Universidad Tecnológica Nacional
Facultad Regional Avellaneda

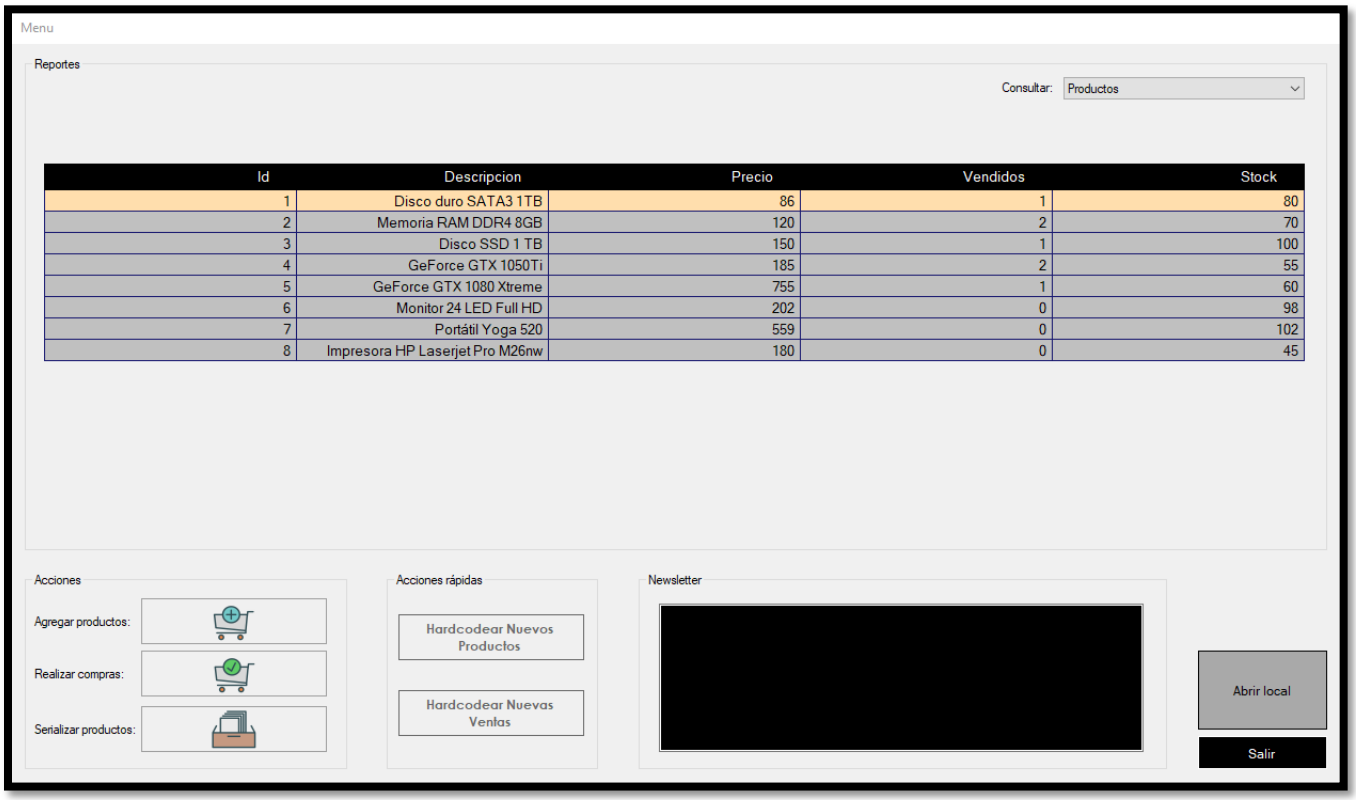
T.P. Nº4 de Laboratorio de Programación II

Acuña Zustovich, Alejo Matías. 2D

La aplicación levanta toda la información del negocio desde una base de datos: empleados, clientes, productos y ventas realizadas. La clase estática Database es la que se encarga de establecer las conexiones y ejecutar las operaciones para importar esta información, así como también para insertar los nuevos registros de productos y ventas que se agreguen desde la aplicación. Por su parte, la clase abstracta Inventario es la encargada de gestionar todas las entidades de nuestro negocio, manejando lista de Empleados, Clientes, Productos y Ventas.

