**PLEASE READ**

The requirements for all assignments are contained in this document. Please make sure to read the assignment carefully and meet all requirements. Don’t forget that in the “Files” section in canvas, there is a file named “Files For Students.zip”. When extracted there will be files to help with various assignments, such as database you may use and SQL help for those databases. Also make sure to reference the below sample programs I have created to help with the listed assignments.

After the requirements for each assignment, I have listed some of the common deductions that student miss for each assignment. Using those listings, you should be able to check your assignments and make sure you have met all requirements correctly.

**Here are the sample programs that will be helpful for each assignment**

**Assignment 1**

Chapter 3 Console

Chapter 3 Example Messagebox

**Assignment 2**

Chapter 5 and 6 Example

Chapter 7 Methods Random Guess

**Assignment 3**

Chapter 8 16 26 27 28 Array

**Assignment 4**

Chapter 10 Classes and Objects

Chapter 11 TicTacVisualElements

**Assignment 5**

Chapter 10 Classes and Objects

Chapter 13 Exceptions

Chapter 14 Windows Form Timer

Chapter 14 WPF DateTime and Timer

Math Game Outline Windows Forms

Math Game Outline WPF

Math Game Sort High Scores

PassingDataAround

**Assignment 6**

Chapter 10 Classes and Objects

Chapter 15 User Control and Event

Chapter 15 Visual Controls

Chapter 15 WPF User Control Example

Chapter 21 DB Class Example

Chpt 21 DB and Add Object Combobox

**Assignment 7**

Threading

Threading WPF

Chapter 17 Files and Streams

**Group Assignment**

Chapter 21 DataGrid Example

Chapter 21 DataGridView adding objects

Advanced WPF Concepts

**Assignment 1**

Make sure to comment each individual class attribute and method using an XML comment. Just type three forward slashes.

**First Part of assignment 1 (Program 1)**

Create a Windows based program that allows the user to do the following:

1. Type a message in a textbox that will be displayed in a Message Box.
2. Display a given type of Message Box, based on the user’s selection.
3. Display what button the user selected on the Message Box.

Your form should have 3 textboxes that the user types a message into, and buttons next to each textbox that when clicked will display a different type of Message Box. Next to each button there should be a label describing what will happen when the button is clicked. Each button should display a Message Box that has different MessageBoxButtons on it. Also, each Message Box should have different MessageBoxIcons on it. The result of each Message Box should also be displayed on the form. So, for instance if a Message Box with an “OK” button is displayed, when the “OK” button is clicked, the form should show that the button “OK” was clicked.

**Second Part of assignment 1 (Program 2)**

Create a console-based program that allows the user to enter two numbers and then prints the results of addition, subtraction, multiplication, division, and the remainder of the two numbers. Then the program prints whether the first number is less than the second number, then prints whether the first number is greater than the second number, then prints whether the first number equals the second number. For this assignment, you do not have to validate user input. For example, if the user enters the letter “A” the program will crash. In later assignments, you will be required to validate input, but for this assignment you do not. However, if you want to practice, I suggest you first finish the assignment, then go back and try to add in user validation.

Here is a sample output of the program:

Please enter the first number: 10

Please enter the second number: 3

10 + 3 = 13

10 - 3 = 7

10 \* 3 = 30

10 / 3 = 3

10 % 3 = 1

10 is not less than 3

10 is greater than 3

10 does not equal 3

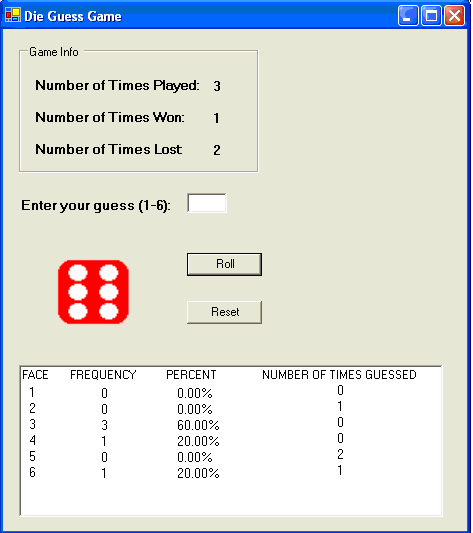
**Commonly Missed Items**

* Make sure to give your UI controls meaningful names.
* On future assignments make sure to use XML comments on all classes, methods, and attributes. To create an XML comment just type three forward slashes.

**Assignment 2**

Create a Windows based program that allows the user to guess what the value of a rolled 6-sided dice will be. The program will display to the user how many times they have played and how many times they have guessed right or wrong. The program will also keep track of the frequency and percentage that each number was rolled, and the number of times the user guessed that number. The program will also have a reset button that returns the game to its initial state.

