

CIS 345 – Business Information Systems Development II – Fall 2014

Assignment 7

Due on Blackboard: Friday, November 7 at 10 PM

Create an Average Calculator using a Windows Form Application within Visual Studio. This average calculator should prompt the user for the number of scores you want to enter. It should then *dynamically* create the specified number of text boxes and reveal a Calculate button. Pressing the calculate button should calculate the average for the scores entered. For testing purposes, limit yourself to creating 3-5 textboxes on the form so that you do not need to manage the GUI elements going off the Form. You do not need to plan for a larger number of text boxes to fit on your screen.

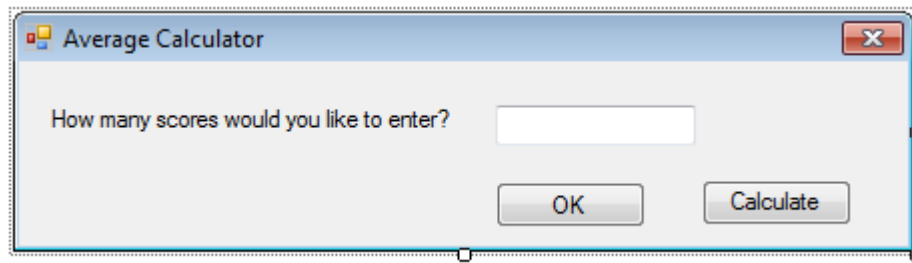
Skills developed: Utilizing Visual Studio Form applications, declaring and instantiating controls, manipulating controls dynamically, dynamically creating arrays of controls/objects.

Project Details and Requirements

A sample executable file is uploaded and available on Blackboard for you to test program behavior.

Project Name: AverageCalculator
Form Name: AverageCalculator

Form AverageCalculator [Design]



Properties: Since AverageCalculator inherits from Form, a host of inherited properties are already available. The following properties will need to be set at *design-time* using the *IDE*.

- Text (this property sets the form title): "Average Calculator"
- FormBorderStyle: FixedSingle
- MaximizeBox: False
- MinimizeBox: False
- Width: 450
- Height: 220
- StartPosition: CenterScreen

1) Set up the AverageCalculator Form as show in the figure above. Use suffixes within the control names so that when you work with them in code, you can easily distinguish a TextBox from a Label. e.g. "scoreTextBox" should easily be interpreted as the TextBox which contains a score, while "scoreLabel" should easily suggest that is the label accompanying a score.

Using the *Designer* in the *IDE*, create the following controls. Set the name and text properties as specified below. These controls are being designed at *design-time*.

- Label, to ask the user how many scores she would like to enter.
 - Name: numberOfScoresLabel
- TextBox, to obtain input of how many scores.
 - Name: numberOfScoresTextBox
- Button, to allow user to press a button after entering the number of scores
 - Name: okButton
 - Text: OK
- Button, to allow user to the user to calculate the average
 - Name: calculateButton
 - Text: Calculate
- Label, to prompt the user to enter scores and to use as a message to output scores
 - Name: scoresLabel

Form AverageCalculator [Partial Class]

1) Within the AverageCalculator partial class, add a private instance variable, scoreTextBoxArray, which is an array of TextBox objects. Do not instantiate the array. ***Do NOT add any code in the designer.cs or Program.cs classes.***

METHODS:

Methods should go inside the class as with all Classes. The precise location (top, bottom, etc.) does not matter so far as methods are located within the class block.

2) Load event handler for the Form

Create a Load event handler for your form.

Purpose: The purpose of this event handler is to set up some of the properties so that the Form is ready to be presented by the user.

General Method Logic:

- Set the Visible property of calculateButton to false. ***You are setting this in code, not using the Designer. This property is being set at “run-time”, not “design-time.”***

3) okButton_Click event handler

Create the Click event handler for the OK button.

Purpose: This method will read the number of scores the user wants to enter and call the CreateTextBoxes method. It will also make changes to the look of the Form and controls.

General Method Logic:

- Declare an integer, numberOfTextBoxes
- Store the number of text boxes to be created into the integer variable. You will still need to use Convert.ToInt32() method. However, now, instead of getting user input from the ReadLine() method, you will access it using the Text property of the TextBox. i.e. *TextBoxName.Text*
- Set the height of the Form to 500 pixels. Access the Form's property using the "this" keyword i.e. "this.Height"
- Set the visible property of calculateButton to True
- Set the Enabled property of okButton to false
- Call CreateTextBoxes and pass it the integer, numberOfTextBoxes as a parameter.

4) CreateTextBoxes method

Parameters: one integer called number.

