To-DoTasks.

Es un organizador de Tareas que se maneja por Tareas y Listas de Tareas.

Las **Tareas Irregulares:**

Son Tareas que no tienen repetición y se dan en una fecha específica.

Por ejemplo: Ir al dentista el 14/2/2023 a las 14:30.

Las **Tareas Rutinarias**, que pueden ser de distintos tipos:

Diarias: Tareas que se repiten todos los dias.

Por ejemplo: Hacer cama, todos los dias, 09:00.

En Fecha específica: Tareas que se repiten cada mes en la fecha seleccionada.

Por ejemplo: Pagar mensualidad gym,28 de cada mes, 18:00.

Dias: Tareas que se repiten en un dia de la semana o un grupo de dias.

Por ejemplo: Ir al gimnasio, Lunes, martes, miércoles, jueves, 18:00.

Descripcion:

Las Tareas tienen una descripción/información que se le puede agregar sobre la Tarea que se va a realizar, algo que se desee especificar o información de la tarea.

La descripción de la Tarea figura en un recuadro al seleccionar las Tareas en el recuadro de Tareas de HOY

Por ejemplo: Ir al dentista el 14/2/2023 a las 14:30

Descripción: Av mitre al 2000, Turno con Perez, llevar plata para la consulta

Tambien estan las **Listas de Tareas,** que son un conjunto de Tareas con el mismo tipo de repetición.

Por ejemplo: Lista de Tareas gimnasio (Rutinaria)

Tareas:

Pecho y Bicep, Lunes, 18:00.

Descripcion : Press banca 4x10 etc..

Espalda y Tricep, martes, 18:00

Descripcion : Remo 4x10 etc..

Las Tareas van a tener el tipo de repetición de la Lista.

Si La lista de tareas es diaria, ejemplo, sus tareas van a ser diarias.

Todas las Tareas que estén asignadas para el día de la fecha (Incluyendo los que estan en listas), van a aparecer en un recuadro que se llama TAREAS DE HOY.

Desde ese recuadro vas a poder visualizar todas las tareas que tocan en ese día.

Ahí vas a poder ver su: El Nombre de la tarea, la hora, y el **estado** de la tarea.

Estado de la tarea:

Existen 3 estados de la tarea:

COMPLETO

PENDIENTE

INCOMPLETO

Estos estados se manejan por el horario.

Si el horario actual supera el horario establecido la tarea se marca como **INCOMPLETA.**

Mientras que esto no suceda la tarea estará PENDIENTE.

Una vez que haya completado la Tarea, la puedo marcar como COMPLETA con el botón de completar.

Si es una equivocación, y marque completa cuando no lo estaba. Puedo volver a tocar el botón que volverá a asignarlo como pendiente. Igual que cuando este incompleta, si la complete luego puedo Marcarlo o Desmarcarlo.

IMPLEMENTACIONES:

THREADS:

```
Thread relojThread = new Thread(ActualizarReloj);
relojThread.Start();
```

```
/// <summary>
/// Reloj (thread)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 referencia
private void ActualizarReloj()
{
    while (true)
    {
        // Actualizar el contenido del Label con la hora actual
        if (labelHoraActual.InvokeRequired)
        {
            labelHoraActual.Invoke(new MethodInvoker(delegate
            {
                labelHoraActual.Text = DateTime.Now.ToString("HH:mm:ss");
            }));
        }
        else
        {
            labelHoraActual.Text = DateTime.Now.ToString("HH:mm:ss");
        }

        Thread.Sleep(1000);
    }
}
```

METODOS DE EXTENSION, GENERICS Y DELEGADOS:

```
//Metodo de EXTENSION
0 referencias
public static class TareasExtensions
{
    1 referencia
    public static List<T> OrdenarPor<T>(this List<T> listaOriginal, Func<T, TimeSpan> obtenerHora)
    {

        for (int i = 0; i < listaOriginal.Count - 1; i++)
        {
            for (int j = 0; j < listaOriginal.Count - 1 - i; j++)
            {
                //DELEGADOS
                TimeSpan horaX = obtenerHora(listaOriginal[j]);
                TimeSpan horaY = obtenerHora(listaOriginal[j + 1]);

                if (TimeSpan.Compare(horaX, horaY) > 0)
                {
                    // Intercambiar elementos si están en el orden incorrecto
                    var temp = listaOriginal[j];
                    listaOriginal[j] = listaOriginal[j + 1];
                    listaOriginal[j + 1] = temp;
                }
            }
        }

        return listaOriginal;
    }
}
```

EXPRESIONES LAMBDA:

```
    //EXPRESION LAMBDA DEL METODO DE EXTENSION
    tareasYListas = tareasYListas.OrdenarPor(t => t is Tareas tarea ? tarea.Hora : TimeSpan.Zero);
```