When the roll button is pressed the dice image should randomly change a couple of times to simulate it rolling. This can be done with a for loop and calling the Thread.Sleep() method in the System.Threading namespace. You will also need to refresh the image. Make sure to check for valid input from the user. Only values of 1 through 6 should be allowed to be entered by the user. Only 1 digit should be allowed to be entered in the textbox, by setting the MaxLength property of the textbox. If an invalid value is entered, a label containing an error message should be displayed next to the user’s guess textbox. For help with the dice roll see example “Chapter 5 and 6 Example”.

****

**For WPF, to set a Gif an Image control:**

1. Create a folder called "Images" in the project.

2. Right click on the folder, select "add", "existing item", then select your image.

3. Select the image, then in the properties window for "Build Action", select "Resource".

4. Set in your code:

MyImage.Source = new BitmapImage(new Uri(@"Images/die1.gif", UriKind.Relative));

**Commonly Missed Items**

- Need to validate user input. Program crashes with invalid input.

- Game should not allow the user to play if invalid input is entered. The dice should not roll.

- Not meeting program requirements to show a random dice image to simulate rolling. Each dice roll should be random, not just the last.

- Each individual class attribute and method need an XML comment.

- Each individual class attribute needs an XML comment.

- Make sure to use meaningful names for all UI controls.

- Did not meet the requirement to only allow one digit to be able to be entered into the user's guess textbox.

- Did not meet the requirement to display an error label when invalid input is entered.

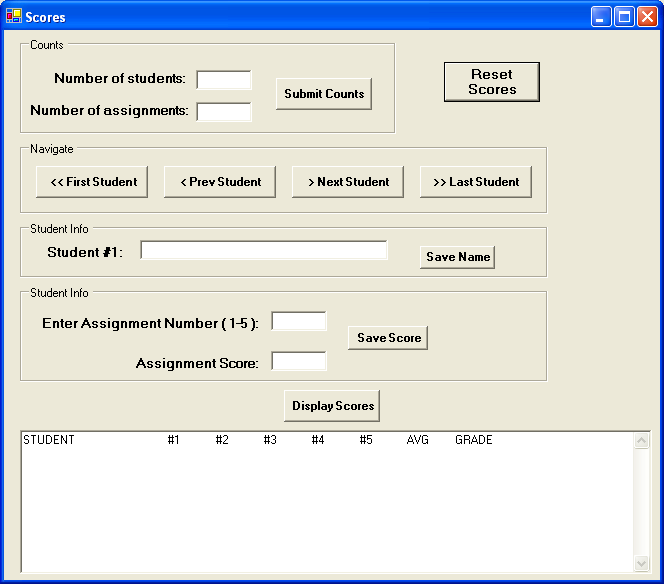
**Assignment 3**

Create a Windows based program that inputs students’ assignment scores and displays them. The program must use arrays to store the student’s information (single dimensional array) and assignment scores (multi-dimensional array).

The program will give the user the ability to enter the number of students (max 10) and the number of assignments (max 99). These values must be validated and error labels given for invalid input. The students’ scores will be defaulted to zero, and the students’ names will be defaulted to “Student #1”, “Student #2”, etc. After the counts have been submitted the user will be allowed to navigate between the students. Each student’s name can be updated and saved by clicking the “Save Name” button. The assignment scores can be updated and saved by entering the assignment number, the assignment score (0 to 100), then clicking the “Save Score” button. The label “Enter Assignment Number (1-X)” should not be hard coded, it should display the correct number of assignments.

At any time after the counts have been submitted the “Display Scores” button may be clicked. This button will display the students, their grades, their average grade, and their letter grade. Use the syllabus for this class to determine the letter grade. Don’t worry too much about if the columns line up perfectly based on how long names are, just get it close. Formatting can be done using tab and new line constants in your string.

At any time, the “Reset Scores” button may be clicked, which will reset the program to its original state of entering the counts.



**Commonly Missed Items**

- Program throws an exception if bottom half of the form is interacted with before the counts are entered.

- Requirement to use error labels for invalid input and not a message box.

- Requirement to check for invalid user input. Program crashes for invalid number of students or assignments.

- Requirement to check for invalid user input. Program crashes for invalid assignment number or score.

- Student name defaults should start with "Student #1", not "Student #0".

- Not meeting requirements for displaying the grades with average and letter grade.

- Can navigate outside the bounds of the student array

- All class variables need XML comments

- All classes, attributes, and methods need XML comments.

