# IT 230 Coding Activity Submission Template

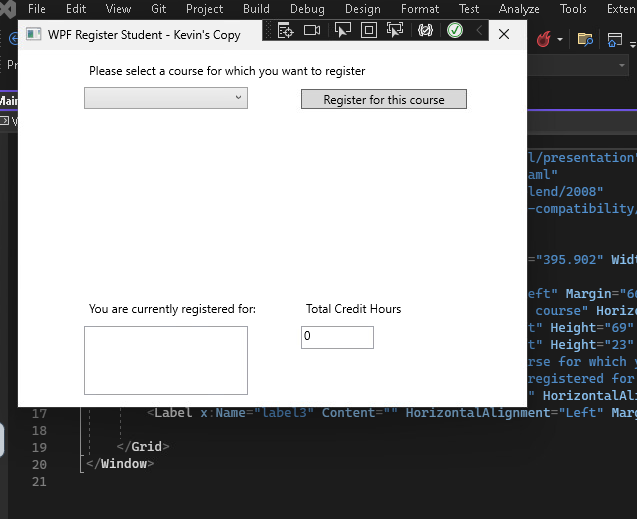
Submit your work on the coding activities for Modules One, Two, Three, Four, and Six in this document. In addition to this document, you should submit a ZIP file containing all your Visual Studio project files and source code that can be run in Visual Studio on a different computer.

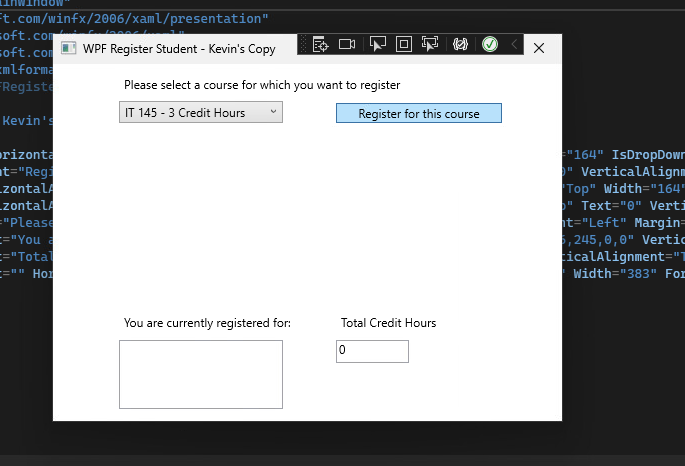
For each coding activity, complete the following steps:

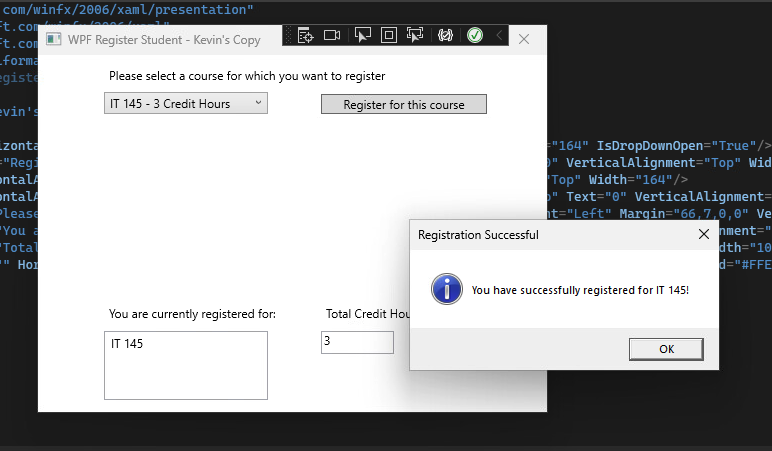
* Download and rename this document to meet the file naming conventions requested in the assignment instructions.
* Fill in the required information below by replacing the bracketed text with the relevant information.
* Submit this document and your ZIP file for grading and feedback. Your ZIP file should follow the same naming conventions.

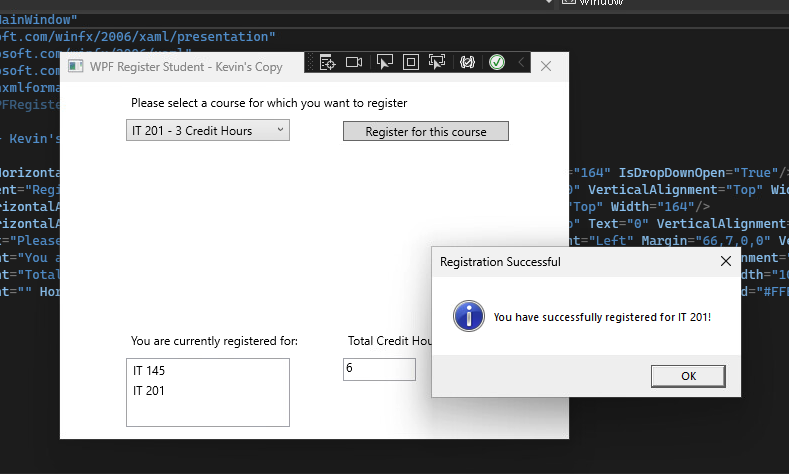
Document your work in the coding activity by completing each of the following items:

