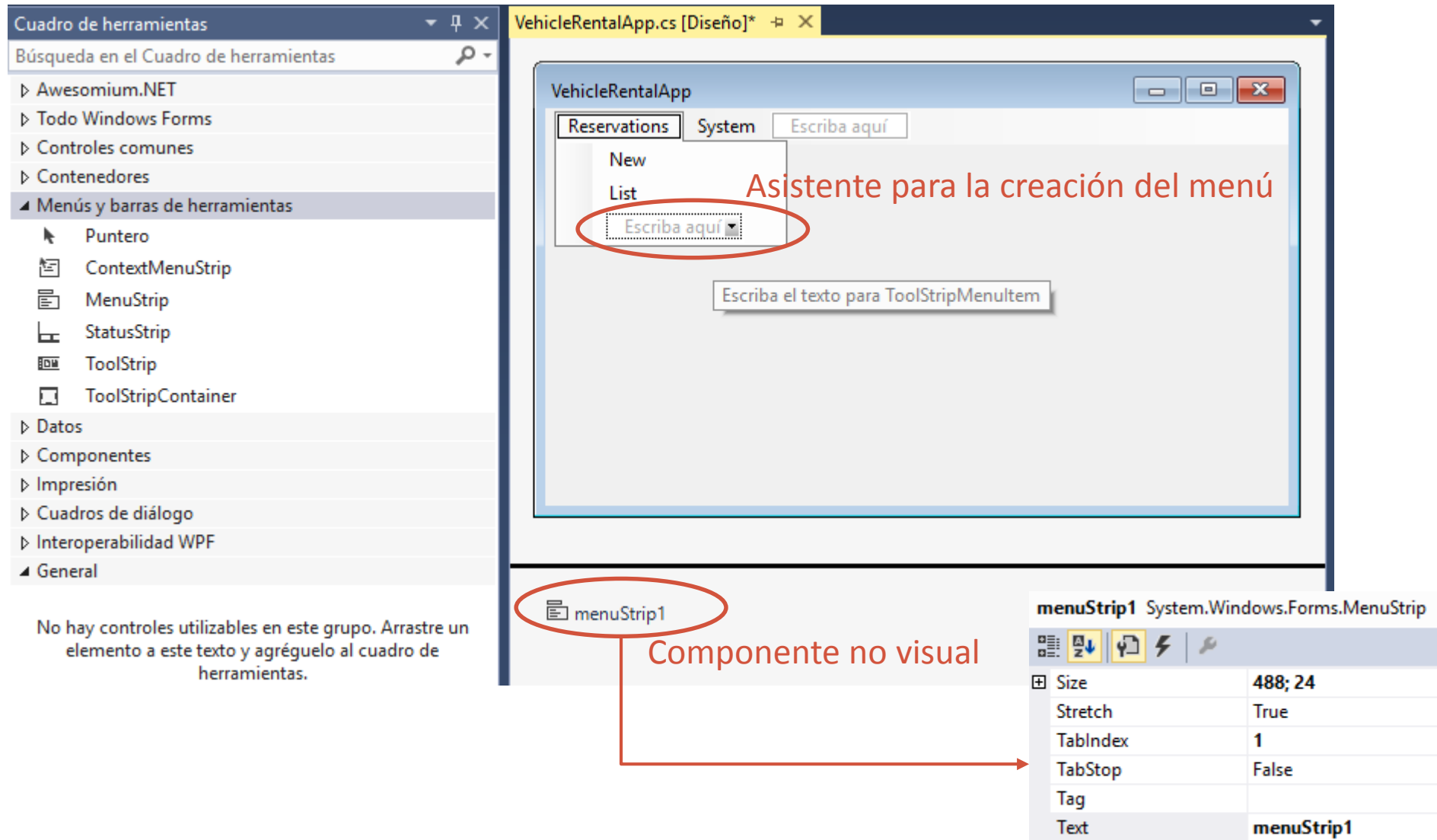
```
private void exitButton_Click(object sender, EventArgs e)  
{  
    Application.Exit();  
}
```

Información
del evento

Diseño y uso de menús

- La mayoría de las aplicaciones Windows poseen menús en sus formularios.
- Hay dos tipos de menús:
 - `MenuStrip`: Es un menú principal.
 - `ContextMenuStrip`: Es un menú contextual.
- Todos los elementos de un menú se almacenan en la propiedad `Item` que es una colección de objetos de la clase `ToolStripMenuItem`. Estos elementos a su vez pueden contener otros submenús.

