DUE: 11/12/2013

Project #2—Fuzzy Dice Order Form

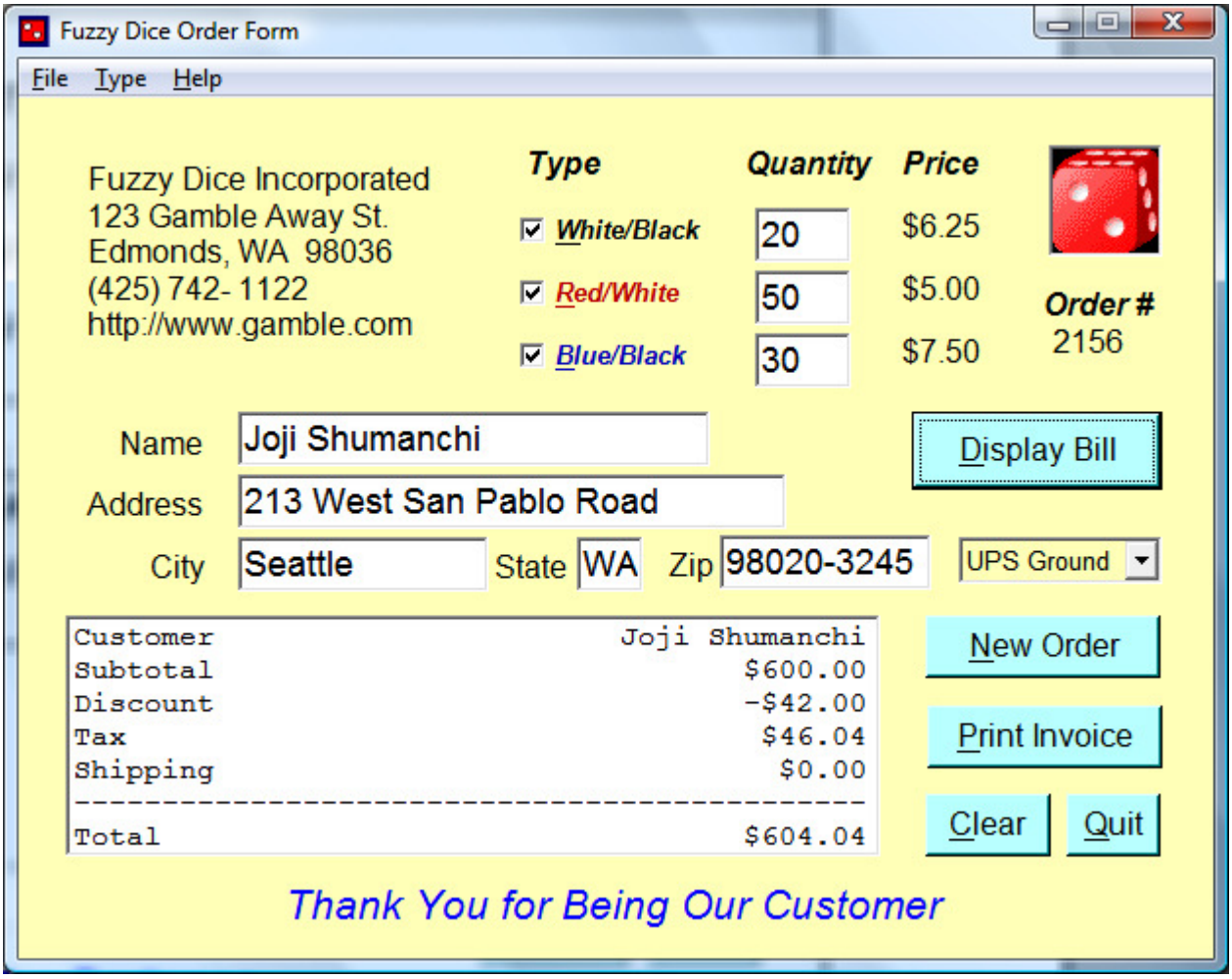
Fuzzy Dice, Inc. produces and sells three types of dice: white/black for $6.25 per pair, red/white for $5.00 per pair, and blue/black for $7.50 per pair. Write a C# application that allows users to process orders for Fuzzy Dice (sample GUI is provided below). The application should calculate the total price of the order, including tax and shipping. Tax is 8.25% and shipping is done via UPS with the following table:

Shipping Charges	
UPS Ground	$2.95
UPS 3 Days	$6.95
UPS Next Day	$15.95
For orders over 20: Free UPS Ground Shipping	

Your C# program should display the order form. The order number, customer's name and address, and the quantities for each type of dice should be entered into the program via text boxes (you may use a masked text box for the order number). Use check boxes for the type of dice ordered. When a Display Bill button is clicked, the customer's name and the invoice should be displayed in a list box, as shown in the sample run below. Note that the discount of 7% applies if customers order more than $500 (before tax and shipping).

