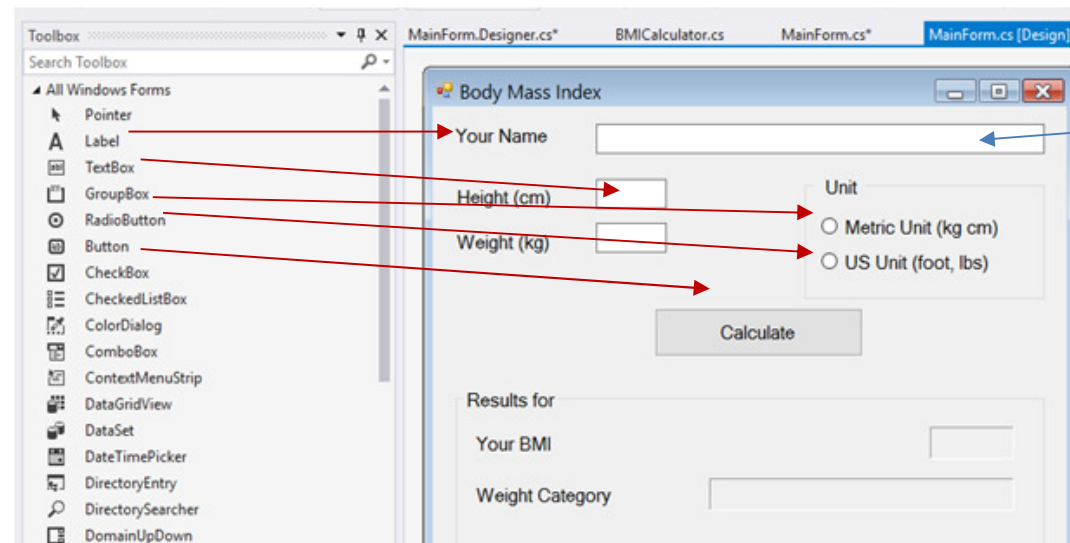


1. Create a new project

Open Visual Studio and create a new project. Select Windows Forms Application from the template, choose a name for your solution and project directory. Visual Studio then prepares the skeleton of the application, and displays a starting Form. When you have the form displayed, you will also see the Toolbox (usually at the left-side). In case you don't see the Toolbox, go to the **View** menu in Visual Studio, and you will find the Toolbox among the menu items. Click on that and the Toolbox will be displayed. The controls needed in this assignment are shown below.

These controls are perhaps listed alphabetically in your Toolbox.

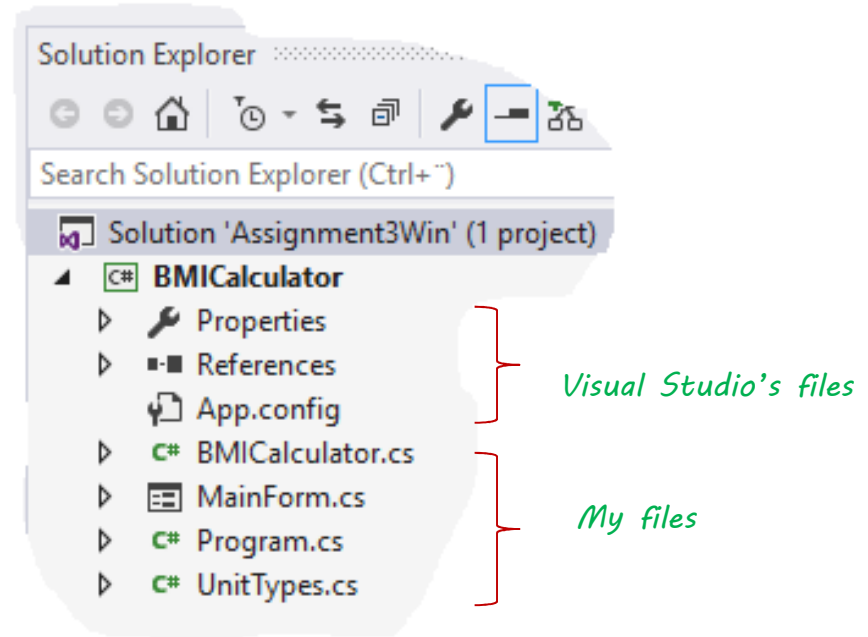


txtName

Design the GUI using the controls from the Toolbox. Note: your Toolbox may not have the order shown in this image. The controls on your editor is most probably sorted in an alphabetical order.

2. The Project

When you create a Windows Forms Application in Visual studio, it also creates a number of files with info about the project, the components from .NET Framework, the DLLs that need to be referenced and also the compilation configurations.



