

Documentacion Tecnica

Nombre del Proyecto: HorarioPlus

Descripción:

HorarioPlus es una aplicación de gestión de empleados que permite llevar un registro de los empleados, sus horarios, asistencias, pagos y otras funciones administrativas.

Capas del Proyecto:

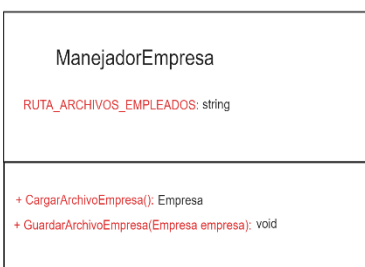
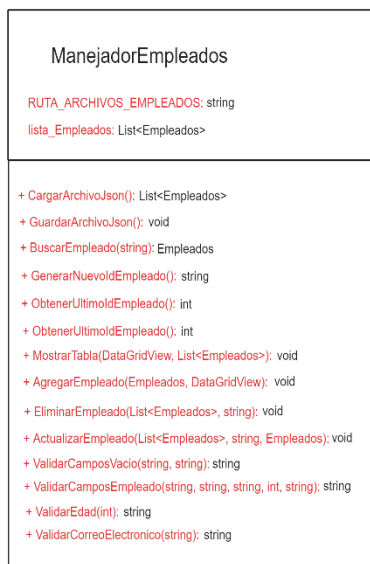
1. Capa de Datos
2. Capa de Entidades
3. Capa de Presentación

Funcionalidades Principales:

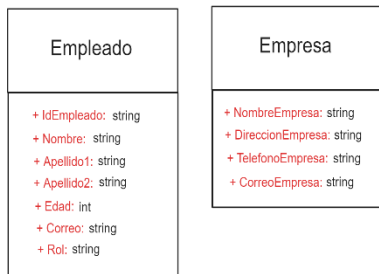
- Cargar y guardar información de empleados desde/hacia archivos JSON.
- Realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre la lista de empleados.
- Validar los datos ingresados por el usuario.
- Interfaz de usuario para el inicio de sesión, gestión de empleados y administración del sistema.

Diagrama de Clases

Capa Datos



Capa Entidades



Capa Presentacion



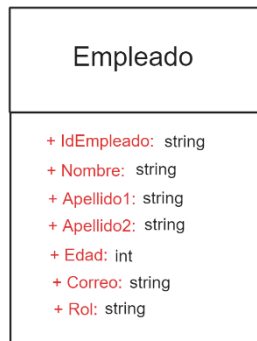
Capa Entidades

Clase Empleados

Esta es una clase que representa un empleado en el sistema, esta siendo parte de la capa entidades según la arquitectura del propio sistema.

Utilizamos {get; set;} para cada propiedad para dar un acceso de lectura y escritura, todos los atributos o propiedades de la clase empleado contienen esta implementación.

Luego creamos dos constructores para esta clase nombrado “Empleados”, uno estará vacío debido a la deserialización desde archivos JSON, y el segundo inicializa un nuevo objeto de esta misma clase con los datos especificados: id, nombre, apellido1, apellido2, edad, correo, rol. Más tarde explicamos donde usamos cada propiedad.



```

public class Empleados
{
    public string IdEmpleado { get; set; }
    10 referencias
    public string Nombre { get; set; }
    10 referencias
    public string Apellido1 { get; set; }
    7 referencias
    public string Apellido2 { get; set; }
    7 referencias
    public int Edad { get; set; }
    7 referencias
    public string Correo { get; set; }
    8 referencias
    public string Rol { get; set; }

    3 referencias
    public Empleados() // Constructor vacio debido a los archivos json
    {
    }

    0 referencias
    public Empleados(string id, string nombre, string apellido1, string apellido2, int edad, string correo, string rol)
    {
        this.IdEmpleado = id;
        this.Nombre = nombre;
        this.Apellido1 = apellido1;
        this.Apellido2 = apellido2;
        this.Edad = edad;
        this.Corrreo = correo;
        this.Rol = rol;
    }
}

```

Clase Empresa

Esta clase también contiene propiedades auto-implementadas por c#, cada una de las propiedades; NombreEmpresa, DireccionEmpresa, TelefonoEmpresa, CorreoEmpresa, tienen este formato lo que significa que se puede acceder a ellas desde fuera de la clase para obtener o establecer sus valores. Más tarde explicamos el uso de cada propiedad en otras clases.

Empresa	
+ NombreEmpresa:	string
+ DireccionEmpresa:	string
+ TelefonoEmpresa:	string
+ CorreoEmpresa:	string

```

public class Empresa
{
    public string NombreEmpresa { get; set; }

    public string DireccionEmpresa { get; set; }

    public string TelefonoEmpresa { get; set; }
    5 referencias
    public string CorreoEmpresa { get; set; }
}

```

Capa Datos

Clase ManejadorEmpleados (gestor de empleados)

```
15 referencias
public class ManejadorEmpleados
{
    #region VARIABLES y PROPIEDADES
    private const string RUTA_ARCHIVO_EMPLEADOS = @"C:\Users\maria magdalena\Desktop\MorarioPlus\MorarioPlus\archivos_empleados\Empleados.json";
    5 referencias
    public static List<Empleados> lista_Empleados { get; set; } = new List<Empleados>();
    #endregion
}
```

Ésta clase estará encargada de hacer la gestión con los datos de los Empleados, se explica más a detalle lo que contiene:

Primeramente, en este apartado se declara una constante de tipo string donde va a estar nuestra ruta exacta donde se encuentran nuestro archivo “Empleados.json” utilizado para almacenar los datos de los empleados, luego se define una Lista que contiene los datos de los empleados cargados desde el archivo JSON creando un objeto dentro de una lista, que su valor predeterminado será una nueva instancia de List<Empleados>.

#Region CRUD

Método CargarArchivoJson()

```
#region CRUD
// Metodo para cargar archivo json
1 referencia
public static List<Empleados> CargarArchivoJson()
{
    if (File.Exists(RUTA_ARCHIVO_EMPLEADOS))
    {
        try
        {
            string json = File.ReadAllText(RUTA_ARCHIVO_EMPLEADOS);
            // Deserializando el archivo JSON en una lista de objetos Empleado
            lista_Empleados = JsonSerializer.Deserialize<List<Empleados>>(json);
            return lista_Empleados;
        }
        catch (Exception error)
        {
            MessageBox.Show("Error al cargar el archivo de empleados en el DataGridView: " +
                error.Message,
                "Error",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    } // Si no existe archivo
    else
    {
        MessageBox.Show("Error: No se encuentra el archivo especificado", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    return null;
}
}
```

Este método se utiliza para cargar los datos de empleados desde un archivo JSON que habíamos definido al principio y que se deserializará en la lista de objetos Empleados. Un método público con modificador “static” para asociarlo en cualquier instancia del tipo. Nos va a retornar una lista de objetos Empleados cargados desde el archivo JSON, o retorna null si ocurre un error durante la carga. Dentro de la estructura try: declaramos una variable local “json” que será igual a “File.ReadAllText” (Se deberá hacer una referencia anteriormente para usar parámetro, using System.IO, y System.Json.Text para usar los archivos json correctamente) luego especificamos nuestra variable RUTA_ARCHIVOS_EMPLEADOS haciendo referencia a donde se encuentra el archivo.