Purpose: This method will receive the number of textboxes to be created as a parameter. It should instantiate the array of textboxes (which is declared as an instance variable) with the specified number. It should further loop through the entire array and create instances of the TextBox class, set their Top and Left property and add them to the Form.

General Method Logic:

- Instantiate the scoreTextBoxArray array and use the parameter “number” as the size of the array.
- Declare an integer, verticalTopPosition, with an initial value of 150.
- Loop through the entire array, scoreTextBoxArray
 - Instantiate a new TextBox using the default constructor for the TextBox class (it is called TextBox()) and assign it to the current element of the scoreTextBoxArray array (scoreTextBoxArray[i]). *Don't forget to use “new” with the constructor!*
 - Set the Left property of the TextBox to 20. You will access the Left property by referring to the current element of the array i.e. scoreTextBoxArray[i] dot Left.
 - Set the Top property of the TextBox to verticalTopPosition.
 - Increment verticalTopPosition by 50.
 - Add the TextBox to the controls collection of the Form.
 - Use this.Controls.Add(scoreTextBoxArray[i]). This passes scoreTextBoxArray[i] which is one instance of a TextBox to the Add method, so that it is added to the Form. *This is exactly how we wrote our AddStudent method for the Gradebook exercise.*

5) calculateButton_Click event handler

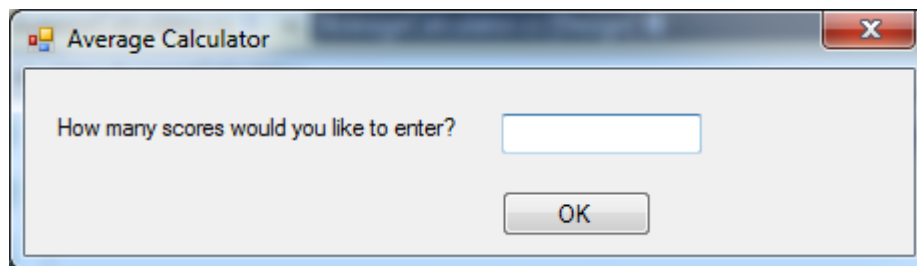
Create the Click event handler for the calculateButton.

Purpose: This method will loop through the array of textboxes, read their Text property, and create a total. It will calculate the average and display it to the user.

General Method Logic:

- Declare double variables total and average. Initialize them to 0.0.
- Loop through the scoreTextBoxArray array and calculate the grand total. Again, this time, you will still have to use Convert.ToInt32() to convert strings to integers but you are not obtaining input using ReadLine(). Instead, you have to access the Text property of each element of scoreTextBoxArray. i.e. scoreTextBoxArray[i].Text, which you will have to convert to an integer and then add to the total.
- Calculate the average. Remember to cast the total as a double.
- Set the Text property of scoresLabel to display the total.
 - Use string concatenation to construct the display text, e.g., "Average is " + Convert.ToString(average)
 - If you want to use placeholders, use the Format method of the String class, e.g., String.Format("{0:F2}", variablename). That will return a formatted string you can use to set the property. You cannot use placeholders in the Text property like you could use with WriteLine methods.

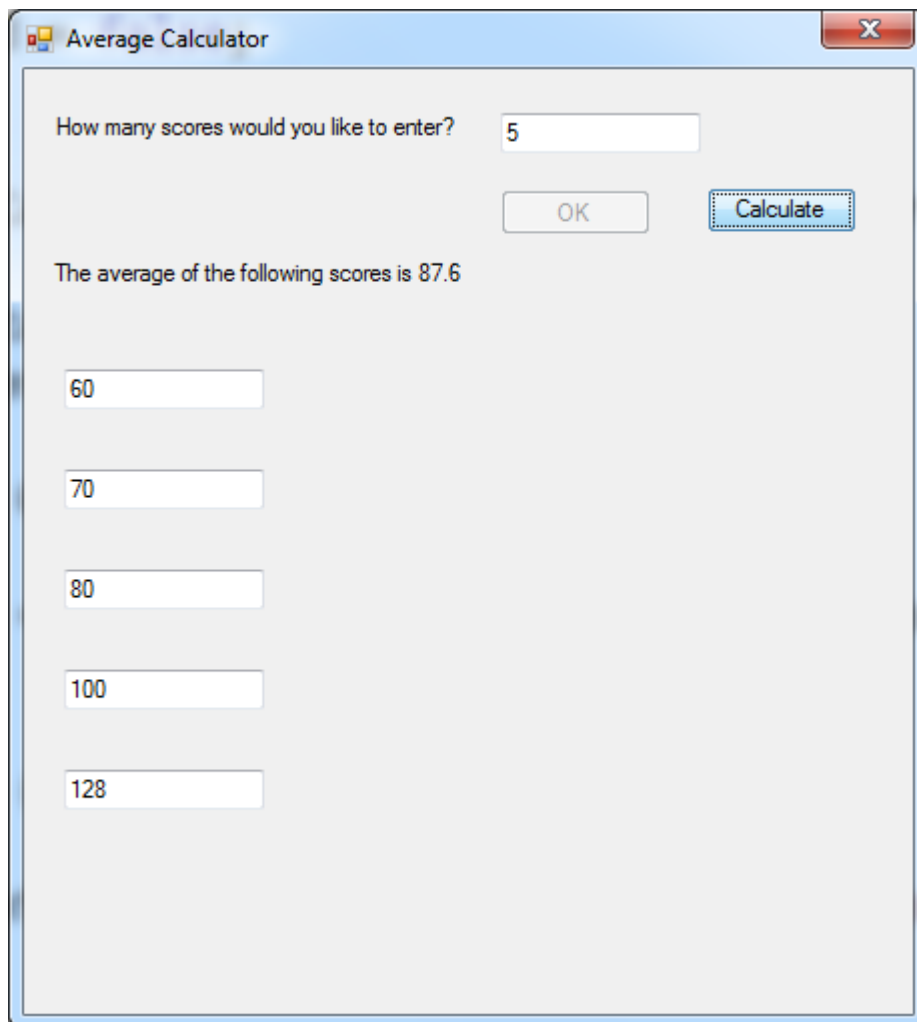
Sample Output:



Average Calculator

How many scores would you like to enter?

OK



Average Calculator

How many scores would you like to enter?

OK Calculate

The average of the following scores is 87.6

Optional Enhancements

- Properly store data in objects rather than having data stored in TextBoxes. *This is how object-oriented programs with GUIs ought to behave:*

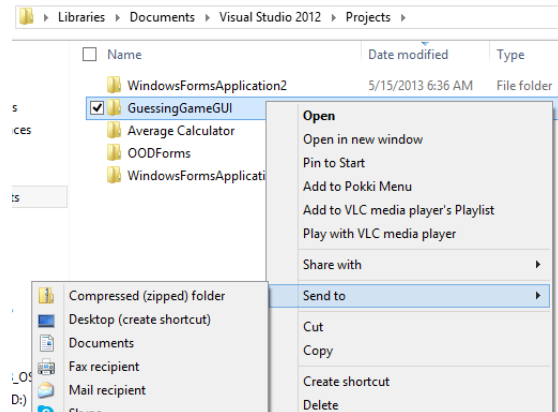
Add your Score or your old Student class, which had room to enter midterm/final scores, to your current project.

- Create an array of Score/Student objects of the same length as the TextBoxes. Copy each score from the TextBox into objects you have in the array.
 - Compute the average using the scores stored in the objects rather than reading them directly from the Textboxes.
 - The advantage of this approach is that objects are far more powerful and can do a lot. In fact, don't you have an Average method already developed for a Score / Student class? That would be software reuse in action!
- Use your existing ReadInteger method, which does exception handling and make it work here within a Utilities.cs class.
 - The ReadInteger method cannot prompt users anymore using the Console ReadLine methods. So, it requires reimagination!
 - The good news is that you can let people have as many attempts as possible (whereas previously we had restricted them to a maximum of 3 errors), so all you need to do is prevent them from proceeding if the text isn't a valid number.

Submission Instructions

Submission should be made using a zip file that contains all of the Visual Studio C# project files. You will need to **zip the entire project folder** along with the .sln and .suo files. The folder will automatically contain the class source files as well as the executable file that is generated in \ProjectName\bin\Debug folder. Upload file to the Blackboard assignment drop box.

Zip the entire top-level folder by right-clicking the folder and selecting Send to | Compressed (zipped) folder.



Using built-in windows zip tools: <http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files>

Make sure you check the following. Your grade is dependent on all these criteria being met.

- You have included your name as a comment within your class.
 - e.g. “// Assignment 7, Jane C. Smith, CIS 345, Tuesday 9:00 AM”
- Zip filename is: FirstNameLastName_Assignment7.zip
- Your code is commented and you are using all prescribed programming conventions.
- Your code utilizes PascalCase and camelCase as appropriate.

Verify your zip file before you submit

- Check for actual class files being present in the folder before you zip it.
- Check your zip file size after zipping – if it is 1K, it likely contains only a shortcut.
- Uncompress your zip file before submitting and verify that files are present.
- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.
- Make sure you have submitted your file and not just saved a draft on Blackboard. A blue clock indicates a submission in progress, i.e. a draft, not a submission.

This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.