UNIVERSIDAD TECNOLOGICA NACIONAL

FACULTAD REGIONAL AVELLANEDA

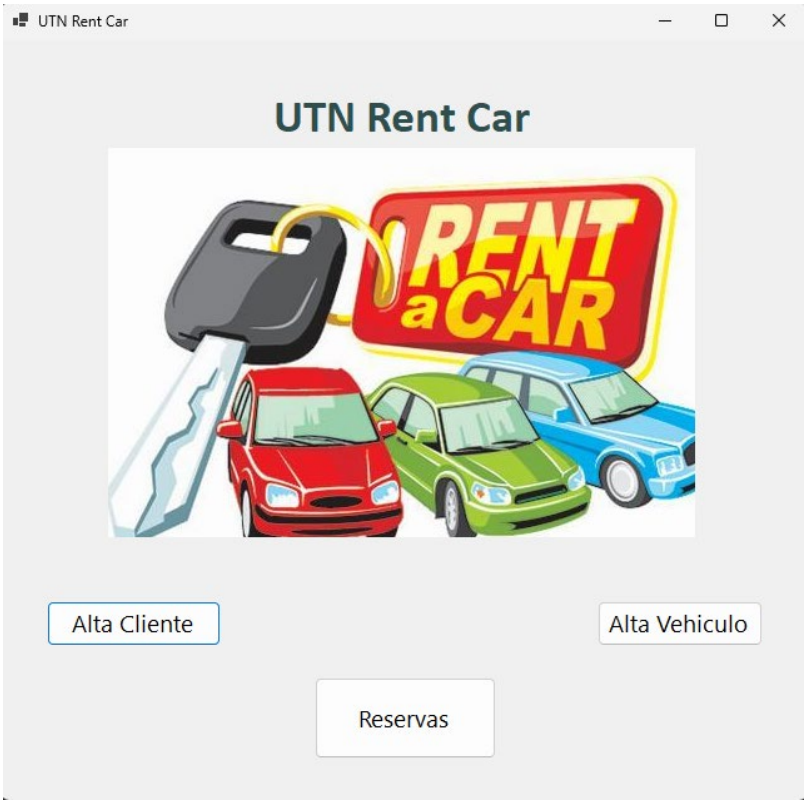
Ejercicio Integrador N° 2 – Nicolás Depetris

La siguiente aplicación fue desarrollada en el marco de la materia Laboratorio de Computación II, como Trabajo Práctico Integrador N°2.

“UTN RENT CAR” tiene como principal objetivo dar soporte a una empresa cuya actividad principal es el servicio de alquiler de vehículos, tanto autos como camionetas.

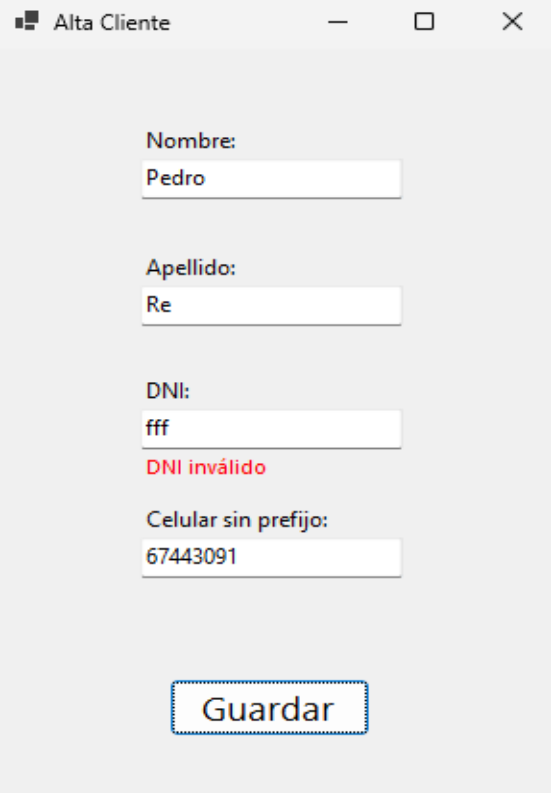
A continuación, se realizará una breve descripción de cuáles son sus funcionalidades y cómo fueron aplicados los distintos temas vistos durante la segunda parte de la materia.