Después de esto pasamos a la siguiente línea inicializando la lista_Empleados que será igual al método JsonSerializer.Deserialize<List<Empleados>>(json) para deserializar los objetos del archivo empleado en una lista objeto, esto se hace para que c# pueda leer estos archivos, y posteriormente retorna la lista de empleados. Dentro de catch está lo siguiente:

Excepciones:

- Si ocurre un error al intentar cargar el archivo JSON, se muestra un mensaje de error utilizando MessageBox.
- Si el archivo especificado no existe, se muestra un mensaje de error indicando que el archivo no se encuentra.

```
// Metodo para guardar lista en archivo json
4 referencias
public static void GuardarArchivoJson()
{
    JsonSerializerOptions opcionSerializar = new JsonSerializerOptions { WriteIndented = true };
    string json = JsonSerializer.Serialize(Lista_Empleados, opcionSerializar);
    File.WriteAllText(RUTA_ARCHIVO_EMPLEADOS, json);
}
```

Método GuardarArchivoJson()

Descripción: Este método es estático, por procedimiento y se utiliza para guardar los datos de empleados en un archivo JSON.

Parámetros: Ninguno

Retorno: Ninguno

Comportamiento: Se utiliza JsonSerializer para serializar la lista de objetos Empleados en formato JSON con opciones de formato indentado esto significa que es mas legible y estructurado. Se escribe el JSON resultante en el archivo especificado por la constante RUTA_ARCHIVOS_EMPLEADOS utilizando File.WriteAllText.

Método BuscarEmpleados()

```
// Metodo para BuscarEmpleado usando como parametro id
1 referencia
public static Empleados BuscarEmpleado(string idEmpleado)
{
    try
    {
        string json = File.ReadAllText(RUTA_ARCHIVO_EMPLEADOS);
        List<Empleados> empleados = JsonSerializer.Deserialize<List<Empleados>>(json);
        // Utilizamos LINQ para buscar el empleados con el id
        Empleados empleado = empleados.FirstOrDefault(e => e.IdEmpleado == idEmpleado);

        if (empleado == null)
        {
            MessageBox.Show("Empleado no encontrado", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        return empleado;
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error al leer el archivo de empleados : " +
            ex.Message,
            "Error",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        return null;
    }
}
```

Descripción: Este método es estático, retorna un valor y se utiliza para buscar un empleado por su ID en el archivo json de empleados.

Parámetros: idEmpleado (string): El ID del empleado que se desea buscar.

Retorno: El objeto Empleados correspondiente al ID proporcionado si se encuentra, o null si no se encuentra.

Comportamiento: Se lee el archivo JSON de empleados, se deserializa el JSON en una lista de objetos Empleados, se utiliza LINQ (recibido en clase 9) para buscar el empleado con el ID proporcionado. Si se encuentra al empleado devuelve el objeto Empleados correspondiente. Si no se encuentra el empleado entonces muestra un mensaje de error y retorna null. Si ocurre un error al intentar leer el archivo de empleados, se muestra el mensaje de error y también retorna null.

Método GenerarNuevoIdEmpleado()

```
// Metodo para generar un nuevo id
1 referencia
public static string GenerarNuevoIdEmpleado()
{
    int ultimoIdEmpleado = ObtenerUltimoIdEmpleado();
    int nuevoIdEmpleado = ultimoIdEmpleado + 1;
    return nuevoIdEmpleado.ToString();
}

//Metodo para obtener el ultimo ID empleado existente
1 referencia
public static int ObtenerUltimoIdEmpleado()
{
    int maxId = 0;
    foreach(var empleado in lista_Empleados)
    {
        int idEmpleado = Convert.ToInt32(empleado.IdEmpleado);
        if(idEmpleado > maxId)
        {
            maxId = idEmpleado;
        }
    }
    return maxId;
}
```

Descripción: Este método es estático, retorna un valor string y se utiliza para un nuevo id para un empleado, el nuevo id se calcula sumando 1 al ultimo ID empleado disponible.

Parámetros: Ninguno

Retorno: string: El nuevo ID generado como una cadena de texto.

Comportamiento: Se obtiene el último ID empleado disponible utilizando el método ObtenerUltimoIdEmpleado(). Se incrementa el último ID en 1 y se guarda en otra variable local llamada nuevoIdEmpleado, para generar nuevo ID. El nuevo ID se devuelve como una cadena de texto porque anteriormente era tipo entero.

Método ObtenerUltimoIdEmpleado()

Descripción: Este método es estático, retorna un valor int y se utiliza obtener el ultimo ID de empleado disponible.

Parámetros: Ninguno

Retorno: int: El nuevo ID de empleado disponible.

Comportamiento:

Se inicializa una variable maxId con el valor 0.

Se recorre la lista de empleados con un bucle foreach y se obtiene el ID de cada empleado.

Se compara cada ID obtenido con el valor actual de maxId.

Si un ID es mayor que maxId, se actualiza el valor de maxId con ese ID.

Una vez que se han recorrido todos los empleados, el valor de maxId será el último ID de empleado.

Por último, se devuelve el valor de maxId como el último ID de empleado.

Método MostrarTabla()

```
// Metodo para mostrar tabla actualizada
// referencia
public static void MostrarTabla(DataGridView dgvTablaEmpleados, List<Empleados> lista_Empleados)
{
    dgvTablaEmpleados.Rows.Clear(); // Limpiar tabla antes de agregar los empleados
    foreach (var empleado in lista_Empleados)
    {
        dgvTablaEmpleados.Rows.Add(empleado.IdEmpleado, empleado.Nombre, empleado.Apellido1, empleado.Apellido2, empleado.Edad, empleado.Correo, empleado.Rol);
    }
}
```

Descripción: Este método es estático, por procedimiento y se utiliza para mostrar una tabla actualizada de empleados en un DataGridView.

Parámetros:

dgvTablaEmpleados, tabla donde se mostrará toda la información.

lista_Empleados, lista de objetos que se utilizará para actualizar la tabla

Retorno: Ninguno

Comportamiento:

Se limpia la tabla usando Clear() antes de agregar los empleados, eliminando cualquier dato desactualizado.

Se recorre la lista de empleados con un bucle foreach.

Se agrega cada empleado a la tabla como una nueva fila, mostrando su ID, nombre, apellidos, edad, correo y rol, justo como están configuradas las columnas del datagridview.

