IVY TECH COMMUNITY COLLEGE

**SDEV 140 / Introduction to Software Development**

**Software Development Project – Part 2**

# Student’s name \_\_\_\_\_Spencer\_Goldwin\_\_\_\_ Section \_\_\_\_\_\_\_ Instructor’s name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If you’re using Visual Studio Community 2015, as requested, the instructions below should be exact but minor discrepancies may require you to adjust. If you are attempting this assignment using another version of Visual Studio, you can expect differences in the look, feel, and/or step-by-step instructions below and you’ll have to determine the equivalent actions or operations for your version on your own.

**INTRODUCTION**: In this assignment, you will develop some of the logic for, and then work with, the Order Input form for the DroneDogs project.

There are two parts to the assignment. In Part 1, you will complete the Algorithm Development Exercise, which will help you determine what programming steps you will need to create in Visual Basic. In Part 2, you will work with a partially complete Drone Dogs project in Visual Basic to implement additional functionality, test it, and submit your completed version of the project as a zipped folder.

**PRELIMINARY:** You should have read the first five chapters from the Books 24x7 Visual Basic textbook prior to attempting this assignment. Be prepared to refer back to that text, to search for tutorial information online as needed (there is a LOT of it), and to ask your instructor questions. You are encouraged to experiment with code samples and examples from the text and other sources to understand better how it works.

**ASSIGNMENT PART 1:**

Complete the Algorithm Development Exercise below. Enter statements where instructed in red. You’ll submit this document with your edits as part of the assignment submittal.

**ALGORITHM DEVELOPMENT EXERCISE:**

To process the customer order, you will need to define constants and variables. This is different than in Python, in that you have to declare data types for both, and you have to declare your variables before using them. For numeric constants or variables, please use the data types **Integer** for whole numbers, and **Double** for values with decimal amounts. The Hungarian Notation prefix for Integer types is **int**, and the prefix for Double types is **dbl**.

1. Let’s start with constants. There are two values that will not change during the program: the cost per hot dog, and the sales tax rate. Since they do not change, you can declare them as **constants** in your program so that the compiler will let you know if you inadvertently try to change them programmatically.

Both the hot dog price and sales tax rate may contain decimal amounts, so they should be declared as type **Double** constants. In Visual Basic, a constant is denoted by the keyword **Const**. By convention, constant names are usually written in all UPPER CASE, with underscores to separate words. You also have to assign the value to a constant AT THE TIME you declare it (because, by definition, the value of a constant cannot be changed).

For example, the declaration of a constant Double representing an delivery charge rate of 3.5% could look like this in Visual Basic:

**Const DBL\_DELIVERY\_CHARGE\_RATE as Double = 0.035**

**YOU: Write statements to declare and initializing constants for:**

**A sales tax rate of 7%: [Const DBL\_SALES\_TAX\_RATE as double = 0.07]**

**The price of a DroneDogs hot dog at $1.99: [Const DBL\_DRONE\_DOG\_PRICE as double = 1.99]**

1. Next, the user will enter the number of each kind of hot dog desired into one of three text boxes on the input form. You’ll need variables for each of these; you might also choose to declare a variable for the total number of dogs ordered (because they are all the same price).

Because people must order a whole number of hot dogs, not fractions of a hot dog, you should declare these as **Integer** variables. (You don’t have to set an initial value for these variables, though it is common practice to initialize all variables – to zero, if no other value is more appropriate.)

Variables in Visual Basic are declared using the keyword **Dim**.

For example, if DroneDogs were selling tacos, an appropriately named Integer variable for the number of tacos could look like this:

**Dim intNumTacos as Integer**

**YOU: Write statements declaring integer variables for:**

**The number of beef dogs ordered: [Dim intNumBeefDogs as Integer]**

**The number of pork dogs ordered: [Dim intNumPorkDogs as Integer]**

**The number of turkey dogs ordered: [Dim intNumTurkeyDogs as Integer]**

**The total number of hot dogs ordered: [Dim intNumTotalDogs as Integer]**

1. Next, you need variables for the subtotal, sales tax amount, and total cost. These are values that your program will calculate, based upon the constants and variables declared above. These should all be **Double** types, because they will contain decimal amounts. Generally, within programs, decimal variables are called floating-point types – Double is an example of a floating-point type.