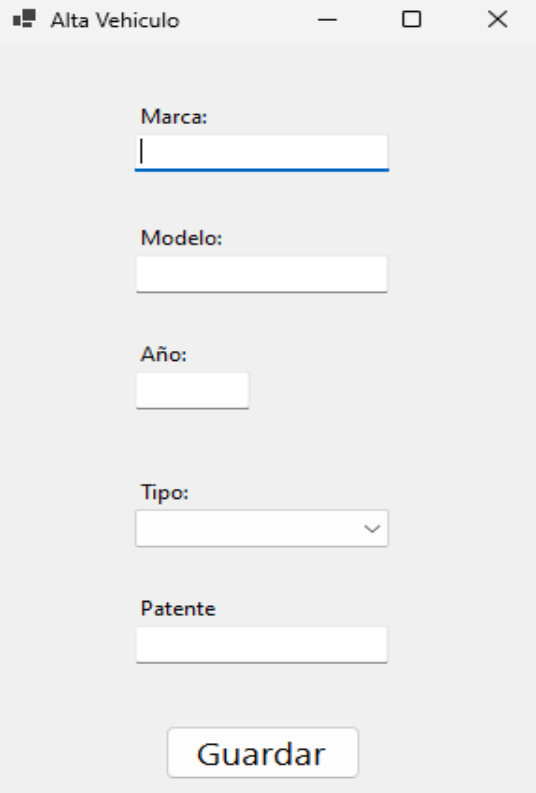
FORMULARIO PRINCIPAL



A través del formulario principal el usuario podrá acceder a los botones “Alta Cliente”, “Alta Vehículo” y “Reservas”.