Método AgregarEmpleado()

```
// Metodo para agregar un nuevo empleado a la lista
// referencia
public static void AgregarEmpleado(Empleados empleadoNuevo, DataGridView dataGridView)
{
    try
    {
        lista_Empleados.Add(empleadoNuevo);
        GuardarArchivo.Json();

        int rowIndex = dataGridView.Rows.Add();
        if (rowIndex >= 0 && rowIndex < dataGridView.Rows.Count)
        {
            dataGridView.Rows[rowIndex].Cells[0].Value = empleadoNuevo.IdEmpleado;
            dataGridView.Rows[rowIndex].Cells[1].Value = empleadoNuevo.Nombre;
            dataGridView.Rows[rowIndex].Cells[2].Value = empleadoNuevo.Apellido1;
            dataGridView.Rows[rowIndex].Cells[3].Value = empleadoNuevo.Apellido2;
            dataGridView.Rows[rowIndex].Cells[4].Value = empleadoNuevo.Edad;
            dataGridView.Rows[rowIndex].Cells[5].Value = empleadoNuevo.Correo;
            dataGridView.Rows[rowIndex].Cells[6].Value = empleadoNuevo.Rol;
        }
        else
        {
            MessageBox.Show("Error al agregar fila al DataGridView", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error al agregar empleado: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Descripción: Este método es estático, por procedimiento y se utiliza para agregar un Empleado a una tabla actualizada hacia DataGridView.

Parámetros:

empleadoNuevo (Empleados): El objeto Empleados que se desea agregar a la lista de empleados. dataGridView, El control DGV que se actualizará con los datos del nuevo empleado.

Retorno: Ninguno

Comportamiento:

- Se agrega el objeto empleadoNuevo a la lista lista_Empleados.
- Se guarda la lista de empleados actualizada en el archivo JSON utilizando el método GuardarArchivoJson().
- Se agrega una nueva fila al control DataGridView especificado.
- Se asignan los valores del nuevo empleado a las celdas de la nueva fila del DataGridView.
- Se manejan las excepciones y se muestran mensajes de error utilizando MessageBox en caso de errores durante el proceso.

Método EliminarEmpleado()

```
// Metodo para eliminar empleado de lista
1 referencia
public static void EliminarEmpleado(List<Empleados> lista_Empleados, string id)
{
    int indiceEmpleado = lista_Empleados.FindIndex(e => e.IdEmpleado == id);
    if(indiceEmpleado != -1)
    {
        lista_Empleados.RemoveAt(indiceEmpleado);
        GuardarArchivoJson();
    }
    else
    {
        MessageBox.Show("No se encontro el empleado con el ID especificado", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Descripción: Este método es estático, por procedimiento y se utiliza para eliminar un empleado de la lista de empleados basado en su ID y guardar los cambios en el archivo JSON.

Parámetros:

- lista_Empleados (List<Empleados>): La lista de empleados de la cual se eliminará el empleado.
- id (string): Usamo como parametro el ID del empleado que se desea eliminar.

Retorno: Ninguno

Comportamiento:

- Primeramente se busca el índice del empleado en la lista de empleados utilizando el ID proporcionado, usamos LINQ para reducir codigo.
- Si se encuentra el empleado (ó índice diferente de -1), se elimina de la lista de empleados.
- Se guarda la lista de empleados actualizada en un archivo JSON utilizando el método GuardarArchivoJson().
- Si no se encuentra el empleado (ó índice igual a -1), se muestra un mensaje de error.

Método ActualizarEmpleado()

```
// Metodo para actualizar empleado de lista
1 referencia
public static void ActualizarEmpleado(List<Empleados> lista_Empleados, string idEmpleadoOrigen, Empleados empleadoModificado)
{
    int indiceEmpleadoOriginal = lista_Empleados.FindIndex(e => e.IdEmpleado == idEmpleadoOrigen);
    if (indiceEmpleadoOriginal != -1)
    {
        lista_Empleados[indiceEmpleadoOriginal] = empleadoModificado;
        GuardarArchivoJson();
    }
    else
    {
        MessageBox.Show("No se encontro el empleado con el ID especificado", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
#endregion
```

Descripción: Este método es estático, por procedimiento y se utiliza para actualizar los datos de un empleado en la lista de empleados y guardar los cambios en el archivo JSON, este es nuestro último método que forma parte de nuestro sistema CRUD.

Parámetros:

- lista_Empleados (List<Empleados>): Esta es la lista de empleados en la que se actualizarán los datos del empleado.
- idEmpleadoOrigen (string): El ID que tomaremos del empleado que se desea actualizar.
- empleadoModificado (Empleados): El objeto Empleados que contiene los nuevos datos del empleado.

Retorno: Ninguno.

Comportamiento:

- Se busca el índice del empleado en la lista de empleados utilizando el ID proporcionado.
- Si se encuentra el empleado (índice diferente de -1), se reemplazan los datos del empleado en la lista con los datos del objeto empleadoModificado.
- Se guarda la lista de empleados actualizada en un archivo JSON utilizando el método GuardarArchivoJson.
- Si no se encuentra el empleado (índice igual a -1), se muestra un mensaje de error.

“Pasamos a los métodos que forman parte del sistema de validación de datos cuando se ingresen en los campos”

Método ValidarCampos()

```
#region VALIDACION DATOS
// Metodo para validarCampo
4 referencias
public string ValidarCampoVacio(string valor, string mensajeError)
{
    if (string.IsNullOrEmpty(valor))
    {
        return mensajeError + Environment.NewLine;
    }
    return string.Empty;
}
}
```

Descripción: Este método retorna un valor y se utiliza para validar si un campo de texto está vacío o nulo.

Parámetros:

- valor (string): Indicamos el parámetro valor del campo que se desea validar.
- mensajeError (string): El mensaje de error que se mostrará si el campo está vacío o nulo.

Retorno: simplemente retorna el mensajeError que hemos definido.

Comportamiento:

- Hacemos uso de la condicional If para comprobar si el valor del campo proporcionado está vacío o nulo utilizando la función **string.IsNullOrEmpty** y que por parámetro indicamos el valor.
- Por ende, si el campo está vacío o nulo, se devuelve el mensaje de error concatenado con un salto de línea utilizando Environment.NewLine.
- Si el campo tiene un valor válido, se devuelve una cadena vacía.

Método ValidarCamposEmpleado()

```
// Metodo para validarCampos en inputs
2 referencias
public string ValidarCamposEmpleado(string nombre, string primerApellido, string segundoApellido, int edad, string correo)
{
    string errorMsg = "";

    errorMsg += ValidarCampoVacio(nombre, "El nombre del empleado no puede estar vacio");
    errorMsg += ValidarCampoVacio(primerApellido, "El primer apellido del empleado no puede estar vacio");
    errorMsg += ValidarCampoVacio(segundoApellido, "El segundo apellido del empleado no puede estar vacio");
    errorMsg += ValidarCampoVacio(correo, "El correo del empleado no puede estar vacio");

    // Llamada al método de validación de edad
    errorMsg += ValidarEdad(edad);

    // Llamada al método de validación de correo electrónico
    errorMsg += ValidarCorreoElectronico(correo);

    return errorMsg;
}
```

Descripción: Aunque este método es similar al anterior lo hicimos con el propósito que el código sea más mantenible y se utiliza para validar los campos, como el nombre, apellidos, edad y correo y devuelve un mensaje de error que contiene todas las validaciones fallidas.

Parámetros:

- nombre (string): El nombre del empleado que se desea validar.
- primerApellido (string): El primer apellido del empleado que se desea validar.
- segundoApellido (string): El segundo apellido del empleado que se desea validar.
- edad (int): La edad del empleado que se desea validar.
- correo (string): El correo electrónico del empleado que se desea validar.

Retorno: string: Un mensaje de error que contiene todas las validaciones fallidas, separadas por saltos de línea.

Comportamiento:

- Aquí se inicializa una cadena de mensajes de error vacía.
- Se realiza la validación de cada campo llamando al método anterior ValidarCampoVacio para verificar si está vacío.
- Se realiza la validación de la edad llamando al método ValidarEdad que lo explicaremos después.
- Se realiza la validación del correo electrónico llamando al método ValidarCorreoElectronico.
- Se concatenan todos los mensajes de error en una única cadena y retornamos.

