Soria Federico Martín. 2A

Universidad Tecnológica Nacional
Facultad Regional Avellaneda

T.P. Nº4 de Laboratorio de Programación II

Funcionamiento general del programa

La aplicación consiste en una plataforma de stock y ventas de productos farmacéuticos (medicamentos y suplementos) que contiene dos Data Table con los productos presentes en la base de datos. En el formulario principal se pueden ejecutar altas de productos nuevos, como también bajas y modificaciones de los que ya hay cargados en el Data Table. Los cambios se ven reflejados en la base de datos únicamente cuando se decide actualizar la misma mediante el botón **Sincronizar con BD** y cuando se ingresa a la plataforma de ventas.

Por otro lado, el FormPrincipal también contiene un botón cuyo nombre es **Vender**, que permite al usuario efectuar una venta de los productos disponibles en el stock. Al generarse una venta, también se crea un archivo de tipo texto que va recopilando todas las operaciones realizadas desde la primera venta, desplegando información como la fecha, hora, datos de los productos y precio total de la venta.

Con respecto a la plataforma de ventas (**FrmVender**), en los catálogos desplegados se ocultan de forma adrede algunos datos como los id's, empaque (en el caso de los suplementos) y origen (en el caso de los medicamentos) con fines estéticos. Para la selección y deselección de productos en este formulario, se deberá apretar "Ctrl" e ir seleccionando los elementos que se desean quitar o agregar.

A continuación, se adjuntan imágenes correspondiente a los winforms utilizados en este programa.

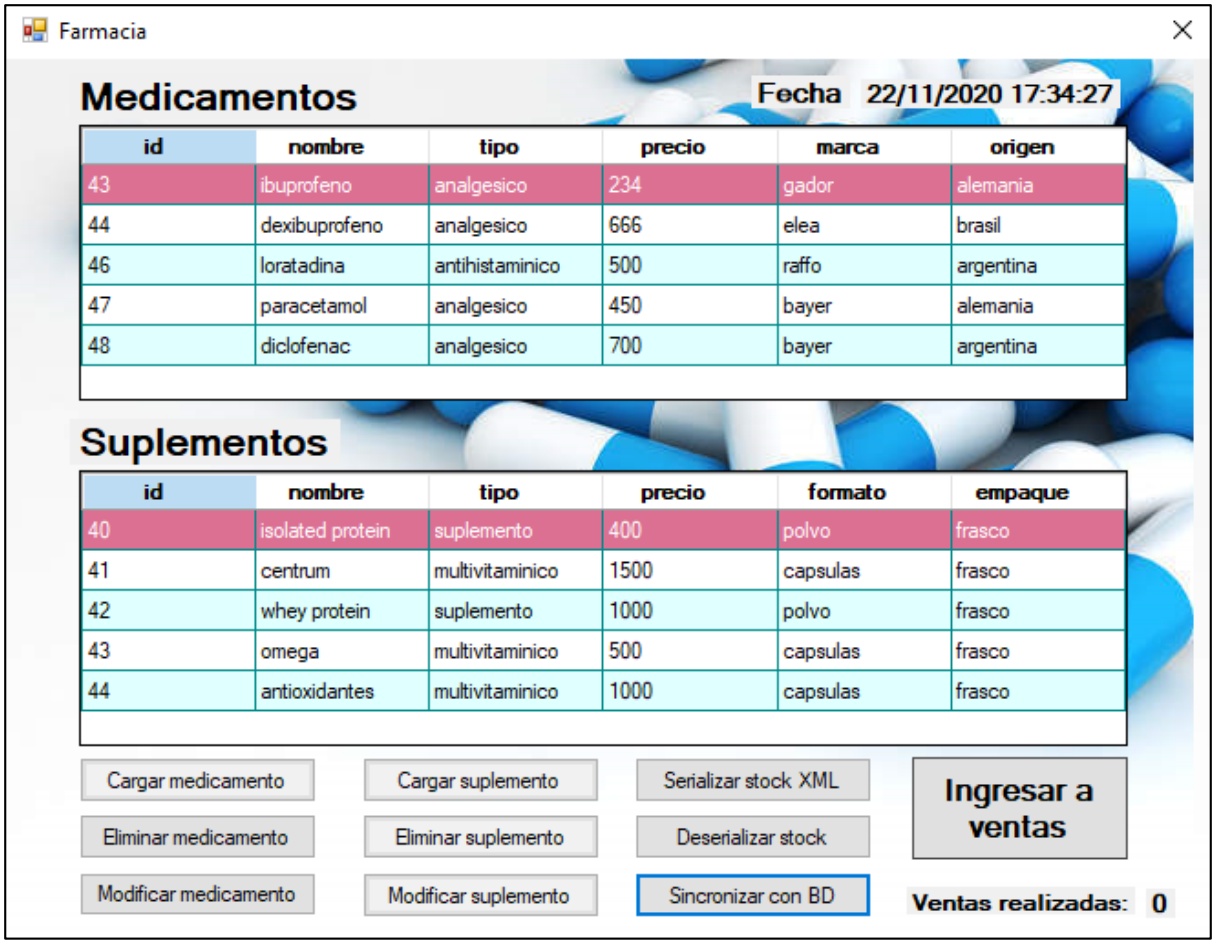
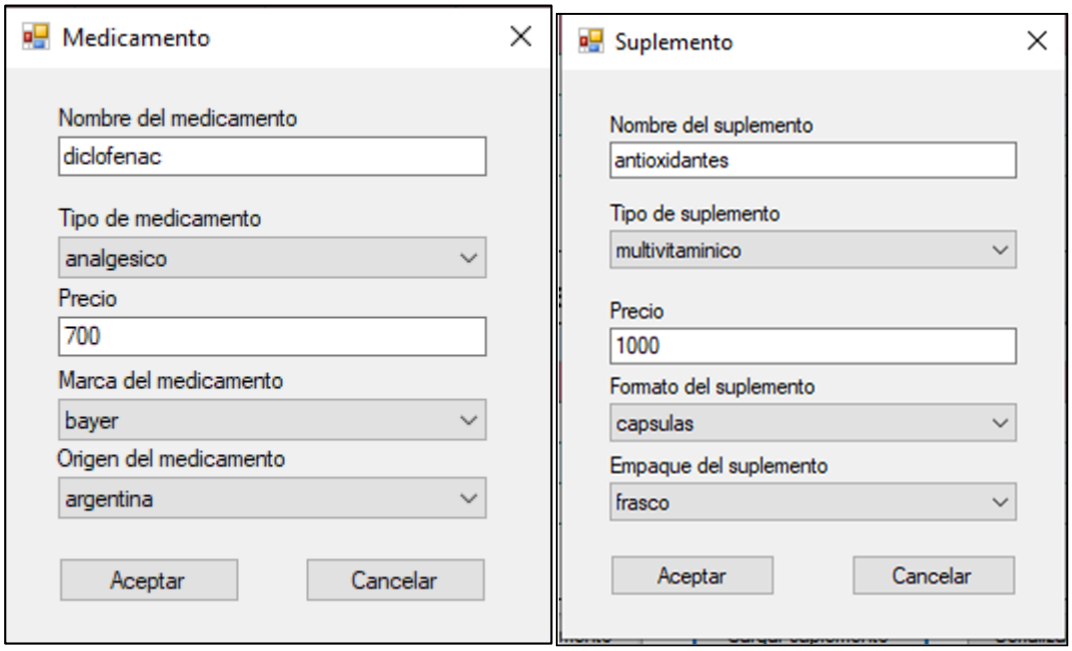