Diseño y uso de menús



Cuadro de herramientas

Búsqueda en el Cuadro de herramientas

- Awesomium.NET
- Todo Windows Forms
- Controles comunes
- Contenedores
- Menús y barras de herramientas
 - Puntero
 - ContextMenuStrip
 - MenuStrip
 - StatusStrip
 - ToolStrip
 - ToolStripContainer
- Datos
- Componentes
- Impresión
- Cuadros de diálogo
- Interoperabilidad WPF
- General

No hay controles utilizables en este grupo. Arrastre un elemento a este texto y agréguelo al cuadro de herramientas.

VehicleRentalApp.cs [Diseño]*

VehicleRentalApp

Reservations System Escriba aquí

New List Escriba aquí

Asistente para la creación del menú

Escriba el texto para ToolStripMenuItem

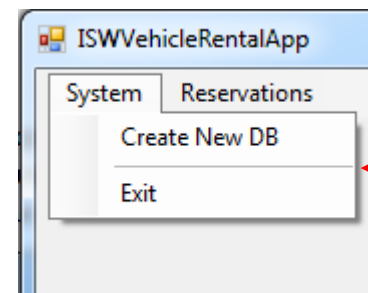
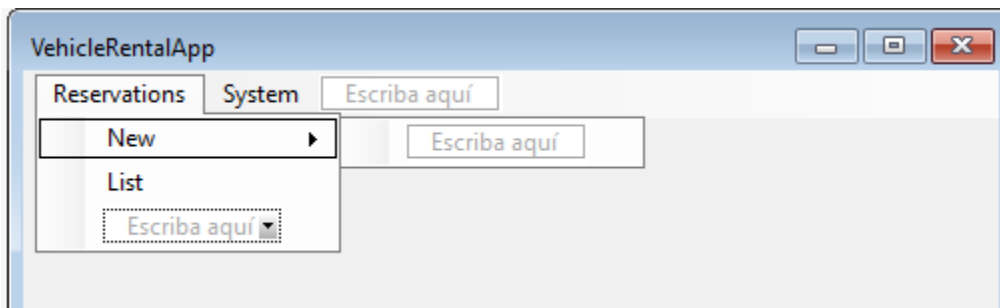
menuStrip1

Componente no visual

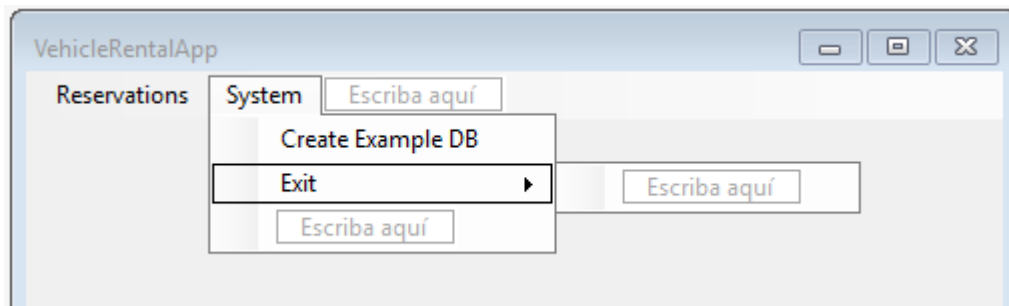
menuStrip1 System.Windows.Forms.MenuStrip

Size	488; 24
Stretch	True
TabIndex	1
TabStop	False
Tag	
Text	menuStrip1

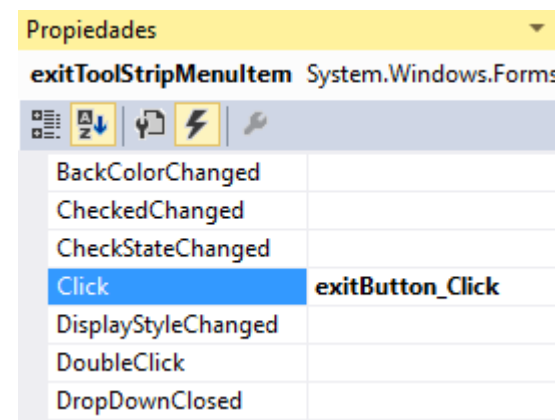
Ejemplo menú



Text: -



Nota: Podemos reutilizar manejadores (métodos) con diferentes controles. Ejemplo: Item de menú Exit y botón Exit



Aplicaciones con varios formularios

- Usualmente en las aplicaciones Windows es necesario utilizar más de un formulario.
- El aspecto predefinido de un formulario común lo define la propiedad `FormBorderStyle`.
- Podemos encontrar formularios:
 - Diseñados por el programador: se añaden con Proyecto|Agregar Windows Forms.
 - Predefinidos por el entorno: cuadros de diálogo.

Modalidad

- Un formulario se puede visualizar de formas
 - **Modal:** Debe ser cerrado para continuar trabajando con la aplicación. Se muestra mediante el método `ShowDialog()`.