En los formularios tenemos distintas funcionalidades, tales como:

- Visualizar reportes actualizados de todas nuestras entidades.
- Ingresar manualmente nuevos productos.
- Registrar manualmente una nueva venta.
- Serializar nuestro listado de productos.
- Acciones rápidas: hardcodear nuevas ventas y productos.
- Y por último una funcionalidad especial para abrir las puertas de nuestro local, permitiendo el ingreso de nuevos clientes mientras continuamos trabajando en la aplicación.

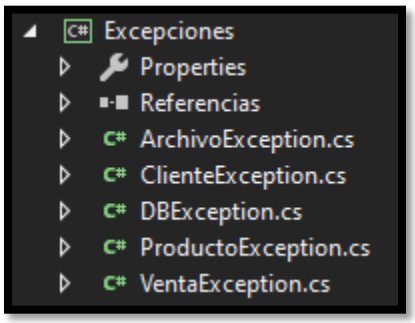


La aplicación de consola "**Test**", por otra parte, únicamente valida funcionalidades tales como la extracción de información desde la Base de datos, los reportes de cada una de las entidades, el ingreso de nuevas ventas y productos (hardcodeados y sólo agregados en la aplicación a modo de prueba, no se retornan a la base de datos) y, en ambos casos, el intento de ingreso de un elemento repetido para probar el lanzamiento de excepciones. Es un código simple y claro que repasa las funcionalidades básicas.

A continuación, se detallan los contenidos de las clases 15 a 25 y su respectiva implementación.

Clase 15: Excepciones

Existen 5 excepciones personalizadas dentro de la biblioteca de clases **Excepciones**:



- ***ArchivoException:*** se lanza si hay un error al guardar/serializar archivos, pudiéndose deberse esto al ingreso de algún parámetro erróneo, como serlo una ruta sin especificar.

- ***ClienteException:*** se lanza al intentar ingresar un Cliente que ya se encuentra registrado en el negocio.

- ***DBException:*** se lanza en caso de que ocurra algún error de mapeo tanto el importar como al exportar información de la base de datos.

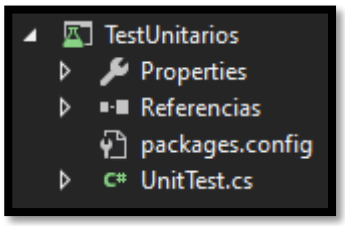
- ***ProductoException:*** se lanza en casos de querer ingresar un Producto que ya se encuentra registrado en el negocio, o bien si se intenta calcular el precio en el Formulario de Venta, sin haber seleccionado previamente un Producto.

- ***VentaException:*** se lanza en casos de intentar ingresar una Venta cuyo número de ticket ya se encuentra registrado en el negocio, o bien si se intenta finalizar una Venta sin completar los debidos campos en la Formulario.

Sumado a ello, también se lanzan, en distintas propiedades de las Entidades, excepciones de tipo ***FormatException*** para controlar que los datos ingresados sean válidos.

Clase 16: Test Unitarios

El proyecto **TestUnitarios** cuenta con la clase UnitTest, dentro de la cual se desarrollan 2 métodos:



`public void ProductoRepetido()`: el cual valida que se lance la excepción ProductoException en caso de que se intenten cargar 2 con el mismo Id.

`public void GuardarArchivoTexto()`: el cual valida que se lance la excepción ArchivoException en caso de que se intente guardar sin especificar la ruta del mismo.

Clases 17, 18, 19: Tipos Genéricos, Interfaces, Archivos y Serialización.