For example, if there were a delivery charge based on the subtotal, an appropriately named variable for that charge could look like this:

**Dim dblDeliveryCharge as Double**

**YOU: Write statements declaring floating-point variables for:**

**The subtotal for the order: [Dim dblSubTotal as double]**

**The amount of sales tax: [Dim dblTotalSalesTax as double]**

**The total cost of the order: [Dim dblTotalCost as double]**

1. Finally, you need to do the arithmetic to determine the total number of hot dogs orderd, find the subtotal for that many hot dogs based upon that constant price, compute the sales tax, and compute the final total. Just as in Python, Visual Basic calculations and assignments require that the receiving variable is alone on the left-hand side of the equals sign, and the computation is on the right-hand side. You have already declared your variables.

For example, if you were determining the gross pay for an employee based upon hours worked and hourly rate, an assignment statement could look like this (if the variables dblGrossPay, dblHoursWorked, and dblPayRate were previously declared and had values).

**dblGrossPay = dblHoursWorks \* dblPayRate**

**YOU: Write statements that will compute the correct values for:**

**The total number of hot dogs ordered: [NumTotalDogs = NumBeefDogs + NumPorkDogs + NumTurkeyDogs]**

**The subtotal (before taxes): [dblSubTotal = NumTotalDogs \* Const DBL\_DRONE\_DOG\_PRICE]**

**The amount of sales tax: [dblTotalSalesTax = dbsubtotal \* Const DBL\_SALES\_TAX\_RATE]**

**The total cost of the order: [dblTotalCost = dblSubTotal + dblTotalSalesTax]**

**ASSIGNMENT PART 2:**

Follow the steps below to finish the DroneDogs project using a starter project provided to you. You’ll open that attached project file for the Drone Dogs project and complete the steps requested. After you’re finished, you’ll have a functioning DroneDogs application, which includes an order input form that calculates the subtotal, sales tax, and total cost for hot dogs from DroneDogs.

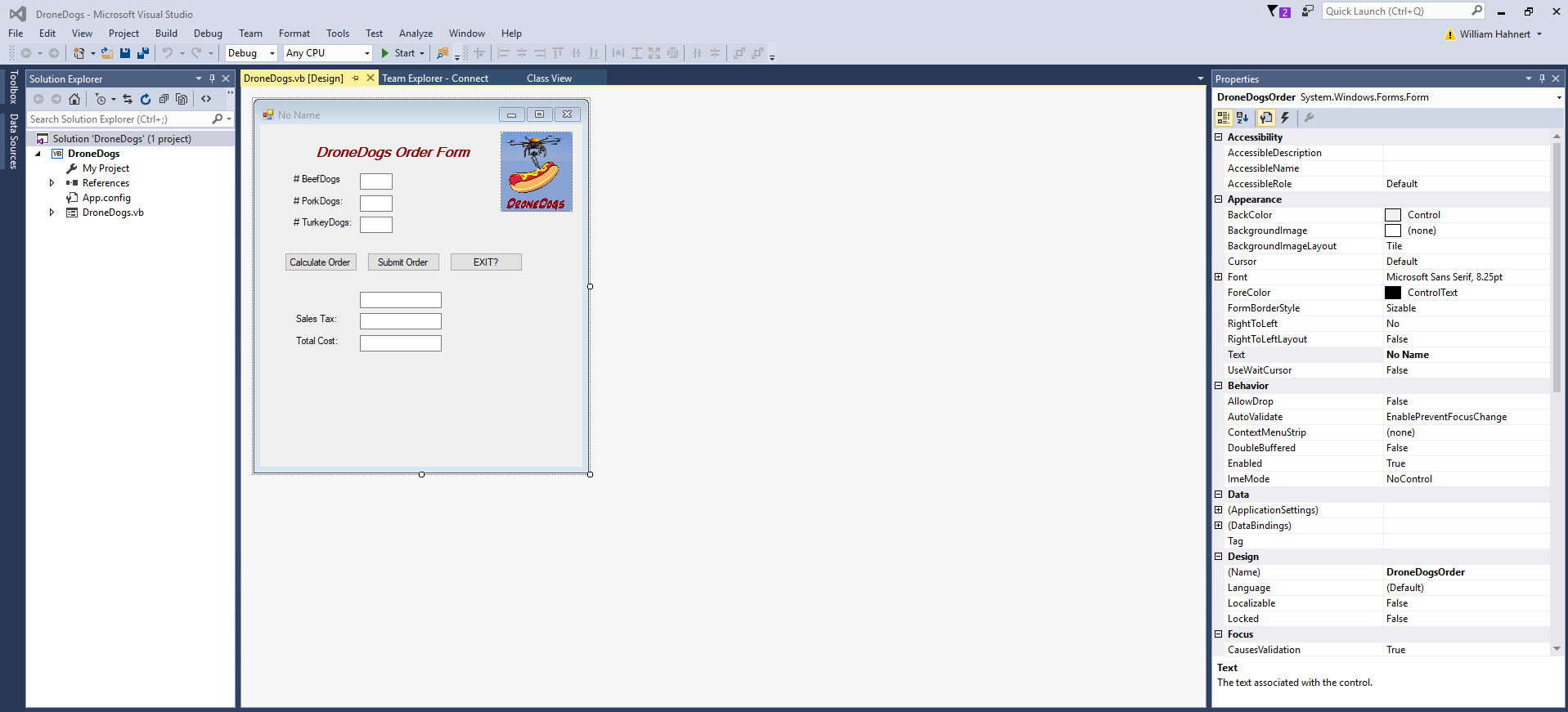
Be sure to aside enough time to do this. You may have to search for additional help in the Books 24x7 text, or find other sources online or via your instructor.

