

ROCHA FRANCISCO
TP 4

FABRICA DE PRODUCTOS INFORMATICOS

Como se ve el form principal completo (por si la resolución del monitor es chica y no se ve completo)



Mi trabajo practico se basa en la fabricación de Computadoras y Celulares mediante la compatibilidad de ciertos componentes, tiene un criterio de ensamblado en el cual no se pueden usar componentes que no sean compatibles con el producto seleccionado, como, por ejemplo:

En consola:

```
FabricaPC PC1 = new FabricaPC(MarcaCPU.Intel, Procesador.i3_7100, Motherboard.GIGABYTE_GA_B250M_D3H,
```

Esto es un pc con productos compatibles, la marca de procesador es Intel, el procesador es Intel socket 1151 y la mother también es Intel y socket 1151, por lo cual se podrá fabricar, en cambio si ponemos componentes incompatibles como los siguientes no se podrá

```
FabricaPC PC3 = new FabricaPC(MarcaCPU.Mediatek, Procesador.i9_10900, Motherboard.GIGABYTE_GA_B250M_D3H,
```

Este pc cuenta con un procesador marca mediatek de celular, un procesador Intel socket 1200 y una mother socket 1151, por lo cual no se fabricará

```
Ya existe un producto con este codigo de barras...  
La mother ingresada no corresponde a el procesador seleccionado: i9_10900  
La marca del procesador no sirve para el esamblado de PC...  
Ya existe un producto con este codigo de barras...
```

Además de la compatibilidad es importante que 2 productos fabricados no cuenten con el mismo UPC (Código de Producto Universal), por lo cual, aunque un producto tenga los componentes correctos si el UPC se repite no podrá ser fabricado

Mi proyecto cuenta con una clase base llamada Producto, la cual contendrá los atributos que se pueden utilizar tanto en una PC como en un Celular, el listado de atributos es el siguiente:

UPC, GPU, RAM, Marca del procesador, Sistema operativo, Almacenamiento

Francisco Rocha TP-4 2ºA

De esta clase heredan las clases FabricaPC y FabricaCelular, las cuales implementan métodos que tiene la clase producto, estas clases cuentan con los siguientes atributos

FabricaPC: Lector de CD, Gabinete, Fuente, Motherboard y CPU.

FabricaCelular: Jack de auricular, Huella dactilar, Camara, Batería, Carcasa, Pulgadas y Resolución.

Trate de darle un enfoque “Realista” a los componentes y que tengan cierto sentido a la hora de fabricar los productos.

Temas vistos entre las clases 15 y 19:

Excepciones:

Utilice excepciones en todo el proyecto para que el este no rompa, al trabajar con prácticamente solo enumerados es muy raro que el programa rompa, pero por ejemplo en el caso del UPC que es un entero me puede llegar a romper el programa si el usuario ingresa de manera forzosa un entero, para esto use la excepción `OverflowException` y `FormatException`, en caso de que llegue a ocurrir alguna de estas excepciones hice que el código de barras se asigne de manera aleatoria así el programa puede seguir funcionando con normalidad, informándole al usuario de lo sucedido.

```
try
{
    GRAFICOS = Gpu;
    MARCA_CPU = Marca;
    MEMORIA_RAM = memoriaRam;
    ALMACENAMIENTO = almacenamiento;
    SISTEMA_OPERATIVO = sistemaOperativo;
    CODIGO_DE_BARRAS =CodigoDeBarras;
}
catch (System.OverflowException ex)
{
    Console.WriteLine(ex.Message + "Se asigno un numero random como codigo de barras");
    Archivos<Producto>.LogErrores(ex.ToString());
    CODIGO_DE_BARRAS = Random();
}
catch (System.FormatException ex)
{
    Console.WriteLine(ex.Message + "Se asigno un numero random como codigo de barras");
    Archivos<Producto>.LogErrores(ex.ToString());
    CODIGO_DE_BARRAS = Random();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message + "Se asigno un numero random como codigo de barras");
    Archivos<Producto>.LogErrores(ex.ToString());
    CODIGO_DE_BARRAS = Random();
}
```

Cree una clase llamada "Excepciones" la cual posee 3 excepciones

"`NoSeEncontroException`" "`ConexionSQLException`"

"`CargarTodosLosDatosException`", por ejemplo, `NoSeEncontroException` lo lanzo cuando no encuentro el UPC en la lista de productos fabricados

```
if (retorno == -1)
{
    throw new NoSeEncontroException();
}
```

Además, cree un método que funcione como log de errores, así cada vez que suceda una excepción el programa guarde en un archivo de texto con la hora a la

que sucedió el error, la finalidad de esto es tener una carpeta con todos los errores que tuvo el programa para poder ser arreglados en un futuro, hice que este log se encuentre en una carpeta aparte en la carpeta de mis documentos así su lectura es más fácil.

