

LABORATORIO 2 (TP 4) – EDGARDO LORENZO (2C)

FABRICA DE PRODUCTOS ELECTRONICOS

De que se trata del programa?

Es una fabrica de componentes electronicos (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. **Warehouse:** Permite ver los productos creados durante la existencia del programa que pueden ser serializados o guardados en DB.
2. **Stock:** Permite ver la cantidad de materiales actual y agregar en caso de ser necesario.
3. **Build:** 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. **Reports:** Permite Guardar lista de 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.

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 excepcion se lanza, en el caso de que aplique, desde la clase Factory.cs desde el metodo 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 evalua con un metodo por TRUE y otro por FALSE la comparacion entre 2 productos en base a su tipo, numero de serie y modelo.

Implementacion del Tema 17 (GENERICOS)

1. **Dentro de la clase Materials.cs se implementa el metodo LoadMaterialsNeeded** parametrizado como <T> que recibe un objeto T. Dicho metodo 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 tambien tiene un atributo de tipo diccionario de materiales
- Se llama al metodo y se le pasa como parametro el objeto (Keyboard, Notebook, el que fuese)
- Se recorre las keys de materiales del objeto y si los materiales NO estan 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 metodo StockAvailable** parametrizado como <T> que recibe un objeto T. Dicho metodo 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, tambien.

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 valores de dichas claves
- Si el value del stock de Materials es \geq 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 metodo SubtractMaterials** parametrizado como <T> que recibe un objeto T. Dicho metodo 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 llamada al metodo 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 metodo LoadMoreStock** parametrizado como <T> que recibe un objeto T. Dicho metodo 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 metodo se implemento de manera Generica para poder agregar STOCKS por separado, dependiendo del tipo de objeto que llegue por parametro.

Implementacion del Tema 18 (Interfaces)

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

- Crea la firma para un metodo SaveXML, el cual recibe un string (path) y un T (objeto generico)
- Crea la firma para un metodo OpenXML, el cual devuelve un T (objeto generico), recibe un string(path) y recibe otro T (objeto 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 servira para guardar objetos en un archivo XML.

Esta clase contiene un metodo 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 generica, el tipo de objeto se resuelve en tiempo de ejecucion.

Dicho metodo 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 metodo OpenXML que devuelve un objeto de tipo T (objeto generico), recibe por parametro un string correspondiente al path del archivo y un objeto T (Generic), permitiendo poder leer tanto List<Product>, Keyboard, Notebook etc. Al ser generica, el tipo de objeto se resuelve en tiempo de ejecucion.

Dicho metodo 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 metodo Deserialize.

Implementacion del Tema 21 (SQL)

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

1. Dentro de la clase se implementa el metodo QueryDB, el cual recibe dos strings. El primero corresponde a la connectionString (la ruta de la Base de Datos) y el segundo corresponde a la query que se quiere hacer sobre dicha Base.

Funciona de la siguiente manera:

- Se crea el objeto de tipo SqlConnection para conectarnos a la Base
- Se le asigna al objeto la ruta de la Base (conexion string)
- 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 metodo 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 segun el nombre de las columnas.

2. Dentro de la clase se implementa el metodo Insert, el cual recibe 4 parametros. El primero corresponde a la conectionString (la ruta de la Base de Datos) y el segundo corresponde a la NonQuery que se quiere hacer sobre dicha Base. El 3 parametro es una lista de strings que contiene los nombres de los atributos del objeto a ingresar en la Base. Y por ultimo el 4 es un parametro de tipo T (generic objetc) con toda la info del objeto a guardar.
Funciona de la siguiente manera:

- Se crea el objeto de tipo SqlConnection para conectarnos a la Base
- Se le asigna al objeto la ruta de la Base (conexion string)
- Se crea el objeto de tipo SqlCommand para ejecutar el insert
- Se realiza la conexion 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 traves 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 parametros de forma dinamica.
- Por ultimo se ejecuta el ExecuteNonQuery().

3. Dentro de la clase se implementa el metodo 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 metodo 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 InvalidOperationException

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 extension los scripts de tablas tambien se agregan a continuacion:

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,
    [SwitchColor] [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

```

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

Como se implemento dicho tema:

- Dentro del namespace FactoryForm se crea un tipo de Delegado DataBaseDelegate generics que recibe como parametro un T (objeto generic).
- Dentro de la clase ReportsFrm.cs (dicha clase pertenece al namespace anterior) se crea una variable de tipo Thread, un contador static de tipo int y un objeto static de tipo Product y un evento de tipo DataBaseDelegate llamado DataBaseModified.
- Dentro del constructor del form se le suscribe al evento el metodo UpdateView.
- En la misma clase se crea el metodo ShowStatistics que recibe un objetc. Dicho metodo castea a Product el object y a traves del InvokeRequired modifica 2 labels del formulario.
- La primer label muestra el Modelo del ultimo objeto guardado en le Base de Datos.
- La segunda label muestra la cantidad total de objetos guardados en la Base de Datos de la sesion actual (durante todo el tiempo que viva el programa).
- El metodo UpdateView encargado de instanciar el hilo con el ParameterizedThreadStart pasandole como parametro el metodo DataBaseModified. Ademas inicia el hilo.
- El evento invoca a su manejador en la linea 68, dentro del metodo btnSaveToDB_Click, de la clase luego de que se guardo un objeto, con exito, en la Base de Datos.
- Se corrobora que dicho manejador se ejecuta en un Thread secundario al del form ya que antes de cerrar el MessageBox.Show del btnSaveToDB_Click las labels ya fueron actualizadas.

Implementacion del Tema 25(METODOS DE EXTENSION)

Se crea la clase ListExtensionMethod (static) la cual implementa el metodo de extension ContainsType generic para cualquier clase de tipo List<T>
Dicho metodo recibe un parametro de tipo string.

El mismo permite corroborar dentro de una lista si existe una clase del tipo de la string recibida por parametro.

Por dentro recorre la lista y compara el `item.GetType().Name` con el string recibido por parametro.

Si hay coincidencia quiere decir que la clase que se recibe por param existe dentro de la lista.

Este metodo 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 (metodo `btnSaveToDB_Click`).