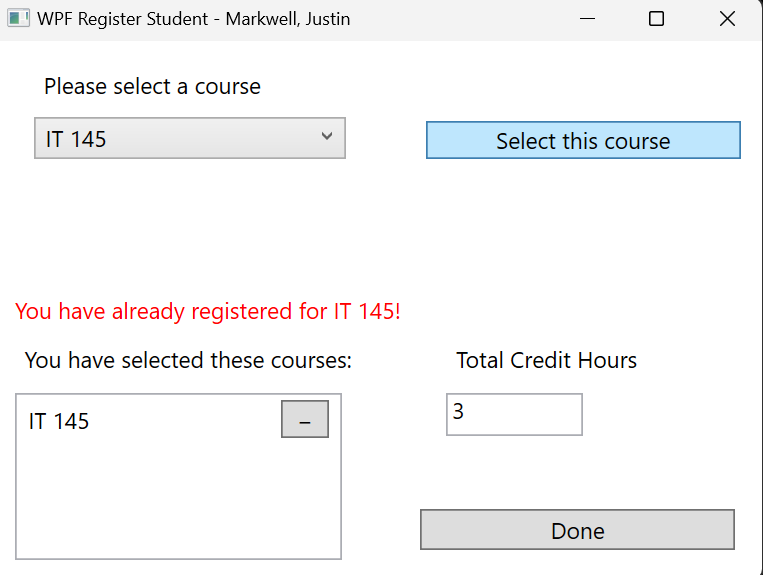
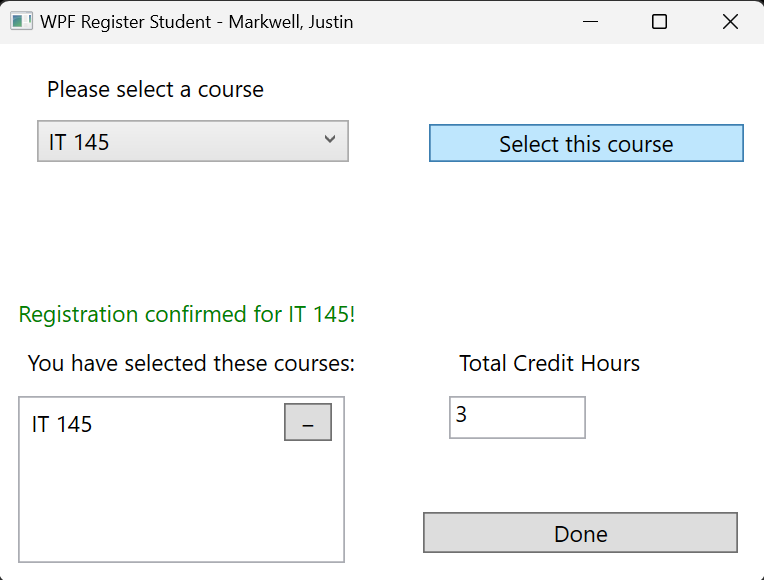
# IT 230 6-2 Final Project Part II Milestone I Coding Activity: Create Classes for Final Project

Justin Markwell

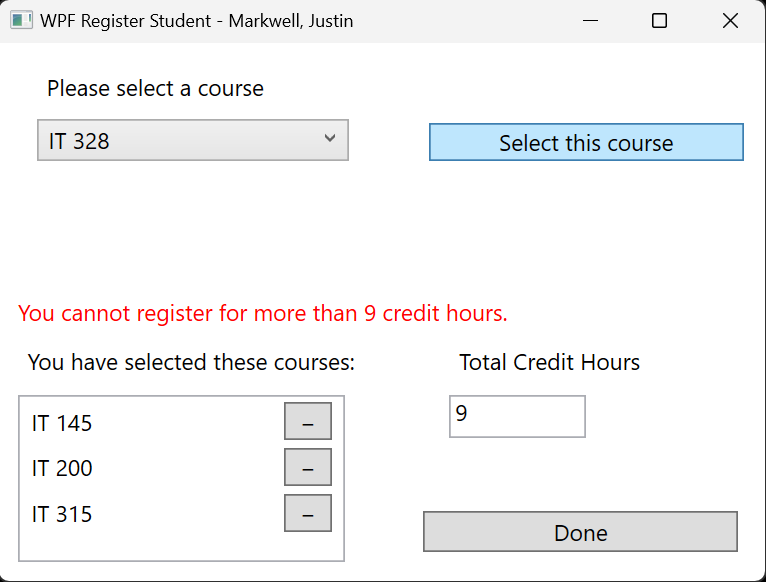
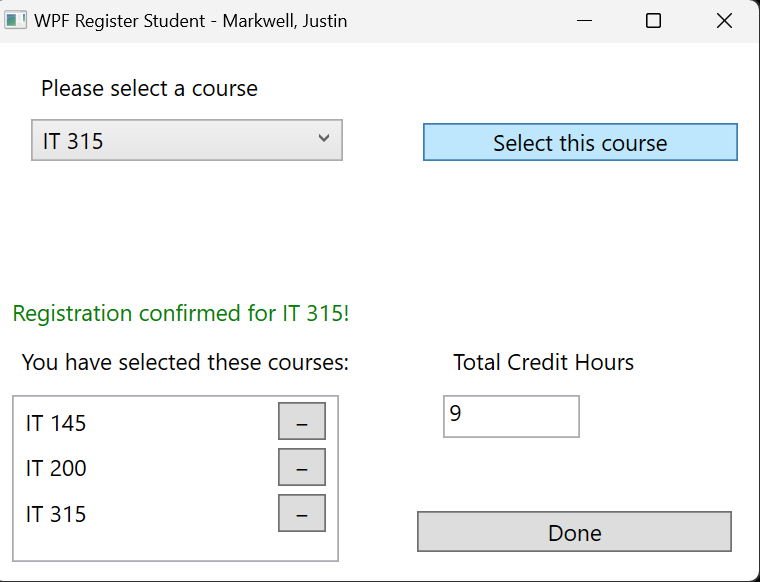
Professor: Zoe **Likoudis**

