Contexto:

Es un programa para atención al cliente de un club de vinos, se asocian clientes pagando una cuota mensual y a cambio se le envían una caja distinta de vino cada mes.

Funcionalidad:

Agregar Cliente: Se deberá rellenar los campos pedidos y presionar el botón AGREGAR. Se deberá ser mayor de edad para agregar al cliente.

Dar de baja Cliente: Se deberá poner el DNI del cliente a eliminar en el txt en pantalla, si esta todo bien se mostrara en el rich a partir de ahí se podrá eliminar el cliente presionando en el botón especifico.

Modificar Cliente: Se deberá poner el DNI y buscar igual que en el apartado de dar de baja, si el cliente existe se mostrarán todos sus datos, desde ahí mismo se podrá modificar el cliente, por último, presionar el guardar.

Stock: Se debe seleccionar el tipo y la cantidad. Luego presionar en agregar o eliminar dependiendo que se quiera hacer.

Temas:

Excepciones: Este tema esta por todo el código, básicamente en cada form hay una excepción. Luego en la clase genérica “listado” en los métodos ADD y REMOVE, lanzo una excepción.

```
2 referencias
public bool add(T obj)
{
    bool retorno = false;
    if (obj is not null)
    {
        if (!EstaEnLaLista(this.lista, obj))
        {
            this.lista.Add(obj);
            retorno = true;
        }
        else
        {
            throw new EstaOnoEnlalista("Ya se encuentra en la lista!");
        }

    }
    return retorno;
```

```
1 referencia
public bool Remove(T obj)
{
    bool retorno = false;
    if (obj is not null)
    {
        if (this.EstaEnLaLista(this.lista, obj))
        {
            this.lista.Remove(obj);
            return true;
        }
        else
        {
            throw new EstaOnoEnlalista("No se encuentra en la lista");

        }
    }
    return retorno;
```

Genéricos: En la biblioteca de clases entidades esta la clase LISTADO que la utilizo tanto para los clientes como para el stock de las cajas de vino. Además, utilizo genéricos en la interfaz Iarchivos para usarla en la clase de archivos txt y la serializadora. Por último en la clase serializadora para serializar tanto el stock de cajas de vino como los clientes.

Interfaces: Utilizo la interfaz Iarchivos en la clase Archivotxt y claseSerializadoraXml.

```
public class ClaseSerializadoraXml<T> : IArchivos<T>
{

    protected string path;
```

```
3 referencias
public class ArchivoTxt : IArchivos<string>
{
    public string path;
    public string nombreArchivo;
```

```
public interface IArchivos<T>
{
    5 referencias
    T Leer(string nombreArchivo);
    5 referencias
    void Escribir(T elemento, string nombreArchivo);
}
```

Archivos: Para el apartado de archivos lo guardo dentro de, librería form/bin/debug/net5.0.

Ahí se encuentra un txt con la cantidad de cajas disponibles.

Serializacion: Se guardan en archivos xml serializados los clientes y las cajas de vino en el mismo lugar del punto anterior. Por otro lado en el código se encuentra el tema cuando se cierra el form principal, llamando al método guardarDatos.

Agrego del TP4:

SQL y conexión: Dentro de la clase CLIENTE DAO se encuentran todas las funciones para traer y modificar data de la base de datos, y estas son llamadas de diferentes partes del código: dentro de agregar cliente, modificar cliente, dar de baja cliente y dentro de sus respectivos botones.

DELEGADO: delegados por un lado uso en mi evento

```
{

    public event VerificarData DatosCargados;
```

Luego uso un delegado en el método cerrar aplicación en el que le paso un método para guardar los datos si se elige esa opción.

```
private void CerrarAplicacion(GuardarInformacion guardar, FormClosingEventArgs e)
{
    DialogResult dialogo = MessageBox.Show("¿desea guardar los datos antes de salir?",
    "Confirmacion de baja", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);
    if (dialogo == DialogResult.Yes)
    {
        guardar.Invoke();
    }
    else
    {
        if (dialogo == DialogResult.Cancel)
        {
            _ = e.Cancel = true;
        }
    }
}
```

Por último, para cargar todos los datos cree un delegado que encapsula dos métodos:

```
cargarInformacion += cargarDatosCaja;
cargarInformacion += cargarDatosClientes;
```

Para así llamar al delegado que encapsula los dos métodos encargados de traer la información.

Lambda:

En la clase Listado uso => en el siguiente método.

```
0 referencias
public string Listar()
{
    StringBuilder sb = new StringBuilder();
    sb.Append("");
    this.lista.ForEach((x) => sb.Append(x.ToString()));

    return sb.ToString();
}
```

Tambien uso lambda para simular que tardo en ir a buscar los datos a la base.

```
        /// </summary>
        1 referencia
        private async void cargarDatosClientes()
        {
            try

            {
                //Simulo que tarda en buscar a la base de datos.
                this.bacos.clientes.lista = await Task.Run<List<Cliente>>(() =>
                {
                    Thread.Sleep(3000);
                    return ClienteDao.Leer();
                });
                if (this.bacos.clientes.Cantidad < 1)
                {
```

MULTI HILO: Al traer la informacion de la base de datos lo hago en un segundo hilo asi se puede seguir utilizando el programa por mas que todavia no se termine de cargar los datos.

```
private async void cargarDatos()
{
    try
    {
        this.bacos.clientes.lista = await Task.Run(ClienteDao.Leer);
        DatosCargados.Invoke();
    }
    catch (BaseDeDatosException ex)
    {
        MessageBox.Show(ex.Message, "Error en carga de datos", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
    catch (Exception)
    {
        MessageBox.Show("Algo salio mal", "Error en carga de datos", MessageBoxButtons.OK,
          MessageBoxIcon.Exclamation);
    }

}
/// <summary>
/// Activa los botones les cambia el texto y les pone el color predeterminado
```

EVENTOS: en el método mencionado en multihilo lanzo un evento cuando se terminan de cargar los datos y asi poder activar todas las funcionalidades del programa.

Métodos de extensión: La clase validaciónExtensión tiene un método que extiende el tipo DateTime que se encarga de validar de si una fecha es correspondiente de un menor o un mayor de edad.


{
0 referencias
public static class ValidacionExtension
{
/// <summary>
/// Verifica que sea menor.
/// </summary>
/// <param name="fecha"></param>
/// <returns>Retorna true si es menor o false si no.</returns>
2 referencias
public static bool EsMenor(this DateTime fecha)
{
if (fecha.AddYears(18) > DateTime.Today)
{
return true;
}
return false;
}