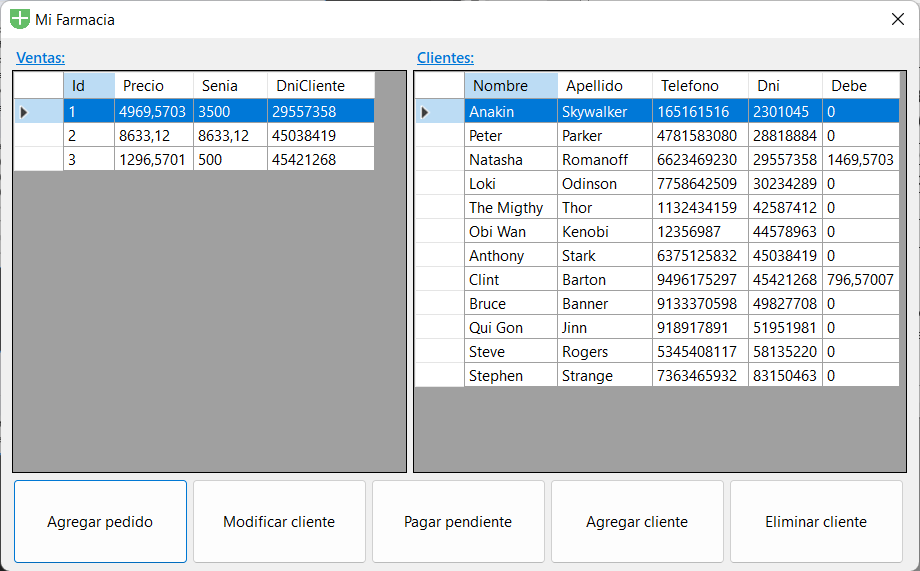


# Flujo del programa

El programa se basa todo en el Form principal, desde este podemos leer datos, abrir las distintas ventanas y realizar acciones con respecto a los clientes y las ventas.

## Inicio



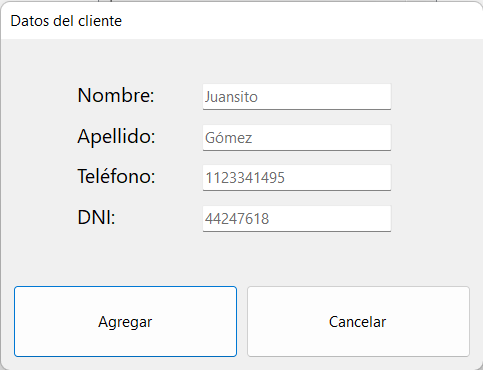
Ya hay algunos datos cargados dentro del programa.

Desde el **Inicio** podemos realizar las siguientes acciones:

## Agregar Cliente

Al presionar este botón, se abrirá una ventana con 4 campos para completar con datos del usuario, seguido de dos botones Agregar y Cancelar. Ninguno de estos campos puede quedar vacío, y deben seguir las siguientes reglas:

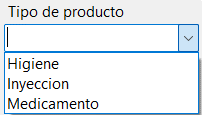
* **Nombre**: Sólo letras y espacios.
* **Apellido**: Sólo letras y espacios.
* **Teléfono**: Entre 3 y 15 números.
* **DNI**: Entre 7 y 9 números.



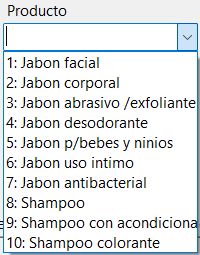
## Agregar pedido

Para agregar un pedido tenemos unos pasos a seguir:

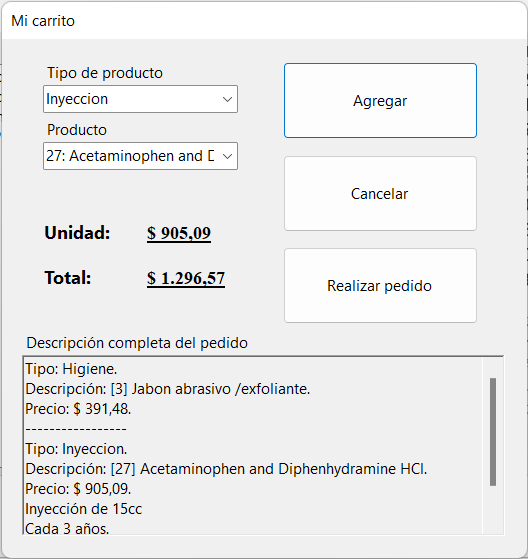
1. Seleccionamos un cliente desde la lista (no podemos realizar un pedido si no hay clientes)
2. Seleccionamos uno de los tres **tipos de productos** en la *primera lista desplegable*.