April 15, 2025

1. Provide a screenshot of the output that resulted from running your program successfully in Visual Studio. See the coding assignment instructions for an example of what should be included in the screenshot. Your screenshot must include the following elements:
   1. The program is fully functional without error
   2. Data results are accurate for the given problem



Registered for 1st Course Cannot register same course



Registered for multiple courses Cannot exceed 9 credit hours

Copy and paste the source code text you wrote for this assignment from the \*.cs file into the space below. Only providing the \*.cs files or a screenshot does not meet the requirements for this part of the assignment. Code should be logically organized. It should also follow proper syntax and conventions noted in the Coding Activity Guidelines and Rubric.

//Optimized changes to MainWindow

public partial class MainWindow : Window

{

Course choice;

List<Course> registeredCourses = new List<Course>(); //Track registered

courses

int totalCredits = 0; //Track total credits

public MainWindow()

{

InitializeComponent();

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

//used an array to hold the courseNames

string[] courseNames = { "IT 145", "IT 200", "IT 201", "IT 270",

"IT 315", "IT 328", "IT 330" };

List<Course> availableCourses = new List<Course>();

//loop through and set name and credits for each available courses

foreach (string courseName in courseNames)

{

Course course = new Course();

course.SetName(courseName);

course.SetCredits(3); //All courses are currently 3 credits

availableCourses.Add(course);

}

//Populate the ComboBox with course choices

foreach (Course course in availableCourses)

{

this.comboBox.Items.Add(course);

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (comboBox.SelectedItem == null) return;

choice = (Course)(this.comboBox.SelectedItem);

//Check if the course is already selected

if (registeredCourses.Contains(choice))

{

messageText.Text = $"You have already registered for

{choice.GetName()}!";

messageText.Foreground = Brushes.Red;//use red to represent an error

}

//Check if the user is trying to register more than 3 courses or exceed

9 credits

else if (totalCredits + choice.GetCredits() > 9)

{

messageText.Text = "You cannot register for more than 9 credit

hours.";

messageText.Foreground = Brushes.Red;//use red to represent an error

}

else

{

//Add course and update total credits

registeredCourses.Add(choice);

messageText.Text = $"Registration confirmed for

{choice.GetName()}!";

messageText.Foreground = Brushes.Green;//use green to represent

successful registration

totalCredits += choice.GetCredits();

this.textBox.Text = totalCredits.ToString();

this.listBox.Items.Add(choice);

//Show Done button if total credits == 9

doneButton.Visibility = (totalCredits >= 3) ?