La biblioteca de clases Archivos contiene la Interface **IArchivos**, la cual posee la firma del método Guardar, el cual recibe un *string* que será la ruta del archivo, y un tipo genérico T el cual serán los datos que se guardarán en el archivo, pudiendo este adoptar el tipo de objeto que se desee en su implementación.

```
bool Guardar(string path, T datos);
```

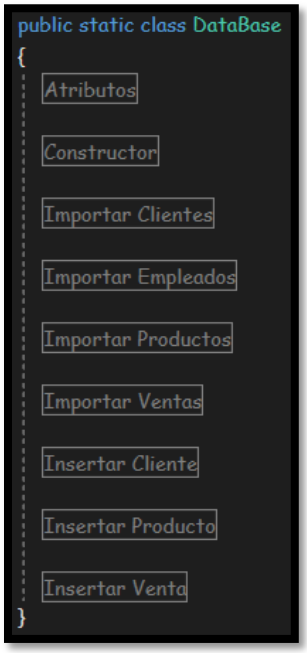
Las clases **Texto** y **XML** implementan la Interface IArchivos, y por ende el método Guardar.

- **Texto:** escribe una cadena de texto en el archivo. Este será próximamente invocado por el método de instancia Guardar, de la clase Compra, el cual imprimirá en un *.txt* los detalles del ticket de la compra que se realice.
- **XML:** serializa un archivo genérico. Este método será próximamente invocado por el método de clase Guardar, de Inventario, el cual serializa el listado completo de los productos que existen en el negocio al momento de generarse.

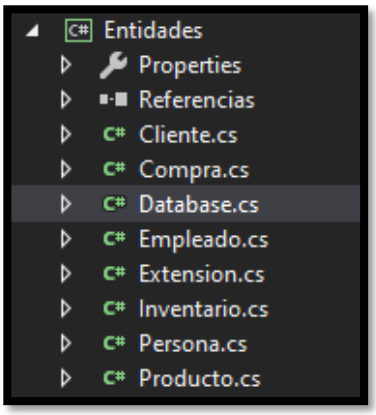
Los archivos se guardan en la carpeta Escritorio.

Clases 21, 22: SQL, Base de Datos.

En la biblioteca de clases Entidades se encuentra la clase estática **Database**, la cual contiene todos los métodos que establecen las acciones de conexión, extracción e inserción de registros de la Base de Datos.



Estos métodos realizan consultas a la base como ***SELECT***, para importar los registros de todas las entidades, e ***INSERT***, para guardar nuevos registros generados desde la aplicación.



En la carpeta ScriptDB se encuentra se encuentra el script **VentasTP4.sql** para generar la base de datos junto con las tablas y los registros iniciales.

Clases 23, 24: Hilos, Eventos.