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

- **No Modales:** Permiten trabajar con varios formularios de la aplicación sin necesidad de cerrarlos. Se muestran mediante el método `Show()`.

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

Ejemplo de formulario principal

```
public partial class VehicleRentalApp : Form
{
    private IVehicleRentalService service;
    private NewReservationForm newReservationForm;    // podría ser var. local
    private ListReservationsForm listReservationForm; // podría ser var. local

    public VehicleRentalApp(IVehicleRentalService service)
    {
        InitializeComponent();
        listReservationForm = new ListReservationsForm(service);
        newReservationForm = new NewReservationForm(service);

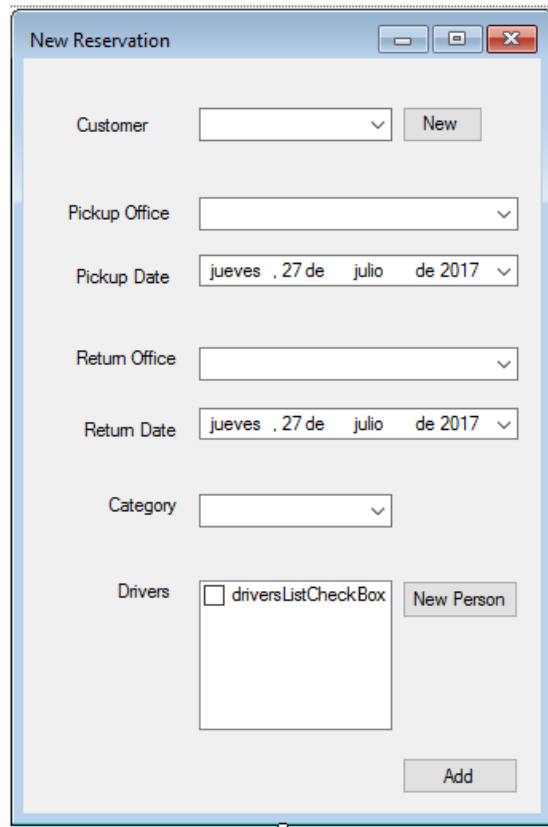
        Paso de parámetros en el constructor.
        Le pasamos el servicio

        private void newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            newReservationForm.ShowDialog();

            Mostrará el nuevo formulario en modo "Modal"
        }
    }
}
```


Ejemplo de formulario *no principal*

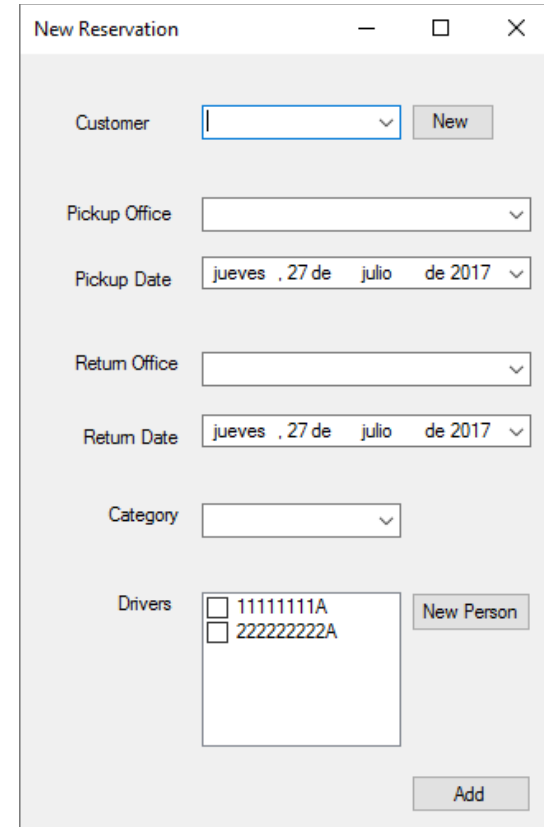
Vista de diseño



The design view of the 'New Reservation' form shows a light blue title bar with standard window controls. The form layout includes:

- Customer:** A text box with a dropdown arrow and a 'New' button.
- Pickup Office:** A text box with a dropdown arrow.
- Pickup Date:** A date picker showing 'jueves , 27 de julio de 2017'.
- Return Office:** A text box with a dropdown arrow.
- Return Date:** A date picker showing 'jueves , 27 de julio de 2017'.
- Category:** A text box with a dropdown arrow.
- Drivers:** A section with a checkbox labeled 'driversListCheckBox', a 'New Person' button, and a list box.
- Add:** A button at the bottom right.

Vista en ejecución



The run view of the 'New Reservation' form shows a light gray title bar with standard window controls. The form layout is identical to the design view, but with the following differences:

- Customer:** The text box is active with a blue border.
- Drivers:** The list box contains two entries: '11111111A' and '22222222A', each with a checkbox.
- Add:** The button is now visible at the bottom right.