Visibility.Visible : Visibility.Hidden;

}

}

//Clear the message text on selection changed in combo box.

private void ComboBox\_SelectionChanged(object sender,

SelectionChangedEventArgs e)

{

messageText.Text = "";

}

//Remove Course (−) button is clicked

private void RemoveCourse\_Click(object sender, RoutedEventArgs e)

{

//Retrieve the button that was clicked

Button btn = sender as Button;

//Retrieve the associated course stored in the button's Tag

Course courseToRemove = btn.Tag as Course;

//If the course is valid and exists in the selected courses list

if (courseToRemove != null &&

registeredCourses.Contains(courseToRemove))

{

//Remove the course from the selected courses list

registeredCourses.Remove(courseToRemove);

//Subtract its credit value from the total credits

totalCredits -= courseToRemove.GetCredits();

//Update the display of total credits

this.textBox.Text = totalCredits.ToString();

//Remove the course from the visible ListBox UI

listBox.Items.Remove(courseToRemove);

//Show confirmation message to the user

messageText.Text = $"You have de-registered from

{courseToRemove.GetName()}.";

messageText.Foreground = Brushes.Green;

//Hide the "Done" button in case credit total is now below the

threshold

doneButton.Visibility = (totalCredits >= 3) ?

Visibility.Visible : Visibility.Hidden;

} }

private void DoneButton\_Click(object sender, RoutedEventArgs e)

{

#if DEBUG

//If running debug, close the debug window

Application.Current.Shutdown(); //close app

#else

//Normal exit

Application.Current.Shutdown();

#endif

}

}

//Requirements for Course class

public class Course

{

//Hold the course name

private string name;

public string Name => name; //for data binding

//credits for each course

private int credits;

//Set the course name

public void SetName(string courseName)

{

name = courseName;

}

//Get the course name

public string GetName()

{

return name;

}

//Set Credits for course

public void SetCredits(int courseCredits)

{

credits = courseCredits;

}

//Get credits for course - in case some course in future is not exactly 3

public int GetCredits()

{

return credits;

}

//Override ToString() >>display course name in the UI

public override string ToString()

{

//return the name field

return name;

}

}

1. Show that you understand the task by explaining the design of your program in the space below. Include the process and steps you took to write your code. Explain how you arrived at the solution to the problem and completed the activity.

## **Design and Development Process**

To complete the course registration application, I designed a WPF-based C# project that allows students to register for courses while following these core rules:

A student may not register for the same course more than once.

The total number of registered credit hours cannot exceed 9.

A "Done" button should only be visible once the user has successfully registered for at least one course.

I began by creating a Course class to store both the name and credit value of each course. The MainWindow handles the user interface and registration logic.

Inside the Window\_Loaded event, I initialized a string array for course names and used a List<Course> to represent the available courses, each defaulting to 3 credit hours. These courses were then added to a ComboBox for user selection. When a student selects a course and clicks the Register button, the program validates two conditions:

1)Whether the course has already been registered.

2)Whether the credit limit will be exceeded.

If either condition fails, a red error message appears using a TextBlock. If registration is successful, the course is added to the student’s list, and a green success message is shown. The current credit total is also updated and displayed.

To improve the visual layout, I used a ViewBox so the UI scales smoothly when the window is resized. This ensures a clean and responsive interface across various screen sizes.

## **Debugging Process and Steps**

### **Issues Identified and Resolved:**

Used PascalCase consistently for method names like SetName() and GetCredits() to adhere to C# standards.

Refined the logic within the Button\_Click event to better handle error messaging for duplicate courses and credit overflow.

Updated the ComboBox to hold Course objects instead of just strings, which simplified retrieval and display by overriding the ToString() method.

Added a null check for the selected item to prevent errors if the user clicks Register without choosing a course.

### **Usability Improvements:**

Added a separate TextBlock (messageText) for displaying error and success messages. Red text is used for errors, and green for successful registrations, giving users clear visual feedback.

Displayed the current total credit hours in real time, updating as courses are added or removed.

Implemented a **de-register** option that allows users to easily remove a course if they selected it by mistake.

Included a **Done** button that only becomes visible once the user has registered at least one course. This helps guide the user experience and provides a natural conclusion to the registration process.

### **Commenting and Readability:**

Included descriptive comments throughout the code to explain logic and method behavior, improving long-term maintainability and clarity.

### **Tested the Program with Various Scenarios:**

Successfully registered three different courses (9 total credit hours).

Tried to register a fourth course and correctly received a credit limit error.

Attempted to register the same course twice and received a duplicate registration warning.

Used the deregister button to remove a course and validated that credit hours and the UI updated accordingly.

Verified that the Done button appeared only after at least one course was registered.

## **Reflection on the Learning Experience**

This project strengthened my understanding of object-oriented design, particularly in separating data (Course class) from UI logic. It also improved my skills in WPF event-driven programming and handling real-time UI updates.

I learned how important it is to provide clear feedback to users, not just through logic, but visually, like using color-coded messages and updating the credit total dynamically. Incorporating features like deregistration and conditional UI elements (such as the Done button) made the application more user-friendly and robust.

Overall, this was a great opportunity to apply clean design principles, enforce logical rules through intuitive UI cues, and improve both my programming and user experience design skills.