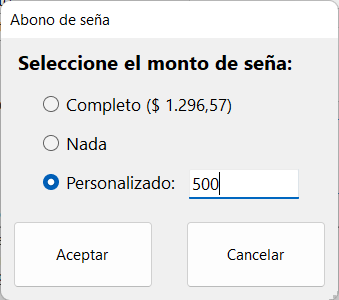
1. Seleccionamos el **producto** que queremos en la *segunda lista desplegable*.



1. Una vez seleccionado, abajo nos muestra el precio del producto, junto a su descripción, si decidimos agregarlo, presionamos el botón Agregar.
2. Si nos arrepentimos del pedido podemos cancelarlo (Cancelar), o si queremos seguir apretamos Realizar pedido.



### Abonar seña

Dentro de este menú tenemos tres opciones para abonar una seña:

El campo **Completo**, es seguido por el precio total de la venta.

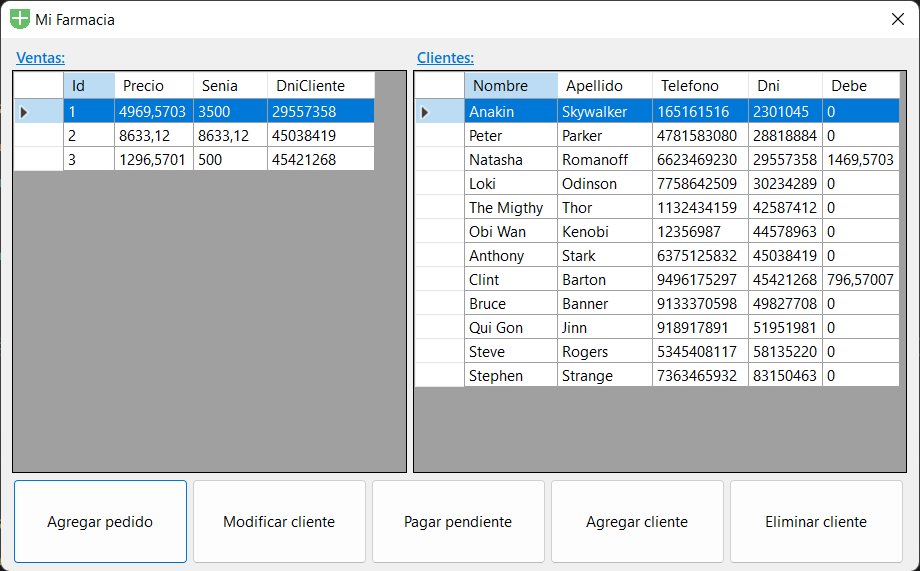
El valor en el campo **Personalizado**, no puede ser negativo ni mayor al valor total. Y si el monto es menor al 5% del valor total de la compra, este será denegado y tomado como 0.

Luego de seleccionar una opción, Aceptamos o Cancelamos.

### Vuelta al [inicio](#_Inicio)

De esta forma, ya tenemos una nueva venta asociada al DNI de un **Cliente**.

Una vez agregamos el pedido, podemos ver que se agrega a lista de la izquierda.



Si presionamos sobre el texto Ventas, este nos abrirá la aplicación de Notas con los datos de todas las ventas. Lo mismo con Clientes.

## Eliminar cliente

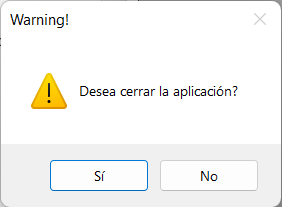
Esta opción es bastante simple y deductiva, al seleccionar un cliente y presionar el botón, nos pide una confirmación. Si confirmamos la acción, se borra el valor de la base de datos y continúa con la ejecución.

## Pagar pendiente

Esta acción es muy parecida a la anterior, tan solo que, en vez de eliminar al cliente, establece su deuda en $0. Esta acción también pide una confirmación, y modifica el campo debe del cliente en la base de datos a 0.

