LABORATORIO 2 (TP 4) – EDGARDO LORENZO (2C)
FABRICA DE PRODUCTOS ELECTRONICOS

De que se trata del programa?
Es una fabrica de componentes electrónicos (Notebooks y Teclados).
En la misma solo se construyen y stockean productos (que se pueden guardar en DB o XML). No se realizan compras ni ventas.
Es un programa sencillo con 5 forms, el principal y 4 secundarios.
Los forms secundarios:

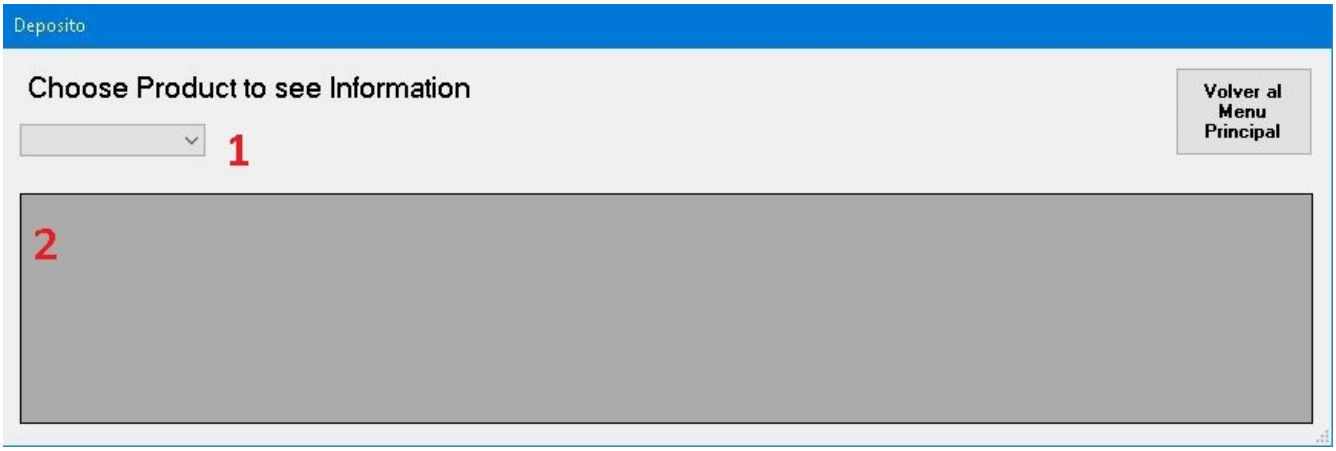
1. **Deposito**: Permite ver los productos creados durante la existencia del programa.
2. **Ver/Cargar Stock**: Permite ver la cantidad de materiales actual y agregar en caso de ser necesario.
3. **Construir Productos**: Permite crear productos de tipo Notebook o Teclado (muestra una imagen previa de cada producto a crear). Contiene un boton para ir directo a Stock en caso de ser necesario re-stockear materiales.
4. **Reportes**: Permite Guardar Productos en BD/XML y Traer desde una BD/XML cada uno con su correspondiente richtextbox. Es en este form donde se implementa el uso de delegados-eventos-hilos.

Breve introducción visual de los Forms:

1. MainFrm

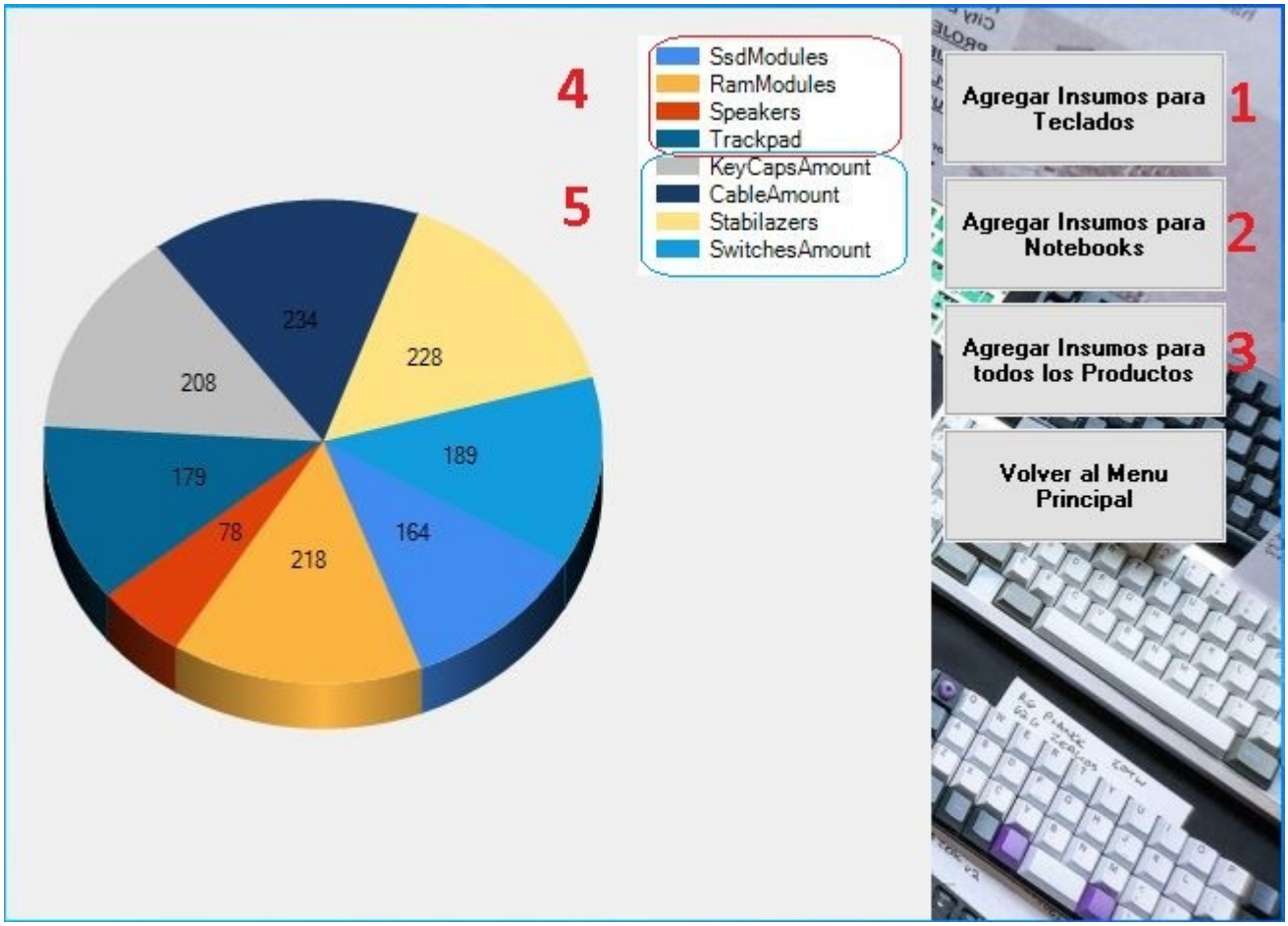
En el Form principal vemos los botones que nos llevan a los Forms secundarios y una label con el titulo de la Fabrica (**AQUÍ se aplica EVENTO + MANEJADOR**).

2. WarehouseFrm (Deposito)



- 1) Un ComboBox que nos permite seleccionar tipo de Productos
- 2) Si existieran Productos en memoria (creados desde el botón Construir Productos) en este DataGrid se desplegarían, cada uno con sus características.