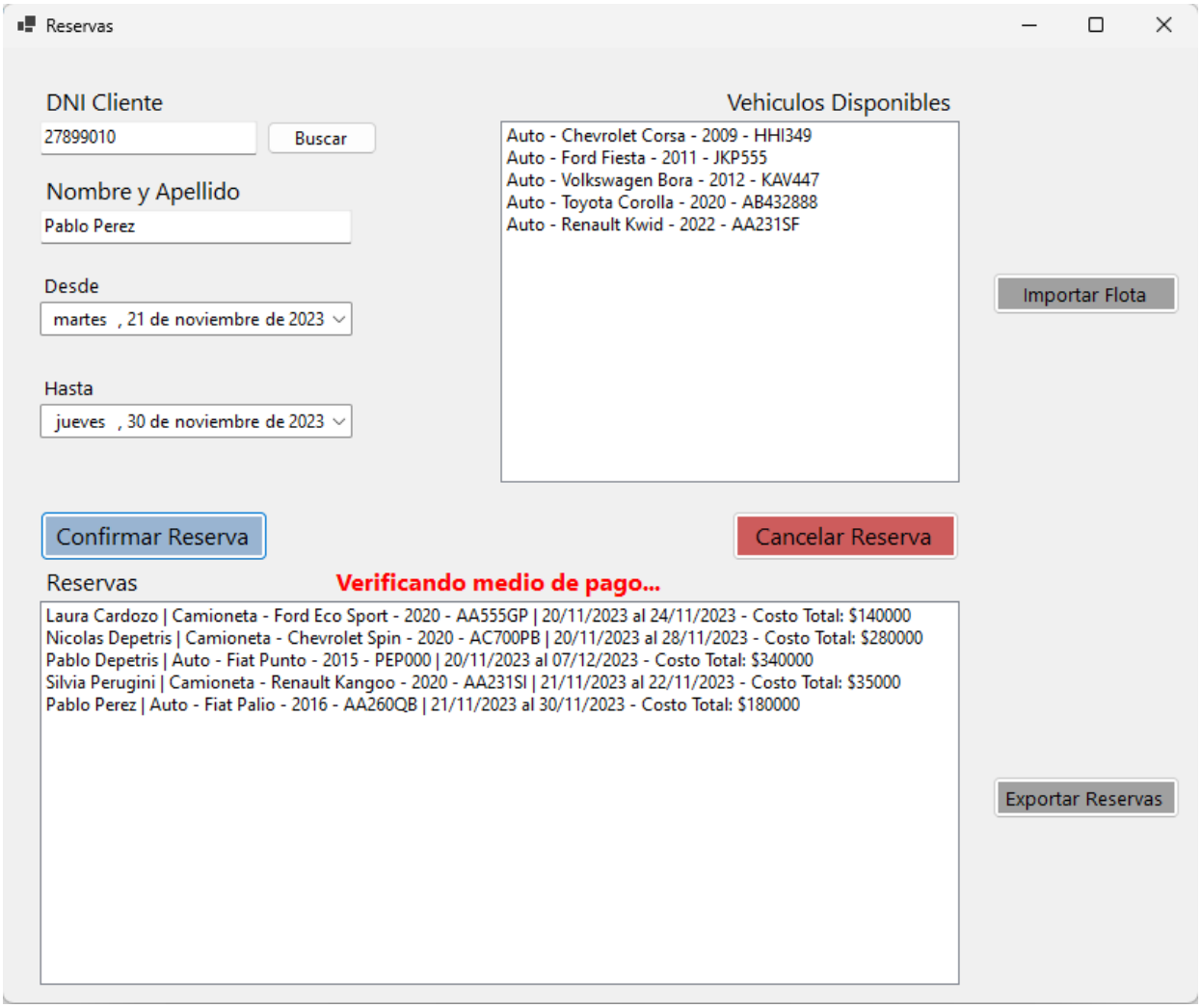
FORMULARIOS ALTA CLIENTE Y ALTA VEHICULO





Mediante el formulario “Alta Cliente” el usuario podrá registrar un nuevo cliente en la base de datos de la empresa para que luego el mismo pueda realizar una reserva y alquilar un vehículo determinado. Por su parte, “Alta Vehículo” permitirá dar de alta un nuevo vehículo cuando la firma haya sumado una nueva unidad a su flota. Cabe aclarar que en ambos formularios se realizan las validaciones correspondientes para que el usuario sólo pueda dar de alta una nueva entidad si los datos son válidos. También se verifica que el cliente/vehículo ingresado ya exista en la base de datos.

FORMULARIO RESERVAS



El siguiente formulario es el encargado de mostrar las reservas de los alquileres que se encuentran activos y los vehículos disponibles para su alquiler. Además, dentro del mismo se podrá realizar una nueva reserva o cancelar aquella que finalmente no se llevará a cabo.

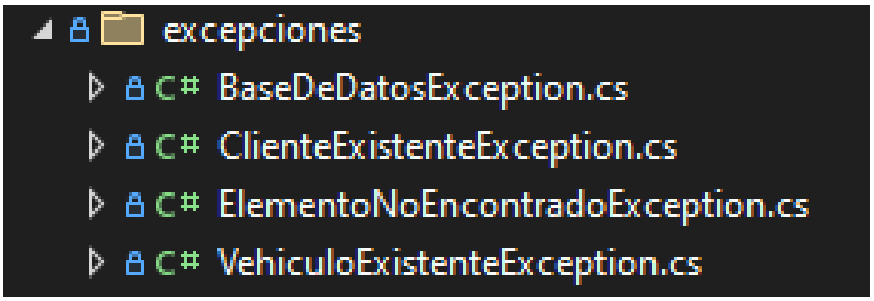
El botón “Buscar” será el encargado justamente de buscar en la base de datos un cliente existente por su DNI, para que el mismo pueda realizar una reserva eligiendo las fechas en las cuales desea alquilar un determinado vehículo, siempre y cuando se encuentre disponible. Posteriormente, se simula el proceso de pago, su verificación y confirmación.

El botón “Importar flota” realiza la importación desde un archivo JSON de nuevos vehículos que se suman a la empresa, siempre y cuando los mismo ya no formen parte de la misma -verificación

por patente-. Por su parte “Exportar Reservas” envía las reservas vigentes a un archivo JSON para llevar un registro de los alquileres que están activos.