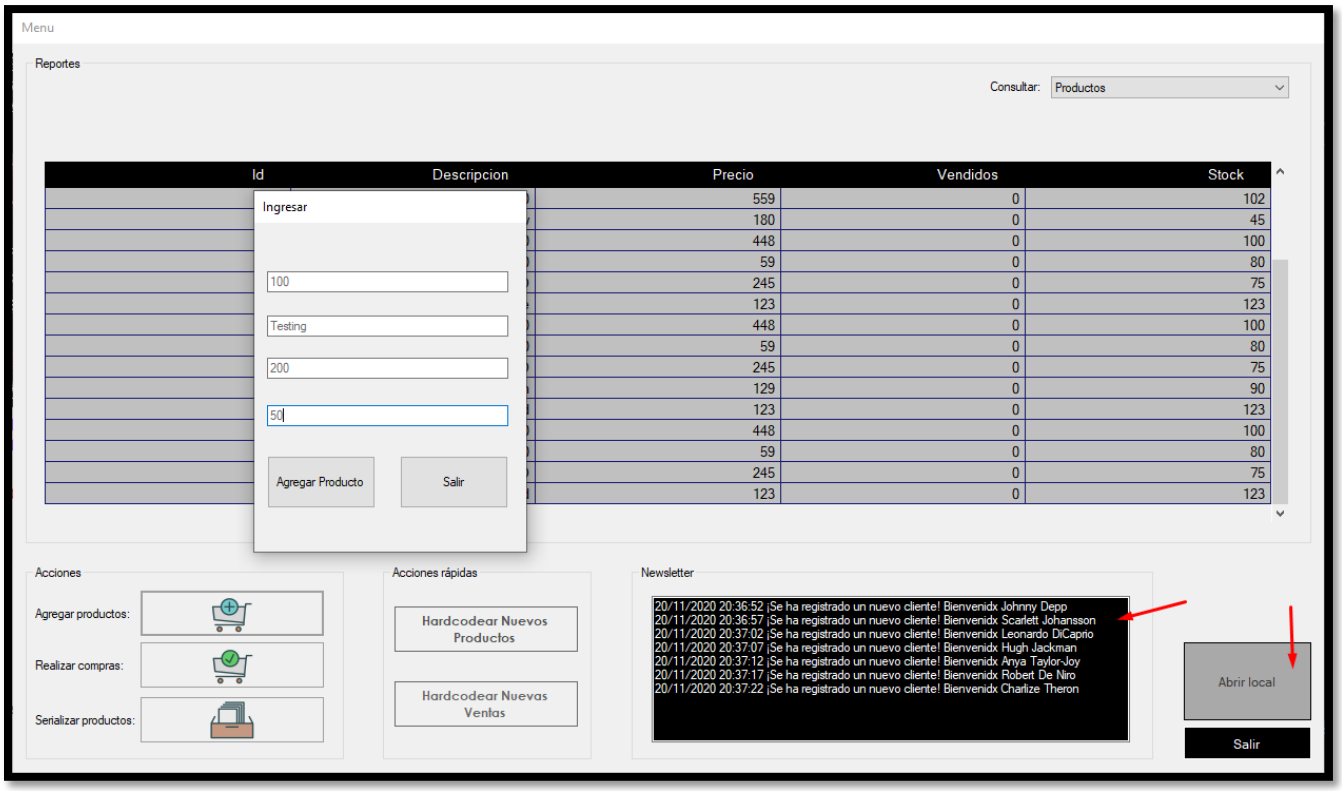
Dentro del Formulario **FormMenu**, se creó la funcionalidad de un Newsletter que va mostrando información relativa al negocio, al mismo tiempo que se continúan usando las demás funcionalidades de la aplicación.

Se crearon los siguientes atributos y métodos:

```
private Thread news;
private delegate void ActualizarTextBox(string news);
private static event ActualizarTextBox Abrir;

private void btnAbrirLocal_Click(object sender, EventArgs e);
private void AbrirLocal(string news);
private void IngresoClientes();
private void NuevosClientes();
private void ActualizarNewsLetter(string texto);
```