3. StockFrm (Stock)



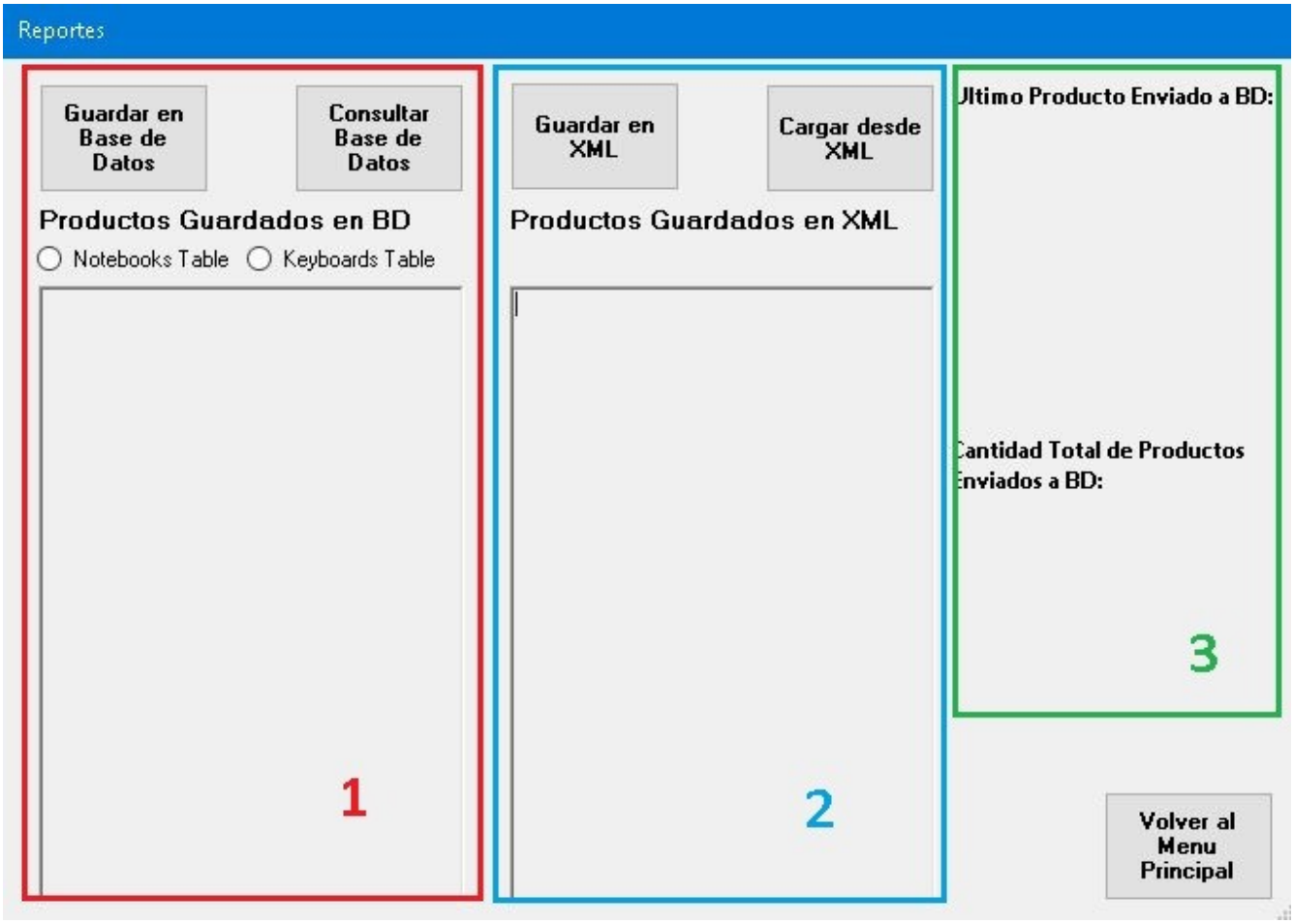
- 1) Un botón solo para cargar insumos de materiales necesarios para construir Teclados

- 2) Un botón solo para cargar insumos de materiales necesarios para construir Notebooks
- 3) Un botón para cargar insumos de materiales necesarios para construir Teclados Y Notebooks
- 4) Los nombres de los materiales necesarios para construir Notebooks
- 5) Los nombres de los materiales necesarios para construir Teclados

4. BuildFrm (Construir Productos)

- 1) Un botón para confirmar la construcción de un Productos
- 2) El botón anterior puede devolver un mensaje de stock insuficiente, en ese caso desde este botón Stock, nos vamos al FrmStock y podremos cargar mas insumos.
- 3) Un GroupBox que en su interior tiene RadioButtons para poder elegir Productos de tipo Teclado
- 4) Otra sección del mismo GroupBox pero que tiene Productos de tipo Notebook
- 5) Un PictureBox que va desplegando las imágenes de los Productos seleccionados cuando hacemos click en algún RadioButton.

5. ReportsFrm (Reportes)



- 1) Recuadro Rojo
    a) Botón GUARDAR EN BASE DE DATOS: Aquí (en caso de existir Productos creados durante la ejecución del Programa) se guardan (de acuerdo a su tipo) los productos en las tablas Notebooks y Keyboards. Una vez enviado los Productos a la base, se eliminan de la lista en memoria del Programa. **(AQUÍ se implementa DELEGADOS-EVENTOS-HILOS). Ver pto *Implementación del Tema 23-24* para mas info.**
    a) Botón CONSULTAR BASE DE DATOS: Aquí se permite consultar la base Products para obtener información de los Productos guardados en la misma. Para hacerlo SI O SI hay que seleccionar alguno de los RadioButtons Notebooks Table o Keyboards Table.
    a) Dicha información seria mostrada por el RichTextBox de abajo.