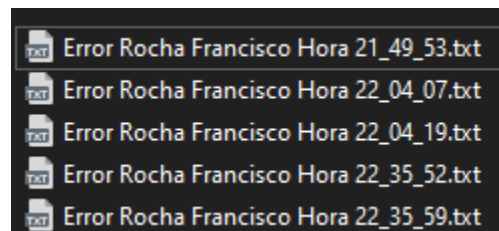
```
public static void LogErrores(string error)
{
    string CarpetaExcepciones = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + @"\LogErrores";

    try
    {
        if (!Directory.Exists(CarpetaExcepciones))
        {
            Directory.CreateDirectory(CarpetaExcepciones);
        }

        CarpetaExcepciones += @"\Error Rocha Francisco Hora " + DateTime.Now.ToString("HH_mm_ss") + ".txt";

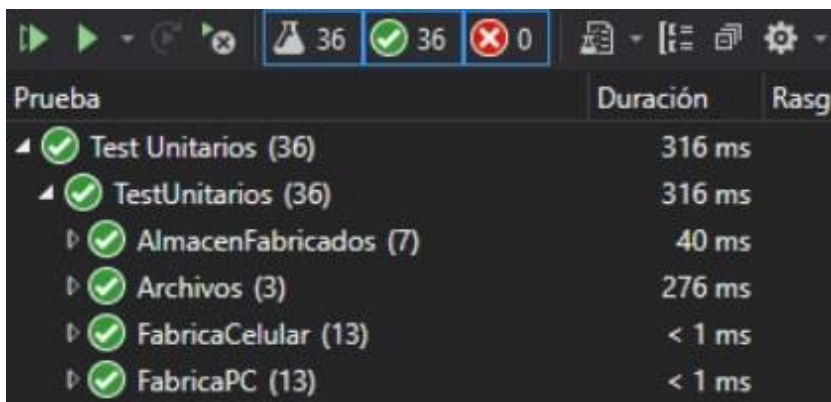
        using (StreamWriter Error = new StreamWriter(CarpetaExcepciones, true))
        {
            Error.WriteLine(error);
        }
    }
    catch (Exception)
    {
    }
}
```

Carpeta y formato de errores:



Test Unitarios:

Utilice test unitarios para probar varias funcionalidades del programa, sobre todo para probar la compatibilidad de los productos y para ver si la carga de los productos fabricados en el almacén funcionaba correctamente.



Tanto en pc como en celular por ejemplo testeé los Equals para probar que el funcionamiento de este sea correcto

```
[TestMethod]
//Pruebo si la sobrecarga del equals del producto es verdadera
0 references
public void ProbarEquals_True()
{
    Entidades.FabricaPC PC1 = new Entidades.FabricaPC(MarcaCPU.I
    Entidades.FabricaPC PC3 = new Entidades.FabricaPC(MarcaCPU.I

    Assert.IsTrue(PC1.Equals(PC3));
}

[TestMethod]
//Pruebo si la sobrecarga del equals del producto es falsa
0 references
public void ProbarEquals_False()
{
    Entidades.FabricaPC PC2 = new Entidades.FabricaPC(MarcaCPU.I
    Entidades.FabricaPC PC3 = new Entidades.FabricaPC(MarcaCPU.I

    Assert.IsFalse(PC2.Equals(PC3));
}
```