Método ValidarEdad()

```
// Método de validación de edad
1 referencia
public string ValidarEdad(int valor)
{
    if (valor == 0)
    {
        return "La edad del empleado no puede ser cero" + Environment.NewLine;
    }
    else if (valor < 18)
    {
        return "La edad del empleado no cumple los requisitos, debe ser mayor de edad" + Environment.NewLine;
    }
    return string.Empty;
}
```

Descripción: Este método lo hicimos con el propósito de utilizarlo para validar la edad del empleado ingresado por el textBox y devuelve un mensaje de error si la edad no cumple con los requisitos establecidos (cuando es menor de 18 años).

Parámetros:

valor (int): El valor de la edad del empleado que se desea validar.

Retorno: string: Un mensaje de error si la edad no cumple con los requisitos, o una cadena vacía si la edad es válida.

Comportamiento:

Aquí hacemos uso de la estructura condicional If anidado con else if.

- Se comprueba si el valor de la edad es igual a cero. Si es así, se devuelve un mensaje indicando que la edad no puede ser cero.
- Si el valor de la edad es menor que 18, se devuelve un mensaje indicando que la edad no cumple con los requisitos de ser mayor de edad.
- Si la edad cumple con los requisitos, se devuelve una cadena vacía.

Método ValidarCorreoElectronico()

```
// Método de validación de correo electrónico
1 referencia
public string ValidarCorreoElectronico(string correo)
{
    try
    {
        new MailAddress(correo);
        return string.Empty;
    }
    catch (Exception)
    {
        return "El correo del empleado debe tener un formato correcto" + Environment.NewLine;
    }
}
#endregion
```

Descripción: Este es nuestro último método de la clase ManejadorEmpleado y se utiliza para validar el formato de una dirección de correo electrónico (Ej: correo@gmail.com) y devuelve un mensaje de error si el formato no es válido.

Parámetros:

correo (string): La dirección de correo electrónico que se desea validar.

Retorno: string: Un mensaje de error si el formato de la dirección de correo electrónico no es válido, o una cadena vacía si el formato es correcto.

Comportamiento:

Usamos estructura de try catch:

- Se intenta crear una instancia de MailAddress con la dirección de correo electrónico proporcionada. Si se crea correctamente, significa que el formato es válido y se devuelve una cadena vacía.
- Si se produce una excepción al intentar crear la instancia de MailAddress, significa que el formato no es válido, y se devuelve un mensaje indicando que la dirección de correo electrónico debe tener un formato correcto.

Clase ManejadorEmpresa (gestor de datos empresa)

```
2. Referencias
public class ManejadorEmpresa
{
    #region RUTA ARCHIVO
    private static string RUTA_ARCHIVO_EMPRESA = @"C:\Users\maria_magdalena\Desktop\HorarioPlus\HorarioPlus\archivos_empleados\Empresa.json";
    #endregion
}
```

En esta region se declara una variable de tipo string donde va a estar nuestra ruta exacta donde se encuentran nuestro archivo "Empresa.json" utilizado para almacenar los datos de la empresa, desde otro archivo JSON creando un objeto nuevo.

Método CargarArchivoEmpresa()

```
#region METODOS
// DESERIALIZANDO ARCHIVO JSON
1. Referencia
public static Empresa CargarArchivoEmpresa()
{
    if (File.Exists(RUTA_ARCHIVO_EMPRESA))
    {
        string empresaJson = File.ReadAllText(RUTA_ARCHIVO_EMPRESA);
        return JsonSerializer.Deserialize<Empresa>(empresaJson);
    } // Si no existe archivo
    else
    {
        return new Empresa(); // podemos devolver una nueva instancia de empresa con valores default
    }
}
// Fin de la clase ManejadorEmpresa
```

Descripción: Este método se utiliza para cargar los datos de una empresa desde el archivo JSON antes declarado e inicializado y devolver una instancia de la clase Empresa con esos datos.

Retorno: Empresa: Retorna una instancia de la clase Empresa con los datos cargados desde el archivo JSON, o una nueva instancia con valores predeterminados si el archivo no existe.

Comportamiento:

Hacemos uso de la condicional if y else para verificar si:

- Si el archivo de la empresa existe en la ruta especificada.
- Si el archivo existe, se lee su contenido y se deserializa en un objeto de tipo Empresa utilizando la clase JsonSerializer.

- Se devuelve una instancia de Empresa con los datos cargados desde el archivo.
- Si el archivo no existe, se devuelve una nueva instancia de Empresa con valores predeterminados.

```
// SERIALIZANDO ARCHIVO JSON
1 referencia
public static void GuardarArchivoEmpresa(Empresa empresa)
{
    JsonSerializerOptions opcionSerializar = new JsonSerializerOptions { WriteIndented = true };
    string json = JsonSerializer.Serialize(empresa, opcionSerializar);
    File.WriteAllText(RUTA_ARCHIVO_EMPRESA, json);
}
#endregion
```

Capa Presentación

Formulario frmLogin

```
6 referencias
public partial class frmLogin : Form
{
    #region INICIALIZACION DE COMPONENTES
    2 referencias
    public frmLogin()
    {
        InitializeComponent();
        timerHora.Start();
    }

    // Metodo para mostrar reloj en pantalla
    1 referencia
    private void timerHora_Tick(object sender, EventArgs e)
    {
        lblHora.Text = DateTime.Now.ToString("hh:mm:ss tt");
    }
    #endregion

    #region EVENTO CLICK
    // Evento Boton Marcar
    1 referencia
    private void btnMarcarRegistro_Click(object sender, EventArgs e) ...
    #endregion
}
```

#region INICIALIZACION DE COMPONENTES

Descripción: Este bloque se encarga de inicializar los componentes del formulario de inicio de sesión y activar un temporizador para mostrar la hora actual.

Constructor:

Aquí el constructor frmLogin() inicializa todos los componentes del formulario y comienza el temporizado para actualizar la hora en tiempo real.

Métodos: timerHora_Tick(): este método está asociado al evento Tick del temporizador timerHora, que actualiza el texto del control lblHora con la hora actual formateada explicada en el siguiente evento.

```
#region EVENTO CLICK
// Evento Boton Marcar
// Referencia
private void btnMarcarRegistro_Click(object sender, EventArgs e)
{
    bool volverAMarcar = true;

    while (volverAMarcar)
    {
        string idEmpleado = txtEmpleadoId.Text;
        Empleados empleado_encontrado = ManejadorEmpleados.BuscarEmpleado(idEmpleado);

        if (empleado_encontrado != null)
        {
            string mensaje = $"{empleado_encontrado.Nombre} {empleado_encontrado.Apellido} registrado a las {DateTime.Now.ToString("hh:mm:ss tt")}";
            MessageBox.Show(mensaje, "Registro Exitoso", MessageBoxButtons.OK, MessageBoxIcon.Information);
            txtEmpleadoId.Clear();

            if (empleado_encontrado.Rol == "Administrador")
            {
                DialogResult resultado = MessageBox.Show("¿Desea ingresar al sistema como administrador?", "Confirmar Acceso", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

                if (resultado == DialogResult.Yes)
                {
                    this.Hide();
                    frmPanelAdministrador frmPanelAdmin = new frmPanelAdministrador(empleado_encontrado.Nombre, empleado_encontrado.Apellido);
                    frmPanelAdmin.ShowDialog();
                    volverAMarcar = false; // No volver a marcar si el usuario decide ingresar como administrador
                }
                else
                {
                    // Si el usuario elige "No", no volvemos a marcar y salimos del bucle
                    volverAMarcar = false;
                }
            }
            else
            {
                txtEmpleadoId.Clear();
                volverAMarcar = false;
            }
        }
        else
        {
            // Si el ID del empleado no existe, mostramos un mensaje de error
            MessageBox.Show("El ID del empleado no existe.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtEmpleadoId.Clear();
            volverAMarcar = false; // No volvemos a marcar si el id del empleado no existe
        }
    }
}
#endregion
```

#region EVENTO CLICK

btnMarcarRegistro_Click(): Cuando se hace click en el botón btnMarcarRegistro. Realiza la acción de marcar el registro del empleado.