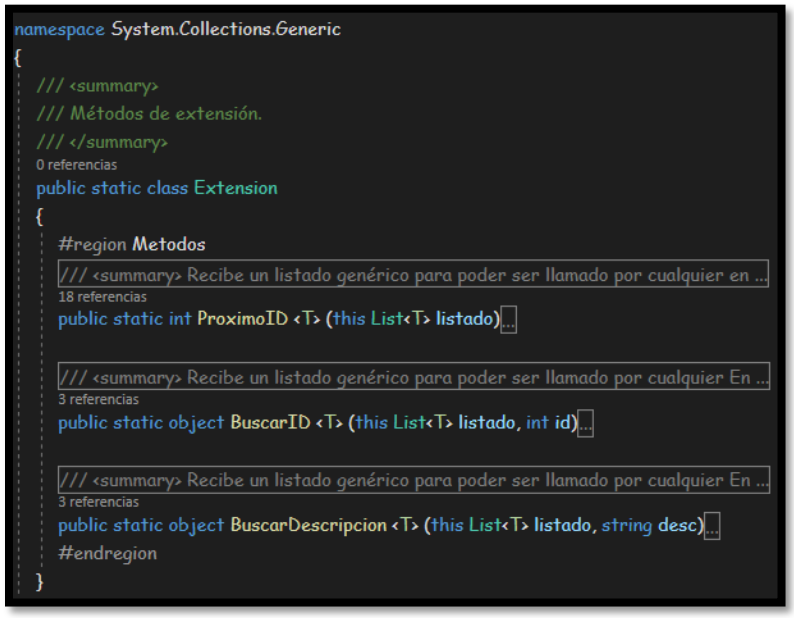
La implementación de Hilos, Delegados y Eventos comprende una simulación de local en el cual, recién abiertas las puertas del local, permitimos el ingreso de una cola de clientes que estaban esperando. A medida que van entrando, estos se van registrando como nuevos clientes nuestros. Mientras esto ocurre, el hilo principal del programa se sigue ejecutando, pudiendo realizar distintas acciones como visualizar informes, agregar productos, realizar compras, etc.



El evento clic del botón **Abrir Local** asigna un método al evento **Abrir**, el cual se ejecuta pasándole el mensaje *"Se abrieron las puertas del local"*. Este método muestra el mensaje por pantalla e invoca al siguiente método **IngresoClientes();** el cual inicia el Hilo, que recibe por parámetro el método **NuevosClientes();** que permite que comiencen a entrar los clientes y se vayan registrando uno a uno. El cliente ingresa, se registra, aparece la notificación en el Newsletter, se esperan 5 segundos y se vuelve a repetir el ciclo. Para poder actualizar el TextBox del Newsletter delego la acción a mi delegado de tipo **ActualizarTextBox**, ya que el control del mismo pertenece al hilo principal.

Clase 25: Métodos de Extensión

Dentro del proyecto Entidades se encuentra la clase estática **Extension**, contenida dentro del namespace System.Collections.Generic. Esta clase contiene 3 métodos estáticos que extienden el listado que reciban por parámetro, el cual es de tipo genérico <T> para que, en la implementación, pueda recibir listados de cualquier tipo de Entidad, ya sean productos, empleados, clientes o ventas.



`public static int ProximoID <T> (this List<T> listado)`: Recorre el listado recibido y retorna el próximo ID a dar de alta para la Entidad correspondiente.

`public static object BuscarID <T> (this List<T> listado, int id)`: Recorre el listado recibido y retorna el objeto de la Entidad correspondiente que coincida con el ID pasado por parámetro.

`public static object BuscarDescripcion <T> (this List<T> listado, string desc)`: Recorre el listado recibido y retorna el objeto de la Entidad correspondiente que coincida con la Descripción pasada por parámetro.

Ejemplos de implementación:

La extensión ***ProximoID()*** se implementa al hardcodear nuevos productos.

```
Producto p1 = new Producto(ListadoProductos.ProximoID(),
ListadoProductos += p1;
DataBase.InsertarProducto(p1);

Producto p2 = new Producto(ListadoProductos.ProximoID(),
ListadoProductos += p2;
DataBase.InsertarProducto(p2);
```

La extensión ***BuscarID()*** se implementa para buscar las Entidades por el ID pasado por parámetro, al importar las ventas desde la Base de Datos.

```
while (dr.Read())
{
    compra += new Compra(int.Parse(dr["id"].ToString()),
        (Empleado)Inventario.ListadoEmpleados.BuscarID(int.Pa
        (Cliente)Inventario.ListadoClientes.BuscarID(int.Parse(
        (Producto)Inventario.ListadoProductos.BuscarID(int.Par
        int.Parse(dr["cantidad"].ToString()), int.Parse(dr["impor
}
```