ARCHIVOS Y SERIALIZACION:

```
// Método para serializar la lista de tareas a JSON
0 referencias
public void SerializarTareas(string filePath)
{
    string jsonString = JsonSerializer.Serialize(Tareas);
    File.WriteAllText(filePath, jsonString);
}

// Método para deserializar la lista de tareas desde JSON
0 referencias
public void DeserializarTareas(string filePath)
{
    if (File.Exists(filePath))
    {
        string jsonString = File.ReadAllText(filePath);
        List<Tareas> tareasDeserializadas = JsonSerializer.Deserialize<List<Tareas>>(jsonString);

        // Reemplazar la lista existente con la lista deserializada
        Tareas.Clear();
        Tareas.AddRange(tareasDeserializadas);
    }
}
```

EXCEPCIONES:

```
1 referencia
private void BtnAceptar_Click(object sender, EventArgs e)
{
    try
    {
        if (tareaSeleccionada != null)
        {
            ModificarTarea();
        }
        else
        {
            AgregarTarea();
        }

        // Solo cerrar el formulario si la tarea se agrega o modifica exitosamente
        this.DialogResult = DialogResult.OK;

        //this.Close();
    }
    catch (Exception ex)
    {
        // Mostrar mensaje de error, pero no cerrar el formulario
        MessageBox.Show($"Error al {(tareaSeleccionada != null ? "modificar" : "agregar")} la tarea: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

INTERFACES:

```
1 referencia
public interface ITareaManager
{
    /// <summary>
    /// Agrega una tarea.
    /// </summary>
    /// <param name="tarea">La tarea a agregar.</param>
    /// <exception cref="Exception">Puede lanzar una excepción genérica si ocurre un error.</exception>
    13 referencias
    void AgregarTarea(Tareas tarea);

    /// <summary>
    /// Modifica una tarea existente.
    /// </summary>
    /// <param name="tarea">La tarea con las modificaciones.</param>
    /// <exception cref="Exception">Puede lanzar una excepción genérica si ocurre un error.</exception>
    1 referencia
    void ModificarTarea(Tareas tarea);

    /// <summary>
    /// Elimina una tarea.
    /// </summary>
    /// <param name="tarea">La tarea a eliminar.</param>
    /// <exception cref="Exception">Puede lanzar una excepción genérica si ocurre un error.</exception>
    6 referencias
    void EliminarTarea(Tareas tarea);
}
```

EVENTOS Y SUSCRIPCIONES:

```
1 referencia
public frmListaDeTareasAgregarModificar(Tareas tarea, ListaDeTareasManager listaDeTareasManager, ListaDeTareas listaSeleccionada)
{
    InitializeComponent();

    this.listaSeleccionada = listaSeleccionada;
    this.listadeTareasManager = listaDeTareasManager;
    this.tareaSeleccionada = tarea;

    TareaAlta += MostrarMensajeAlta;
    TareaModificada += MostrarMensajeModificacion;
    TareaModificadaEnLista += MostrarMensajeModificacionLista;

    //ESTE CONSTRUCTOR ES PARA MODIFICAR UNA TAREA DE LA LISTA SELECCIONADA.

    if (this.listaSeleccionada is not null && this.listadeTareasManager is not null && tareaSeleccionada is not null)
    {
        // Configurar controles desde la lista seleccionada
        ConfigurarControlesDesdeListaSeleccionada();