Variables: volverAMarcar (bool): Declaramos e inicializamos variable de manera local que indica si se debe volver a marcar el registro después de una acción.

Proceso:

Hacemos uso del bucle while para que realice todo su interior mientras volverAMarcar sea verdadero.

Se obtiene el ID del empleado desde el control txtEmpleadold.

Se busca el empleado en la lista de empleados utilizando el método BuscarEmpleado de la clase ManejadorEmpleados.

Si se encuentra el empleado:

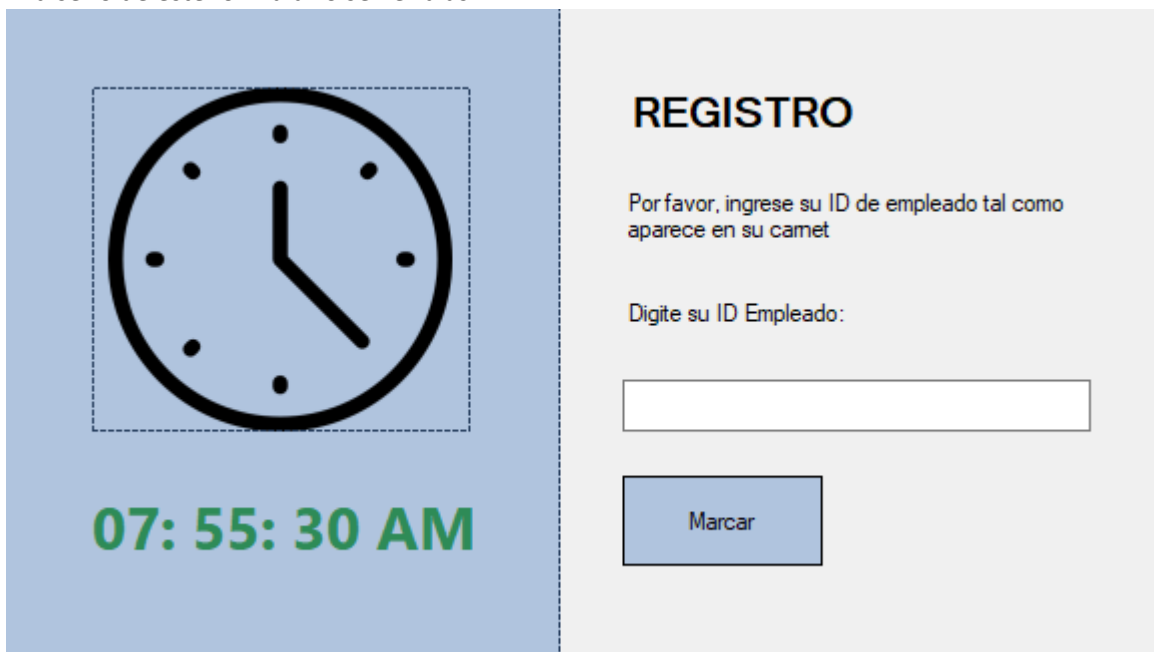
- Se muestra un mensaje de registro exitoso con el nombre del empleado y la hora de registro.
 - Si el empleado es un administrador, se ofrece la opción de ingresar al sistema como administrador.
 - Si se elige ingresar como administrador, se oculta el formulario actual, se crea una instancia del formulario frmPanelAdministrador y se muestra ese mismo formulario en forma de Dialogo y ponemos volverAMarcar en falso para que no vuelva a hacer el mismo proceso.

Si el ID del empleado no existe, se muestra un mensaje de error.

Si el empleado no es administrador, se limpia el control txtEmpleadold.

Se establece volverAMarcar a false para evitar volver a marcar si el ID del empleado no existe o si el usuario elige no ingresar como administrador.

El diseño de este formulario se vería así:



REGISTRO

Por favor, ingrese su ID de empleado tal como aparece en su carnet

Digite su ID Empleado:

Marcar

Formulario frmPanelAdministrador

```
4 referencia
public partial class frmPanelAdministrador : Form
{
    #region PROPIEDADES Y VARIABLES
    2 referencia
    public string NombreAdministrador { get; set; }
    2 referencia
    public string ApellidoAdministrador { get; set; }

    private static IconMenuItem MenuActivo = null;

    private static Form FormularioActivo = null;

    1 referencia
    public frmPanelAdministrador(string nombre, string apellido)
    {
        InitializeComponent();
        NombreAdministrador = nombre;
        ApellidoAdministrador = apellido;
    }
    #endregion
}
```

#region PROPIEDADES Y VARIABLES

Este bloque define las propiedades y variables utilizadas en el formulario de panel de administrador.

Propiedades:

NombreAdministrador (string): esta propiedad almacena el nombre del administrador para luego mostrarlo en un label.

ApellidoAdministrador (string): esta propiedad es similar a la anterior con el mismo propósito pero almacena el apellido del mismo.

Variables:

MenuActivo (IconMenuItem = estamos usando una librería para el diseño de botones y menús así que toma como nombre este y por tanto lo usamos) : esta variable es estática y almacena el menu activo del formulario.

FormularioActivo (Form) : esta también es una variable static que almacena el formulario activo o en uso.

Constructor:

frmPanelAdministrador(string nombre, string apellido): inicializa los componentes del formulario y asigna los valores de nombre y apellido que anteriormente habíamos definido en las propiedades.


```

#region CARGA && CIERRE FORMULARIO
1 referencia
private void frmPanelAdministrador_Load(object sender, EventArgs e)
{
    // Configuramos el lblUsuario con el nombre y apellido del administrador
    lblUsuario.Text = $"{NombreAdministrador} {ApellidoAdministrador}";
}

#endregion

```

#region CARGA FORMULARIO

Este bloque de código se encarga de realizar acciones durante la carga del formulario.

En el interior se ejecuta la configuracion del componente lblUsuario para convertirlo con el nombre y apellido del administrador, cuando X administrador se haya registrado antes.

#region EVENTOS CLICK

```

#region EVENTOS CLICK
1 referencia
private void btnCerrarSesionAdmin_Click(object sender, EventArgs e)
{
    DialogResult resultado = MessageBox.Show("Estas seguro que deseas cerrar tu sesion?", "Confirmar Cierre Sesion", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (resultado == DialogResult.Yes)
    {
        this.Close();
        frmLogin formularioLogin = new frmLogin(); // Crea instancia y llamado al login
        formularioLogin.Show();
    }
}

1 referencia
private void btnAjustes_Click(object sender, EventArgs e)
{
    frmAjustes formulario_Ajustes = new frmAjustes();
    formulario_Ajustes.ShowDialog();
}

#endregion

```

Este bloque contiene los manejadores de eventos para los clicks en los botones del formulario de panel de administrador.