3. .The UnitType enum

Enums are types, like classes, and therefore they should be public (or internal) so they can also be used in other classes in the project. It is a recommended practice to save each type (class, struct, enum, etc.) in a separate file.

*The whole
UnitTypes.cs file*

```
namespace Assignment3Win
{
    enum UnitTypes
    {
        Metric,
        American
    }
}
```

4. The MainForm Class

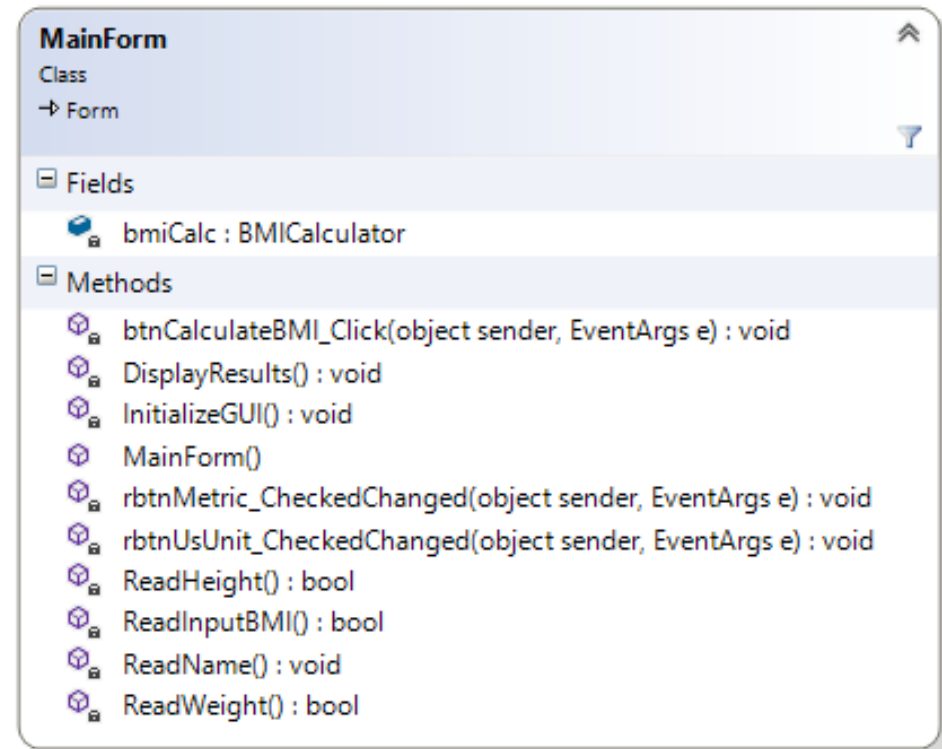
4.1.1 Only and only **MainForm** should work with the user interface via its controls (the components it contains).

4.1.2 **BMI calculator** (and other classes if any) should not have anything to do with **MainForm** or its components. Instead, **MainForm** will be using the **BMICalculator** class. It will provide input to and gets output from the object of the **BMICalculator** class which is named **bmiCalc** in this example..

- **MainForm** declares and creates an instance of the **BMICalculator** (**bmiCalc**).

```
public partial class MainForm : Form
{
    // create an instance of BMICalculator
    private BMICalculator bmiCalc = new BMICalculator ( );
```

- **MainForm** uses methods (those which are public) of the instance **bmiCalc** to set (save) values from the input controls (textboxes, option buttons, etc.) in the instance's (**bmiCalc**'s) private fields (name, height, weight, unit type).
- **MainForm** calls the relevant **bmiCalc**'s methods to calculate and provide output.
- **MainForm** updates the GUI by displaying the output using the dedicated controls.



4.1.3 Write a method **InitializeGUI** and use it for clearing input/output controls and setting default values (default RadioButton).

Note: Each set of **RadioButtons** must be encapsulated inside a container component such as a **GroupBox**. Otherwise different **RadioButtons** sets contained within the same component (Windows Form) will be considered as one set. When one of the buttons is checked, all other buttons in the same set will be unchecked. Although we have only one set of radio buttons on our GUI, it is still a good idea to enclose them inside a **GroupBox** control

```
/// <summary>
/// The MainForm constructor - will run as soon as
/// the GUI is created
/// </summary>
1reference
public MainForm ( )
{
    InitializeComponent ( ); //VS's initializations
    InitializeGUI ( );       //Your initializations.
}
/// <summary>
/// Initialize the GUI with some preset options
/// </summary>
1reference
private void InitializeGUI ( )
{
    this.Text = "The Body Mass Calculator"; //this = me --> MainForm

    //input controls
    rbtnUsUnit.Checked = true;
    lblHeight.Text = "Height (feet)";
    lblWeight.Text = "Weight (lbs)";

    //output controls
    txtHeight.Text = string.Empty;
    txtWeight.Text = string.Empty;
}
```