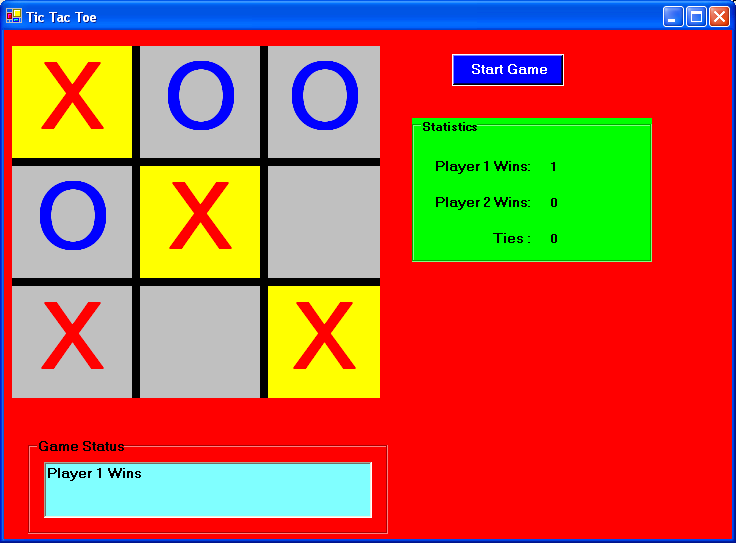
**Assignment 4**

Create a Tic-Tac-Toe game that can be played by two players.  The form will consist of a Tic-Tac-Toe board in which the users’ click on to choose their space.  As the game is being played the Game Status section will tell whose turn it is.  When someone wins or there is a tie, a message will be displayed in the Game status section telling the users the status.  When someone wins the game the winning move needs to be indicated. There also needs to be a section that keeps track of the number of wins for each player, and the number of ties. When the game is finished, the user may click the “Start Game” button to start a new game.

This program will consist of the main form and at least one class that will define the rules of the game.  This class will have an array that is passed in through a property that represents the game board.  The class will then have methods within it that determines if someone won, if there is a tie, or if neither has occurred yet. Make sure all business logic is in a separate class and not behind the UI.

**EXTRA CREDIT (10 Points)**

Create a computer player that can be played against. The computer player will need to be smart enough to make a winning move or to block a winning move. The AI code will need to be in a separate AI business logic class.



**Commonly Missed Items**

- Should have one method that takes care of all button clicks, which was described in the lecture and sample code. If the program had to be updated it would have to be fixed in every method that has been copied and pasted.

- All attributes and methods need XML comments.

- All class attributes need an XML comment.

- All business logic is behind the UI. The major concept behind this assignment is using classes to abstract out the business logic into a separate class.

- Requirement to highlight the winning move.

- After the game is over, moves shouldn't be allowed to be made

- Make sure to watch the lectures. The lecture I gave on Tic Tac Toe outlined the entire assignment and showed every method and attribute that should have been created.

**Assignment 5**

Create a math game for young kids. This game should be themed with material a small child would enjoy. For instance, if you had a son/daughter that liked Star Wars, use pictures and sounds from the Star Wars movies. Or you could do a sports themed game so that when they get an answer correct, they see and hear a crowd cheering. The Game will consist of 3 Windows, which will be the main menu window, the game window, and the final score window.

**Main Menu Window**

When the game first starts the main menu window will be displayed. This window will allow the user to enter their information, choose the game type, and begin playing the game. The user’s information will consist of their name and age. The game type options the user may choose from are Add, Subtract, Multiply, and Divide. These options must be displayed using radio buttons. The window will also have a “Begin Game” button. When the user clicks this button, the entered information must be validated. If the name is blank, or the age is not a valid number between 3 and 10, or a game type has not been selected, then a red error label describing what the user needs to fix will be displayed. For instance, if the user didn’t enter a name, then a label with red text will appear telling the user “Please enter your name to continue”.

**Game Window**

If the user’s entered information is valid, then the main menu window is hidden and the game window displayed. As the game is being played the user should have the option to cancel the current game and return to the main menu. The game will consist of ten randomly generated questions. The random numbers should be kept between 1 and 10 so that the game is easy to play for a young child. So, based on the game type, each question must be randomly generated. For instance, if the game is addition, then two random numbers will need to be generated (between 1 and 10) for each question and displayed to the user.

The game window will have a “Start” button that will begin the game. When the “Start” button is clicked a timer, which goes off every second, will begin. This timer is used to time how long it takes the player to answer all ten questions. When the user answers the final question, this timer is stopped, and that is the game time. As the user is playing the game, the game timer will be displayed to the user so they know how many seconds has elapsed.

When the game starts, the first randomly generated question will appear. So the first question may be “1 + 2 = \_\_\_”. At this point the program waits for the user to enter their answer, then click a “Submit” button. The user may also press the “Enter” key to submit their answer. When the user submits their answer, they are told, via a label, whether or not their answer was correct. At this point, the current question and answer are cleared away and the next question is displayed. This will continue until all ten questions have been guessed (right or wrong). Do not force the user to get answers correct before moving to the next question.

After all ten questions have been answered the final score window should be automatically displayed, and the game window hidden.

**Final Score Window**

When the final score window is displayed, the following information should be displayed: user’s name, age, number of correct answers, number of incorrect answers, and time took to complete the game. The background for final score window should be changed based on the number of correct answers. If the user gets between 0 and 4 correct answers, display an image that conveys a low score. If the user gets between 5 and 7 correct answers, display an image that conveys an average score. If the user gets between 8 and 10 correct answers, display an image that conveys a high score.