El primer evento click es el manejador del click cuando se presione el botón “Cerrar Sesión”.

En el se encuentra la funcion “DialogResult” y luego inicializamos la variable local resultado que toma la respuesta del usuario mediante los botones del dialogo, en este caso “Si” y “No”. Creamos una condicional if: si el resultado es igual a sí o yes, entonces cierra el formulario presente (frmPanelAdministrador), luego creamos instancia del frmLogin y hacemos el llamado. Luego lo mostramos con Show().

El segundo evento click se ejecuta cuando se presiona el botón Ajustes.

En su interior creamos la instancia del formulario frmAjustes y la llamamos, luego lo mostramos en forma de dialogo.

#region EVENTOS FORMULARIOS

```
#region EVENTOS FORMULARIOS (ABRIR)
// Metodo para abrir formularios
// referencias
private void AbrirFormulario(IconMenuItem menu, Form formulario)
{
    if (MenuActivo != null)
    {
        MenuActivo.BackColor = Color.LightBlue;
    }
    menu.BackColor = Color.Silver;
    MenuActivo = menu;

    if (FormularioActivo != null)
    {
        FormularioActivo.Close();
    }
    FormularioActivo = formulario;
    formulario.TopLevel = false;
    formulario.FormBorderStyle = FormBorderStyle.None;
    formulario.Dock = DockStyle.Fill;
    formulario.BackColor = Color.SteelBlue;

    pnlContenedor.Controls.Add(formulario);
    formulario.Show();
}
// referencia
```

Este método se encarga de abrir un formulario en el panel de contenedor dentro del formulario principal. Toma como argumentos un ítem de menú y el formulario que se desea abrir. Hicimos este formulario para ahorrar código para cada vez que se abre un formulario en el menú.

Parámetros: Toma 2 parámetros

menu: El ítem de menú que se ha seleccionado para abrir el formulario.

formulario: El formulario que se abrirá en el panel de contenedor.

Acciones:

1. Si hay un menú activo anteriormente, se restaura su color de fondo a LightBlue para crear un efecto hover activo.
2. Se establece el color de fondo del menú actual como Silver para indicar que está activo.
3. Se actualiza la variable MenuActivo con el menú actual.
4. Si hay un formulario activo anteriormente, se cierra para que no se sobrecargue de formularios.
5. Se asigna el formulario especificado como el formulario activo.
6. Se configuran las propiedades del formulario para que se ajuste al panel de contenedor, como removerle los bordes, llenar el interior del formulario padre, y un color de fondo SteelBlue.
7. Se agrega el formulario al panel de contenedor.
8. Y se muestra el formulario.

EVENTOS CLICK PARA CADA FORMULARIO

```
1 referencia
private void menuAsistencias_Click(object sender, EventArgs e)
{
    AbrirFormulario((IconMenuItem)sender, new frmAsistencias());
}
1 referencia
private void subMenuNuevoRegistro_Click_1(object sender, EventArgs e)
{
    AbrirFormulario(menuEmpleados, new frmNuevoRegistroEmpleado());
}
1 referencia
private void subMenuHorarios_Click(object sender, EventArgs e)
{
    AbrirFormulario(menuEmpleados, new frmHorarioEmpleados());
}
1 referencia
private void subMenuPagos_Click(object sender, EventArgs e)
{
    AbrirFormulario(menuPagos, new frmPagos());
}
1 referencia
private void subMenuDeducciones_Click(object sender, EventArgs e)
{
    AbrirFormulario(menuPagos, new frmDeducciones());
}
1 referencia
private void menuNominas_Click(object sender, EventArgs e)
{
    AbrirFormulario((IconMenuItem)sender, new frmNominas());
}
1 referencia
private void menuCarnet_Click(object sender, EventArgs e)
{
    AbrirFormulario((IconMenuItem)sender, new frmCarnet());
}
#endregion
```

Para estos formulario y sus eventos click hay dos tipos de formularios, el primer modelo sería un menú, y el segundo un submenú, ahora explicamos sus acciones:

Acciones: Para los menús se llama al método AbrirFormulario con el sender (IconMenuItem) y la nueva instancia del formulario a llamar como argumento.

Para los submenús se llama al mismo método AbrirFormulario con el menú padre como primer parámetro sender y luego la nueva instancia del formulario a llamar como argumento.

Este formulario PanelAdministrador se vería así:

Panel Administrador

Administrador: Nombre Usuario

Ajustes Cerrar Sesión

Asistencias Empleados Pagos Nominas Carnet

Escriba aquí

Nuevo Registro

Horarios

Escriba aquí

Formulario frmNuevoRegistroEmpleado

#region VARIABLES

```
1 referencia
public partial class frmNuevoRegistroEmpleado : Form
{
    #region VARIABLES
    private List<Empleados> lista_Empleados = new List<Empleados>();
    private string nuevoIdEmpleado;
    private bool nuevoBotonPresionado;
    private Empleados empleadoSeleccionado;
    #endregion
}
```

Estas variables son utilizadas en el formulario para almacenar una lista de empleados, un nuevo ID de empleado, controlar el estado de un botón y almacenar el empleado seleccionado.

Variables:

1. lista_Empleados: esta es una lista que almacena el objeto de tipo Empleados.
2. nuevoIdEmpleado: Una cadena que almacena el nuevo ID de un empleado.
3. nuevoBotonPresionado: Un booleano que indica si se ha presionado un nuevo botón.
4. empleadoSeleccionado: Una instancia de la clase Empleados que representa al empleado seleccionado.

#region INICIO Y CIERRE FORMULARIO

```
#region INICIO Y CIERRE FORMULARIO
1 referencia
public frmNuevoRegistroEmpleado()
{
    InitializeComponent();
}

1 referencia
private void frmNuevoRegistroEmpleado_Load(object sender, EventArgs e)
{
    lista_Empleados = ManejadorEmpleados.CargarArchivoJson();
    ManejadorEmpleados.MostrarTabla(dgvTablaEmpleados, lista_Empleados);

    cbxRol.Items.Add(new OpcionCombo() { Valor = 1, Texto = "Empleado" });
    cbxRol.Items.Add(new OpcionCombo() { Valor = 2, Texto = "Administrador" });
    cbxRol.DisplayMember = "Texto";
    cbxRol.ValueMember = "Valor";
    cbxRol.SelectedIndex = 0;

    foreach (DataGridViewColumn columna in dgvTablaEmpleados.Columns)
    {
        if (columna.Visible && columna.Name != "btnSeleccionar")
        {
            // Obtener el nombre de la columna
            string nombreColumna = columna.HeaderText;

            // Agregar el nombre de la columna directamente al ComboBox
            cbxCategoriaBuscar.Items.Add(nombreColumna);
        }
    }
    dgvTablaEmpleados.SelectionChanged += SeleccionEmpleado_SelectionChanged;
}

0 referencias
private void frmPanelAdministrador_FormClosing(object sender, FormClosedEventArgs e)
{
    ManejadorEmpleados.GuardarArchivoJson();
}

#endregion
```

Estas regiones contienen métodos relacionados con el inicio y cierre del formulario “frmNuevoRegistroEmpleado”. En el inicio del formulario, se cargan datos, se configuran controles como ComboBox y se maneja el evento de cambio de selección en el DataGridView. En el cierre del formulario, se guarda el archivo JSON con los datos de los empleados.