Equals devuelve true siempre que dos productos tengan el mismo código de barras o si tienen componentes incompatibles. En el almacén probé por ejemplo que el funcionamiento del operador + o que la cantidad de productos fabricados funcioné correctamente

```
[TestMethod]
//Pruebo si la cantidad de elementos en el almacen
0 references
public void ProbarCantidadGenerica_True()
{
    AlmacenProdutosFabricados<Producto> Almacen;
    Almacen = "Test Method Francisco Rocha";

    Entidades.FabricaPC PC1 = new Entidades.FabricaPC(MarcaCPU.I
    Entidades.FabricaCelular Celular1 = new Entida

    Almacen += PC1;
    Almacen += Celular1;

    Assert.IsTrue(2 == Almacen.CantidadGenerica);
}
```

```
[TestMethod]
//Pruebo si funciona correctamente la sobrecarga
0 references
public void ProbarSobrecargaOperadorMas_True()
{
    AlmacenProdutosFabricados<Producto> Almacen;
    Almacen = "Test Method Francisco Rocha";

    Entidades.FabricaPC PC1 = new Entidades.Fabri

    Almacen += PC1;

    int indice = Almacen | PC1;

    Assert.IsTrue(indice != -1);
}
```

En los archivos testeé que el guardado y la carga de los datos funcione correctamente.

```
[TestMethod]
//Pruebo deserializar el almacen de productos fabricados
0 references
public void ProbarCargarListadoProductosFabricados()
{
    AlmacenProdutosFabricados<Producto> Almacen;
    Almacen = "TestMethod";

    AlmacenProdutosFabricados<Producto> Almacen2;
    Almacen2 = "TestMethod";

    Almacen.Directorío = AppDomain.CurrentDomain.BaseDirectory + "TestMethod.xml";
    Almacen2.Directorío = AppDomain.CurrentDomain.BaseDirectory + "TestMethod.xml";

    Entidades.FabricaPC PC1 = new Entidades.FabricaPC(MarcaCPU.Intel, Procesador.i3);
    Almacen += PC1;

    Archivos<Producto>.GuardarFabrica(Almacen);

    Almacen2 = Archivos<Producto>.CargarFabrica(Almacen2);

    Assert.IsTrue(Almacen.CantidadGenerica == Almacen2.CantidadGenerica);
}
```

Hice varios test más como se ve en la primera imagen, pero esta es la vista general

Tipos Genéricos:

La clase AlmacenProdutosFabricados la hice de tipo genérico

```
39 references
public class AlmacenProdutosFabricados<T> where T : IProductos
{
```

La cual se encarga de agregar mediante la sobrecarga del operador pipe y +, además se encarga listar todos los productos fabricados por el usuario.

Cuenta con dos constructores privados, el primero se encarga de crear la lista genérica y el segundo se encarga de asignarle un nombre al almacén de productos fabricados

```
private AlmacenProdutosFabricados()
{
    try
    {
        Producto = new List<T>();
    }
    catch (Exception ex)
    {
        Archivos<Producto>.LogErrores(ex.ToString());
    }
}
```



```
1 reference
private AlmacenProductosFabricados(string nombre) : this()
{
    try
    {
        this.nombreAlmacen = nombre;
    }
    catch (Exception ex)
    {
        Archivos<Producto>.LogErrores(ex.ToString());
    }
}
#endregion
```

Interfaces:

Cree una interfaz IProducto la cual tiene 2 métodos y 6 propiedades que utilice en la clase producto

```
public interface IProductos
{
    int CODIGO_DE_BARRAS { get; set; }

    GPU GRAFICOS { get; set; }

    RAM MEMORIA_RAM { get; set; }

    MarcaCPU MARCA_CPU { get; set; }

    SistemaOP SISTEMA_OPERATIVO { get; set; }

    Almacenamiento ALMACENAMIENTO { get; set; }

    bool NoMostrar();

    string Informacion();
}
```

```
99+ references
public class Producto : IProductos
{
```

Además, cree una interfaz ISQL la cual no tiene métodos ya que se los agregue mediante un método de extensión que utilice en la clase SQLPC y SQLCelular

```
3 references
public interface ISQL
{
    ...
}
```

```
9 references
public class SQLPC : ISQL
{
```

```
9 references
public class SQLCelular : ISQL
{
```


En el almacén de productos fabricados establecí que T sea de tipo IProductos

```
[Serializable]
39 references
public class AlmacenProdutosFabricados<T> where T : IProductos
{
```

esto significa que el el almacén de productos fabricados acepta tanto a la clase Producto, FabricaPC y FabricaCelular, estas dos últimas debido a que heredan de producto por lo tanto estas clases también son de tipo IProducto