*Do not change the call order InitializeComponent
() must always be the first statement*

4.1.4 Where to go next? You have to think about what to do and when to do things. A possible scenario:

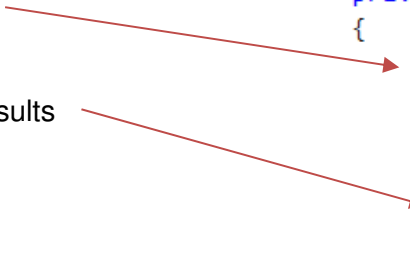
- The applications starts and **MainForm** is displayed with the initializations coded in the **MainForm's** constructor (see [public MainForm \(\)](#) above)
- The program then waits (at run time) until the user clicks the **Calculate** button.

When the user clicks the button, do the following:

- ✓ Read and validate input
- ✓ If input is ok
- ✓ Calculate and display results

```
private void btnCalculateBMI_Click (object sender, EventArgs e)
{
    bool ok = ReadInputBMI ( );

    if (ok)
    {
        DisplayResults ( ); //calculate and display results
    }
}
```



Get the user input from the controls and save them in bmiCalc object.

- Read the name-input given in the name-textbox (txtName.Text)
 - If txtName.Text is empty call bmiCalc.SetName (method that you will need to write) and send “No name” as a parameter
 - Otherwise call bmiCalc.SetName and send txtName.Text as a parameter
- Read the height value, txtHeight.Text, but this is a string and need to be converted to a double

- Use `double.TryParse` to convert the string to a valid double value
- Convert feet to inches and save in `bmiCalc`'s height field. (Code is given for this part later in this document).
- Read the weight value and validate it using `double.TryParse`
 - If good value save in `bmiCalc`'s weight field
- **If the input is valid,**
 - Call `bmiCalc`'s method that calculates the BMI value.
 - Call `bmiCalc`'s methods to do other jobs and get values.
 - Display the values in the labels that you have drawn for them. Remember that a label's `Text` property is the text that will be displayed on the control. Numeric values must be converted to string using the **`ToString()`** method of the value or **`string.Format`**.

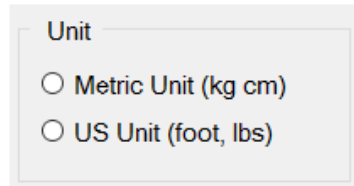
```
private void DisplayResults ( )
{
    lblResultYourBmi.Text = bmiCalc.CalculateBMI ( ).ToString ( "f2" );
    lblResultWeightCategory.Text = bmiCalc.BmiWeightCategory ( ).ToString ( );
    grpResults.Text = "Results for " + bmiCalc.Name;
}
```

The first line can be also written as:

```
lblResultYourBmi.Text = bmiCalc.CalculateBMI ( ).ToString ( "0.00" );
```

In both cases, the value of BMI will be rounded off upwards to two decimal positions.

4.1.5 Double-click on Metric Unit **RadioButton**, and write the following code in the handler method that VS prepares for you:



Unit

☐ Metric Unit (kg cm)

☐ US Unit (foot, lbs)

```
private void rbtnMetric_CheckedChanged (object sender, EventArgs e)
{
    if (rbtnMetric.Checked) //if false - unchecked
    {
        lblHeight.Text = "Height (cm)";
        lblWeight.Text = "Weight (kg)";
        bmiCalc.SetUnit (UnitTypes.Metric);
    }
}
```

4.1.6 Do similar coding for the other **RadioButton**.

4.2 How to read the value of height from the TextBox: See the figure on the next page

Text (**txtHeight.Text**) on Line 99 in the code excerpt below is a property in the class **TextBox** (.NET Framework) and it has a string data type. Properties are like methods but they are used just as variables (discussed in the next module).

The **SetHeight** method is to be written in the **BMICalculator** class.

The **BMIcalculator** class expects height in meters (m) or inches (in) and weight in kilograms (kg) or pounds (lb). Because we let the user give the height in centimeters (cm) and feet (ft) as a common convention, we need to convert them to m and inches, respectively.

Note: This conversion could be done in the **BMICalculator** as well – depends on how you would like to construct the **BMICalculator** class.