Métodos:

frmNuevoRegistroEmpleado_Load(): Este método ejecuta las instrucciones cuando el formulario carga. Carga datos de empleados desde el archivo JSON haciendo un llamado a la clase ManejadorEmpleados + el método que se usará, hace otro llamado haciendo la misma acción e invoca al método Mostrar Tabla para que cargue la tabla con la información que toma como parámetros el componente del datagrid + la lista de empleados, configura el ComboBox de roles, y configura el ComboBox de búsqueda con las columnas visibles del DataGridView, y por último invoca un evento de SelectionChanged para que se ejecute la selección de empleados en una fila de la tabla.

frmPanelAdministrador_FormClosing(): este método se ejecuta al cerrar el formulario, donde hacemos el llamado del método de la clase ManejadorEmpleados que guarda los datos de los empleados en el archivo JSON con todos los cambios elaborados.

#region CRUD – Eventos Click

```
#region CRUD – EVENTOS CLICK
1 referencia
private void btnNuevo_Click(object sender, EventArgs e)
{
    NuevoEmpleado();
}
1 referencia
private void btnGuardar_Click(object sender, EventArgs e)
{
    InsertarNuevoEmpleado();
}
1 referencia
private void btnActualizar_Click(object sender, EventArgs e)
{
    ActualizarEmpleadoExistente();
}
1 referencia
private void btnEliminar_Click(object sender, EventArgs e)
{
    EliminarEmpleadoTabla();
}
1 referencia
private void btnCancelar_Click(object sender, EventArgs e)
{
    nuevoBotonPresionado = false;
    btnNuevo.Enabled = true;
    LimpiarEntradasTexto();
}
#endregion
```

Resumiendo un poco, estos son eventos para cada botón: Nuevo, Insertar, Actualizar, Eliminar y Cancelar.

Estos eventos hacen solamente llamados a los métodos que se mostrarán a continuación:

Método NuevoEmpleado()

```
#region METODOS PROCEDIMIENTOS
1 referencia
private void NuevoEmpleado()
{
    if (!nuevoBotonPresionado) // !enModoEdicion
    {
        nuevoBotonPresionado = true;
        btnNuevo.Enabled = false;
        btnInsertar.Enabled = true;
        btnActualizar.Enabled = false;
        btnEliminar.Enabled = false;
        nuevoIdEmpleado = ManejadorEmpleados.GenerarNuevoIdEmpleado();
        txtIdEmpleado.Text = nuevoIdEmpleado;
        LimpiarEntradasTexto();
    }
}
1 referencia
```

Se utiliza para agregar un nuevo empleado. Deshabilita el botón de nuevo lo que hace que no pueda presionarlo de nuevo, habilita el botón de insertar, y deshabilita los botones de actualizar y eliminar para que no se hagan estas acciones mientras se agrega un nuevo empleado. Además, genera un nuevo ID de empleado usando el método de la clase ManejadorEmpleado, luego asigna el nuevo ID de empleado al campo de texto txtIdEmpleado y limpia las entradas de texto del formulario.

Método InsertarNuevoEmpleado()

```
private void InsertarNuevoEmpleado()
{
    if (!string.IsNullOrEmpty(nuevoIdEmpleado))
    {
        OpcionCombo rolSeleccionado = (OpcionCombo)cbxRol.SelectedItem;
        string textoRol = rolSeleccionado.Texto;

        ManejadorEmpleados manejador_Empleados = new ManejadorEmpleados();
        string errorMsg = manejador_Empleados.ValidarCamposEmpleado
        (
            txtNombre.Text,
            txtPrimerApellido.Text,
            txtSegundoApellido.Text,
            (int)numEdad.Value,
            txtCorreo.Text
        );
        if (string.IsNullOrEmpty(errorMsg))
        {
            Empleados nuevoEmpleado = new Empleados()
            {
                IdEmpleado = nuevoIdEmpleado,
                Nombre = txtNombre.Text,
                Apellido1 = txtPrimerApellido.Text,
                Apellido2 = txtSegundoApellido.Text,
                Edad = (int)numEdad.Value,
                Correo = txtCorreo.Text,
                Rol = textoRol
            };

            ManejadorEmpleados.AgregarEmpleado(nuevoEmpleado, dgvTablaEmpleados);
            ManejadorEmpleados.MostrarTabla(dgvTablaEmpleados, lista_Empleados);
            LimpiarEntradasTexto();

            nuevoBotonPresionado = false;
            btnNuevo.Enabled = true;
        }
        else
        {
            MessageBox.Show("No hay informacion para agregar un nuevo empleado, llene todos los campos vacios",
                "Error",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
    else
    {
        MessageBox.Show("No hay informacion para agregar un nuevo empleado, por favor haga clic en 'Nuevo Registro' ",
            "Error",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```

Este método se utiliza para insertar un nuevo empleado en la lista de empleados y actualizar la tabla de empleados en el formulario. Antes de agregar al empleado, valida los campos del formulario para asegurarse de que están completos y en un formato válido.

1. Verifica si el ID del nuevo empleado no está vacío ni nulo.
2. Obtiene el rol seleccionado del ComboBox cbxRol
3. Valida los campos del empleado llamando al método ValidarCamposEmpleado de la clase ManejadorEmpleados pero antes creando la instancia.
4. Si no hay errores entonces procede a crear un nuevo objeto Empleados con los datos ingresados. Agrega el nuevo empleado llamando al método agregarEmpleado y actualiza la tabla también llamando al método MostrarTabla de la clase ManejadorEmpleados.
5. Limpia las entradas y restaura el estado de los botones.
6. Si hay mensajes de error con la validacion entonces muestra mensaje de error
7. Si el ID del nuevo empleado está vacío o nulo, muestra error que se debe hacer click en botón Nuevo para hacer un nuevo registro

Método ActualizarEmpleadoExistente()

```
1 referencia
private void ActualizarEmpleadoExistente()
{
    if (empleadoSeleccionado != null)
    {
        ManejadorEmpleados manejador_Empleados = new ManejadorEmpleados();
        string errorMsg = manejador_Empleados.ValidarCamposEmpleado
        (
            txtNombre.Text,
            txtPrimerApellido.Text,
            txtSegundoApellido.Text,
            (int)numEdad.Value,
            txtCorreo.Text
        );

        if (string.IsNullOrEmpty(errorMsg))
        {
            OpcionCombo rolSeleccionado = (OpcionCombo)cbxRol.SelectedItem;
            string textoRol = rolSeleccionado.Texto;

            // Crear un objeto Empleados con los datos modificados
            Empleados empleadoModificado = new Empleados()
            {
                IdEmpleado = empleadoSeleccionado.IdEmpleado,
                Nombre = txtNombre.Text,
                Apellido1 = txtPrimerApellido.Text,
                Apellido2 = txtSegundoApellido.Text,
                Edad = (int)numEdad.Value,
                Correo = txtCorreo.Text,
                Rol = textoRol
            };

            // Actualizar el empleado en la lista y mostrar la tabla actualizada
            ManejadorEmpleados.ActualizarEmpleado(lista_Empleados, empleadoSeleccionado.IdEmpleado, empleadoModificado);
            ManejadorEmpleados.MostrarTabla(dgvTablaEmpleados, lista_Empleados);
            LimpiarEntradasTexto();
        }
        else
        {
            // Mostrar mensajes de error de validación
            MessageBox.Show("Se encontraron los siguientes errores: " + Environment.NewLine +
                errorMsg,
                "Error de validación",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
    else
    {
        MessageBox.Show("No se ha seleccionado ningún empleado para actualizar.",
            "Error",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```

Se utiliza para actualizar los datos de un empleado existente en la lista de empleados. Primero valida los campos del formulario para asegurarse de que están completos y en un formato válido. Luego, crea un nuevo objeto Empleados con los datos modificados y llama al método ActualizarEmpleado del ManejadorEmpleados para actualizar el empleado en la lista. Finalmente, muestra la tabla actualizada en el formulario y limpia las entradas de texto.

