

Assignment #1 (15 points)

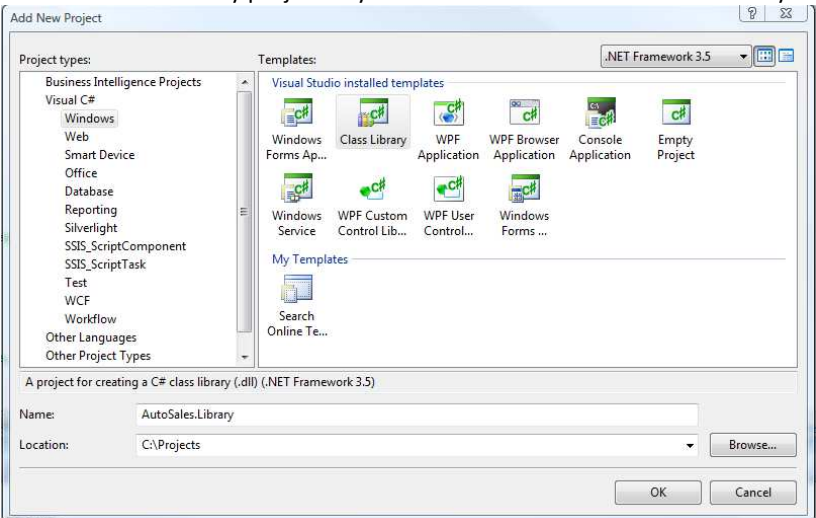
Creating a windows application with Visual Studio .NET

Exercise 1

Building a windows application

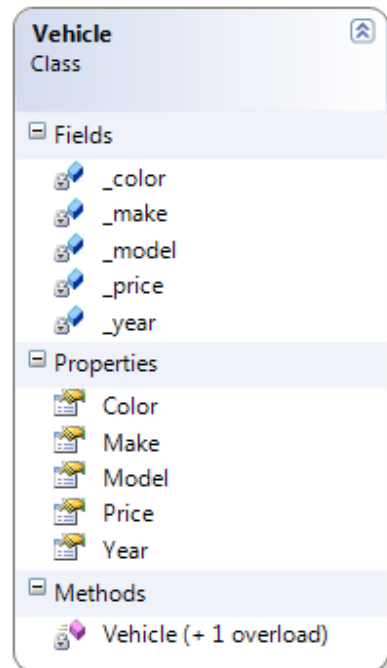
Scenario

Your employer, Student's Used Auto Sales, has found their current inventory management system to be too cumbersome and would like to upgrade from a command-line application (console) to a windows application. The new application should support the same functions of entering a vehicle into inventory and browsing the inventory catalog.

Tasks	Detailed Steps
1. Setting up the lab	<p>a. Open Visual Studio and create a new C# windows application project named AutoSales.UI. Windows applications are also known as winform applications.</p> <p>Note: For convenience, you should create the project on your flash drive.</p> <p>b. Delete the default form, Form1.</p> <p>c. Add three new forms to the project. Name these forms:</p> <ul style="list-style-type: none">• frmMain• frmAddVehicle• frmListInventory
2. Create a library project	<p>a. Add a new class library project to your solution. Name it AutoSales.Library.</p> 

3. Create the types

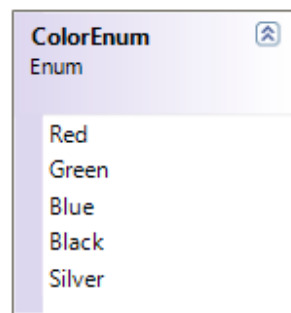
- a. Add a source file named vehicle.cs. Create the Vehicle class with these members:



Add business rules to the Year and Price properties :

- Year between 1960 and next year.
- Price between 1 and \$100,000

- b. Add a source file named ColorEnum.cs. Create an enumeration for the vehicle color.



4. Create a reference

- a. Create a reference from the AutoSales.UI project to AutoSales.Library project. This will make the types in your library available to your windows application.