Ejemplo de formulario *no principal*

```
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();
```

```
    }...
```

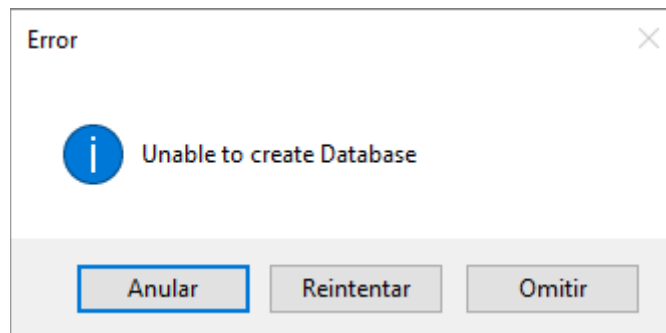
Recibe parámetros en el constructor

Método a **implementar** para cargar los datos en este formulario

Cuadros de diálogo simples: MessageBox

- La clase `MessageBox` nos proporciona cuadros de diálogo simples y de comportamiento modal (sólo permite la llamada a `Show()`).
- Se puede definir el título de la ventana, el mensaje a mostrar, los botones que aparecen y un icono gráfico, dependiendo de los parámetros pasados al invocar el método `Show()`.

```
DialogResult answer = MessageBox.Show(this, // Owner
    "Unable to create DB", // Message
    "Error", // Title
    MessageBoxButtons.AbortRetryIgnore, // Buttons included
    MessageBoxIcon.Exclamation); // Icon
if (answer == DialogResult.Retry)
{
    // Retry operation...
}
```