- 2) Recuadro Azul
    a) Botón GUARDAR EN XML: Aquí (en caso de existir Productos creados durante la ejecución del Programa) se permite serializarlos en un archivo XML.
    Si la intención es enviar a un archivo, hacerlo antes del paso anterior (Recordá que Guardar en Base, una vez efectuado, los elimina de memoria)
    a) Botón CARGAR DESDE XML: Aquí se pueden cargar archivos XML, deserializarlos y mostrar la información de los Productos en el RichTextBox de abajo.

- 3) Recuadro Verde
    a) A medida que vamos guardando Productos en la Base (**Recuadro Rojo**) se van actualizando 2 labels (lblModel y lblCount) en un hilo secundario.
    b) LblModel: Muestra Modelo, Numero de Serie y Hora en que el Producto se tiro a la base.
    c) LblCount: Muestra la cantidad TOTAL de Productos que se tiraron a la Base, durante toda la existencia del Programa. Podemos Salir de este Form, volver a entrar y el contador sigue funcionando.

Implementacion del Tema 15 (EXCEPCIONES)

Se crea la Clase OutOfStockException.cs para controlar la cantidad de materiales disponibles para poder fabricar un producto ya sea de tipo Keyboard o Notebook.
La excepción se lanza, en el caso de que aplique, desde la clase Factory.cs desde el método StockAvailable, la atrapa dentro de la misma clase por el set de la propiedad Create (llamadora de StockAvailable) y la vuelve a lanzar para ser finalmente atrapada y mostrada por la clase BuildFrm.cs dentro de su manejador btnBuild_Click (Donde se utiliza la propiedad Factory.Create = new TIPO PRODUCTO).

Implementacion del Tema 16 (TEST UNITARIOS)

Se crea el proyecto UnitTestEntidades, dentro del mismo la clase Product_Test.cs en la cual se evalúa con un método por TRUE y otro por FALSE la comparación entre 2 productos en base a su tipo, numero de serie y modelo.

Implementacion del Tema 17 (GENERICS)

1. Dentro de la clase Materials.cs se implementa el método LoadMaterialsNeeded parametrizado como <T> que recibe un objeto T. Dicho método se utiliza para cargar un diccionario <string,int> de materiales por cada tipo de objeto, sin necesidad de tener una lista hardcodeada de entrada.
   Funciona de la siguiente manera:

- Cada objeto también tiene un atributo de tipo diccionario de materiales
- Se llama al método y se le pasa como parámetro el objeto (Keyboard, Notebook, el que fuese)
- Se recorre las keys de materiales del objeto y si los materiales NO están dentro del diccionario de materiales de la clase Materials, se agrega el nombre del material y un valor random.

2. Dentro de la clase Factory.cs se implementa el método StockAvailable parametrizado como <T> que recibe un objeto T. Dicho método se utiliza para corroborar que en el diccionario de materiales haya stock suficiente para poder crear lo que cuesta crear un elemento de tipo T. Al ser Generics puede ser una instancia de MechanicalKeyboard, Thinkpad o si en el futuro se decide agregar otro tipo de producto mas, también.
   Funciona de la siguiente manera:

- Como mencionamos en el pto. 1 c/objeto tiene un atributo de tipo diccionario de materiales
- Se recorren las claves del diccionario de materiales del objeto
- Se recorren las claves del diccionario de materiales de Materials (desde Factory, ya que esta tiene un atributo de tipo Materials)
- Se comparan los values de dichas claves
- Si el value del stock de Materials es >= al value del stock del objeto se setea bool en true. Se realiza un break y se procede a evaluar otro material.
- Caso contrario no existen suficientes materiales para crear un objeto de dicho tipo y se procede a lanzar una OutOfStockException

3. Dentro de la clase Factory.cs se implementa el método SubstractMaterials parametrizado como <T> que recibe un objeto T. Dicho método se utiliza para restar de la lista de materiales (en poder de la clase Factory) los materiales que consumen crear una instancia de un objeto.
   Al ser generico se le pueden pasar objetos tanto de tipo MechanicalKeyboards como Thinkpad sin distincion.
   Funciona de la siguiente manera:

- Si se realiza la llamda al método significa que StockAvailable devolvio true y existen suficientes materiales para crear una instancia del objeto.
- Se crea una copia local del diccionario en poder de la Factory y se le asigna el diccionario de Materials
- Se recorren las claves del diccionario de materiales del objeto
- Se recorren las claves del diccionario de materiales de Materials
- Se comparan las claves
- Si hay coincidencia se resta al diccionario Original (no la copia que esta siendo recorrida) la clave del diccionario del objeto.

