Para esta segunda parte se agregaron las clases correspondientes y se hizo una refactorización del código, basándose en los 3 pilares, los **objetos**, la habilidad de **leer y guardar XML** y de impactar los **cambios en mi base de datos**.

Para el diseño de las base de datos:

Nuestra base de datos de disponibles (los diseños de la indumentaria) mantiene una conexión constante con la base de datos. Esta se carga y se mantiene actualizada con nuestro listbox.

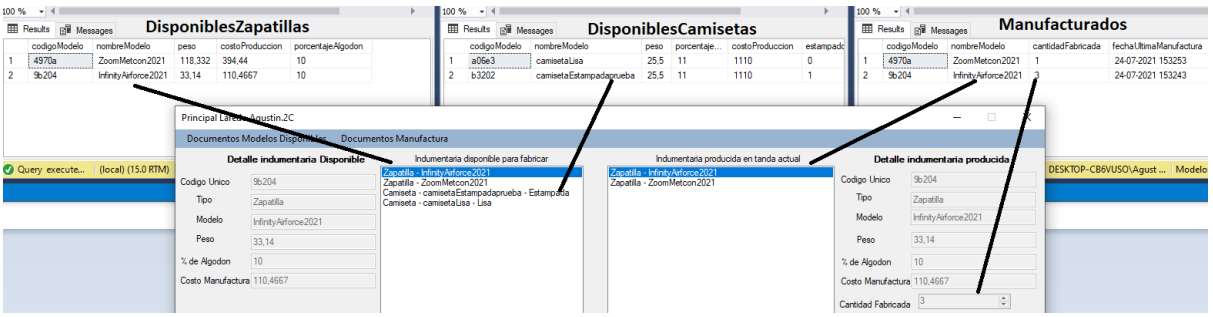
Nuestra base de datos de manufacturadas persiste las veces que mandamos a producir una indumentaria. En cada ejecución nuestra lista de manufacturados va a estar vacia, ya que esto seria nuestra tanda actual. Una vez mandado a fabricar podemos eliminar una indumentaria de la tanda actual viendo como esto impacta en el listbox y en la base de datos, nunca pudiendo eliminar lo que fue manufacturado en tandas anteriores.

Un ejemplo para demostrar este funcionamiento.

Una zapatilla fue fabricada 2 veces en la tanda anterior, cuando en la tanda/ejecución actual la mande a fabricar 3 veces en nuestra base de datos nos mostrara que fabricamos 5, guardando tambien el fecha de ultima manufactura. Si en esta misma ejecución borramos las 3 veces que mandamos a fabricar en nuestra base de datos seguirá mostrandonos 2 como al principio.

Clase 21 : SQL. Decidí usar 3 tablas, dos para los diseños disponibles que se cargan automaticamente al cargar la aplicacion. Estas dos tablas solo pueden consultarse, agregarle y borrarle indumentarias, no necesitamos que se actualice ningun registro. Almacena la info de cada diseño.

Por otro lado la tabla de manufacturados agrega los items que voy fabricando en mi aplicacion, pero si la indumentaria ya esta en la lista, actualizará la fecha de ultima manufactura y le agregara +1 a la cantidad fabricada



Clase 22: Base de datos.

Aqui la implementacion para borrar una indumentaria de la base de datos de manufacturados.

Mediante una validación con el metodo LeerBaseDatosManufacturados decidimos si borraremos el registro o solamente restaremos -1 a la cantidad manufacturada.

```
public static void BorrarIndumentariaManufacturadaBaseDatos(Indumentaria indumentariaBorrar)
{
    try
    {
        //primero debo consultar a mi base de datos de manufacturados
        Dictionary<string, int> manufacturadosDB = new Dictionary<string, int>();
        LeerGuardarBaseDatos.LeerBaseDatosManufacturados(out manufacturadosDB);

        //Ahora tenemso un diccionario con llave siendo el modelos y con la cantidad de fabricados que tiene nuestro modelo
        int cantidadFabricados = manufacturadosDB[indumentariaBorrar.Modelo];

        //Abrimos nuestra conexion
        LeerGuardarBaseDatos.conexion = new SqlConnection("Data Source=.; Initial Catalog=Modelos; Integrated Security=True");
        LeerGuardarBaseDatos.comando = new SqlCommand();
        LeerGuardarBaseDatos.comando.Connection = LeerGuardarBaseDatos.conexion;

        if (LeerGuardarBaseDatos.conexion.State != System.Data.ConnectionState.Open && LeerGuardarBaseDatos.conexion.State != System.Data.ConnectionState.Connecting)
        {
            LeerGuardarBaseDatos.conexion.Open();
        }

        //limpio mis parametros
        LeerGuardarBaseDatos.comando.Parameters.Clear();

        //Si nuestra indumentaria ya está en nuestra base de datos, osea que tiene mas de una cantidad, solamente le restare uno a esta cantidad
        if (manufacturadosDB.ContainsKey(indumentariaBorrar.Modelo) && cantidadFabricados > 1)
        {
            LeerGuardarBaseDatos.comando.CommandText = "UPDATE Manufacturados SET cantidadFabricada = @cantFab WHERE nombreModelo = @nombreMod";

            LeerGuardarBaseDatos.comando.Parameters.AddWithValue("@cantFab", cantidadFabricados-1);
        }
        else//si nuestra cantidad es 1, debemos borarla
        {
            LeerGuardarBaseDatos.comando.CommandText = "DELETE FROM Manufacturados WHERE nombreModelo = @nombreMod";
        }

        LeerGuardarBaseDatos.comando.Parameters.AddWithValue("@nombreMod", indumentariaBorrar.Modelo);
        LeerGuardarBaseDatos.comando.ExecuteNonQuery();
    }
    catch (Exception e)
    {
        throw new Exception("Hubo un error al borrar el modelo en la base de datos", e);
    }
    finally
    {
        LeerGuardarBaseDatos.conexion.Close();
    }
```

Clase 23: Hilos

Para la parte de Hilos decidí usar Tasks que es una forma de implementar hilos en la aplicación que se libera de la tarea de estar pendiente de abrir o cerrarlo o de administrarlos.