        ConfigurarCamposTareaSeleccionada();
    }
}
```

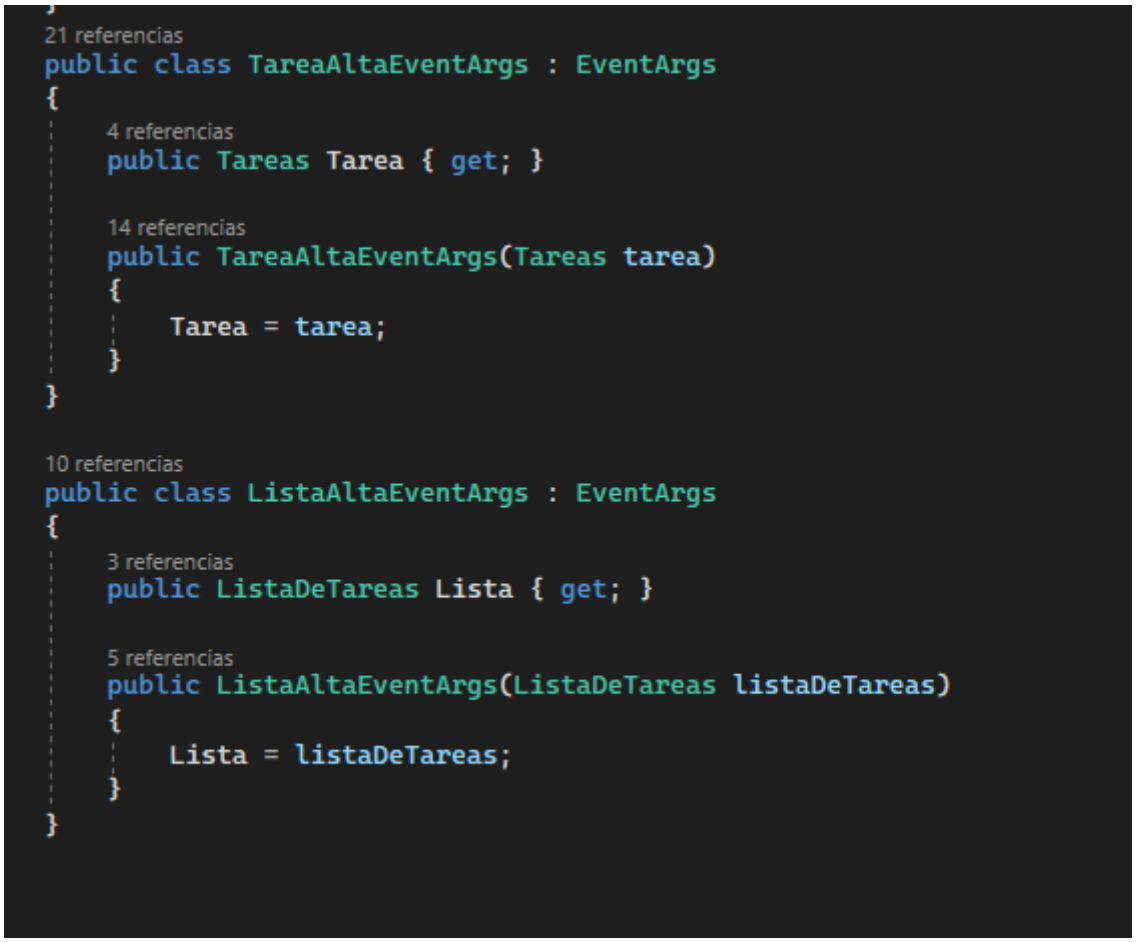
```
2 referencias
private void MostrarMensajeAlta(object sender, TareaAltaEventArgs e)
{

    MessageBox.Show($"La TAREA | '{e.Tarea.Nombre}' dada de alta.", "Éxito!", MessageBoxButtons.OK, MessageBoxIcon.Information);
}

2 referencias
private void MostrarMensajeModificacion(object sender, TareaAltaEventArgs e)
{

    MessageBox.Show($"La TAREA | '{e.Tarea.Nombre}' modificada.", "Éxito!", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
1 referencia
private void MostrarMensajeModificacionLista(object sender,ListaAltaEventArgs lista)
{

    MessageBox.Show($"fue modificada la tarea en la LISTA | '{lista.Lista.Nombre}'.", "Éxito!", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```



```
21 referencias
public class TareaAltaEventArgs : EventArgs
{
    4 referencias
    public Tareas Tarea { get; }

    14 referencias
    public TareaAltaEventArgs(Tareas tarea)
    {
        Tarea = tarea;
    }
}

10 referencias
public class ListaAltaEventArgs : EventArgs
{
    3 referencias
    public ListaDeTareas Lista { get; }

    5 referencias
    public ListaAltaEventArgs(ListaDeTareas listaDeTareas)
    {
        Lista = listaDeTareas;
    }
}
```

TEST UNITARIOS

```
namespace TestUnitarios
{
    [TestClass]
    0 referencias
    public class ValidarTextoNoVacioTests
    {
        [TestMethod]
        | 0 referencias
        public void ValidarTextoNoVacio_TextoNoVacio_NoDeberiaLanzarExcepcion()
        {
            // Arrange
            string texto = "Ejemplo";
            string nombreCampo = "NombreCampo";

        Clases.ValidarTextoNoVacio(texto, nombreCampo);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        | 0 referencias
        public void ValidarTextoNoVacio_TextoVacio_DeberiaLanzarExcepcion()
        {
            // Arrange
            string texto = string.Empty;
            string nombreCampo = "NombreCampo";

            Clases.ValidarTextoNoVacio(texto, nombreCampo);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        | 0 referencias
        public void ValidarTextoNoVacio_TextoNulo_DeberiaLanzarExcepcion()
        {
            // Arrange
            string texto = null;
            string nombreCampo = "NombreCampo";

            Clases.ValidarTextoNoVacio(texto, nombreCampo);
        }

    }
}
```