4. Por ultimo dentro de la clase Factory.cs se implementa el método LoadMoreStock parametrizado como <T> que recibe un objeto T. Dicho método se utiliza para cargar mas materiales al diccionario de materiales.
   Funciona de la siguiente manera:

- Se crea una copia local del diccionario en poder de la Factory y se le asigna el diccionario de Materials
- Se recorren las claves del diccionario de materiales del objeto
- Se recorren las claves del diccionario de materiales de Materials
- Se comparan las claves
- Si hay coincidencia y el objeto es de tipo Keyboard se agrega stock basado en un random
- Si hay coincidencia y el objeto es de tipo Notebook se agrega stock basado en otro random
- El método se implemento de manera Generica para poder agregar STOCKS por separado, dependiendo del tipo de objeto que llegue por parámetro.

Implementacion del Tema 18 (Interfaces)

Se crea la interfaz Ifile la cual se parametriza como Generic.

- Crea la firma para un método SaveXML, el cual recibe un string (path) y un T (objeto generico)
- Crea la firma para un método OpenXML, el cual devuelve un T (objeto generico), recibe un string(path) y recibe otro T (objecto generico).

La misma se implementa en la clase Serializator.cs (pto. 19)

Implementacion del Tema 19 (Archivos)

Se crea la clase Serializator.cs a la cual se parametriza como Generic e implementa una interfaz y servirá para guardar objetos en un archivo XML.
Esta clase contiene un método Save que devuelve un bool, recibe por parametro un string correspondiente al path del archivo y un objeto T (Generic), permitiendo poder recibir tanto List<Product>, Keyboard, Notebook etc. Al ser genérica, el tipo de objeto se resuelve en tiempo de ejecucion.
Dicho método hace uso del XmlTextWriter que obtiene un puntero al archivo, si el mismo no existe lo crea. Y del XmlSerializer encargado de serializar el objeto en el archivo.

Esta clase contiene un método OpenXML que devuelve un objeto de tipo T (objeto generic), recibe por parámetro un string correspondiente al path del archivo y un objeto T (Generic), permitiendo poder leer tanto List<Product>, Keyboard, Notebook etc. Al ser genérica, el tipo de objeto se resuelve en tiempo de ejecución.
Dicho método hace uso del XmlTextReader que obtiene un puntero al archivo, si el mismo no existe lo crea. Y del XmlSerializer encargado de deserializar el objeto a traves de su método Deserialize.

Implementacion del Tema 21 (SQL)

Se crea la clase SQL a la cual se parametriza como Generic para poder utilizarla en su mayoría con cualquier tipo de Objeto.

Para la ConnectionString se agrega ruta desde *EntidadesCore → Properties → Settings:*

Data Source=.;Initial Catalog=Products;Integrated Security=True

(MODIFICAR SOURCE, punto por localhost, EN CASO DE ERROR)

Se pasa siguiente param al SqlConnection: Properties.Settings.Default.StringConnection

1. Dentro de la clase se implementa el método QueryDB, el cual recibe un parámetro de tipo string. Corresponde a la query que se quiere hacer sobre dicha Base.

   Funciona de la siguiente manera:

- Se crea el objecto de tipo SqlConnection para conectarnos a la Base (se le pasa ruta por Property)
- Se crea el objeto de tipo SqlCommand para ejecutar la query
- Se crea el objeto de tipo SqlDataReader para traernos los objetos de la Base
- Se crean variables temporales para objetos tanto de tipo Notebooks como de tipo Teclados (El método sirve para ambos tipos de objetos)
- Se realiza la conexion a la base y en caso de ser exitosa se le asigna al objeto DataReader el resultado de la query
- Se buclea el objeto DataReader reconstruyendo cada objeto de tipo Notebook o Teclado según el nombre de las columnas.