When the user closes the final score window, they will be taken back to the main menu window where they can choose to play again.

**General Guidelines**

Be careful when choosing questions for the user. Remember that this is for small children. You want the questions to have random numbers (between 1 and 10), but you also want them to make sense. For instance, if the subtraction game is being played, the answer should not be negative. If the game being played is division, then the answer should be a whole number (10 / 2 = \_\_).

The purpose of this game is to become more familiar with the use of classes. You will need to create classes of at least types “User” and “Game”. Make sure to keep all your business logic out of the UI. This means no logic behind the UI for generating or answering questions.

The “User” class will hold all of the user’s information (Name and age). The “Game” class will be used to create the game questions. The type of game will be passed into this class so that the class can generate the appropriate questions. Also, the answer to each question will be passed into this class and the class will determine if the answer was correct or not.

All methods need to have exception handling. Top level methods need to handle the exception and alert the user to the exception, and lower level called methods need to raise the exception up to calling methods. See example “Chapter 13 Exceptions”.

**How to play a sound (use only .wav files):**

Put the .wav files in the same folder as the exe. If you get an exception playing the file make sure it is a .wav file. If you find other file types, such as .mp3, that you want to play, you will have to convert them to .wav files, there are websites that can do this for you.

SoundPlayer simpleSound = new SoundPlayer("My.wav");

simpleSound.Play();

**To set an Image as the background in XMAL:**

1. Create a folder called "Images" in the project.

2. Right click on the folder, select "add", "existing item", then select your image.

3. Select the image, then in the properties window for "Build Action", select "Content".

3. Next select your main window in the XAML, then in the properties window, select "Brush", "Background", select a "Tile Brush”, then for "ImageSource" click on the drop down and selected the image.

**To set an Image as the background in XAML in the code at runtime:**

1. Do steps above 1 through 3, then use this code:

ImageBrush myBrush = new ImageBrush(new BitmapImage(new Uri(@"Images/MyImage.jpg", UriKind.Relative)));

wndMain.Background = myBrush;

**EXTRA CREDIT for Math Game (10 Points)**

The extra credit for the math game is to create a high score feature in the math game. This will consist of converting the final scores screen into a top 10 high scores screen.

Once the user has completed a game, the user should be taken to the top 10 high scores screen. The user’s score should be displayed on the high score screen along with whether or not they made the top ten high scores. The score will consist of the number of correct answers, the number of incorrect answers, and their time. The high score list will be based off of the number of correct answers, then time. So, if two scores have the same number of correct answers, then the score with the lowest time will decide which one is a higher score. After the high score screen is displayed the user may return to the main screen to play the next game of their choice

The “Scores” class will be used to keep track of the top ten high scores. This class should have the current score passed into it, and it should sort the top ten scores.

**Commonly Missed Items**

- Didn't meet the requirement "red error label describing what the user needs to fix will be displayed"

- Didn't meet the requirement to validate the user's age "If the name is blank, or the age is not a valid number between 3 and 10".

- Requirement to theme the game for children.

- The division part of the program, questions need to divide evenly.

- The requirements state that there should be 10 timed questions, not that each question should be timed.

- Timer Should display just number of seconds, not a date/time.

- Program crashes if invalid data is entered in user info screen.

- Program crashes if invalid data is entered for the user's guess.

- Didn’t meet requirement to allow the user to press "enter" to submit an answer

- Didn’t meet requirement “The user is then told whether the answer is correct or not, and the current answer is cleared away.”

- Final score screen requirements not implemented.

- Didn't meet requirement "a Timer will appear, showing elapsed seconds"

- Didn’t meet requirement “After all ten questions have been answered the final score screen should be displayed. “

- Didn't meet requirement "After the final score screen is displayed the user may return to the main screen to play the next game of their choice"

- Didn't meet requirement "If the user gets between 0 and 4 correct answers, display an image that conveys a low score. If the user gets between 5 and 7 correct answers, display an image that conveys an average score. If the user gets between 8 and 10 correct answers, display an image that conveys a high score."

- Final score screen not displaying the user's data and the user's score.

- Game doesn't allow the user to play more than one game

- Before the game is started, the user may press "enter" in the answer textbox and questions start appearing.

- User can submit an answer before the game is started.

- All class attributes and methods need a XML comment.

- All methods need a try catch.

- Try catch not implemented correctly. Top level methods need to handle the exception and calling methods need to throw the exception with meaningful information.

- A requirement was to have all logic in “User” and “Game” classes.

**Assignment 6 (First Part)**

Create an initial GUI for an airline flight reservation system that will have minimal functionality. Please see the picture for an example of what the GUI should look like. You may also look at the airline reservation system example program I gave you on Canvas. I recommend you use WPF for this assignment.

There will be two different flights the user may choose from. Each flight will have a different seating arrangement. Two flights will need to be graphically created, each with a different seating arrangement. How you do this is up to you. Some ideas on how to do each flight would be to use a panel or user control filled with labels or buttons to represent the seats.