Acciones:

1. Verifica si se ha seleccionado un empleado para actualizar.
2. Valida los campos del empleado llamando al método ValidarCamposEmpleado del ManejadorEmpleados.
3. Si no hay mensajes de error:
 - a. Obtiene el rol seleccionado del ComboBox cbxRol.
 - b. Crea un nuevo objeto Empleados con los datos modificados.
 - c. Llama al método ActualizarEmpleado del ManejadorEmpleados para actualizar el empleado en la lista.

- d. Muestra la tabla de empleados actualizada llamando al método `MostrarTabla` del `ManejadorEmpleados`.
 - e. Limpia las entradas de texto del formulario llamando al método `LimpiarEntradasTexto`.
4. Si hay mensajes de error, muestra un mensaje de error indicando los problemas de validación.
5. Si no se ha seleccionado ningún empleado, muestra un mensaje de error indicando que se debe seleccionar un empleado para actualizar.

Método `EliminarEmpleadoTabla()`

```
1 referencia
private void EliminarEmpleadoTabla()
{
    if(empleadoSeleccionado != null)
    {
        DialogResult resultadoEliminacion = MessageBox.Show($"Estas seguro que deseas eliminar a: {empleadoSeleccionado.Nombre} {empleadoSeleccionado.Apellido1} de los arch
        "Confirmacion", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
        if (resultadoEliminacion == DialogResult.OK)
        {
            ManejadorEmpleados.EliminarEmpleado(Lista_Empleados, empleadoSeleccionado.IdEmpleado);
            ManejadorEmpleados.MostrarTabla(dgvTablaEmpleados, Lista_Empleados);
        }
    }
    else
    {
        MessageBox.Show("No se ha seleccionado ningun empleado para eliminar",
            "Error",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```

Este método se utiliza para eliminar un empleado seleccionado de la tabla de empleados. Antes de realizar la eliminación, muestra un mensaje de confirmación para confirmar si se desea eliminar al empleado. Si el usuario confirma entonces se elimina al empleado de la lista y se actualiza la tabla de empleados mostrando los cambios actualizados.

Las instrucciones que ejecuta son:

1. Con una condición `if` verifica si hay un empleado seleccionado para eliminar.
 2. Luego muestra un mensaje con un `DialogResult` con opción de `OkCancel`, para confirmar la eliminación.
 3. Luego un `if` anidado que verifica si el usuario confirma la eliminación mediante un `OK`. Si `ok`, entonces se llama a los métodos `EliminarEmpleado()` y `MostrarTabla()` de la clase `ManejadorEmpleados` con sus respectivos parámetros.
- Si no hay ningún empleado seleccionado entonces muestra el mensaje de error.

Método `LimpiarEntradasTexto()`

```
4 referencias
public void LimpiarEntradasTexto()
{
    txtNombre.Text = "";
    txtPrimerApellido.Text = "";
    txtSegundoApellido.Text = "";
    txtCorreo.Text = "";
    numEdad.Value = 0;
    cbxRol.SelectedIndex = 0;
}
```

Este método simplemente manipula los controles `textbox` del formulario y los convierte en una cadena vacía, seguido del valor de `Edad` a cero y el `comboBox` al índice seleccionado a cero igualmente.

Método SeleccionarEmpleado()

```
1 referencia
public Empleados SeleccionarEmpleado()
{
    if (dgvTablaEmpleados.SelectedRows.Count > 0)
    {
        btnInsertar.Enabled = false;
        btnActualizar.Enabled = true;
        btnEliminar.Enabled = true;
        DataGridViewRow row = dgvTablaEmpleados.SelectedRows[0];
        // Creamos objeto para los datos de la fila seleccionada
        Empleados empleadoSeleccionado = new Empleados()
        {
            IdEmpleado = row.Cells["IdEmpleado"].Value.ToString(),
            Nombre = row.Cells["Nombre"].Value.ToString(),
            Apellido1 = row.Cells["Apellido1"].Value.ToString(),
            Apellido2 = row.Cells["Apellido2"].Value.ToString(),
            Edad = Convert.ToInt32(row.Cells["Edad"].Value),
            Correo = row.Cells["Correo"].Value.ToString(),
            Rol = row.Cells["Rol"].Value.ToString()
        };
        // Asignar los valores del empleado seleccionado a los campos de texto
        txtIdEmpleado.Text = empleadoSeleccionado.IdEmpleado;
        txtNombre.Text = empleadoSeleccionado.Nombre;
        txtPrimerApellido.Text = empleadoSeleccionado.Apellido1;
        txtSegundoApellido.Text = empleadoSeleccionado.Apellido2;
        numEdad.Value = empleadoSeleccionado.Edad;
        txtCorreo.Text = empleadoSeleccionado.Correo;
        cbxRol.SelectedIndex = cbxRol.FindStringExact(empleadoSeleccionado.Rol);

        return empleadoSeleccionado;
    }
    else
    {
        return null;
    }
}
```

Este método se utiliza para seleccionar un empleado de la tabla de empleados cuando se hace clic en una fila de la tabla. Una vez seleccionado el empleado, se muestran sus datos en los campos de texto o controles textBox correspondientes para su visualización y posible modificación.

Comportamiento:

1. Verifica si hay al menos una fila seleccionada en la tabla de empleados.
2. Si hay una fila seleccionada:
 - a. Deshabilitamos el botón de insertar para que no agregue un nuevo empleado pero sí habilita los botones de actualización y eliminación porque se ocuparán.
 - b. Obtenemos la fila seleccionada y extrae los datos del empleado.
 - c. Creamos un nuevo objeto de tipo Empleados con los datos del empleado seleccionado.
 - d. Asigna los valores del empleado seleccionado a los campos de texto correspondientes en el formulario.
 - e. Devuelve el objeto del empleado seleccionado.
3. Si no hay ninguna fila seleccionada, devuelve null.

El formulario “frmNuevoRegistroEmpleado” se vería algo así:

Detalle Empleado

ID Empleado
 + Nuevo Registro

Nombre


Primer Apellido



Segundo Apellido

Edad

Correo

Rol

 Insertar  Cancelar

 Actualizar  Eliminar

Lista Empleados

Buscar por: 

ID	Nombre	Apellido 1	Apellido 2	Edad	Correo	Rol
----	--------	------------	------------	------	--------	-----