Archivos y serialización:

Para archivos cree una clase Archivos la cual tiene 3 métodos, el primero se encarga de guardar todas las excepciones en un archivo de texto que se genera dependiendo del horario del error en una carpeta destinada para todos los errores como mostré en el punto de excepciones

```
66 references
public static void LogErrores(string error)
{
    string CarpetaExcepciones = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + @"\LogErrores";

    try
    {
        if (!Directory.Exists(CarpetaExcepciones))
        {
            Directory.CreateDirectory(CarpetaExcepciones);
        }

        CarpetaExcepciones += @"\Error Rocha Francisco Hora " + DateTime.Now.ToString("HH_mm_ss") + ".txt";

        using (StreamWriter Error = new StreamWriter(CarpetaExcepciones, true))
        {
            Error.WriteLine(error);
        }
    }
    catch (Exception)
    {
    }
}
```

Para la serialización utilice 2 métodos de guardado y cargado los cuales utilizan XML

El primero se encarga de guardar los datos del almacen de productos fabricados pasado por parámetro

```
4 references
public static void GuardarFabrica(AlmacenProdutosFabricados<T> Datos)
{
    try
    {
        using (XmlTextWriter auxArchivo = new XmlTextWriter(Datos.Directorio, Encoding.UTF8))
        {
            XmlSerializer auxEscriitor = new XmlSerializer(typeof(AlmacenProdutosFabricados<T>));

            auxEscriitor.Serialize(auxArchivo, Datos);
        }
    }
    catch (Exception ex)
    {
        LogErrores(ex.ToString());
    }
}
```

El segundo se encarga de retornar el almacen pasado por parámetro con todos los datos cargados

```
3 references
public static AlmacenProdutosFabricados<T> CargarFabrica(AlmacenProdutosFabricados<T> Datos)
{
    try
    {
        if (File.Exists(Datos.Directoriot))
        {
            using (XmlTextReader auxArchivoLeer = new XmlTextReader(Datos.Directoriot))
            {
                XmlSerializer auxLector = new XmlSerializer(typeof(AlmacenProdutosFabricados<T>));







                Datos = (AlmacenProdutosFabricados<T>)auxLector.Deserialize(auxArchivoLeer);
            }
        }
    }
    catch (Exception ex)
    {
        LogErrores(ex.ToString());
    }

    return Datos;
}
```

Temas vistos entre las clases 20 y 25:

SQL:

En el Microsoft SQL Server Management Studio cree 3 tablas para guardar los productos fabricados y para guardar el nombre del almacén y la cantidad de productos.

  **dbo.Tabla_Celular**
  **dbo.Tabla_PC**
  **dbo.Table_Almacen**

La tabla llamada **Tabla_Celular** se encarga de guardar los datos de todos los celulares fabricados desde el formulario y posee este formato:

	UPC	Marca	GPU	Ram	SistemaOperat...	Almacenamie...	Camara	Bateria	Material	Pulgadas	Resolucion	Jack	Huella
	279085970	0	0	0	0	0	_16MP	_5000_mAh	Aluminio	_6_1	_2560x1440_Pix...	0	0
	567873500	QualcommSna...	Adreno_615	_8GB	Android_Oreo	_256GB	_16MP	_4000_mAh	Polycarbonato	_6_3	_1440x1080_Pix...	0	1
	826280674	HiSilicon	Adreno_650	_16GB	Android_Nougat	_256GB	_8MP	_2000_mAh	Plastico	_6_1	_2560x1440_Pix...	1	1
	890464318	HiSilicon	Adreno_608	_12GB	Android_Oreo	_24GB	_12MP	_2000_mAh	Plastico	_5_5	_1440x1080_Pix...	1	0

La tabla llamada **Tabla_PC** se encarga de guardar los datos de todas las computadoras fabricadas desde el formulario y posee este formato:

	UPC	Marca	Procesador	Motherboard	GPU	Ram	Fuente	Gabinete	SistemaOperat...	Almacenamie...	LectorCD
	278772068	Intel	i7_7700k	MSI_H270_TOM...	GTX_1080_8GB	XPG_Spectrix_D...	Corsair_HX850	NOX_Hummer_...	Linux_Ubuntu	SSD_Samsung_...	1
	306499562	AMD	AMD_Ryzen_5_...	MSI_MPG_B550...	GTX_1060_3GB	XPG_Spectrix_D...	Corsair_HX850	Razer_Tomaha...	Windows_XP	SSD_Samsung_...	0
	460310254	Intel	i7_7700	MSI_H270_TOM...	GTX_1070_8GB	Patriot_Viper_R...	Corsair_AX1600i	NOX_Hummer_...	Linux_Xubuntu	SSD_Samsung_...	0
	476204260	AMD	AMD_Ryzen_5_...	MSI_MPG_B550...	GTX_1060_3GB	XPG_Spectrix_D...	Corsair_RM650x	NOX_Hummer_...	Windows_7	HDD_SATA_2TB	0
	758856552	AMD	AMD_Ryzen_9_...	ASUS_TUF_GA...	GTX_1050_3GB	GSkill_TridentZ...	Corsair_HX850	NOX_Hummer_...	Windows_XP	HDD_SATA_5TB	0

La tabla llamada **Table_Almacen** se encarga de guardar el nombre del almacén y la cantidad de productos fabricados que poseía el almacén a la hora de cerrar el formulario

	NombreDelAlamcen	CantidadDeProductos
	Productos fabricados Francisco Rocha	0
	Productos fabricados Fede Dávila	1

Base de datos:

Cree dos clases llamada SQLPC y SQLCelular las cuales poseen 3 métodos que se conectan a la base de datos del SQL

SQLPC:

(No pongo capturas de los métodos enteros porque son muy largos)

En InsertarPC uso el `INSERT INTO Tabla_PC`

```
2 references
public bool InsertarPC(FabricaPC PC)
{
```

En BorrarPC uso el `DELETE FROM Tabla_PC`

```
1 reference
public bool BorrarPC(int UPC)
{
```

En el ObtenerListaDatos uso el `SELECT * FROM Tabla_PC`

```
1 reference
public AlmacenProductosFabricados<Producto> ObtenerListaDato(AlmacenProductosFabricados<Producto> Almacen)
{
```

SQLCelular:

(No pongo capturas de los métodos enteros ya que como en el caso de PC porque son muy largos)

En InsertarCelular uso el `INSERT INTO Tabla_Celular`

```
2 references
public bool InsertarCelular(FabricaCelular Celular)
{
```

En BorrarCelular uso el `DELETE FROM Tabla_Celular`

```
public bool BorrarCelular(int UPC)
{
```

En el ObtenerListaDatos uso el `SELECT * FROM Tabla_Celular`

```
public AlmacenProductosFabricados<Producto> ObtenerListaDato(AlmacenProductosFabricados<Producto> Almacen)
{
```

Ambas clases poseen un constructor sin parámetros el cual inicializa la cadena de conexión

```
4 references
public SQLCelular()
{
    this.conexion = new SqlConnection(Properties.Settings.Default.Conexion);
}
```

```
public SQLPC()
{
    this.conexion = new SqlConnection(Properties.Settings.Default.Conexion);
}
```

Por ultimo tengo una clase llamada SQLAlmacen la cual no usa un Delete, pero si usa un Update y la cual también posee un constructor sin parámetros en la que se inicializa la cadena de conexión

SQLAlmacen:

En el InsertarAlmacen uso el `INSERT INTO Table_Almacen`

```
1 reference
public void InsertarAlmacen(AlmacenProdutosFabricados<Producto> Almacen)
{
```

En el ModificarCantidad uso el `UPDATE Table_Almacen`

```
1 reference
public void ModificarCantidad(AlmacenProdutosFabricados<Producto> Almacen)
{
```

En el ExisteAlmacen uso el `SELECT NombreDelAlamcen FROM Table_Almacen`

```
1 reference
public bool ExisteAlmacen(AlmacenProdutosFabricados<Producto> Almacen)
{
    string nombreAlmacen =
```

(Todos estos métodos se utilizan dentro del FormPrincipal)

Hilos:

Cree dos atributos Thread para lograr que no esté abierto más de una vez el formulario de mostrar y el formulario de eliminar

```
Thread mostrar;  
Thread eliminar;
```

Utilice hilos en 5 partes del FormPrincipal para que el resto de formularios se puedan ejecutar al mismo tiempo