The “Choose Flight” combo box should be filled up with the two different flights that are extracted from the database Flight table. In the combo box, both the Flight Number and Aircraft Type should be displayed. When the form first loads, the flights should be extracted from the database and inserted into the combo box. All other controls on the form should be disabled at this point.

When the user selects a flight from the “Choose Flight” combo box two things should happen. The first is that the appropriate flight should be displayed on the left of the program, and the second is that the “Choose Passenger” combo box should be enabled and loaded with the passengers on the selected flight from the database.

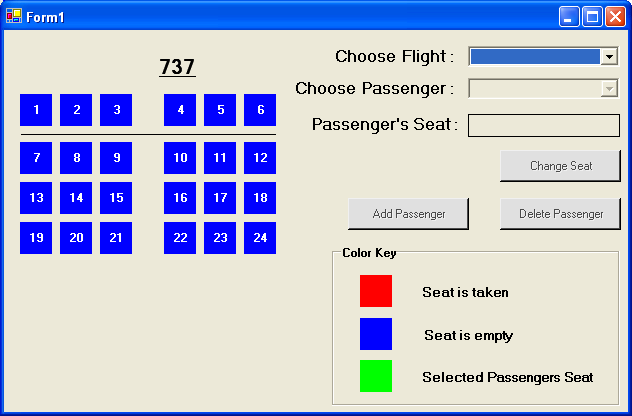
The “Change Seat” and “Delete Passenger” buttons will not have any functionality for the first part of this assignment.

When the user clicks the “Add Passenger” button another form should be displayed. This form needs to be created for this part of the assignment. The other form is where the new passenger’s information will be entered. This form should allow the user to enter the passenger’s first name and last name. It should also have a “Save” and “Cancel” button on it. For the first part of this assignment both of these buttons will do nothing but close this form. **NOTE**: Microsoft Access should be used to store the data.

A good way to make sure no business logic is behind the UI is to create no database code (SQL or DataSet) behind the UI. A requirement, that you must do, is to get the flights in the combo box using a separate method in another class, that gets the flights, creates an object for each flight, adds each object to a generic list, returns that list to the UI, then that list is bound to the combo box to display the data. (See example below). This same requirement must also be done for Passengers.

All methods need to have exception handling. Top level methods need to handle the exception and alert the user to the exception, and lower level called methods need to raise the exception up to calling methods. Make sure all business logic is in separate classes and not behind the UI.

**Note**: If you have a 64bit OS, then connecting to Access may be a problem. If you get a strange error try compiling in 32 bit in Visual Studio. To do this right click on the project and choose properties. Then select "Build", then for "Platform Target" choose "x86".



**WPF**

cboFlights.ItemsSource = clsFlightManager.GetFlights();

**Windows Forms**

cboFlights.DataSource = clsFlightManager.GetFlights();

Where the method GetFlights returns a List<clsFlight> objects.

The class clsFlight is a class that just holds the data associated with a flight.

**Commonly Missed Items**

- Your business logic is behind the UI. There shouldn't be any SQL statements behind the UI. Requirements say "A good way to make sure no business logic is behind the UI is to create no database code (SQL or DataSet) behind the UI. For instance, to get the flights in the combo box, the UI should call a separate method in another class, that gets the flights, creates an object for each flight, adds each object to a list, returns that list to the UI, then that list is bound to the combo box to display the data"

- All attributes and methods need XML comments.

- Need try catch blocks

- Try catch blocks not implemented correctly. Top level methods, such as button clicks, should handle the exception, and all lower level methods should throw the exception.

- Didn't meet requirement "In the combo box, both the Flight Number and Aircraft Type should be displayed."

- Didn't meet requirement "When the user selects a flight from the “Choose Flight” combo box two things should happen. The first is that the appropriate flight should be displayed on the left of the program".

- Didn't meet the requirement "When the user selects a flight from the “Choose Flight” combo box the “Choose Passenger” combo box should be enabled and loaded with the passengers on the selected flight from the database."

- Your business logic classes should not be referencing the the UI like in method SetupPassengerSelectionBox in class Flight. This method should be returning a list of flight objects that is then bound to the combo box

- Didn't meet requirement "Two flights will need to be graphically created, each with a different seating arrangement."

- Add passenger form not created.

- In the future just bind your list of flights and passengers to the combo boxes.

- Program should load flights and passengers from the database.

- The main window should not be using objects of type DataSet. This should all be in the business logic where you query the flights/passengers, fill up a list of objects, then bind those lists to the combo boxes.

**Assignment 7 (Extra Credit Only / Not Required)**

Using the Week 3 assignment, add another button on the form that says “Output to file”. This button will be located next to the “Display Scores” button. When the “Output to file” button is pressed, instead of displaying the student’s data to the textbox, it will be outputted to a file on a new thread. The exact same text that is displayed in the student’s data textbox, should be saved to the file. Since a new thread is being used the UI should not lock up when the button is pressed. The file name should be entered by the user in a textbox next to “Output to file” button. Make sure to check that the file does not already exist. You may default the directory location where the file is written to, to "C:\Users\Public" just to make things easier.