A continuación, se detalla la inclusión de todos los temas vistos durante el curso:

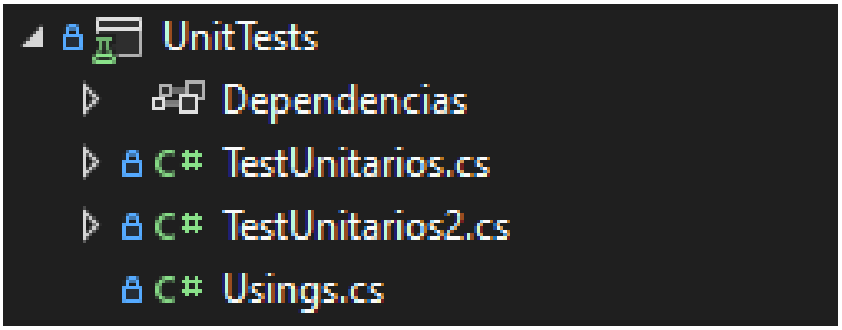
EXCEPCIONES



Ejemplo:

```
namespace Entidades.excepciones
{
    /// <summary>
    /// Excepción lanzada cuando un cliente ya existe en la base de datos.
    /// </summary>
    5 referencias
    public class ClienteExistenteException : Exception
    {
        1 referencia
        public ClienteExistenteException(string? message) : base(message)
        {
        }

        0 referencias
        public ClienteExistenteException(string? message, Exception? innerException) : base(message, innerException)
        {
        }
    }
}
```

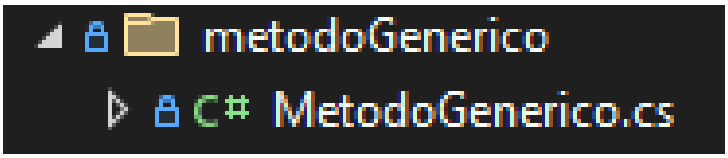
TEST UNITARIOS



Ejemplo:

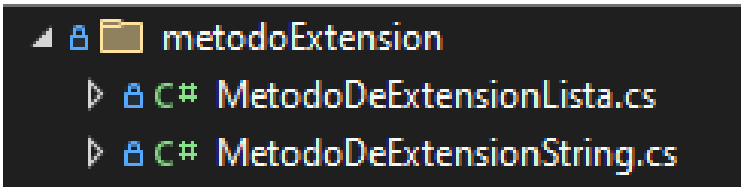
```
[TestClass]
0 referencias
public class TestUnitarios
{
    [TestMethod]
    /// <summary>
    /// Verifica que al invocar el método CalcularCostoReserva con un vehículo de tipo "Auto", devuelva el costo correcto
    /// según la tarifa y los días alquilados.
    /// </summary>
    0 referencias
    public void Al_InvocarAlMetodoCalcularCostoReserva_SiSeRecibeUnVehiculoDeTipoAuto_DeberiaDevolverElCostoSegunTarifaYDiasAlquilados()
    {
        // Arrange
        Cliente cliente = new Cliente("Pedro", "Gonzalez", 26577620, "65881452");
        Vehiculo vehiculo = new Vehiculo("Fiat", "Uno", 2011, "Auto", "JSP222", true);
        DateTime fechaInicio = new DateTime(2023, 11, 1);
        DateTime fechaFin = new DateTime(2023, 11, 6);

        Reserva reserva = new Reserva(cliente, 12345678, vehiculo, "PatenteAuto", fechaInicio, fechaFin, true);

        // Act
        float costoReserva = reserva.CalcularCostoReserva();

        // Assert
        Assert.AreEqual(Reserva.TarifaAuto * 5, costoReserva);
    }
```

GENERICS




```
2 referencias
public class MetodoGenerico
{
    /// <summary>
    /// Busca un elemento en una lista por el valor de una propiedad específica.
    /// </summary>
    /// <typeparam name="T">Tipo de elemento en la lista.</typeparam>
    /// <typeparam name="TPropiedad">Tipo de propiedad a comparar.</typeparam>
    /// <param name="valorPropiedad">Valor de la propiedad a buscar.</param>
    /// <param name="lista">Lista en la que se realizará la búsqueda.</param>
    /// <param name="obtenerPropiedad">Función que obtiene el valor de la propiedad del elemento.</param>
    /// <returns>El primer elemento encontrado que coincide con el valor de la propiedad.</returns>
    /// <exception cref="ElementoNoEncontradoException">Se lanza si no se encuentra ningún elemento con el valor de la propiedad
    /// proporcionado.</exception>
    2 referencias
    public static T LeerElementoPorPropiedad<T, TPropiedad>(TPropiedad valorPropiedad, List<T> lista, Func<T, TPropiedad> obtenerPropiedad)
        where T:class
    {
        try
        {
            foreach (T elemento in lista)
            {
                if (EqualityComparer<TPropiedad>.Default.Equals(obtenerPropiedad(elemento), valorPropiedad))
                {
                    return elemento;
                }
            }

            throw new ElementoNoEncontradoException($"Ningún elemento encontrado para ese valor de propiedad: {valorPropiedad}");
        }
        catch (Exception ex)
        {
            throw new ElementoNoEncontradoException($"Error al buscar elemento de tipo {typeof(T).Name}", ex);
        }
    }
}
```