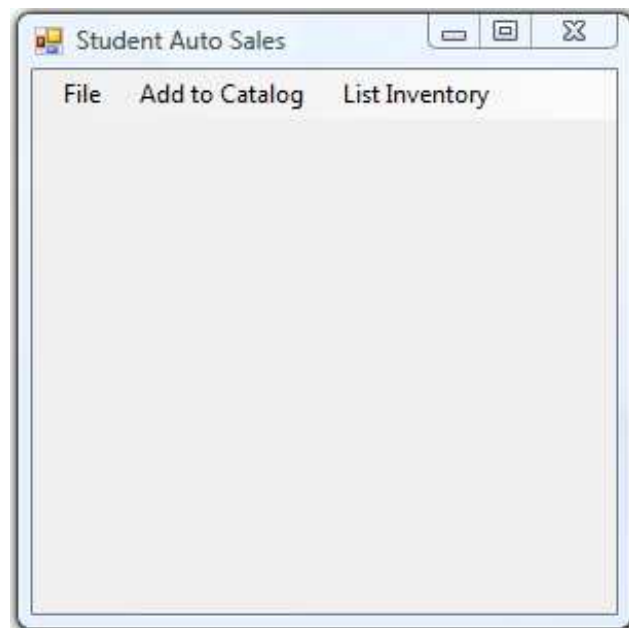
Note: From your AutoSales.UI winform project, right-click on the References folder and choose "Add Reference". In the Add Reference window choose the Projects tab and select your AutoSales.Library project.

5. Specify the startup form

- a. Since we deleted form1, we'll need to modify the code that kick-starts our application to create an instance of the correct form.

6. Design the main form

- b.** Open the Program.cs source file and look in the Main() method. Recall that the Main method is the entry point for applications.
- c.** Find the line that loads the initial form and change it from
`Application.Run(new Form1());`
to `Application.Run(new frmMain());`
- a.** Open frmMain in design view by double-clicking on the source file.
- b.** Modify the title of this form by changing its Text property to “Student Auto Sales”.
- c.** Set the form’s IsMdiContainer property to true. This makes the form a container for other windows.
- d.** Drag a StatusStrip control onto the form. Notice that the StatusStrip control shows up in the component tray.
- e.** Drag a MenuStrip control onto the form. Modify this control to create the following menu structure:
 - File → Exit (Name = exitToolStripMenuItem)
 - Add to Catalog (Name = addToolStripMenuItem)
 - List Inventory (Name = listToolStripMenuItem)



7. Code the main form

- a.** Go to code view by hitting the F7 button.
- b.** Declare a field to represent the inventory catalog at the top of the class:
Create a public static field of the frmMain class. Type the field as List<Vehicle> and name it catalog.

```
public static List<Autosales.Library.Vehicle> catalog;  
catalog = new List< Autosales.Library.Vehicle>();
```

<p>8. Wire up the menu items</p>	<p>a. Double click on the Quit menu item to bring up its default event (click) eventhandler. Here you will simply quit the application:</p> <pre>private void exitToolStripMenuItem_Click(object sender, EventArgs e) { Application.Exit(); }</pre> <p>b. Double click on the Add Vehicle menu item to bring up its default event (click) eventhandler. Here you will create an instance of the frmAddVehicle form and display it as a MDI child form:</p> <pre>private void addToolStripMenuItem_Click(object sender, EventArgs e) { frmAddVehicle frm = new frmAddVehicle(); frm.MdiParent = this; //"this" indicates the current class frmMain frm.Show(); }</pre> <p>c. Double click on the List Inventory menu item to bring up its default event (click) eventhandler. Here you will create an instance of the frmListInventory form and display it as a MDI child form:</p> <pre>private void listToolStripMenuItem_Click(object sender, EventArgs e) { frmListInventory frm = new frmListInventory(); frm.MdiParent = this; //"this" indicates the current class frmMain frm.Show(); }</pre>
<p>9. Design the add form</p>	<p>a. Open the frmAddVehicle form in design view by double-clicking on the source file.</p> <p>b. Modify the title of this form by changing its Text property to “Add Vehicle to Inventory”.</p> <p>c. Add a ColorDialog control.</p> <p>d. Add a header label to the top of the form. Set its Text property to “Add a Vehicle”.</p> <p>e. Add a textbox and a label for the Make, Model, and Color properties. Give the input controls meaningful names such as txtModel, txtMake, and txtColor.</p> <p>f. Add a button to the right of the color textbox. When clicked, this button will open the color dialog window that allows the user to pick a color. Name this button btnColorPicker and set its Text property to “...”.</p> <p>g. Add a NumericUpDown control to capture the year and a label. Name this control numericYear and set its Minimum property to 1960. We’ll set its Maximum property to the current year in the page’s load event. This control will make user input more reliable.</p> <p>h. Add a MaskedEditControl to capture the price and a label. Name this control masktxtPrice and set its Mask property to #####. This control will make user input more reliable.</p> <p>i. Add a button to the bottom of the form. Set its name to btnAdd and its Text property to “Add to Inventory”.</p>

j.