When the user presses the button, the button should be disabled, and the text next to the button should say “Writing to file”. Next the background thread (not the UI) should write the data to a file. Since writing to the file will be fast, we need to simulate a process that will take a while. We will do this by putting the thread to sleep for 2 seconds. After the thread wakes up update the text next to the button to “Finished writing to file”, and enable the button.

Make sure the work of writing the data to the file is done on the background thread and not the UI thread.

All methods need to have exception handling. Top level methods need to handle the exception and alert the user to the exception, and lower level called methods need to raise the exception up to calling methods. See example “Chapter 13 Exceptions”.

Don’t worry about updating your old code. Just write your code for the new requirements. However, your new code should be commented, have exception handling, and business logic should be in a separate class. That means all the code that is running on a separate thread and writing to the file, should be in a separate class.

**Commonly Missed Items**

- You are not writing the file on a separate thread, you are putting the UI to sleep.

- Requirement to have all business logic in a separate class.

- Business logic shouldn't be calling a method in the main window.

- Missing XML comments.

- Need to use a delegate to display the finished writing to screen after the thread is actually done.

**Group Assignment Prototype**

The first part due for the group project is a prototype of the final project. The full project requirements can be viewed in the next section. Please make sure to read all requirements for the Final Group Project. This part of the assignment is to get each group member engaged and to assign each role for the assignment. The final project must be done in WPF. **The prototype will be turned in by one member of the group.**

The group project will be broken up so that each member is responsible for everything for a single Window. So, for a 3 person team, one person will do everything for the Main Window, another person does the Search Window, and the last does the Edit Items Window.

The prototype of the group project will consist of a preliminary design of the UI, how each person’s code will interface with other code, and the SQL for the assignment in a class. Again, make sure to read the final group project requirements to fully understand what the UI will look like and how the program will work when completed.

The Visual Studio project will be created and 3 folders will be added to the project. The folders will be called “Search”, “Main”, and “Items”. Inside these folders, each member will put their respective code files.

For the “Search” folder, there should be a XAML file for the UI called “wndSearch.xaml”, another file named “clsSearchSQL” which contains all SQL statements for the Search Window, and the last file should be “clsSearchLogic” which will contain all business logic for the Search Window.

For the “Main” folder, there should be a XAML file for the UI called “wndMain.xaml”, another file named “clsMainSQL” which contains all SQL statements for the Main Window, and the last file should be “clsMainLogic” which will contain all business logic for the Main Window.

For the “Items” folder, there should be a XAML file for the UI called “wndItems.xaml”, another file named “clsItemsSQL” which contains all SQL statements for the Items Window, and the last file should be “clsItemsLogic” which will contain all business logic for the Items Window.

**GUI**

All of the screens should be created with all controls needed to complete the requirements. For instance, on the search screen, there should be 3 drop down boxes for selection, a DataGrid, and select and cancel buttons. Once each screen has been created the flow of the program needs to be completed. So, for example, on the main form, there should be a menu with the selection of “Search for Invoice” that when clicked should open the search window, then when the user clicks the “Select” or “Cancel” buttons the search window should close and the main form get focus.

**Interfaces**

This part of the assignment is to put together a plan on how each screen will pass the data to the other screens. This will be done by putting the appropriate comments in the sections of stubbed out code to explain how the data will be passed around. This will get you thinking about how each screen will interface with the others. So for example, on the search screen, behind the button click event for the “Select” button, there should be a detailed comment about how the selected InvoiceID will be passed back to the main form. For example, if a property is set in the Search screen window with the selected Invoice ID, then the comment will explain how the variable is set and the Main screen may access this data via a property.

**SQL**

This part of the assignment is to create a class for each window that contains the SQL needed for that window. Each SQL class will be nothing but methods that contain different statements of SQL. Make sure to create SQL statements that will help in meeting all requirements that use the database. So, SQL statements needed will be to select different types of data on each window, to update/insert/delete data on each form. Use Microsoft Access to run the queries ahead of time to make sure the queries give you the expected data and work correctly. Your SQL statements should be tested and working. Below is an example of a class/method that should be used as a guide for your code.

class clsSQL

{

/// <summary>

/// This SQL gets all data on an invoice for a given InvoiceID.

/// </summary>

/// <param name="sInvoiceID">The InvoiceID for the invoice to retrieve all data.</param>

/// <returns>All data for the given invoice.</returns>

public string SelectInvoiceData(string sInvoiceID)

{

string sSQL = "SELECT \* FROM Invoices WHERE InvoiceNum = " + sInvoiceID;

return sSQL;

}

}

**NOTE**: Only one person should submit the protype assignment with all members’ name in canvas.

**Commonly Missed Items**

- Didn't meet the following requirement - "The Visual Studio project will be created and 3 folders will be added to the project. The folders will be called “Search”, “Main”, and “Items”. Inside these folders, each member will put their respective code files."