1. The project is worth 30 points and it will be graded on accuracy, completeness, presentation, comments, and the like.
2. Make sure that your name, project number, and due date appear as comments in your code (preferably on top).
3. Submit a zipped file with all files and subfolders via the Canvas by the deadline.

EXTRA CREDIT (Optional—3 points). Enhance the form by adding an about form, splash screen and ability to print the invoice. Consider using a combo dropdown for the customer and read the customer information from a text file.



Project Guidelines

Problem Statement: The following list reviews the GUI design and programming guidelines you have learned so far. You can use this list to verify that the applications and projects you create adhere to the standards outlined herein.

- Information should flow either vertically or horizontally, with the most important information always located in the upper-left corner of the screen.
- Maintain a consistent margin of two or three dots from the edge of the window.
- Related controls should be grouped together using either white space or a group box.
- Position related controls on succeeding dots.
- Buttons should either be centered along the bottom of the screen or stacked in either the upper-right or lower-right corner.
- If the buttons are centered along the bottom of the screen, then each button should be the same height; their widths, however, may vary.
- If the buttons are stacked in either the upper-right or lower-right corner of the screen, then each button should be the same height and the same width.
- Use no more than six buttons on a screen.
- The most commonly used button should be placed first.

- Button captions should:
  - ✓ be meaningful
  - ✓ be from one to three words appear on one line
  - ✓ be entered using book title capitalization
  - ✓ contain & for "hot" keys activation
- Use labels to identify the controls in the interface.
- Identifying labels should:
  - ✓ be from one to three words
  - ✓ appear on one line
  - ✓ be aligned by their left borders
  - ✓ be positioned either above or to the left of the control that they identify
  - ✓ end with a colon (:)
  - ✓ be entered using sentence capitalization
  - ✓ have their BackStyle property set to 0-Transparent
  - ✓ have their BorderStyle property set to 0-None
- Labels that display program output (for example, the results of calculations):
  - ✓ should have their BorderStyle set to 1-Fixed Single
  - ✓ should have their Appearance property set to 0-Flat
  - ✓ can have their BackStyle property set to either 0-Transparent or 1-Opaque
- Align labels and controls to minimize the number of different margins.
- If you use a graphic in the interface, use a small one and place it in a location that will not distract the user.
- Use no more than two different font sizes, which should be 10, 12 or 14 points.
- Use only one font type, which should be a sans serif font, in the interface.
- Avoid using italics and underlining.
- Use light colors such as white, off-white, light gray, pale blue, or pale yellow for an application's background, and very dark color such as black for the text. Alternately, use dark colors for the form's background with light colors for the text—contrast is the name of the game.
- Use color sparingly and don't use it as the only means of identification for an element in the interface.
- Set each control's TabIndex property to a number that represents the order in which you want that control to receive the focus (begin with 0).
- A text box's TabIndex value should be one more than the TabIndex value of its identifying label.
- Lock the controls in place on the form.
- Assign a unique access key to each essential element of the interface (text boxes, buttons, and so on).
- Document the program internally.
- Use variables to control the preciseness of numbers in the interface.
- Set the form's BorderStyle, MinButton, MaxButton, and ControlBox properties appropriately:
  - ✓ Splash screens should not have a Minimize, Maximize, or Close button, and their borders should not be sizable.

  - ✓ Forms that are acting as dialog boxes should have a BorderStyle property of 3-Fixed Dialog.
  - ✓ Forms other than splash screens and dialog boxes should always have a Minimize button and a Close button, but disable the Maximize button.
  - ✓ Forms other than splash screens and dialog boxes typically have a BorderStyle property setting of either 1-Fixed Single or 2-Sizable.
- Test the application with both valid and invalid data (test the application without entering any data; also test it by entering letters where numbers are expected). Test the application for correct results with known data.
- In the InputBox function, use sentence capitalization for the prompt, and book title capitalization for the title.
- Center a form by including the appropriate formulas in the form's Load event.
- Display an hourglass or an arrow and an hourglass if a procedure will take a long period of time.
- Remove excess or duplicated code from the application.

Grading Guidelines: Your project will be graded on the following:

- Correct solution to the proposed problem.
- GUI design (as outlined above).
- Elegance, simplicity, style, and readability of the algorithms and code.
- Comments:
  - ✓ Use of header.
  - ✓ Description of project.
  - ✓ Description of procedures.
  - ✓ Description of complicated code.

Hints for Project #2: *Given that your GUI resembles the given one, follow these guidelines:*

1. Consider using a masked text box with appropriate mask for the quantity, order number, state and zip. Alternatively, consider using numericupdowns for the quantities.
2. Use Courier New for font in the list box and use a formatting string to display the bill with the appropriate justification as shown in the sample GUI.
3. Show the price of each type die at form_load using predefined class-level constants. This allows for easy change of prices.
4. The Clear button differs from the New Order button in that the Clear button clears everything but order number and the customer information. The New Order button clears everything, including the customer information and increments the current order number by one.
5. Disable initially all quantity text boxes and enable each one with focus set to it only when a corresponding check box is selected (or chosen from the Type menu). Similarly, un-checking a check box should clear and disable the corresponding text box.