2. Dentro de la clase se implementa el método Insert, el cual recibe 3 parámetros. El primero corresponde a la NonQuery que se quiere hacer sobre dicha Base. El segundo parámetro es una lista de strings que contiene los nombres de los atributos del objeto a ingresar en la Base. Y por ultimo eltercero es un parámetro de tipo T (generic objetc) con toda la info del objeto a guardar.
   Funciona de la siguiente manera:

- Se crea el objecto de tipo SqlConnection para conectarnos a la Base
- Se crea el objeto de tipo SqlCommand para ejecutar el insert
- Se realiza la conexión a la Base
- Se recorre la lista de propiedades del objeto (Con la lista de propiedades)
- Se recorren las propiedades del objeto con un GetType().GetProperties()
- Si hay coincidencia a través de un Parameters.WithValue se utiliza el nombre de la propiedad como primer argumento y el nombre de la propiedad punto GetValue como segundo argumento para ir creando los parámetros de forma dinámica.
- Por ultimo se ejecuta el ExecuteNonQuery().

3. Dentro de la clase se implementa el método BuildInsertQuery, el cual recibe 2 parametros. El primero corresponde a la lista de propiedades y el segundo corresponde a la tabla donde se insertara el objeto.
   Funciona de la siguiente manera:

- Se realiza un foreach por cada propiedad de la lista y se va creando un string con la NonQuery que eventualmente se le pasara al método Insert (ver pto 2).
- En caso de que la lista de propiedades o la string correspondiente al nombre de la tabla sean NULL se lanzara excepcion de tipo InvalidQueryException

Implementacion del Tema 22(BASE DE DATOS)

Utilizar siguientes 3 scripts para crear Base Products, Tablas Notebooks y Tabla Keyboards, necesarias para testear los temas del pto anterior (SQL).

Los mismos se dejan en:

- **TP4\DataBase Scripts**
  - **Keyboards_Table.sql**
  - **Notebooks_Table.sql**
  - **Products.DataBase.sql**

Ademas se deja una Base ya funcional para poder testear el programa (ya tiene tablas/registros cargados):

- **TP4\DataBase BackUp**
  - **Products.bak**

Por su poca extensión los scripts de tablas también se agregan a continuación
(PREFERENTEMENTE utilizar los scripts guardados en TP4\DataBase Scripts\):

1. **Tabla Notebooks:**

```
USE [Products]
GO
/****** Object:  Table [dbo].[Notebooks]    Script Date: 12/7/2021 00:18:49 ******/
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[Notebooks]')
AND type in (N'U'))
DROP TABLE [dbo].[Notebooks]
GO
/****** Object:  Table [dbo].[Notebooks]    Script Date: 12/7/2021 00:18:49 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Notebooks](
	[Serial_Number] [numeric](18, 0) NOT NULL,
	[Model] [varchar](50) NULL,
	[Price] [float] NULL,
	[ScreenSize] [numeric](18, 0) NULL,
	[SsdModules] [numeric](18, 0) NULL,
	[RamModules] [numeric](18, 0) NULL,
	[Battery] [numeric](18, 0) NULL,
	[Speakers] [numeric](18, 0) NULL,
	[Trackpad] [numeric](18, 0) NULL,
	[HasDockingStation] [numeric](18, 0) NULL,
 CONSTRAINT [PK_Notebooks] PRIMARY KEY CLUSTERED
(
	[Serial_Number] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

2. **Tabla Keyboards:**

```
USE [Products]
GO
/****** Object:  Table [dbo].[Keyboards]    Script Date: 12/7/2021 00:19:15 ******/
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[Keyboards]')
AND type in (N'U'))
DROP TABLE [dbo].[Keyboards]
GO
/****** Object:  Table [dbo].[Keyboards]    Script Date: 12/7/2021 00:19:15 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Keyboards](
	[Serial_Number] [numeric](18, 0) NOT NULL,
	[Model] [varchar](50) NULL,
	[Price] [float] NULL,
	[KeyboardSize] [numeric](18, 0) NULL,
	[CableAmount] [numeric](18, 0) NULL,
	[KeyCapsAmount] [numeric](18, 0) NULL,
	[SwitchesAmount] [numeric](18, 0) NULL,
	[SwtichColor] [varchar](50) NULL,
	[SwitchType] [varchar](50) NULL,
	[Stabilazers] [numeric](18, 0) NULL,
	[HasBluetooth] [numeric](18, 0) NULL,
 CONSTRAINT [PK_Keyboards] PRIMARY KEY CLUSTERED
(
	[Serial_Number] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Implementación del Tema 23-24(HILOS, EVENTOS Y DELEGADOS)

Delegado, Atributos, Eventos y Manejador:

- Dentro del namespace FactoryForm se crea un tipo de Delegado DataBaseDelegate generics que recibe como parametro un T (objeto generic).
- En la clase ReportsFrm.cs (dicha clase pertenece al namespace anterior) se crea una variable de tipo Thread, una variable productsSentToDB_Count static de tipo int, una variable timeStampProductWasSentoToDB static de tipo DateTime y un objeto static de tipo Product.
- Ademas se crean 2 eventos de tipo DataBaseDelegate, uno llamado DataBaseModified y otro llamado SetInitialLabel.

