TP3. Proyecto de Heladeria

Alumno: Francisco Allende
División: 2A
Docente: Maximiliano Neiner

El proyecto empieza permite procesar un pedido al venderlo y manejar el Stock desde la app. Esas son las dos opciones principales
Vale aclarar que tenia una fondo de imagen mucho más bonito pero no pude convertirlo a .resx

El programa comienza con el Form padre cargando varias listas dinámicas como atributos que se usan a lo largo del programa. **Estas listas son deserializadas de un xml si existe el folder y el file, sino, las hardcodea.** Esto sucede en el método **fill list del form1.cs**

```
if (File.Exists(directoryPath + nameFileStock)
    && File.Exists(directoryPath + nameFileRemovidos)
    && File.Exists(directoryPath + nameFileStockCafeteria)
    && File.Exists(directoryPath + nameFileVentas)
    && File.Exists(directoryPath + nameFileMesas))
{
    //No deserealizo en json porque ya lo hago con XML y no tiene sentido hacerlo do
    this.heladeraStock.ListaGenerica = Serializador.DeserializarXML(nameFileStock, t
    this.removidosStock.ListaGenerica = Serializador.DeserializarXML(nameFileRemovid
    this.cafeteria.ListaCafes = Serializador.DeserializarXML(nameFileStockCafeteria,
    this.ventas.ListaVentas = Serializador.DeserializarXML(nameFileVentas, this.vent
    this.listaMesas = Serializador.DeserializarXML(nameFileMesas, this.listaMesas);
```

```
else
{
    //No existe la ruta al directorio,
    this.HardcodearListaHelados();
    this.HardcodearListaCafes();
    this.HardcodearListaVentas();
    this.HardcodearListaMesas();
    this.SaveAndExport();
}
```

Cuando uno clickea un botón, por ejemplo, el de stock, el form padre se envía a sí mismo y a sus atributos para acceder a la misma lista dinámica, cuestión de poder editarla y que los cambios perduren.

```
case "Stock":
    Stock formStock = new Stock(this, this.heladeraStock, this.removidosStock);
    this.Hide();
    formStock.Show();
    break;
```

Ya deserializada la lista, o hardcodeada si no existe, accedemos, siguiendo el caso del stock, a un **datagrid con la lista como base de datos de la heladería, en la cual se establece un sistema CRUD.**
La heladera es el stock, siendo una heladera industrial de gran capacidad

- El form de stock permite remover un postre de la heladera
- Añadirlo si fue removido
- Editar su cantidad
- Update de la lista y de la tabla, cualquier operación es de inmediato reflejada en la tabla y guardada tanto en memoria, como en archivo si se decide a guardar
- Ordenar por Id

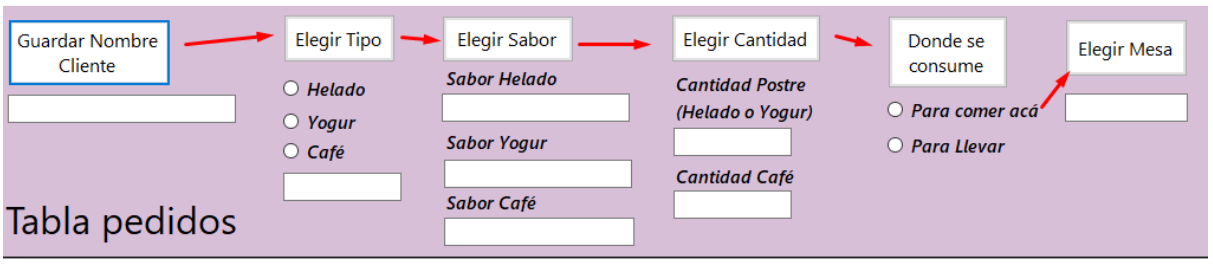
- Ordenar por mayor a menor cantidad
- Guardar los cambios, ya sea de manera voluntaria o preguntado como recordatorio antes de salir.

También calcula el stock total, el espacio disponible en la heladera y el sobrante. Permite advertir si se edita la cantidad por una superior a la capacidad disponible, anulando esa edición y pidiendo por una cantidad de stock más baja (a menos que se libere)

Todas las operaciones del CRUD se realizan con el Id del postre
Stock, al igual que TomarPedido, implementan una interface ITabla con operaciones comunes a tener una tabla con BBDD.

Cada pedido de id y **cada operación del crud posee un try catch en el evento**, con numerosos catchs según la excepción con mensaje **personalizado en cada caso, llegando a tener excepciones de mi biblioteca propia de excepciones**

El proceso de venta es un breadcrum, un paso a la vez. No se puede adelantar un paso sin haber hecho el primero.
Es por cliente el pedido, cada cliente puede tener numerosos pedidos, que se toman de manera individual.
No se puede, ya confirmado un pedido, cambiarle el nombre al cliente, ni donde consume, ni su mesa si es para comer en el local y mucho menos su id. **Esto es intencional y buscado**, ya que se atiende un cliente a la vez, pero con múltiples pedidos
Quiere decir, **si un cliente pide 1kg de helado, medio de yogur y un café, serán tres pedidos distintos**, tres recorridos de este form, pero a un mismo cliente
A la opción de la mesa solo se la requiere si el cliente come en el lugar, sino no es requerida



Al igual que la lista del stock, esta tabla posee try catch para mayor seguridad con catchs personalizados y algunos de mi propia clase de excepciones

- También permite el proceso del CRUD, con el confirmar como un add a la tabla
- El Cancel como un edit (borra los campos para empezar de nuevo pero no borra ningún pedido existente)

- El Remover para quitar pedidos ya confirmados, quitandolo de la tabla de pedidos confirmados
- y la opción de cobrar, que es el final del ciclo.
- Al igual que el stock, esto se actualiza según que se decida hacer (remover, añadir un nuevo pedido, etc).

El cobrar guarda el/los pedido/s y su respectivo cliente en la lista dinámica de ventas.
Además. genera un messagebox informando que se está imprimiendo la factura del pedido y siendo elaborado. Además, printea en el listbox la factura que simula el proceso de impresión. Una vez hecho esto, borra los campos para un cliente nuevo.