- Didn't meet the following requirement - "For example, inside the “Search” folder, there should be a XAML file for the UI called something like “wndSearch.xaml”, another file named something like “clsSearchSQL” which contains all SQL statements for the Search Window, and the last file should be something like “clsSearchLogic” which will contain all business logic for the Search Window."

- Didn't meet the following requirement - "Once each screen has been created the flow of the program needs to be completed. So, for example, on the main form, there should be a menu with the selection of “Search for Invoice” that when clicked should open the search window, then when the user clicks the “Select” or “Cancel” buttons the search window should close and the main form get focus."

- Didn't meet the following requirement - "This will be done by putting the appropriate comments in the sections of stubbed out code to explain how the data will be passed around. This will get you thinking about how each screen will interface with the others. So for example, on the search screen, behind the button click event for the “Select” button, there should be a detailed comment about how the selected InvoiceID will be passed back to the main form." Also should be comments explaining how the main window will be refreshed when the Items window is closed.

Main

-----------

- Didn't meet the requirement "Main window will have a menu (at the top, use a menu control) that will have two functionalities. The first will allow the user to update a def table that contains the items. The next will be to open a search screen used to search for invoices."

- Main window class clsMainSQL does not have all of the SQL needed for the window. All SQL was given in the Database Help document.

- Main window class clsMainSQL has business logic in it, the requirements were to create SQL classes that had just the SQL statements and that's it.

- Main window needs comments to talk about how it will interact with the other windows to get the selected invoice or know when to refresh the items.

- Main window, when the Item's Window is closed needs to have comments talking about refreshing the combo boxes with the items in it, in case they got updated.

Search

----------

- Search window class clsSearchSQL does not have all of the SQL needed for the window. All SQL was given in the Database Help document.

- Search window class clsSearchSQL has business logic in it, the requirements were to create SQL classes that had just the SQL statements and that's it.

- Search window should have a comment about when the invoice is selected, the Invoice ID is saved in a local variable that the main window can access.

Item

---------

- Item window class clsItemSQL does not have all of the SQL needed for the window. All SQL was given in the Database Help document.

- Items window class clsItemsSQL has business logic in it, the requirements were to create SQL classes that had just the SQL statements and that's it.

**Group Assignment All Requirements**

Create a Windows WPF program that can be used as in invoice system for a small business. The type of business is up to you. Examples of a business would be a Supplement Store, Jewelry Store, Shoe Store, Equipment Rental Store, etc. A Microsoft Access database should be used as the backend database to store the invoice data. **The group assignment will be turned in by one member of the group.**

The group project will be broken up so that each member is responsible for everything for a single Window. So, for a 3-person team, one person will do everything for the Main Window, another person does the Search Window, and the last does the Edit Items Window.

The Visual Studio project will be created and 3 folders will be added to the project. The folders will be called “Search”, “Main”, and “Items”. Inside these folders, each member will put their respective code files.

For the “Search” folder, there should be a XAML file for the UI called “wndSearch.xaml”, another file named “clsSearchSQL” which contains all SQL statements for the Search Window, and the last file should be “clsSearchLogic” which will contain all business logic for the Search Window.

For the “Main” folder, there should be a XAML file for the UI called “wndMain.xaml”, another file named “clsMainSQL” which contains all SQL statements for the Main Window, and the last file should be “clsMainLogic” which will contain all business logic for the Main Window.

For the “Items” folder, there should be a XAML file for the UI called “wndItems.xaml”, another file named “clsItemsSQL” which contains all SQL statements for the Items Window, and the last file should be “clsItemsLogic” which will contain all business logic for the Items Window.

The main window should allow the user to create new invoices, edit existing invoices, or delete existing invoices. There should be just one window for all functionality of the main window. So, the main window will NOT open other windows to add/edit/delete invoices. It will also have a menu (at the top, use a menu control) that will have two functionalities. The first will allow the user to update a def table that contains the items. The next will be to open a search screen used to search for invoices.

If a new invoice is created the user may enter data pertaining to that invoice. Since an auto-generated number is used in the database for the invoice number, when a user creates a new invoice, just display “TBD” for the Invoice Number. An invoice date will also be assigned by the user. Next different items will be entered by the user. The items will be selected from a drop-down box and the cost for that item will be put into aread only textbox. This will be the default cost of an item. Once the item is selected, the user can add the item. As many items as needed should be able to be added. All items entered should be displayed for viewing in a list (something like a DataGrid). Items may be deleted from the list. A running total of the cost of all items should be displayed as items are entered or deleted.

Once all the items are entered the user can save the invoice. Once the Invoice is saved, query the max invoice number from the database, to display for the invoice number (for our project, this will work, since the last inserted invoice, will be the max). This will lock the data in the invoice for viewing only. From here the user may choose to Edit the Invoice or Delete the Invoice.