**DRONEDOGS PROJECT GUIDELINES:**

1. Download and unzip the project folder provided in this module to a location on your computer.
2. Start Visual Studio, and open the Visual Basic project file **DroneDogs.sln** inside the root level of the project folder. That project is a Windows Forms application named DroneDogs and has a mostly complete order form already built into it.
3. Use the Build menu, Rebuild Solution to rebuild the DroneDogs project - this will compile the program into executable form or will let you know of any syntax errors you have (there should be NONE at this point). Once you have rebuilt DroneDogs successfully, select the Debug Menu, Start Without Debugging to run the program. Your input form should appear but you won’t be able to do much with it. You’ll have to exit the ‘run’ by clicking the Close button (X) at the top right of the form.

Use the visual editing capabilities of the Visual Studio IDE to make the following changes to the form and controls on it (buttons, labels, etc.) itself. You’re editing the attributes (variable values) of the various controls – setting the default values of those attributes. Note that these can be changed again, programmatically, at run time.

1. If you cannot already see the DroneDogs form displayed, in the Solution Explorer Window, double click on *DroneDogs.vb* to open the visual editor for the form. You’ll see it tabbed in the main window of Visual Studio as *DroneDogs.vb [Design]*
2. Right-Click on the entire form and view it’s **Properties**. You should see a Properties window open on the right for *DroneDogsOrder System.Windows.Forms.Form*

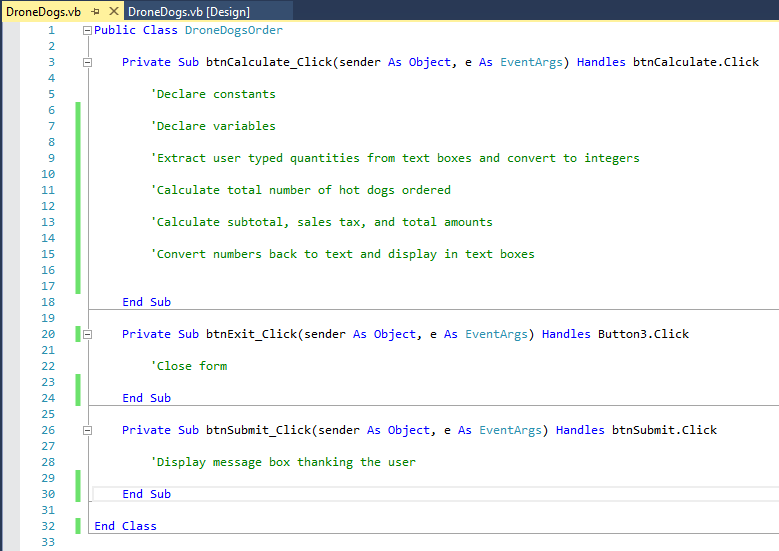


1. Change the **Text property** of the form from *No Name* to your name (for example – Fred Garvin). You should see the name on the title bar of the form change immediately!
2. Check each of the labels, text boxes, buttons and picture box controls on the form to make sure that the **Name property** and **Text property** are appropriate. For example: The first label should be named lblBeefDogs, and it’s text property should be # Beef Dogs.  
     
   If either property is incorrect or poorly/inconsistently chosen, (e.g. a button named Button1 instead of something meaningful like btnCalculateSubTotal – or – a button used to calculate the subtotal, but which shows “Calculate Income Tax” on screen), then change the Name property and/or Text property as appropriate to what the Order Input form indicates.
3. For the last three text boxes, change the **Read-Only property** to True. This will ensure that the user is not able to click into those text boxes when the program runs in order to type values there - they are only used by you, programmatically, to display output information.
4. You’ve noticed by now that the Subtotal Text box has no label. Find the Toolbox (if you don’t see it, you can choose it from the View Menu). Double click on Label and you’ll see Label1 added to your form. Drag and resize that Label to the appropriate location, and change its Name and Text properties to appropriate values.