- 1- Hilo para cuando se fabrica una PC (ButtonFabricarPC_Click)

```
Thread threadPC = new Thread(HiloPC);  
threadPC.IsBackground = true;  
threadPC.Start();
```

- 2- Hilo para cuando se fabrica un Celular (ButtonFabricarCelular_Click)

```
Thread threadCelular = new Thread(HiloCelular);  
threadCelular.IsBackground = true;  
threadCelular.Start();
```

- 3- Hilo para cuando se muestran los productos fabricados (ButtonMostrar_Click)

```
this.mostrar = new Thread(HiloMostrar);  
this.mostrar.IsBackground = true;  
this.mostrar.Start();
```

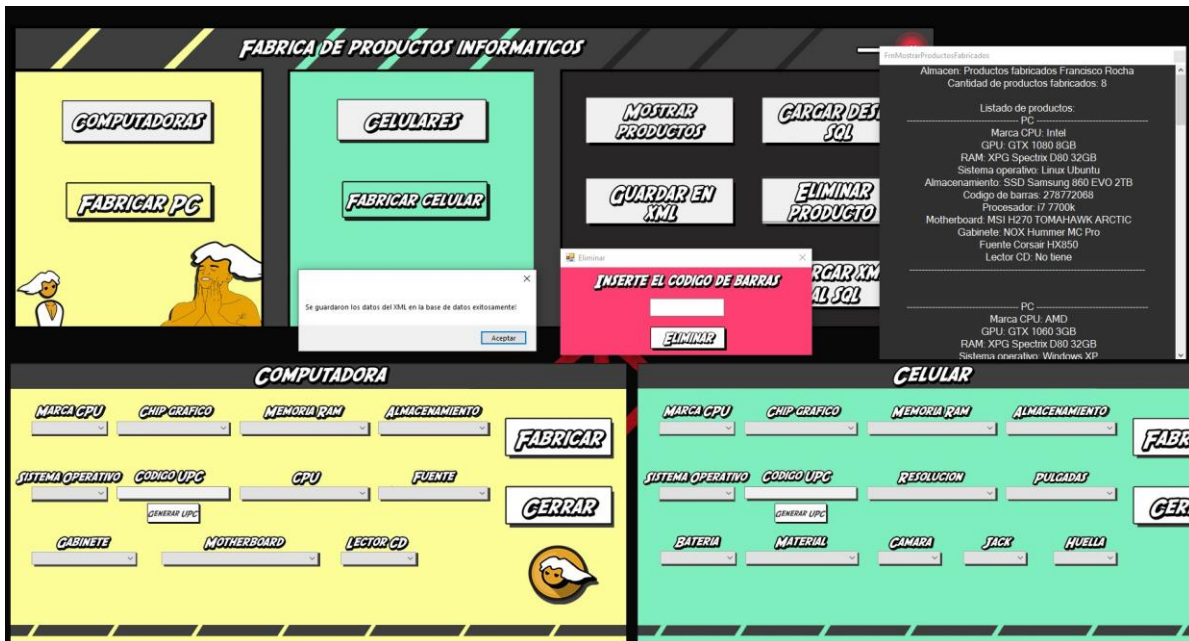
- 4- Hilo para cuando se elimina un producto (buttonEliminar_Click)

```
this.eliminar = new Thread(HiloBorrar);  
this.eliminar.IsBackground = true;  
this.eliminar.Start();
```