Figura 1: Formulario principal

Figura 2: Plataforma de ventas



Figuras 3 y 4: Formularios de carga de Medicamento y Suplemento

Resumen de los temas aplicados de las clases 15 a 25:

Excepciones

Se crearon diversas excepciones para tratar los problemas más frecuentes que podrían suceder a la hora de trabajar con esta aplicación. En este caso, se crearon **ArchivosException**, **PrecioInvalidoException** y **StringInvalidoException,** siendo éstos implementados en diversas clases.

Test Unitarios

Los test unitarios realizados para este programa se encuentran en el proyecto **TestsUnitarios**. En el mismo se prueban el lanzamiento de excepciones y sobrecarga de operadores

Tipos Genéricos

Los tipos genéricos se utilizan en la interfaz **Iarchivo<T>** y la clase instanciable **Venta<T>**, estando esta última restringida para objetos de tipo Producto y derivados.

```
/// <summary>
/// Clase pública y genérica Venta<T>, restringida para Productos o derivados
/// Implementación de Generics (tema requerido por la consigna del TP Nº4)
/// </summary>
/// <typeparam name="T"> de tipo Producto o derivados </typeparam>
10 referencias
public class Venta<T> where T : Producto
{
    #region Atributos

    public List<Producto> listaDeCompras;
    public float precioTotal;

    #endregion
```

