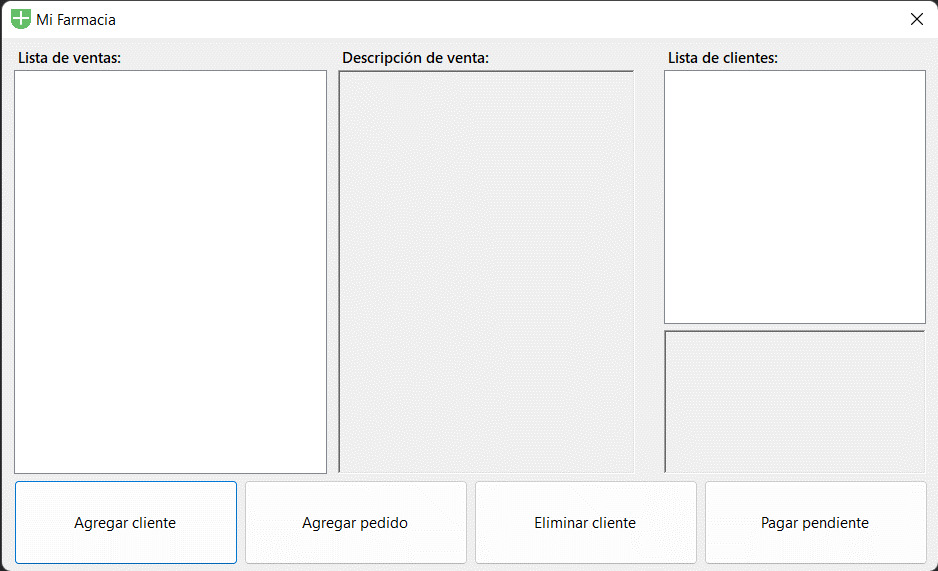


# Flujo del programa

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

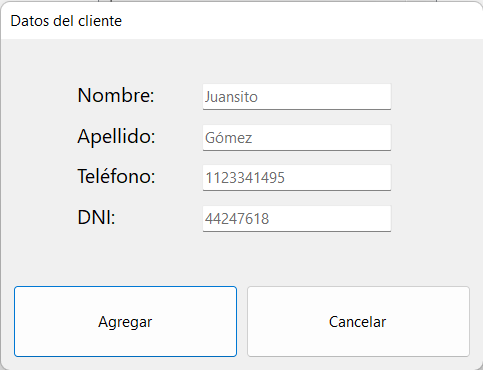


Desde el **Inicio** podemos realizar las siguientes acciones:

## 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:

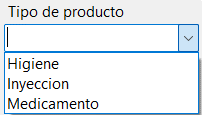
* **Nombre**: Sólo letras.
* **Apellido**: Sólo letras.
* **Teléfono**: Entre 3 y 15 números.
* **DNI**: Entre 7 y 9 números.



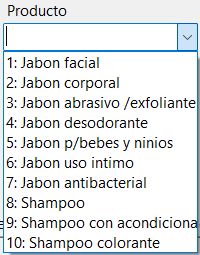
## Agregar pedido

Para agregar un pedido tenemos unos pasos a seguir:

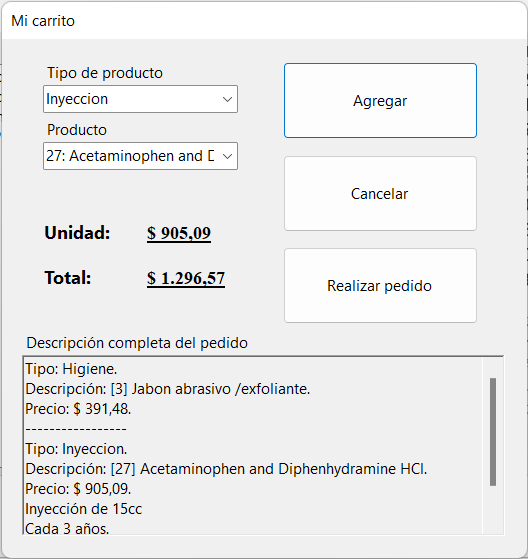
1. Seleccionamos un cliente desde la lista (no podemos realizar un pedido si no hay clientes)
2. Seleccionamos uno de los tres **tipos de productos** en la *primera lista desplegable*.



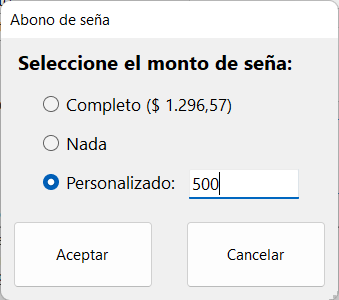
1. Seleccionamos el **producto** que queremos en la *segunda lista desplegable*.



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.

