**Course Project
DeVry University
College of Engineering and Information Sciences**

**Course Number: CEIS209**

Background

This course project is designed to help you practice using your skills. Each week, you will learn important concepts. Then, you will apply these concepts to build an amazing course project over the next seven weeks. What’s more, this project will have a Graphical User Interface (GUI) instead of using the console. This course project has two parts. For the first couple of weeks, you will be create a basic GUI and learn how to work with GUI controls while practicing basic programming concepts. After that, you will create an object oriented project.

Part 2: Scenario

## You have been hired as a consultant to create a basic Payroll System for a small business. The business wants to keep track of basic employee information and print paychecks for the employees every two weeks. Your client has some basic functionality in mind, such as the ability to add and remove employees, display basic employee information, and print paychecks. Maintenance is key! You realize that Object-Oriented programming will provide the most efficient application that is easy to maintain and update as needed. Let’s get started!

# Week 6: Encapsulating Data and Writing to a File

# Objectives

1. Integrate Employee class
2. Data hiding

# Introduction

This week, you will use your Employee class in your project to encapsulate the employee information and make your application more maintainable. When a user clicks the AddButton, you will get the information for the employee, create an Employee object, and add the Employee object to the employees listbox. Finally, you will save all of the employee information to the file.

# Steps

Picking up from last week, we will include the Employee class in our forms application.

1. When a user clicks the Add button, we need to get the employee’s information, create an Employee object, and add the Employee object to the employees listbox. How can we get the employee’s information? We need to create a second form and then call it when the Add button is clicked.

Create an input form to get the employee’s information. Add a Submit button and a Cancel button to the form. When you finish, the form should look similar to this one:

 

The InputForm has a property called **CancelButton**. As a result, we will need to rename our **CancelButton** to **ExitButton**. We can keep the text as Cancel. Change the button name now!

The textboxes on the **InputForm** are set to **Private**. This means that the second form will not be able to see the textboxes. Change the accessibility to **Public** so the second form will be able to see the textboxes. Click in the firstName text box and in the properties the Modifiers should be Public. 

When the user clicks the Add button, the code should create an InputForm and show it as a Dialog. The user will enter the employee’s information and then click the Submit button. Otherwise, the user can click the Cancel button. If the user clicks the Submit button, set the DialogResult property of the InputForm to **DialogResult.OK**. If the user clicks the Cancel button, set the DialogResult property to **DialogResult.Cancel**. When you are finished, your code should look like this:

private void SubmitButton\_Click(object sender, EventArgs e)
{
 this.DialogResult = DialogResult.OK;
 this.Hide();
}

private void ExitButton\_Click(object sender, EventArgs e)
{
 this.DialogResult = DialogResult.Cancel;
}

Let’s change the code for the Add button on the Main form.

* 1. Create an InputForm object and show it as a Dialog.
	2. Get the information off the InputForm and create an Employee object.
	3. Dispose of the InputForm
	4. Add the Employee object to the employees listbox

When you are finished, your AddButton code should look like this:

private void AddButton\_Click(object sender, EventArgs e)

 {

 // add item to the employee listbox

 InputForm frmInput = new InputForm();

 using (frmInput)

 {

 DialogResult result = frmInput.ShowDialog();

 // see if input form was cancelled

 if (result == DialogResult.Cancel)

 return; // end the method since the user cancelled the input

 // get user's input and create an Employee object

 string fName = frmInput.FirstNameTextBox.Text;

 string lName = frmInput.LastNameTextBox.Text;

 string ssn = frmInput.SSNTextBox.Text;

 string date = frmInput.HireDateTextBox.Text;

 DateTime hireDate = DateTime.Parse(date);

 Employee emp = new Employee(fName, lName, ssn, hireDate);

 // add the Employee object to the employees listbox

 EmployeesListBox.Items.Add(emp);

 }

}

1. We set up the Remove button last week based on the index of the object. This code will continue to work even though we changed the type of object that we are using! Test the functionality of the Remove button and confirm that it works.
2. Every time we start our application, we have to type in new Employee information. It is important to have persistent data. In the business world, you will be reading and writing to databases. In this course project, we will read and write to files.

When the user adds an Employee object, we need to write the updated Employee data to the file. When a user removes an Employee object, we also need to write the updated Employee data to the file. On the other hand, when a user clicks the Display button, we need to read the Employee data from the file.

Since we want to keep the code as maintainable as possible, let’s do the reading and writing to files using methods

Add a **WriteEmpsToFile( )** method call to the end of your Add button code and then create the method.
	1. Create a string variable for the filename
	2. Open a pipe to the file using the **StreamWriter** object
		1. Add “**using System.IO**” to the top of your code so you can read and write to files
	3. Write all of the Employee object data to the file using Comma Separated Values (CSV) format
	4. Remember to close your **StreamWriter** object!
3. Add this same method call to the end of your **Remove** button code so it updates the file when you remove an Employee object.

Add a **ReadEmpsFromFile( )** method call to the end of your Display button code and then create the method.
	1. Create a string variable for the filename
	2. Open a pipe from the file using the **StreamReader** object
	3. Read all of the Employee object data from the file. Remember that the data is in Comma Separated Values (CSV) format.
		1. Read a line
		2. Split the line into a string array called “parts” or something similar
		3. Use the items in the string array to build an Employee object
		4. Add the Employee object to the employees listbox
	4. Remember to close your **StreamReader** object!

If you want the Employee data to be automatically loaded when you start your application, add the ReadEmpsFromFile( ) method to the MainForm\_Load event method

# Deliverables Week 6

* Use the Word document template and put your information at the top. Take a screenshot while your application is running. YOU MUST ENTER YOUR NAME AS ONE OF THE EMPLOYEES. Paste the screenshot to a Word document below your information. Finally, copy-paste your Form code (only the code for the form, not program.cs) to the Word document below the screenshot. Save and close your Word document. Submit the Word document for your first Course Project deliverable.