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. Your last name as the first printed text on the screen
   2. Verification that the program is fully functioning and data results are accurate for the given problem

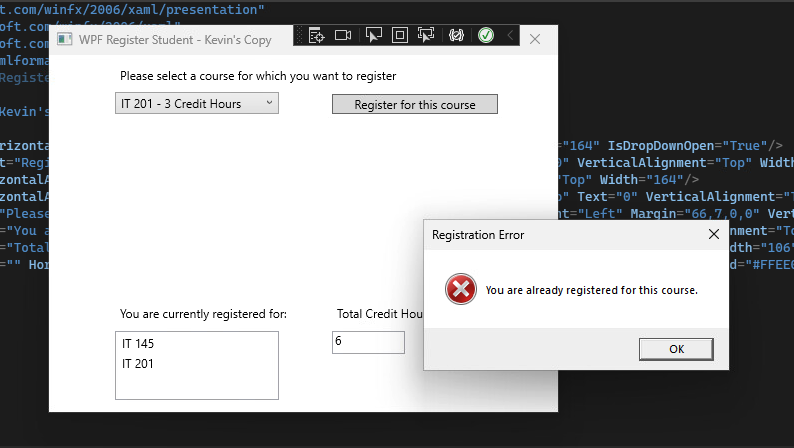
Screenshot 1:

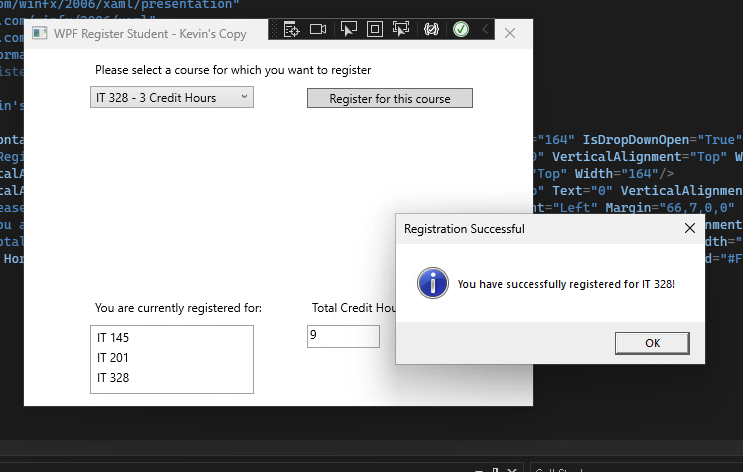
Screenshot 2:

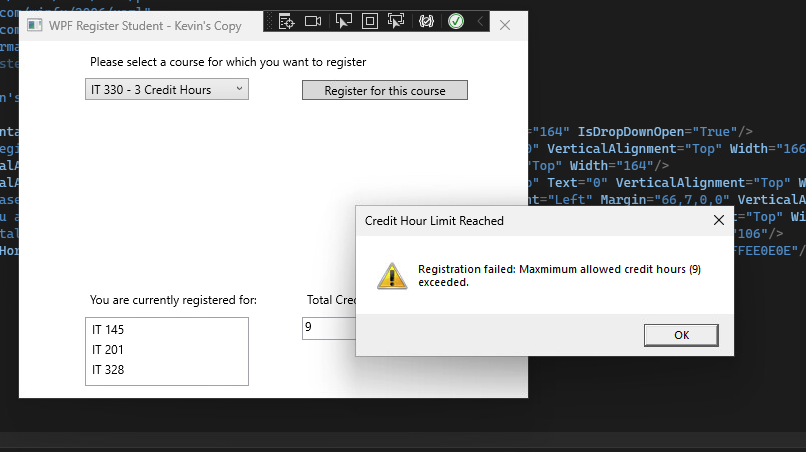
Screenshot 3:

Screenshot 4:

Screenshot 5:



Screenshot 6:

Screenshot 7:

1. 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.