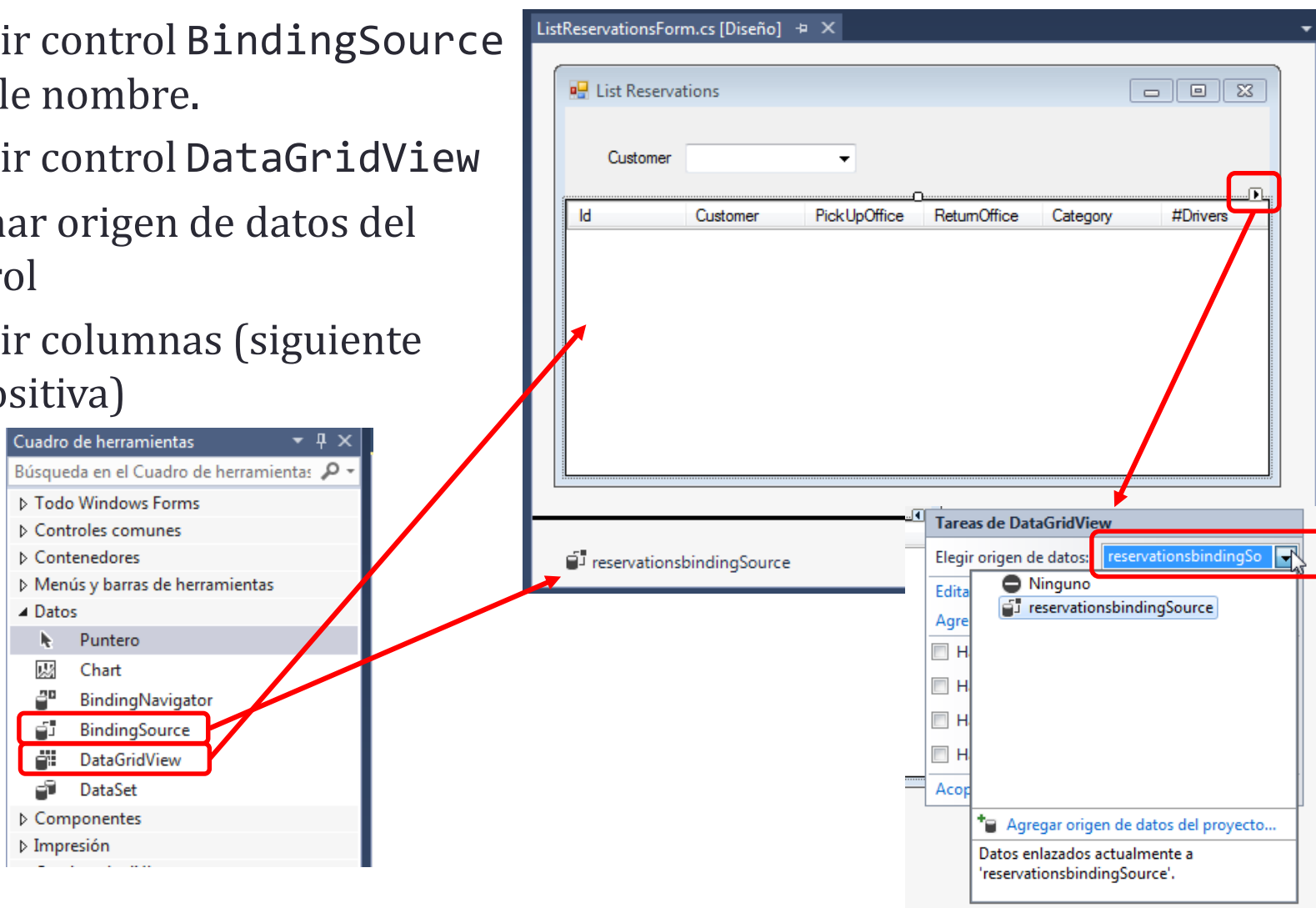


Otros cuadros de diálogo

- Operaciones comunes como abrir y guardar ficheros, imprimir, seleccionar colores, etc.
 - `OpenFileDialog`, `SaveFileDialog`, `FolderBrowserDialog`, `ColorDialog`, `FontDialog`, `PageSetupDialog` y `PrintDialog`.
- Extienden a la `CommonDialog`, que define los métodos y eventos comunes a todas estas clases. Se muestran con el método `ShowDialog()`, el cual devuelve un objeto de tipo `DialogResult` que puede tomar dos valores:
 - `DialogResult.OK` si el usuario pulsa el botón OK en el cuadro de diálogo, o
 - `DialogResult.CANCEL` en otro caso.

Visualización de colección de datos

1. Añadir control BindingSource y darle nombre.
2. Añadir control DataGridView
3. Asignar origen de datos del control
4. Añadir columnas (siguiente diapositiva)

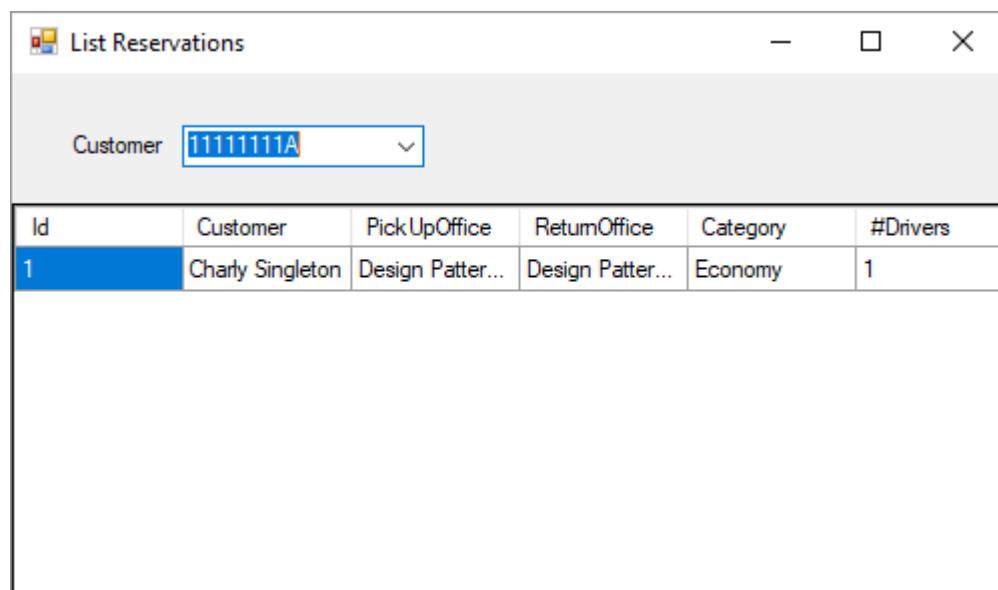


Visualización de colección de datos

The screenshot illustrates the steps to configure a DataGrid in a Windows Forms application. It shows the 'ListReservationsForm.cs [Diseño]' window with a 'List Reservations' form. The 'Tareas de DataGridView' context menu is open, with 'Agregar columna...' highlighted. The 'Agregar columna' dialog is shown, with 'Columna sin enlazar' selected. The 'Editar columnas' dialog is also open, showing the 'Id' column selected. The 'Propiedades de columnas enlazadas' section is visible, with 'DataPropertyName' set to 'ds_Id'.

Después de añadir las columnas se deben editar para asignar el nombre de la propiedad.

Visualización de colección de datos



Id	Customer	Pick Up Office	Return Office	Category	#Drivers
1	Charly Singleton	Design Patter...	Design Patter...	Economy	1

Funcionalidad deseada:

- Cuando se muestre el formulario se podrá seleccionar un cliente (Customer) del ComboBox
- Al seleccionar un cliente se mostrarán sus reservas en el DataGridView.