The screenshot shows a Windows form titled "Add Vehicle to Inventory". Inside the form, there is a section titled "Add a Vehicle". Below this title, there are five input fields: "Make:" with the value "Jeep", "Model:" with the value "Wrangler", "Year:" with a numeric up-down control showing "2007", "Price:" with a masked edit control showing "21000", and "Color:" with the value "Red" and a color picker button. At the bottom of the form is a button labeled "Add to Inventory". Annotations with arrows point to specific controls: an orange arrow labeled "NumericUpDown" points to the Year control; a green arrow labeled "MaskedEdit" points to the Price control; and two red arrows labeled "TextBox" point to the Make and Model controls.

10. Code the Add Vehicle form

- a. Initialize the form by double-clicking the form to bring up the event handler for the load event; this is the default event for forms. Here we will set the minimum/maximum year range for the NumericUpDown control

```
private void frmAddVehicle_Load(object sender, EventArgs e)
{
    numericYear.Maximum = (decimal)DateTime.Today.Year;
    numericYear.Minimum = 1960;
    numericYear.Value = (decimal)DateTime.Today.Year - 1;
}
```

- b. Wire up the color picker button by double-clicking on the button. Doing so will automatically switch to code view and the button's click event handler will be created. Here is where you add code that is executed in response to a click event.

```
private void btnShowColor_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() != DialogResult.Cancel)
    {
        txtColor.Text = colorDialog1.Color.ToKnownColor().ToString();
    }
}
```

- c. Wire up the add vehicle button by double-clicking on the button to bring up the default event (click) handler. Here you will create an instance of a Vehicle object, populate its state and add it to the catalog before closing the form. The catalog exists on the Main form.

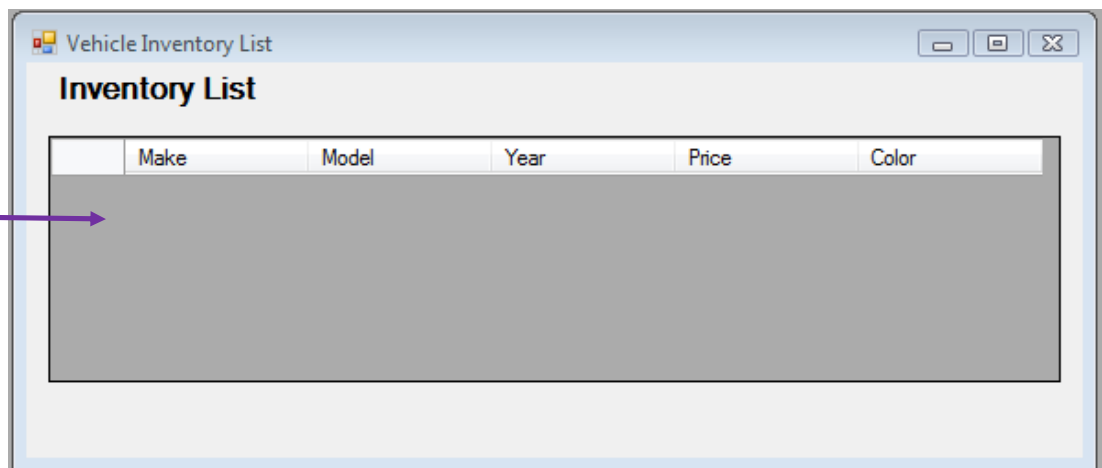
```
private void btnAdd_Click(object sender, EventArgs e)
{
    Autosales.Library.Vehicle v = new Autosales.Library.Vehicle();
    v.Make = txtMake.Text;
    v.Model = txtModel.Text;
    v.Year = (int)numericYear.Value;
    v.Price = Convert.ToInt32(masktxtPrice.Text);

    try //convert the user-supplied color to the enum value
    {
        Type t = typeof(Autosales.Library.ColorEnum);
        object obj = Enum.Parse(t, txtColor.Text);
        v.Color = (Autosales.Library.ColorEnum)obj;
    }
    catch
    {
        v.Color = Autosales.Library.ColorEnum.Other;
    }

    frmMain.catalog.Add(v);
    this.Close();
}
```

11. Design the List Inventory form

- Open the frmListInventory form in design view by double-clicking on the source file.
- Modify the title of this form by changing its Text property to "List Vehicle Inventory".
- Add a label control for the form header. Set its Text property to "Inventory List".
- Add a DataGridView control to the form. Name this control gdInventory.



12. Code the List Inventory form

- From the List Inventory form, hit F7 to go to code view. Alternatively, you can right-click on the source file and choose "View Code".
- In the form constructor, after the call to InitializeComponent(), bind the data grid: `gdInventory.DataSource = frmMain.catalog;`