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

**Course Number: CEIS247**

Background

**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 5: Create a Windows forms application with basic controls and add an Employee Class

# Objectives

1. To start the course project
2. Design and create two Windows forms with GUI controls
3. Read from and update GUI control variables

# Introduction

# Steps

## Start Visual Studio. You will see a Start Page that looks something like this. Read through it, and then click the “Create a new project” tile near the bottom, right corner.

##

1. **Create a new Windows Forms App (.NET Framework) – C# version**. If you do not see Windows Forms App (.NET Framework), simply type “Windows Forms App” in the search box and choose the C# version. Then, hit the **Next** button.


2. **When you configure your new project,** call it “**Lastname\_CourseProject\_part2**”. Make sure you change the location to somewhere that you will find it easily! Then, click the **Create** button.

 
3. **Make a new form**. You should be presented with a blank form like the one below. You are now ready to start designing your form! Currently it is called Form1, which is both boring and poor programming practice. A form, as well as forms controls, should have meaningful names, just like all other variables.



Rename the form’s file in the Solution Explorer to MainForm. Then, add four Buttons and a ListBox to your form. Feel free to play with the Font, BackColor, or any other property! Have fun! The form above was set to a font size of 20. Feel free to do the same so your form looks like the one below.

|  |  |  |
| --- | --- | --- |
| Object | Name | Text |
| Form | MainForm | Payroll System |
| Button | AddButton | Add |
| Button | RemoveButton | Remove |
| Button | DisplayButton | Display |
| Button | PrintPaychecksButton | Print Paychecks |
| ListBox | EmployeesListBox | n/a |

Your main form should look something like this. If you hit the green **Start** arrow, your application will run! Congratulations, you have created your first form! Click on the little red square to stop the program execution and go back to the editor.

 

When we click the buttons, nothing happens. We have not “activated” the buttons yet. We need to add the code for each button to execute. When you double-click on a button, Visual Studio will create the Event Handler and the method to go with the event. For example, when you double-click on the Add button, Visual Studio will write this line to handle the event:

this.AddButton.Click += new System.EventHandler(this.AddButton\_Click);

When a user clicks the AddButton, C# will run the “AddButton\_Click” method. I highlighted the event and the method for clarity. In other words, the code that you put into the AddButton\_Click will be executed when the user clicks the AddButton!

When the user clicks the AddButton, simply add a “New Employee” string to the employees listbox. When the user clicks the RemoveButton, get the selected item number (index) and remove the item at that item number. Check and make sure something is selected! When the user clicks the DisplayButton or the PrintPaychecksButton, show a message so the user knows that the button code has not been completed. When you finish, the code should look something like this:

using System;

using System.Windows.Forms;

namespace LastName\_CourseProject\_part2

{

 public partial class MainForm : Form

 {

 public MainForm()

 {

 InitializeComponent();

 }

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

 {

 // add item to the employee listbox

 EmployeesListBox.Items.Add("New Employee");

 }

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

 {

 // remove the selected item from the employee listbox

 int itemNumber = EmployeesListBox.SelectedIndex;

 if(itemNumber > -1 )

 {

 EmployeesListBox.Items.RemoveAt(itemNumber);

 }

 else

 {

 MessageBox.Show("Please select employee to remove.");

 }

 }

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

 {

 MessageBox.Show("Displaying all employees...");

 }

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

 {

 MessageBox.Show("Printing paychecks for all employees...");

 }

 }

}

Create an Employee class to encapsulate the employee information. It is much easier to work with a single Employee object instead of working with many separate variables. If I have fifty employee objects, I can get the details for an employee by simply asking the employee using the dot operator. For example, we can get the employee’s first name by using the dot operator and selecting the FirstName property:

employees[27].FirstName

Think about it. If I do not use encapsulation, I will need to create an array for each attribute that I want to track. Imagine having to maintain an application with as many as thirty arrays to track the details for the fifty employees! In the past, we used “parallel arrays” to track all of this information. Now, we use object-oriented programming with encapsulation to make our lives **a lot** easier! With encapsulation, the Employee object encapsulates or contains all of the details that we need to know about the employee. When we need a piece of information, we simply ask the Employee object using the dot operator!

We also need to implement “data hiding”. Data hiding simply means that we make our attributes or “data” hidden outside of the class. If someone wants to change one of the attributes, they will need to use our property. Our property will contain accessors and mutators. Accessors allow the outsider to “access” or get the attribute value. Mutators allow the outsider to “mutate” or “change” or set the attribute value. Since the outsider is forced to use our property, we can validate the information that the outsider is trying to give us. If the information is not valid, we can ignore it. By validating the inputs, we can protect our data.

Create an Employee class that encapsulates the important information that we want the Employee objects to contain. In addition, implement data hiding techniques to help reduce hacking and to protect our data! Here is the UML class diagram for the Employee class. Since we have no idea how to pay the generic employee, the CalculatePay( ) behavior should simply return zero.



# Deliverables Week 5

* Use the Word document template and put your information at the top. Take a screenshot while your application is running. 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. Paste a screenshot of your Employee class. Save and close your Word document. Submit the Word document for your first Course Project deliverable.