MainWindow.xaml.cs Source code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace WPFRegisterStudent

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private List<Course> courses;

private int totalCreditHours = 0;

private const int MAX\_CREDIT\_HOURS = 9;

public MainWindow()

{

InitializeComponent();

}

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

{

courses = new List<Course>

{

new Course("IT 145", 3),

new Course("IT 200", 3),

new Course("IT 201", 3),

new Course("IT 270", 3),

new Course("IT 315", 3),

new Course("IT 328", 3),

new Course("IT 330", 3),

};

foreach (Course course in courses)

{

comboBox.Items.Add(course);

}

textBox.Text = "0";

}

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

{

if (comboBox.SelectedItem == null)

{

MessageBox.Show("Please select a course to register.", "Selection Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

Course selectedCourse = (Course)comboBox.SelectedItem;

if (selectedCourse.IsRegisteredAlready())

{

MessageBox.Show("You are already registered for this course.", "Registration Error", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

if (totalCreditHours + selectedCourse.GetCreditHours() > MAX\_CREDIT\_HOURS)

{

MessageBox.Show("Registration failed: Maxmimum allowed credit hours (9) exceeded.", "Credit Hour Limit Reached", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

selectedCourse.SetToRegistered();

totalCreditHours += selectedCourse.GetCreditHours();

textBox.Text = $"{totalCreditHours}";

listBox.Items.Add(selectedCourse.getName());

MessageBox.Show($"You have successfully registered for {selectedCourse.getName()}!", "Registration Successful", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

}

Course.cs source code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WPFRegisterStudent

{

class Course

{

private string name;

private bool isRegisteredAlready;

private int creditHours;

public Course(string name, int creditHours)

{

this.name = name;

this.creditHours = creditHours;

this.isRegisteredAlready = false;

}

public string getName()

{

return name;

}

public bool IsRegisteredAlready()

{

return isRegisteredAlready;

}

public void SetToRegistered()

{

isRegisteredAlready = true;

}

public int GetCreditHours()

{

return creditHours;

}

public override string ToString()

{

return $"{getName()} - {creditHours} Credit Hours";

}

}

}

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.

The objective of this project was to create a WPF Register Student program that enables students to select and register for courses, while ensuring that business rules such as a maximum credit hour limit are enforced. The program uses a graphical user interface (GUI) to enhance user interaction, allowing students to select courses from a dropdown list, register for them, and view their registered courses along with the total credit hours.

Design Process and Steps Taken:

I first began by reviewing the task requirements, focusing on features like course registration, credit hour tracking, and validation against business rules such as no duplicate course registration and ensuring to keep to the 9-credit-hour limit.

With all that in mind I created a Course class to encapsulate course properties, including the course name, credit hours, and registration status. The class includes methods to check if a course is already registered and to mark it as registered. This was all done in the MainWindow.xaml.cs:, This file handles the core logic of the application such as:

    •    Populating the ComboBox with available courses.

    •    Handling course selection and registration logic.

    •    Implementing checks to prevent duplicate registrations and to ensure total credit hours do not exceed 9.

    •    Updating the ListBox to display registered courses and the TextBox to show total credit hours.

I then ensured that UI elements in MainWindow.xaml (ComboBox, ListBox, TextBox) were properly named and linked to their counterparts in the code behind (MainWindow.xaml.cs). This step was crucial for updating the UI dynamically as users interacted with the program.

I then dealt with error handling and user feedback for this I added MessageBox alerts to inform users about errors, such as attempting to register for the same course twice or exceeding the credit hour limit, and successful registrations.

Finally came the testing and debugging I thoroughly tested the application, ensuring that the user can register for multiple courses without exceeding the credit limit, that the program prevents duplicate registrations, and the UI updates correctly, showing registered courses and total credit hours. By following these steps, I was able to meet all project requirements and create a user-friendly registration system.

1. Reflect on your learning experience and what you learned from completing the activity.

This project was a valuable learning experience that strengthened my understanding of WPF application development and the Model-View-Controller (MVC) principles. Some key takeaways were first, WPF and XAML Integration, I gained hands-on experience with XAML for designing the UI and understood how it integrates with C# in the code-behind to create dynamic, interactive applications. Second lesson was in event handling in WPF, specifically managing user interactions through events (like button clicks) helped me understand how to create responsive interfaces. I learned how important it is to link UI components correctly and how to handle user input effectively. A Third lesson was problem solving and debugging, a significant part of the process involved identifying and fixing issues, such as mismatched element names between XAML and C# code. This taught me the importance of attention to detail and the value of systematic debugging. Finally user centered design and ensuring that the application provided appropriate feedback, through MessageBox alerts in order to enhance the user experience. I learned how small details like validation messages and UI updates play a crucial role in building intuitive applications.

Initially, I faced challenges with UI element naming and ensuring the data flow between the UI and the logic layer. However, working through these issues deepened my understanding of WPF structure and best practices. By the end of the project, I felt more confident in my ability to design functional WPF applications and implement logic that directly interacts with UI components.