Interfaces

La interfaz creada para esta solución fue **IArchivo<T>**, que a su vez también es genérica. Esta interfaz se utiliza en las clases **Xml<T>** y **Txt**

```
namespace Archivos
{
    /// <summary>
    /// Interfaz IArchivo <T> genérica
    /// Implementación de Generics y Archivos (tema requerido por la consigna del TP Nº4)
    /// </summary>
    /// <typeparam name="T"> de tipo T genérico </typeparam>
    2 referencias
    public interface IArchivo<T>
    {
        /// <summary>
        /// Método abstracto Guardar
        /// </summary>
        /// <param name="archivo"> de tipo string </param>
        /// <param name="datos"> de tipo T genérico </param>
        /// <returns> un bool </returns>
        3 referencias
        bool Guardar(string archivo, T datos);

        /// <summary>
        /// Método abstracto Leer
        /// </summary>
        /// <param name="archivo"> de tipo string </param>
        /// <param name="datos"> de tipo out T genérico </param>
        /// <returns> un bool </returns>
        3 referencias
        bool Leer(string archivo, out T datos);
    }
}
```

Soria Federico Martín. 2A

Archivos y Serialización

Los archivos generados en esta solución son de tipo xml y Txt. Los xml se crean a partir de los datos del Data Table y el txt se genera cuando se efectúa una venta, recolectando la información de la misma (fecha, hora, productos, precio total). El mismo se guarda en la ruta por defecto:
(Soria.Federico.2A.TP4\Entidades de WinForms\bin\Debug)

Por otro lado, cuando se aprietan los botones **Serializar stock XML** y **Deserializar Stock**, se ejecutan dichas acciones. En el caso de **Serializar stock XML**, la función que tiene es la de serializar por separado los stocks presentes en los DataGrid de Medicamentos y Suplementos, generando así **dos archivos .xml**. **Deserializar Stock** despliega dos ventanas con la información contenida en los archivos XML en caso de existir. Caso contrario muestra una ventana informativa con el mensaje “No se pudo leer el archivo”.

Nota: Existe la posibilidad de que el proyecto venga con archivos XML de pruebas realizadas previamente. En tal caso, al serializar de nuevo, se sobreescriben los mismos.

Estos archivos xml se guardan en la ruta por defecto:

(Soria.Federico.2A.TP4\Entidades de WinForms\bin\Debug)

SQL y Bases de Datos

El núcleo del funcionamiento del programa reside en la conexión a una base de datos y la ejecución de comandos de SQL. El nombre de la base de datos es **storage_pharmacy**, que a su vez contiene dos tablas: **storage_meds** para los medicamentos y **storage_supplements,** donde se guardan los datos de los respectivos productos.

El funcionamiento del mismo comienza al iniciar el programa, trayendo automáticamente la información de la base de datos y desplegándola en las grillas correspondientes.

El alta, baja y modificación de cada elemento está distribuida en los botones del **FormPrincipal**, cuyos cambios no se reflejan hasta que se sincroniza con la base de datos o se entra al form de ventas.

Para configurar los comandos Select, Insert, Update y Delete, se utilizan objetos de tipo SqlDataAdapter.

Nota: En la carpeta de la solución se dejan tanto los archivos mdf y ldf como el script para crear la base de datos utilizada en este proyecto.

Hilos

Se utiliza un hilo de nombre **hiloFecha** que contiene un método llamado **MostrarFecha()** dentro de la clase **FrmPrincipal**, cuyo funcionamiento es mostrar la fecha y hora actualizada cada 1 segundo, independientemente de las acciones que se realicen por separado durante el uso del programa.

Delegados y Eventos

Al iniciar el programa, se ejecuta un evento de nombre **Bienvenida** de tipo **DelegadoBienvenida()** que tiene cargado un método que presenta un mensaje de bienvenida al usuario.

También se utiliza un evento de nombre **VentaFinalizada** de tipo **DelegadoVenta(),** cuya función es actualizar las grillas cuando se cierra el form de Ventas.

Por último, se utiliza el evento **ContarVentaEvent** que se define en **FrmVender**, que contiene un método privado de nombre **ActualizarContadorVentas()**, que se encarga de actualizar el **labelContador** cada vez que se realiza una venta, mostrando así el número de ventas realizadas en el **FormPrincipal**.

Método de extensión

Se utiliza un método de extensión llamado **ValidarNombreProducto**, que se encarga de validar que el string ingresado como nombre de un producto no contenga números. El método funciona como método de extensión de la clase **Producto** y se utiliza en la propiedad **Nombre**.

```
/// <summary>
/// Clase pública de extensión
/// Implementación de Método de extensión (tema requerido por la consigna del TP Nº4)
/// </summary>
0 referencias
public static class ClaseExtension
{
    /// <summary>
    /// Método de extensión que valida el ingreso de un string
    /// </summary>
    /// <param name="p"> de tipo Producto </param>
    /// <param name="dato"> de tipo string </param>
    /// <returns> un string </returns>
    1 referencia
    public static string ValidarNombreProducto(this Producto p, string dato)
    {
        string strRegex = @"^[a-zA-Z,\s]+$";
        Match validador = Regex.Match(dato, strRegex);
        if (!validador.Success)
        {
            dato = "";
        }
        return dato;
    }
}
```