6. Calculate subtotal first, based on number of dice ordered—note that each color combination dice have a different price. Make sure you apply discount first (7% off the subtotal if it is >= $500), if any, before calculating the tax (8.25%).
7. Calculate the shipping charges next—it helps to draw a flowchart here. Essentially, consider the shipping choice in the combo box (check for the text property of the combo box). If it is “UPS Ground” then the shipping charge will be 0 if 20 or more items are ordered else it is $2.95. Otherwise, if the shipping choice is “UPS 3 Days”, the shipping charge is $6.96, else it is $15.95.
8. Calculate the total by adding to the subtotal the tax and the shipping and subtracting the discount. Use a formatting string for displaying columns with print zones and justification in a list box.
9. Provide input validation. Do not allow wrong type (use KeyPress or Validating event handler or Try/Catch blocks) or excessively large number (you decide what is that maximum) or negative number of dice, for example.
10. Declare constants for all constants used (such as 8.25% tax, 7% discount, dice prices, shipping charges, etc., including maximum number of dice to order, minimum constants for discount or for free shipping).
11. Make the check boxes and the shipping combo box dynamic—any change in their state should automatically recalculate the invoice without having to press the Display Bill button.
12. Make sure the menus are functional without repeating the code. The sample GUI has a File, Type, and Help main menu items. File contains Clear, New Order, Separator, Print, and Quit. Type contains White, Red, and Blue, and Help contains About for an optional About form. All menu choices can be activated with a corresponding “Hot” key combination, such as Alt-F for File.
13. Set the Tab Order index appropriately so that the form can be filled in proper order. You can invoke the Tab Order from View menu.
14. For test purposes do not require completion of customer information.
15. Optionally and for extra credit, consider adding an explanation via an About form or a MessageBox that each price is for a pair of dice—thus a quantity of one (1) yields two dice.

Pay attention to the following criteria for grading:

/*~~~~~~
'Comments by the prof:
'Great effort. Here are suggestions for improving:
'1. Use comments through the program, not just at the top heading
'Use comments to describe each procedure and complicated code.
'2. Convert entries in textboxes from strings to numbers with the appropriate convert functions:
Convert.ToByte, Convert.ToDecimal, etc.
'3. Program crashes on empty.

'4. Make the Display Bill button default.
'5. Correct the formulas—results are incorrect.
'6. Apply 8.25% tax to bill.
'7. Use Const definition for all constants and follow naming convention of ALL CAPITALS.
'8. Apply 7% discount on orders over $500.
'9. Provide Clear, Quit, New Order and Print Invoice (disabled if not implemented) buttons.
'10. Use Application.Exit() to exit program.
'11. Use a ComboBox with DropDownList property for the three UPS shipping rates.
'12. Free shipping applies if order has over 20 items and UPS ground is chosen for shipping.
'13. Name the project/form appropriately; a generic WindowsApplication1, Project1 or Form1 just won't do.
'14. Use the Heading.txt heading for the project.
'15. Quantity textboxes are enabled only if corresponding checkbox is checked.
'16. Display bill as currency.
'17. Follow the naming convention for naming variables and constants.
'18. Avoid excessive use of class variables-declare variables inside procedures.
'19. Clean up the empty procedures code.
'20. Separate code in main procedure into 4 section: Declaration, Assignment, Calculation, and Display.
'21. Implement input validation for all text boxes (avoid -3 dice, for example).
'22. Select UPS Ground as default text for shipping combo box.
'25. Declare and use variables for input.
'26. Use the Clear() method to clear text in a text box.
'27. Nice effort on the printing--printing the bill alone would suffice (without all controls).
'28. Text boxes should be empty initially.
'29. Dice quantities should be integers; all $$ variables should be Decimal type.
'30. Replace if (chkRed.Checked = true) with if chkRed.Checked.
'31. Implement menus & printing. Synchronize Types in menu with check boxes.
'32. Use mono-spaced font such as Courier for the list box--it helps set columns correctly.
'33. Use print zones with string formatting to set up columns in display list box.
'34. Add an image to the form.
'35. Check the Tab Order on your form.
'36. Clear should clear check boxes, list box and quantity text boxes—leave customer information intact.
'37. Disable maximization of form.
'38. Make check boxes and combo box dynamic.
'39. Take care of unused local variables warnings.
'40. Why require order of all type of dice? Allow for orders with only one type of dice.
'The ones that apply to your project are:
'3, 7, 15, 21, 30, 31, 37, 40
'
'23+2=25/30
'~~~~~~*/