On Line 128, the second string sets the caption of the **MessageBox**. The method **MessageBox.Show** is overloaded several times. This means that the method is written in its class in several versions with the same name but with different order of method parameters.

```
91  /// <summary>
92  /// Read the input height from the dedicated textbox (txtHeight)
93  /// Validate input and if ok writes data to BMI object
94  /// Height must be in converted to m (if read as cm) or inch (from feet)
95  /// </summary>
    1 reference
96  private bool ReadHeight ( )
97  {
98      double outValue = 0;
99      bool ok = double.TryParse ( txtHeight.Text, out outValue );
100     if (ok)
101     {
102         if (outValue > 0) //height cannot be zero or negative
103         {
104             if (bmiCalc.GetUnit () == UnitTypes.American)
105             {
106                 bmiCalc.SetHeight ( outValue * 12.00 ); //feet to inches
107             }
108             else
109             {
110                 bmiCalc.SetHeight (outValue / 100.0); //cm --> m
111             }
112         }
113         else
114         {
115             ok = false;
116         }
117     }
118     if (!ok)
119     {
120         MessageBox.Show ( "Invalid height value!", "Error" );
121     }
122     return ok;
123 }
```


5. The Body-Mass Index (BMI) Calculator (BMICalculator class)

An effective way of reducing the number of instance variables is to let methods return output. This way, you do not have to use instance variables for saving output values and keeping the value updated all the time. Whenever an output value is desired by another method, it should call the method that returns that value.

5.1 BMI formulas:

Metric Units

$BMI = \text{weight in kg} / \text{height}^2 \text{ (in m}^2\text{)}$

U.S. Units

$BMI = 703.0 * \text{weight (in lb)} / \text{height}^2 \text{ (in inch}^2\text{)}$

where $\text{height}^2 = \text{height} * \text{height}$

To calculate the expression height to power of two, you can use **Math.Pow ()** but this is slower operation than using $\text{height} * \text{height}$. Use **Math.Pow ()** when the power exceed 3 or when the power has decimal part (ex. 3 to the power of 1.2). Otherwise, use multiplication.

Note: The **BMICalculator** class defines instance variables only for storing input, as input is data that comes from the user (or other objects). As far as output is concerned, the class uses methods to return output. Output data can always be reproduced, but that is not always possible with input data. The reason for having getter methods (methods that returns an input value) is that when MainForm passes input to BMICalculator to store the data, it may then lose that data. Whenever MainForm or any other object later wants to get the value from the BMICalculator, its calls the related getter method.

Getter methods usually are methods with no parameter that return the value of an instance variable.

Setter methods usually are void methods with one parameter (new value) that is to be saved into an instance variable.

Examples of these method types are given below.

The methods in the code examples below are to be written in the **BMICalculator** class. These methods are called by **MainForm** but can be called by any other objects (if you had a bigger application). Keep this in mind when writing a class – it should be usable for any other object (client).

```

/// <summary>
/// This class computes the BMI and has private fields that are
/// accesable via getter and setter methods.
/// </summary>
2 references
class BMICalculator
{
    private string name = "No Name";
    private double height = 0; //m, inch
    private double weight = 0; //kg, lb
    private UnitTypes unit;

    /// <summary>
    /// Getter method - connected to name-field.
    /// </summary>
    /// <returns>The value stored in the field name.</returns>
    1 reference
    public string GetName ( )
    {
        return name;
    }

    /// <summary>
    /// Setter method: Changes (overwrites) the value saved in the
    /// instance variable name by a new value that comes to this
    /// method through the parameter value.
    /// </summary>
    /// <param name="value">Input: new value to be saved in name</param>
    2 references
    public void SetName(string value)
    {
        //validate value before accepting it!
        if (!string.IsNullOrEmpty ( value ))
            name = value;
    }

    1 reference
    public double GetHeight()
    {
        return height;
    }

    3 references
    public void SetHeight (double value)
    {
        if (value >= 0)
            height = value;
    }
}

```

Write similar methods for the **weight** (**double**) and **nit** (**UnitTypes**) fields. In the setter method connected to unit, you don't have to do any validation.

Factors that related to a certain calculation must be stored manipulated in the object (class) that is responsible for the calculation and provided to other objects when they ask for it. The BMI categories depending on the value of BMI is to be saved in the BMICalculator class, not in the MainForm. This way, the BMICalculator will not be dependent on the MainForm and therefore will be usable by other objects.

```
/// <summary>
/// Method that sets the weight category depending on the value
/// of BMI. Notice that it call the method Calculate BMI before
/// setting the categories.
/// </summary>
/// <returns>Weight category based on the bmi-value.</returns>
1 reference
public string BmiWeightCategory ( )
{
    double bmi = CalculateBMI ( );
    string stringout = string.Empty;
    if (bmi > 40)
        stringout = "Overweight (Obesity class III)";
    else if (bmi < 40)

        //Continue with the rest

    return stringout;
}
```

Good Luck!

Farid Naisan