- En el Constructor del form se le suscribe a ambos eventos el Manejador UpdateView.
- La primer label (lblModel) muestra el Modelo del ultimo objeto guardado en le Base de Datos.
- La segunda label (lblCount) muestra la cantidad total de objetos guardados en la Base de Datos de la sesión actual (durante todo el tiempo que viva el programa).

Como funciona?:

- Durante el ReportsFrm_Load() se corrobora que la variable static de tipo Product (finalProductSentToDB) sea distinta de NULL, caso afirmativo se genera/invoca al evento SetInitialLabel y se pasa finalProductSentToDB como argumento. **Linea 44.**
- finalProductSentToDB llega como OBJECT al Menejador UpdateView y se vuelve a pasar como parámetro cuando se inicia el Thread. Este Thread llama al ShowStatistics.
- ShowStatistics corrobora que el Object sea de tipo Product, lo castea a Product y se lo pasa al método UpdateLabel quien a través del InvokeRequired lo utilizara para obtener Modelo y Numero de Serie para actualizar lblModel

- Caso contrario, donde finalProductSentToDB es NULL, el evento nunca se genera/invoca dentro del ReportsFrm_Load().

- Suponiendo que tenemos una lista de Productos (creados dsde BuildFrm) y queremos guardarlos en la Base de Datos, sucede lo siguiente:
- Una vez apretado el boton Guardar en Base de Datos, se dispara su Manejador btnGuardarBase_Click, una vez guardados los objetos en la base, se genera/invoca el evento DataBaseModified y se le pasa como arguemento una copia de la lista. **Linea 170.**
- La lista de Productos llega como OBJECT al Menejador UpdateView y se vuelve a pasar como parámetro cuando se inicia el Thread. Este Thread llama al ShowStatistics.
- ShowStatistics corrobora que el Object sea de tipo List<Product>, lo castea a List<Product> y recorre dicha lista con un foreach, por cada item se llama al método UpdateLabel quien atravez del InvokeRequired lo utilizara para obtener Modelo y Numero de Serie para actualizar lblModel. Una vez finalizado UpdateLabel, se asigna el item a finalProductSentToDB, se aumenta el productsSentToDB_Count en 1, se elimina dicho Producto de la Lista de la Fabrica (no la copia de la lista que sea pasa a ShowStatistics, esta ultima es temporal y solo sirve para poder recorrerla) y se duerme el Thread por 1,5s.

ACLARACION:

- DELEGADOS-EVENTOS-HILOS (la combinación de los 3) solo funciona desde ***btnGuardarEnBase_Click***

- ***btnConsultarBase_Click*** solo muestra productos de la base (en caso de que haya) por medio del rich text box **rtbFromDB.** No guarda objetos en memoria durante el transcurso del programa.

- Por ultimo se crea un Manjeador llamado Parpadear, para el evento Tick de un objeto de tipo Timer. Dicho evento cambiar el ForeColor de lblTitle (según un intervalo de tiempo) para dar el efecto de cartel luminoso en el MainFrm.

Implementacion del Tema 25(métodoS DE EXTENSION)

Se crea la clase ListExtensionMethod (static) la cual implementa el método de extensión ContainsType generic para cualquier clase de tipo List<T>
Dicho método recibe un parámetro de tipo string.
El mismo permite corroborar dentro de una lista si existe una clase del tipo de la string recibida por parámetro.
Por dentro recorre la lista y compara el item.GetType().Name con el string recibido por parámetro.
Si hay coincidencia quiere decir que la clase que se recibe por param existe dentro de la lista.
Este método se utiliza para validar si dentro de una lista de Productos existen objetos de tipo Notebook, Teclado o ambos, para luego (en caso afirmativo) crear una lista de propiedades y una NonQuery para insertar dichos objetos en la Base de Datos (método btnSaveToDB_Click).