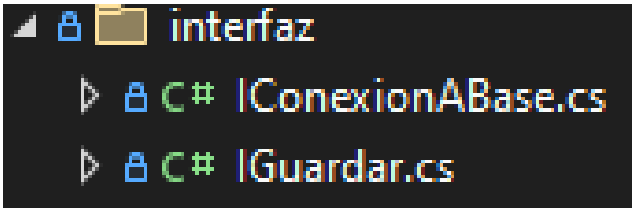
METODOS DE EXTENSION



Ejemplo:

```
0 referencias
public static class MetodoDeExtensionLista
{
    /// <summary>
    /// Filtra las reservas vigentes de una lista de reservas.
    /// </summary>
    /// <param name="listaReservas">La lista de reservas a filtrar.</param>
    /// <returns>Una nueva lista que contiene solo las reservas vigentes.</returns>
    4 referencias
    public static List<Reserva> FiltrarReservasVigentes(this List<Reserva> listaReservas)
    {
        List<Reserva> listaReservasVigentes = new List<Reserva>();
        foreach (Reserva reserva in listaReservas)
        {
            if (reserva.Vigente)
            {
                listaReservasVigentes.Add(reserva);
            }
        }
        return listaReservasVigentes;
    }
}
```

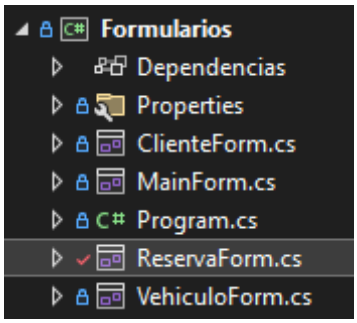
INTERFACES



Ejemplo:

```
/// <summary>
/// Define un contrato para las clases que pueden realizar operaciones de guardado en la base de datos.
/// </summary>
/// <typeparam name="T">El tipo de entidad que se va a guardar.</typeparam>
3 referencias
public interface IGuardar<T> where T : class
{
    /// <summary>
    /// Guarda la entidad en la base de datos.
    /// </summary>
    /// <param name="entidad">La entidad que se va a guardar, de tipo genérica.</param>
    10 referencias
    void Guardar(T entidad);
}
```

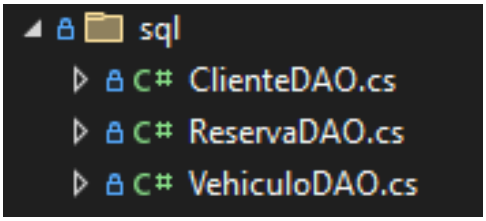
ARCHIVOS – SERIALIZACION



Ejemplo:

```
/// <summary>
/// Exporta la lista de reservas proporcionada a un archivo JSON en el escritorio del usuario.
/// </summary>
/// <param name="listaReservas">Lista de reservas a exportar.</param>
1 referencia
private void ExportarJson(List<Reserva> listaReservas)
{
    string escritorioPath = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

    string carpetaDeLaSolucion = Path.Combine(escritorioPath, "ArchivosIntegrador");
    if (!Directory.Exists(carpetaDeLaSolucion))
    {
        Directory.CreateDirectory(carpetaDeLaSolucion);
    }
    string rutaCompleta = Path.Combine(carpetaDeLaSolucion, "reservasVigentes.json");
    JsonSerializerOptions options = new JsonSerializerOptions();
    options.WriteIndented = true;
    string jsonListaReservas = JsonSerializer.Serialize(listaReservas, options);
    File.WriteAllText(rutaCompleta, jsonListaReservas);
}
```