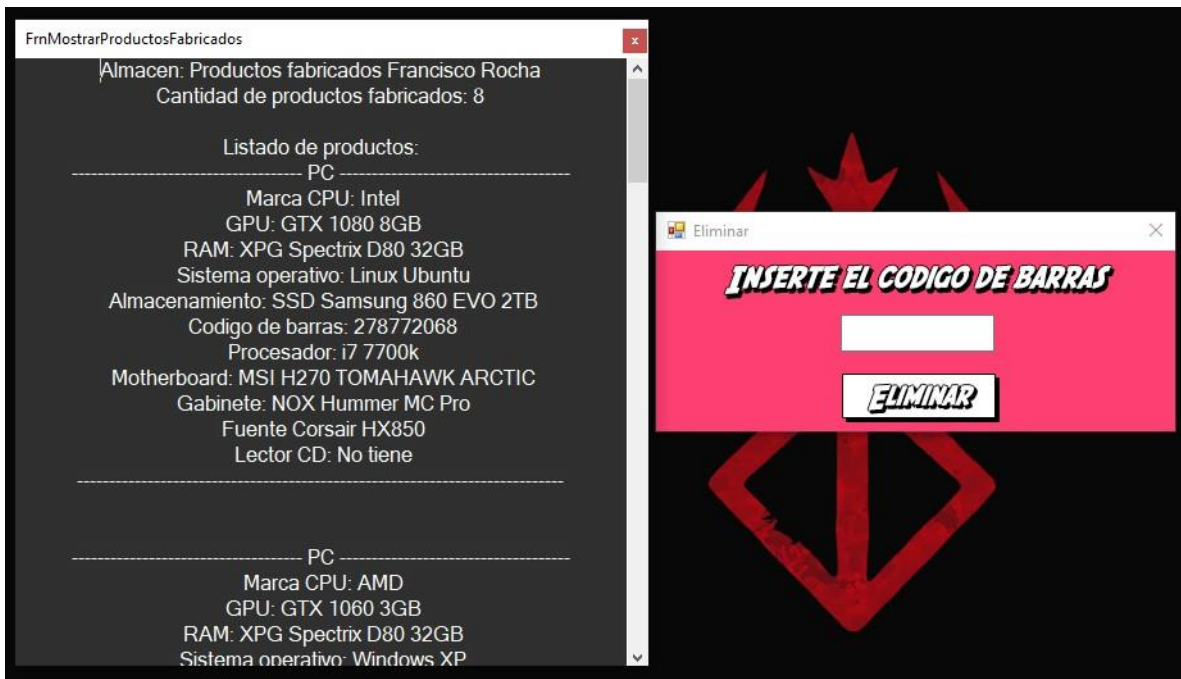
- 5- Hilo para cargar los datos del XML al SQL en segundo plano (buttonXMLaSQL_Click)

```
Thread threadXML = new Thread(HiloXMLtoSQL);  
threadXML.IsBackground = true;  
MessageBox.Show("Se estan guardando los datos del XML a la base de datos...");  
threadXML.Start();
```

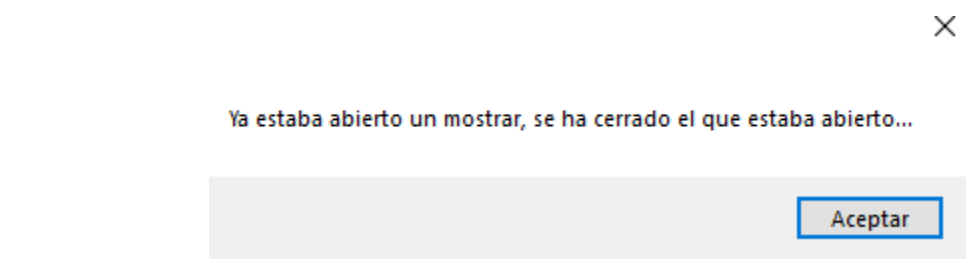
Todos los hilos ejecutados al mismo tiempo se verían así:



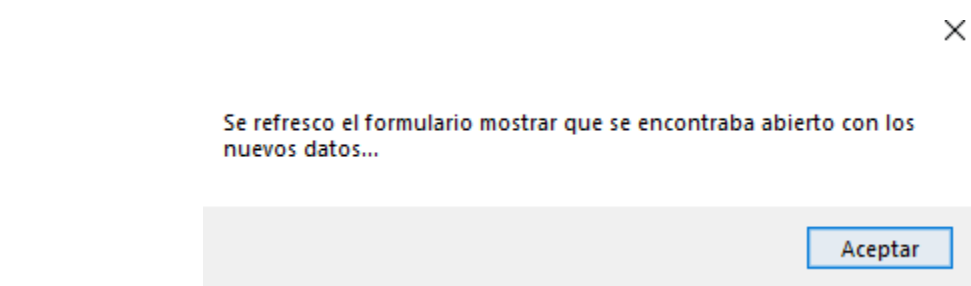
El hilo encargado de eliminar abre un hilo de mostrar si es que no existe ya abierto un hilo mostrando los datos del almacén, caso contrario no abre un nuevo mostrar y deja el que ya estaba abierto