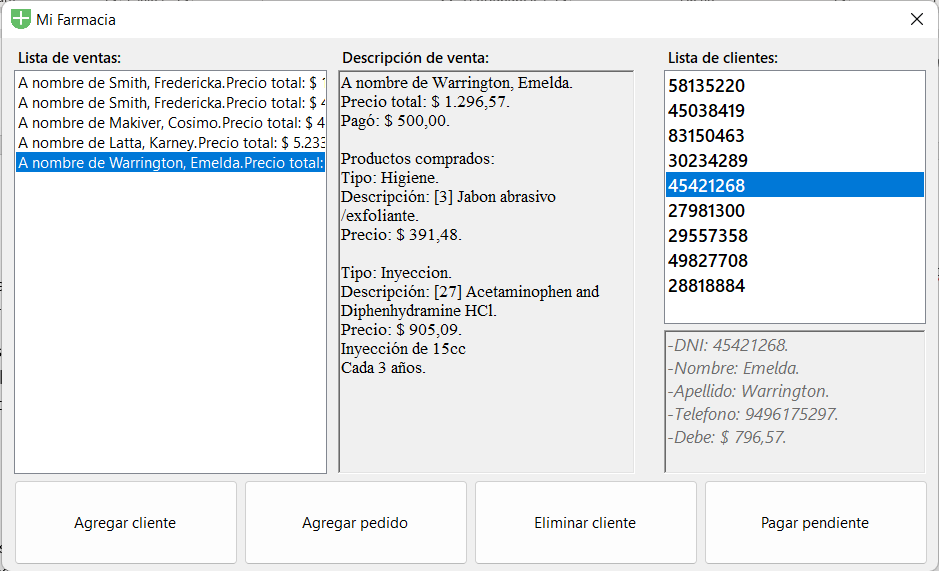
El valor en el campo **Personalizado**, no puede ser negativo ni mayor al valor total.

Luego de seleccionar una opción, Aceptamos o Cancelamos.

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

De esta forma, ya tenemos una nueva venta asociada a un **Cliente**, con todos sus atributos completos.

Una vez agregamos el pedido, podemos ver que se agrega a lista (izq) y cuando clickeamos, podemos ver su información más detalladamente en el cuadro (der).



Con el cliente es lo mismo, lo seleccionamos y abajo se muestran sus datos. También podemos ver que, al abonar $500 de seña (con un precio total de $1.296,57), el cliente queda debiendo $796,57.

## Eliminar cliente

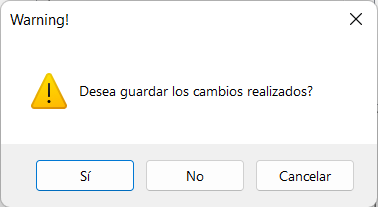
Esta opción es bastante simple y deductiva, al seleccionar un cliente y presionar el botón, nos pide una confirmación. Si confirmamos la acción, el programa busca en la lista de clientes al que concuerde con los datos del DNI seleccionado; una vez que lo encuentra, lo remueve y continúa con la ejecución.

## 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, tal como [Eliminar](#_Eliminar_cliente), busca el cliente con el DNI correspondiente y modifica su atributo debe en 0.

## Cerrar

No hay un botón *cerrar* como tal, pero podemos cerrar el programa desde la **X** en la esquina superior derecha. Cuando decidimos cerrarlo, nos pregunta si queremos guardar los datos modificados o no, o si queremos cancelar la acción. Si decidimos guardar, los nuevos datos se escriben y/o eliminan en el archivo.

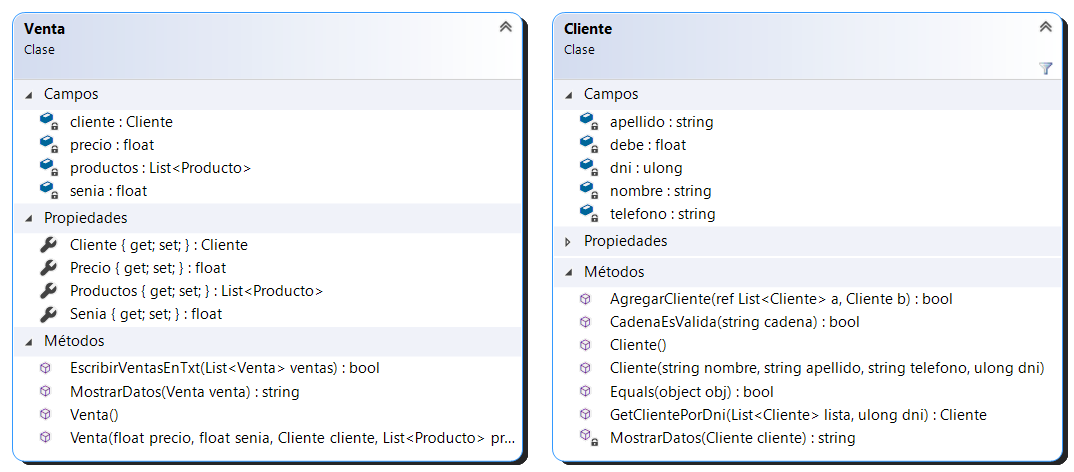


# Diagrama de clases

## Producto y derivadas



## Venta Cliente



# 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 5 pruebas de cuatro métodos de la clase [Cliente](#Cliente):

* GetClientePorDni
* Constructor de Cliente (2)
* CadenaEsValida
* AgregarCliente

### ProductoTest

En esta clase realizo dos test del mismo método, 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 un archivo .txt donde se muestran todas las ventas realizadas (se guarda en Documentos):

* ListadoDeVentas.