Visualización de colección de datos

1. Cuando se crea el formulario se deben añadir los clientes existentes al control de tipo ComboBox (customersComboBox). En el ejemplo, se hace en el método LoadData()

```
public ListReservationsForm(IVehicleRentalService service) : base(service)
{
    InitializeComponent();
    LoadData(); // se cargan los datos en el formulario
}

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


Visualización de colección de datos

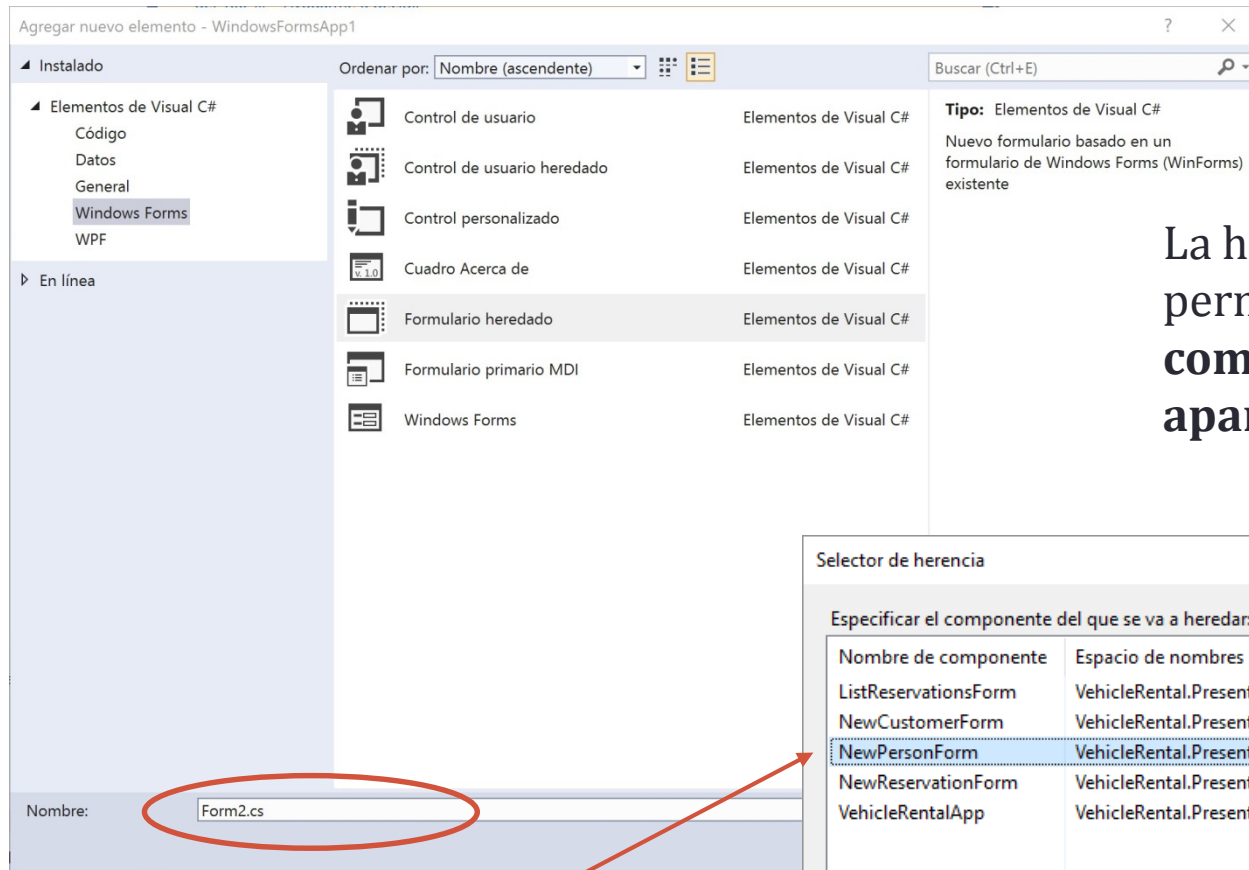
Al seleccionar un elemento en el control ComboBox se debe rellenar el control DataGridView. Se llamará al manejador del evento `SelectedIndexChanged` del control ComboBox.

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

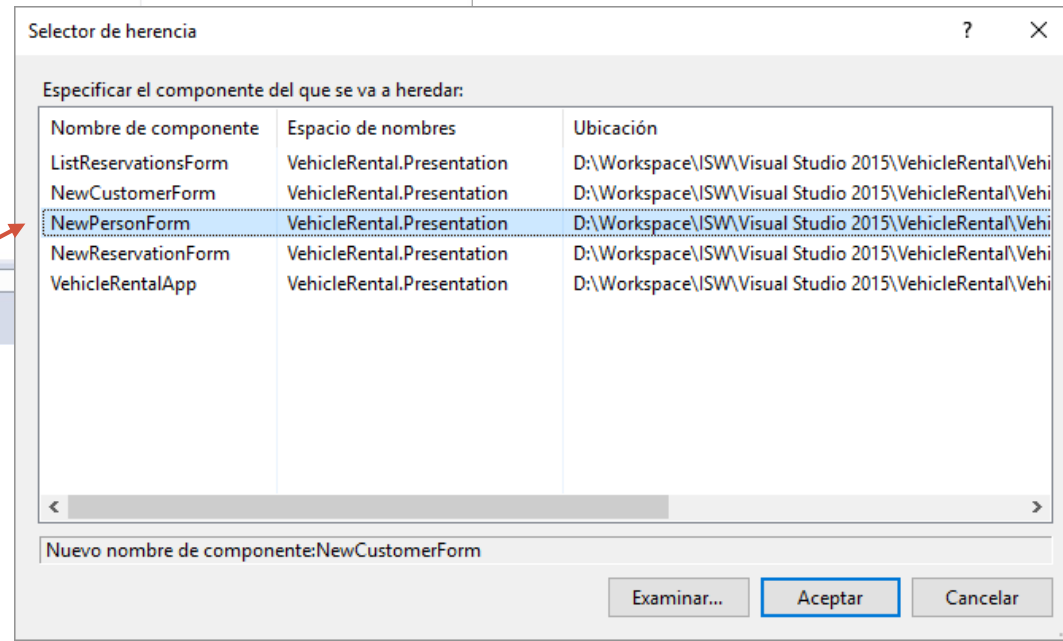
    //A BindingList of anonymous objects is used to provide the data model to the DataGridView

    BindingList<object> bindinglist = new BindingList<object>();
    foreach (Reservation r in reservations)
        //Adding one anonymous object for each reservation obtained
        bindinglist.Add(new
        {
            //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.PickUpOffice.Address,
            ds_ReturnOffice = r.ReturnOffice.Address,
            ds_Category = r.Category.Name,
            ds_NumDrivers = r.Drivers.Count
        });
    reservationsbindingSource.DataSource = bindinglist;
}
```