The user also needs to be able to search for invoices, which will be a choice from the menu. On the search screen, all invoices should be displayed in a list (like a DataGrid) for the user to select. The user may limit the invoices displayed by choosing an Invoice Number from a drop down, selecting an invoice date, or selecting the total charge from a drop-down box. The total charges in the drop-down box should be the unique set of total charges sorted from smallest to largest. When a limiting item is selected, the list should only reflect those invoices that match the criteria. So, the user should be able to select a date and a total charge, and only invoices that match both will be displayed. A clear selection button should reset the form to its initial state. Once an invoice is selected the user will click a “Select Invoice” button, which will close the search form and open the selected invoice up for viewing on the main screen. From there the user may choose to Edit or Delete the invoice.

The last form needed is a form to update the def table which contains all the items for the business. This form can be accessed through the menu and only when an invoice is not being edited or a new invoice is being entered. This form will list all the items in a list (like a DataGrid). The items will consist of a code, cost, and description. From here the user can add new items, edit existing items, or delete existing items. If the user tries to delete an item that is on an invoice, don’t allow the user to do so. Instead warn them with a message that tells the user which invoices that item is used on. When an item is updated, the code must not be allowed to be updated because it is the primary key, only the description and cost may be updated. When the user closes the update def table form, make sure to update the drop-down box as to reflect any changes made by the user. Also update the current invoice because its item name might have been updated.

Since this is the final project all lessons learned throughout the course should be used and implemented. Don’t forget to abstract your business logic into classes and keep you UI code clean. Make sure to test all user inputs so your program doesn’t crash, have another group member test your code thoroughly. All methods should handle exceptions. Since this a WPF application you should use styles for your applications. At a minimum, a theme should be applied to the application, such as one talked about in the Microsoft Blend lecture. Visual properties shouldn’t be hard coded into controls, they should be put into styles and applied to controls.

**Guidelines**

- Project Submission: you must turn it in by midnight on the due date.

- Only one person should submit the assignment with all members’ name in the email.

**Common Mistakes**

- Student's didn't unit test each other’s code

- Forgot to use styles

- Business logic behind the UI

- Validate all user input

**Tips**

- Run each other’s code to test for bugs

- Look at each other’s code

- Verify all requirements

- Run through the presentation together

- Break up the project so that each member is responsible for everything for a single Window.

**DataGrid Help**

* Keeps thing simple.
* On the edit item screen, instead of using the DataGrid like an excel spreadsheet, just show each selected item in textboxes next to the DataGrid
* To get rid of the extra row on the bottom of the DataGrid set the following property:
  + CanUserAddRows="False"

**Microsoft Access Help**

To view data in a table: In the “Tables” menu on the left-hand side of the screen, double click the data you wish to the view the data for.

To create and test queries: Click the “Create” ribbon item, click the “Query Design” button. This brings up the ability to create queries in a designer, if you know how to work this go ahead, if you want to write queries manually and test them, then close the “Show Table” window, then click the “SQL View” button at the top left corner, select “SQL View”. Now create your SQL statements, then click the “Run” button on the ribbon.

**Commonly Missed Items**

Search

- Requirement to “On the search screen all invoices should be displayed in a list”.

- On the search screen the combo boxes should only have distinct values, not duplicates, and should be ordered so users can find the values they need.

- Search window has business logic behind the UI.

- Search window, the date and total charge should be combined as a filter if both are selected.

- Requirement to have a clear selection button.

- Each time the search window is opened it should refresh invoices and combo boxes because an invoice could have been updated or deleted.

- Requirement to theme the application.

- All Classes, Attributes, and Methods need XML comments.

- Try catch blocks not implemented correctly.

- All methods need a try catch block

- All business logic has been put into the classes that are just supposed to be SQL statements.

Main

- I can only add 1 type of item to an invoice. I should be able to select an item and click the Add item button multiple time for the same item.

- When I create a new invoice, the total charge is always zero in the search screen so it is not being saved to the db.

- Main window has business logic behind the UI.

- When creating an invoice, the Invoice number is not given

- Requirement to have a menu control to get to the search and edit screens.

- After an invoice is selected it should be displayed in read only mode and an edit button needs to be clicked.

- After a new invoice is created and saved, it should be displayed in read only mode.

- Main window items need to be refreshed after items have been updated/deleted/added on the items window.

- Shouldn't be able to save an invoice without entering a date.

- Didn't meet the requirement "An invoice date will also be assigned by the user."

- After an invoice is searched for and selected, the main window should display it in a read only mode, then an edit button needs to be clicked to edit it.

Items

- When deleting an item on the def table form, a requirement says “Instead warn them with a message that tells the user which invoices that item is used on.”. Your message doesn’t tell which invoices it is on.

- When deleting an item on the def table form, if the item exists on invoices, then distinct invoices should be displayed.

- Need to validate user input on the update def table window. If I enter very long strings the program raises an exception.

- Def table window has business logic behind the UI.