El botón Mostrar productos fabricados se encarga de verificar si no existe ya un hilo mostrar, en caso de que ya exista un hilo que abra el FormMostrar se elimina el hilo y se crea uno nuevo, informando por pantalla lo ocurrido



Tanto el hilo de ButtonFabricarPC_Click, ButtonFabricarCelular_Click y buttonEliminar_Click refrescan los datos del hilo mostrar mediante un evento si es que este último se encuentra abierto



Eventos y Delegados:

En el namespace FormFranciscoRocha cree el delegado llamado Callback

```
namespace FormFranciscoRocha
{
    public delegate void Callback();
}
```

Dentro del form principal cree el evento llamado RefreshMostrar

```
public partial class FormPrincipal : Form
{
    public event Callback RefreshMostrar;
}
```

Dentro del botón que se encarga de abrir el formulario de mostrar le suscribí al evento el método RefreshRichMostrar

```
RefreshMostrar += Mostrar.RefreshRichMostrar;
```

El código de este método es el siguiente:

```
3 references
public void RefreshRichMostrar()
{
    try
    {
        if (this.InvokeRequired)
        {
            Callback callback = new Callback(this.RefreshRichMostrar);
            object[] args = new object[] { };

            this.Invoke(callback);
        }
        else
        {
            this.WindowState = FormWindowState.Normal;
            this.Mostrar();
        }
    }
    catch (Exception ex)
    {
        Archivos<Producto>.LogErrores(ex.ToString());
    }
}
```

El método se encarga de que cuando se dispara el evento refresque el FormMostrar actualizando el RichTextBox.

Hago el Invoke del evento en HiloPC, HiloCelular y en HiloBorrar.

```
RefreshMostrar.Invoke();
```

Métodos de extensión:

Utilice métodos de extensión para extender 3 clases, las cuales son IProductos, ISQL y NoSeEncontroException mediante una clase que cree llamada extensiones, decidí extender la clase IProductos con métodos que se repitan en PC y Celular pero que no se utilizan en la clase Producto, esto me sirve para no tener métodos repetidos en la clase PC y Celular, los métodos que cree son más que nada validaciones, con el caso de la extensión de la clase ISQL es lo mismo, lo utilice para codear una sola vez lo que se repetía en la clase SQLCelular y SQLPC, por ultimo extendí la clase NoSeEncontroException en la que cree un método el cual informa el mensaje de la excepción.

Extension de la clase IProductos:

```
public static string Fix(this IProductos extension, string arreglar)
{
    StringBuilder correcion = new StringBuilder(arreglar);

    try
    {
        for (int i = 0; i < correcion.Length; i++)
        {
            if (correcion[i] == '_' && i == 0)
            {
                correcion.Remove(0, 1);
            }
            else if (correcion[i] == '_')
            {
                correcion[i] = ' ';
            }
        }
    }
    catch (Exception ex)
    {
        Archivos<Producto>.LogErrores(ex.ToString());
    }

    return correcion.ToString();
}
```

```
public static bool ValidarMarca(this IProductos extension, MarcaCPU marca)
{
    bool retorno = false;

    try
    {
        if ((int)marca == 1 || (int)marca == 2)
        {
            retorno = true;
        }
    }
    catch (Exception ex)
    {
        Archivos<Producto>.LogErrores(ex.ToString());
    }

    return retorno;
}
```

(estos son algunos ejemplos, hay mas)

Extension de la clase ISQL:

```
3 references
public static bool BoolSelec(this ISQL extension, int index)
{
    bool retorno;

    if (index == 0)
    {
        retorno = true;
    }
    else
    {
        retorno = false;
    }

    return retorno;
}
```

Extension de la clase NoSeEncontroException:

```
public static string InformarExcepcion(this NoSeEncontroException excepcion)
{
    return "No se encontro el UPC indicado";
}
```

Aca dejo un ejemplo de la implementacion de uno de los metodos de extension:

```
this.Fix(PLACA_MADRE.ToString()), this.Fix(GABINETE.ToString()), this.Fix(FUENTE.ToString()),
(extension) string IProductos.Fix(string arreglar)
Metodo para rempalzar los _ de los enum por puntos '.', en caso de iniciar con _ elimina el caracter
Returns:
Retorna el string ingresado pero sin guines bajos
```