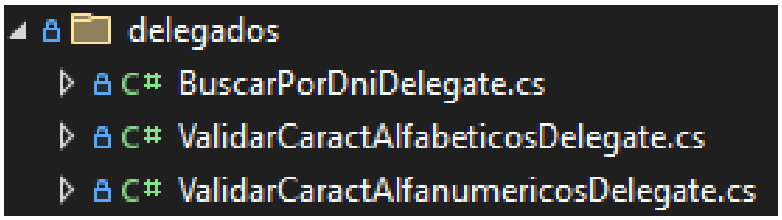
SQL – CONEXIÓN A BASE DE DATOS



Ejemplo:

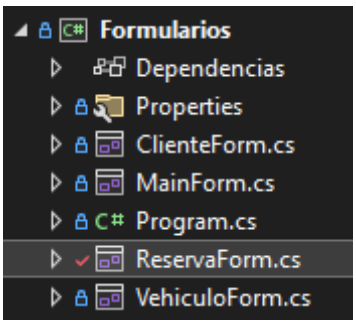
```
/// <summary>
/// Modifica el estado de disponibilidad de un vehículo en la base de datos.
/// </summary>
/// <param name="vehiculoEditado">Vehículo con los cambios a aplicar.</param>
/// <param name="patente">Patente única que identifica al vehículo a modificar.</param>
/// <exception cref="BaseDeDatosException">Se lanza cuando ocurre un error al interactuar con la base de datos.</exception>
2 referencias
public static void Modificar(Vehiculo vehiculoEditado, string patente)
{
    try
    {
        using (SqlConnection connection = new SqlConnection(VehiculoDAO.StringConnection))
        {
            string query = "UPDATE VEHICULOS set DISPONIBLE=@disponible WHERE PATENTE=@patente";

            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("patente", patente);
            command.Parameters.AddWithValue("disponible", vehiculoEditado.Disponible);
            connection.Open();
            command.ExecuteNonQuery();
        }
    }
    catch (Exception ex)
    {
        throw new BaseDeDatosException("Error al actualizar el vehiculo", ex);
    }
}
```

DELEGADOS Y EXPRESIONES LAMBDA




```
/// <summary>
/// Delegado utilizado para buscar un cliente según su dni en una lista de clientes.
/// </summary>
/// <param name="numDni">El dni del cliente que se va a buscar.</param>
/// <param name="listaClientes">La lista de clientes en la que se buscará el cliente.</param>
/// <returns>Un mensaje indicando si la cadena contiene únicamente caracteres alfabéticos o no.</returns>
public delegate Cliente BuscarPorDniDelegate(int numDni, List<Cliente> listaClientes);
```

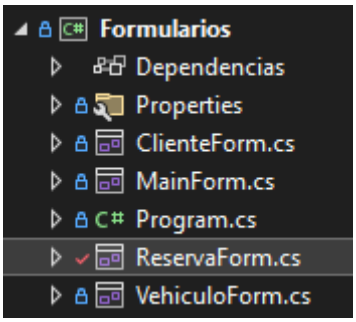
Ejemplo delegado:



```
this.delegadoBuscarPorDni = new BuscarPorDniDelegate(this.BuscarPorDni);
```

```
/// <summary>
/// Busca un cliente por su número de DNI en la lista de clientes.
/// </summary>
/// <param name="numDni">Número de DNI del cliente a buscar.</param>
/// <param name="listaClientes">Lista de clientes en la que se realizará la búsqueda.</param>
/// <returns>El objeto Cliente si se encuentra.</returns>
/// <exception cref="ElementoNoEncontradoException">Se lanza si no se encuentra ningún cliente con el DNI proporcionado.</exception>
2 referencias
private Cliente BuscarPorDni(int numDni, List<Cliente> listaClientes)
{
    foreach (Cliente cliente in listaClientes)
    {
        if (cliente.Dni == numDni)
        {
            return cliente;
        }
    }
    throw new ElementoNoEncontradoException("No se encontró ningún cliente con ese DNI.");
}
```