Herencia visual



La herencia visual nos permite **reutilizar** tanto **comportamiento** como **apariencia**



El nuevo formulario Form2 va a heredar de NewPersonForm

NOTA: es necesario compilar primero el formulario base NewPersonForm

Herencia visual. Reutilización de comportamiento

Todos los formularios utilizan `IVehicleRentalService`. Por tanto, crearemos un formulario base `VehicleRentalFormBase` con dicha referencia y **todos los formularios** heredarán de él

// 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;
    }
}
```

Herencia visual. Reutilización de comportamiento

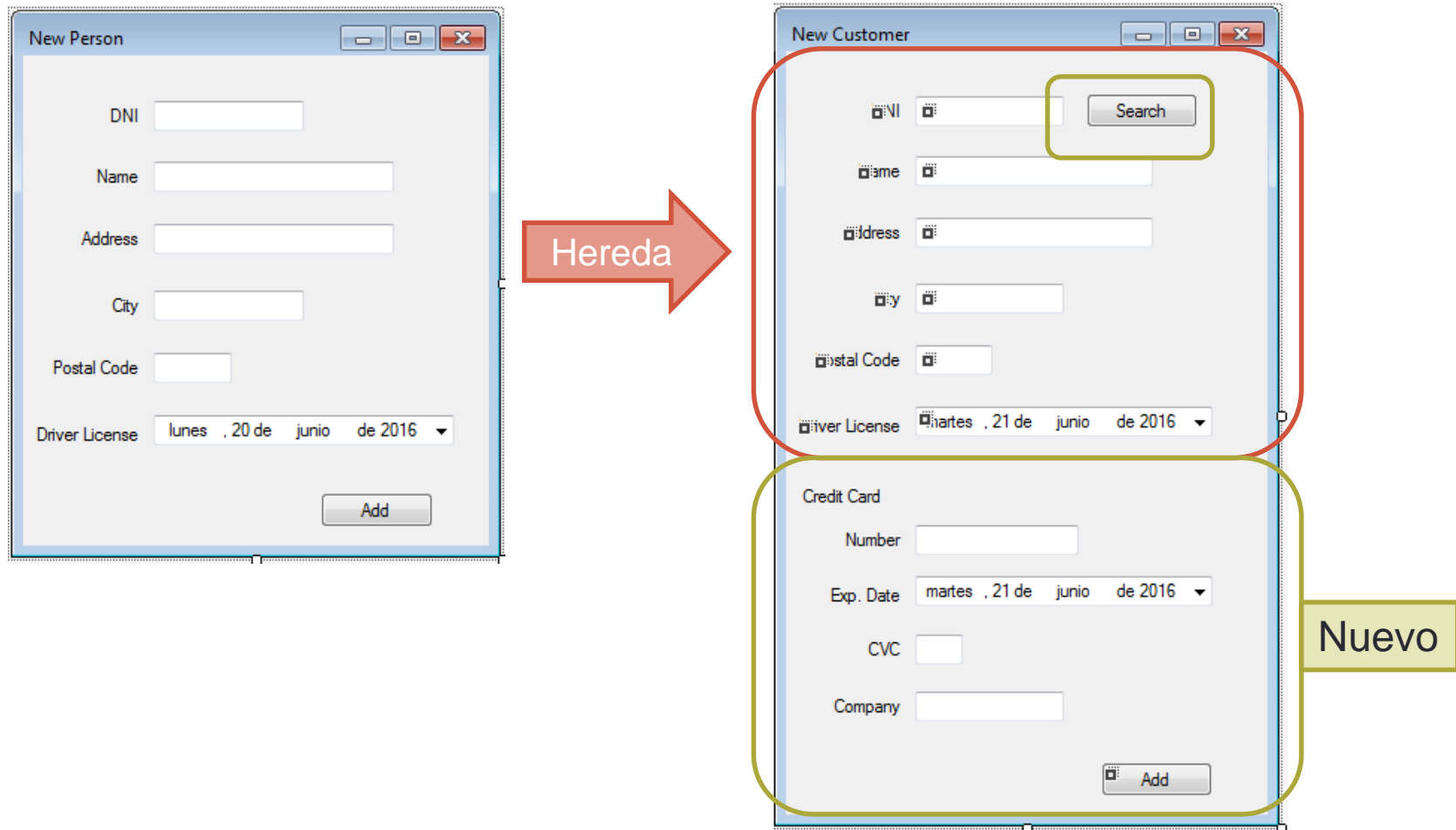
Por **ejemplo**, el formulario VehicleRentalApp...

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

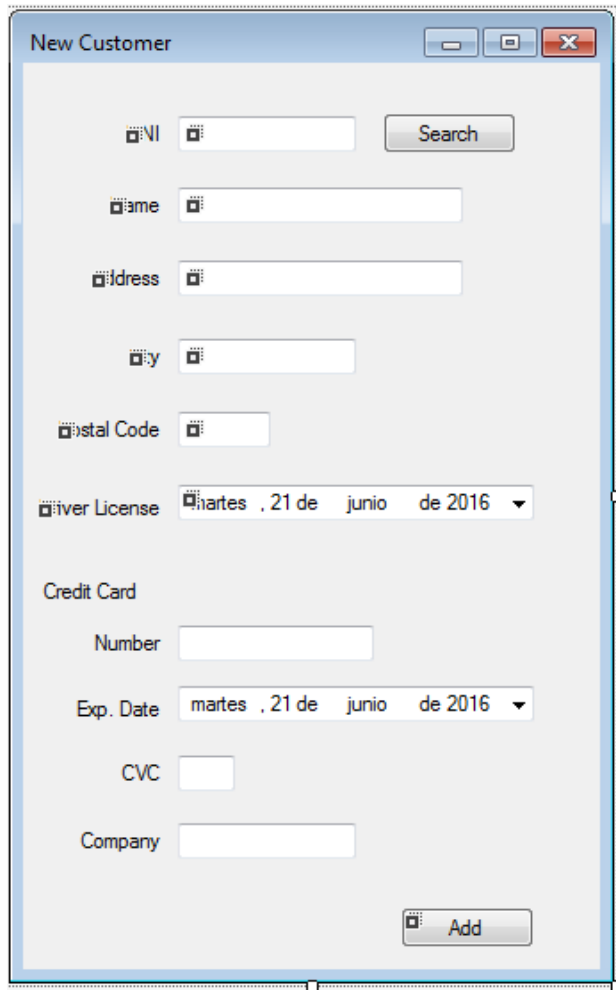
    public VehicleRentalApp(IVehicleRentalService service) : base(service)
    {
        InitializeComponent();
        listReservationForm = new ListReservationsForm(service);
        newReservationForm = new NewReservationForm(service);
    }
    ...

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

Herencia visual. Reutilización de la apariencia



Herencia visual. Reutilización de la apariencia



New Customer

NI Search

Name

Address

City

Postal Code

Driver License martes , 21 de junio de 2016

Credit Card

Number

Exp. Date martes , 21 de junio de 2016

CVC

Company

Add

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

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

Bibliografía básica

- 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

Recursos

- Tutoriales sobre los formularios Windows Forms
[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: Crear un visor de imagen
<https://msdn.microsoft.com/es-es/library/dd492135.aspx>