## Cerrar

No hay un botón *cerrar* como tal, pero podemos cerrar el programa desde la **X** en la esquina superior derecha. Al presionar este botón, nos pide una confirmación. Si decidimos cerrar, se escriben los datos de clientes en un XML y ventas en XML y .txt.



# Diagrama de clases

## Producto y derivadas



## Venta Cliente



## Extensiones



# Temas incluidos

## Excepciones

Decidí centrar las excepciones en el objeto [**Cliente**](#Cliente), las cuales se encargan de que los datos de este sean válidos y no haya fallos más adelante. Respecto a estas excepciones, cree una clase del tipo *Exception* como clase padre, para luego derivar de esta las otras excepciones y así hacerlo más fácil de atrapar si se lanza alguna.

Estas excepciones pueden encontrarse controladas en *FrmPrincipal* y en *FrmAgregarCliente*.

## Unit Testing

Dentro de la solución, se encuentra el proyecto **UnitTesting**, la cual tiene dos clases:

### ClienteTest

Aquí realizo 3 pruebas de cuatro métodos de la clase [Cliente](#Cliente):

* Constructor de Cliente (2)
* CadenaEsValida

### ProductoTest

En esta clase realizo dos test del mismo método de [Producto](#_Producto_y_derivadas), GetProductoPorId.

## Generics<T>

El tipo genérico lo incluí en la clase **Serializadora**, la cual se encarga de la lectura y escritura de archivos. Decidí utilizarla allí para hacer más fácil la tarea de manejo de archivos y no hacer sobrecarga de métodos por solo un tipo de dato.

## Interfaces

En este trabajo utilicé una sola interfaz, la cual tiene un método.

Esta interfaz la utilizo en dos [derivadas de **Producto**](#_Producto_y_derivadas:), **Inyección** y **Medicamento**.

## Archivos y serialización

El manejo de archivos fue especialmente el tema más interesante, me encontré con varios problemas por los datos que contenía y las propiedades que requería. Finalmente pude resolver los problemas y terminé con tres archivos .XML:

* ListadoDeProductos,
* ListadoDeClientes,
* ListadoDeVentas;

y dos archivos .txt donde se muestran todas las ventas realizadas y los clientes respectivamente (se guardan en Documentos):

* ListadoDeVentas,
* ListadoDeClientes.

## SQL y base de datos

En esta instancia de trabajo práctico decidí cambiar un poco lo hecho en el trabajo anterior. Los datos se almacenan en la base de datos **Farmacia**,con 4 tablas en vez de listas en la clase. Las tablas son:

* **Clientes**: Se guardan los datos de los clientes.
* **Productos**: Aquí se guardaría el "stock" de productos disponibles. Contiene un ID incremental.
* **Ventas**: Tiene 4 datos de las ventas (precio, seña, DNI del cliente y un ID incremental).
* **Carrito**: Esta tabla se encarga de conectar las últimas dos. Consta de un id para la venta y otro para el producto.

Estas tablas son manipuladas con sus clases respectivas dentro de la carpeta ADOs.



## Eventos y delegados

El programa cuenta con dos eventos, los cuales decidí crearlos como tipo de dato de delegado. Los eventos que cree son los siguientes:

### [SeniaInvalida](#EventoSeñaInvalida) (FrmMontoSenia)

**Delegado**: public delegate void DelegadoSeniaInvalida(float valor)

**Manejador**: public void ManejadorSeniaInvalida(float valor)

### AbrirArchivoTxt (FrmPrincipal)

**Delegado**: public delegate void DelegadoAbrirArchivo(string nombreArchivo)

**Manejador**: private void ManejadorAbrirArchivoTxt(string nombreArchivo)

## Hilos

Decidí que, al invocar [*AbrirArchivoTxt*](#_AbrirArchivoTxt_(FrmPrincipal)), este realice la acción dentro de un **Task**. Ubicado en FrmPrincipal, es invocado cuando se presiona el label de clientes o el de ventas.

## Métodos de extensión

Para finalizar, realicé 2 métodos de extensión, para facilitar el acceso a métodos bastante intuitivos.

* CortarEnCaracter(this string, char)
* GetProductoPorId(this int, List<Producto>)

Ambos en la clase [**Extensiones**](#_Extensiones).