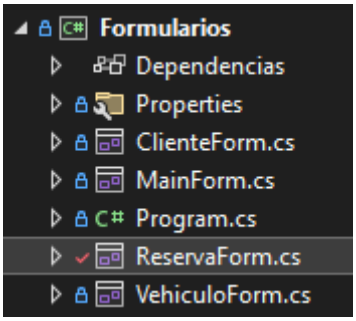
Ejemplo expresiones lambda:




```
/// <summary>
/// Valida si un vehículo ya existe en la lista de vehículos proporcionada, a través de su patente.
/// </summary>
/// <param name="vehiculo">Vehículo a validar.</param>
/// <param name="listaVehiculos">Lista de vehículos en la que se realiza la búsqueda.</param>
/// <returns>True si el vehículo ya existe en la lista, False en caso contrario.</returns>
1 referencia
private bool ValidarVehiculoExistente(Vehiculo vehiculo, List<Vehiculo> listaVehiculos)
{
    return listaVehiculos.Any(Vehiculo item => item.Patente == vehiculo.Patente);
}
```

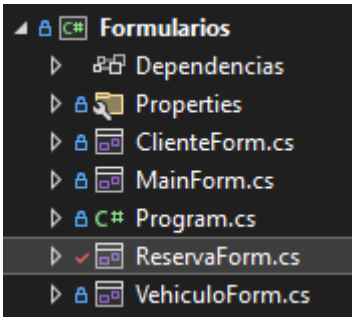
HILOS

Ejemplo:




```
/// <summary>
/// Inicia el procesamiento simulado de un pago utilizando eventos de verificación, éxito y finalización del pago.
/// </summary>
1 referencia
private void ProcesarPago()
{
    // Inicia una nueva tarea en un hilo secundario.
    // Utiliza un ciclo do-while para simular la verificación, éxito y finalización del pago.
    this.cancellation = new CancellationTokenSource();
    Task.Run(() =>
    {
        do
        {
            if (this.OnVerificacionPago is not null && this.OnPagoOk is not null)
            {
                // Invoca el evento de verificación del pago y espera simulada.
                this.OnVerificacionPago.Invoke("Verificando medio de pago...");
                Thread.Sleep(3500);
                // Invoca el evento de éxito del pago.
                this.OnPagoOk.Invoke("El pago se realizó con éxito");
                Thread.Sleep(2000);
                // Invoca el evento de finalización del pago y cancela la tarea.
                this.OnPagoOff.Invoke();
                this.cancellation.Cancel();
            }
        } while (!this.cancellation.IsCancellationRequested);
    }, this.cancellation.Token);
}
```

EVENTOS



```
//EVENTOS
public event DelegateVerificacionPago OnVerificacionPago;
public event DelegatePagoOk OnPagoOk;
public event DelegatePagoOff OnPagoOff;
```

```
1 referencia
private void ReservaForm_Load(object sender, EventArgs e)
{
    try
    {
        this.dtpDesde.MinDate = DateTime.Now;
        this.dtpHasta.MinDate = DateTime.Now.AddDays(1);
        this.ListaReservas = ReservaDAO.LeerReservas();
        this.CargarListaReservas();
        this.CargarListaVehiculosDisp();
        this.OnVerificacionPago += this.MostrarVerificacionPago;
        this.OnPagoOk += this.MostrarPagoOk;
        this.OnPagoOff += this.BorrarPago;
    }
    catch (BaseDeDatosException ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Ejemplo:

```
/// <summary>
/// Método manejador del evento OnVerificacionPago que muestra un mensaje de verificación de pago en el formulario.
/// </summary>
/// <param name="mensaje">Mensaje de verificación de pago.</param>
2 referencias
private void MostrarVerificacionPago(string mensaje)
{
    // Verifica si es necesario invocar el método en el hilo de la interfaz de usuario.
    if (this.InvokeRequired)
    {
        // Invoca el método en el hilo de la interfaz de usuario utilizando.
        this.BeginInvoke(() => this.MostrarVerificacionPago(mensaje));
    }
    else
    {
        this.lblPago.Visible = true;
        this.lblPago.ForeColor = Color.Red;
        this.lblPago.Text = mensaje;
    }
}
```