MAKE SURE YOU HAVE ALL ITEMS ABOVE COMPLETE BEFORE CONTINUING ON

So far you’ve used the visual editing properties of Visual Studio to edit the initial look of the application. Now you have to do some actual coding to handle things that happen/change during execution.

1. To code the Calculate Order button, double-click that button in the visual editor (on the form), and the code header will appear in the code window. You’ll see the following:



NOTE: If you double click on any of the controls on the form except for the three buttons you see code fragments for above, empty ‘event handlers’ for those controls will automatically be added to the code. You can just delete those event handlers (from Private Sub XXXXX through End Sub).

* 1. Declare the variables and constants, by adding the code from the Algorithm Development Exercise Steps #1-4 above into the appropriate locations. You’ll see comments such as **'Declare constants** or **'Declare variables** above each location where you have code to add. (Normally, you’d be adding these comments yourselves).
  2. You’ll need to extract the user typed text from the appropriate text boxes and convert those values into the Integer variables you declared. Enter those statement where the comments indicate.

To convert the entries in the text boxes into numerical amounts that can be used in calculations, there is a function called **Convert.ToInt32**. For example, if you want to convert the number of beef dogs from the beef dogs text box into its corresponding integer variable, depending on what you named that text box and variable, the statement might be:

**intNumBeef = Convert.ToInt32(txtBeefDogs.Text)**

* 1. You’ll have to write the code to calculate the total number of hotdogs ordered, using the appropriate variable you’ve already declared to hold the total, and adding up those values you just extracted from the text boxes.
  2. You’ll have to write the code to use those numerical values and the constants you’ve already declared, to calculate the Subtotal, Sales Tax, and Total Cost amounts. For instance, you might calculate a delivery charge as:

**dblDeliveryCharge = DBL\_DELIVERY\_CHARGE\_RATE \* dblOrderSubtotal**

* 1. After you have calculated the Subtotal, Sales Tax, and Total Cost amounts, you’ll need to convert those numerical amounts back to text, so that you can display them in the text boxes, using a function called **ToString**. To make your money amounts display with a dollar sign and 2 decimal places, there is an argument **("c2")** meaning currency format, 2 decimals. For example, to display the subtotal amount in the subtotal text box, depending on what you named them, the statement might be:

**txtOrderSubtotal.Text = dblOrderSubtotal.ToString("c2")**

1. To code the Submit button, double-click it, and its code header will appear in the code window. Enter a comment that you are displaying a message box thanking the user. Then, the only code line you need for the Submit button is the line:

**MessageBox.Show("Thank you for ordering your meal from DroneDogs!")**

1. To code the Exit button, double-click it, and its code header will appear in the code window. Enter a comment that you are exiting the program. Then, the only code line you need for the Exit button is the line:

**Me.Close()**

1. After you enter your code, select File Menu, Save All; and then use the Build menu, Rebuild Solution to rebuild DroneDogs. This will compile the program into executable form or will let you know of any syntax errors you have. If you have errors, look closely at the lines that are mentioned and read the error message. Check your spelling, spacing and punctuation and determine where the error is.
2. When the program compiles (builds) successfully, select the Debug Menu, Start Without Debugging to run the program. Your input form should appear, and you should be able to enter the number of each of the three types of hot dogs. NOTE: Enter 0 for any type of hot dog you are not ordering (do not leave it blank). Click Calculate Order, and the program should correctly compute the subtotal, the amount of sales tax, and the total cost of the order. If there are error messages, read them carefully, look at how you spelled your variable names, and check punctuation. Once it runs, use a calculator to verify that your program actually runs correctly – you may have logic errors that cause your program to calculate the wrong values, even if it ‘runs’. Correct all errors, save, build and run your program again until you get it working correctly.

There is a sample screen shot below.



1. Verify that your program works for the sample screenshot data, and for other data values that are appropriate to test with.
2. Once you have verified that your program works, you can exit Visual Studio. Navigate to the location where you’re saving your work and created a zipped folder for the entire project DroneDogs folder, just like you did in Python IDLE. The DroneDogs.sln file will be one of the items in the root of the main DroneDogs folder.

Attach that zipped folder to submit for Part II of this assignment.