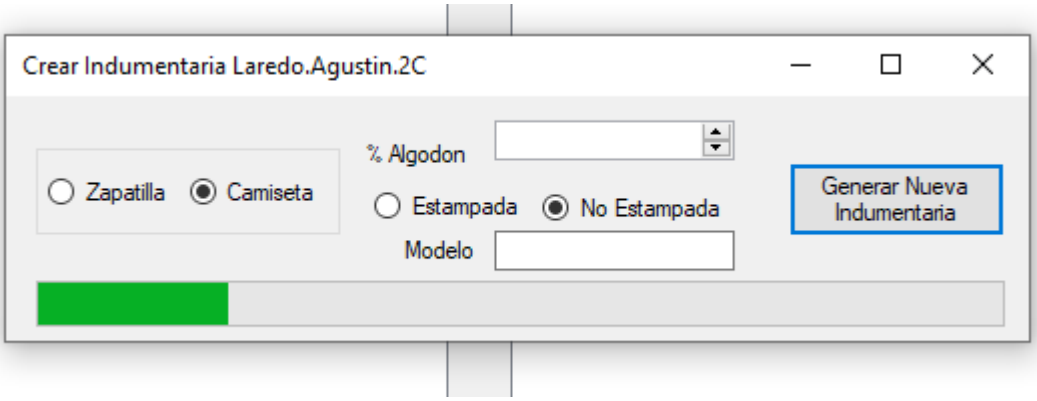
```
private void btnNuevaIndumentaria_Click(object sender, EventArgs e)
{
    if (this.rdbZapatilla.Checked  )
    {
        Zapatilla zapatillaNueva = new Zapatilla(((ECapelladas)this.cmbCapellada.SelectedItem), (ESuelas)this.cmbSuela.SelectedItem);
        try
        {

            this.FabricandoIndumentaria.Invoke(this, new IndumentariaEventArgs(zapatillaNueva));

            Task MostrarProgreso = new Task(() =>
            {
                AvanzarBarra(zapatillaNueva);
            });

            MostrarProgreso.Start();
        }
        catch(Exception error)
        {
            MessageBox.Show(error.Message);
        }
    }
```

De esta forma se nos mostrar una barra de progreso que simulará el proceso de diseño desde otro hilo creado por el Task




```
2 referencias
private void ActualizarBarraProgreso ()
{
    if (this.pbCreacion.InvokeRequired)
    {
        //si el flujo de aplicacion entra aqui desde un thread secundario debemos invocarlo nuevamente desde el thread primario apra que modifique el form
        DelegadoActualizarForm actualizarBarra = new DelegadoActualizarForm(ActualizarBarraProgreso);
        this.Invoke(actualizarBarra);
    }
    else
    {
        //actualizo la barar de progreso de mi form
        this.pbCreacion.Value = this.progreso;
        if(this.pbCreacion.Value == 100)
        {
                //Una vez llena la barra de progreso, reseteamos el contador y limpiamos la barra
                this.progreso = 0;
                this.pbCreacion.Value = this.progreso;
        }
        else
        {
            //avanzamos +1 el progreso
            this.progreso++;
        }
    }
}
2 referencias
private void AvanzarBarra(Indumentaria zapatillaNueva)
{
    for (int i = 0; i < 101; i++)
    {
        //cada 50 ms actualizo mi bara de progreso uno mas
        this.ActualizarBarraProgreso();
        Thread.Sleep(50);
    }
    // una vez que llenamos la barra de progreso avisamos que ya se creó la indumentaria que pasamos por parametro
    this.AvisarCreacion.Invoke(zapatillaNueva);
}
```

Clase 24: Eventos y delegados

Al momento de terminar esta barra de progreso vemos como invoca el evento AvisarCreacion pasandole como parámetro la indumentaria que acaba de crear.

En el Constructor vemos que método se suscribe

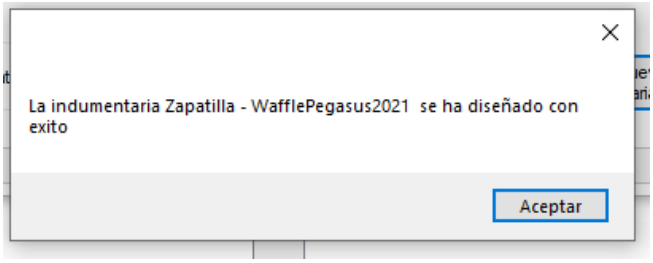
```
public delegate void DelegadoIndumentaria(Indumentaria indu);
public delegate void DelegadoActualizarForm();
4 referencias
public partial class FrmCrearIndumentaria : Form
{
    public event EventHandler<IndumentariaEventArgs> FabricandoIndumentaria;
    public event DelegadoIndumentaria AvisarCreacion;
    public ListBox listadoDisponibles;
    private int progreso;

    1 referencia
    public FrmCrearIndumentaria(ListBox listaDisponibles)
    {
        InitializeComponent();
        this.progreso = 0;
        this.listadoDisponibles = listaDisponibles;
        FabricandoIndumentaria += AltaBajaConsultaListas.AgregarIndumentariaDisponible;
        FabricandoIndumentaria += LeerGuardarBaseDatos.GuardarDisponibleEnDB;
        AvisarCreacion += MostrarMensajeCreacion;
    }
```

```
1 referencia
public void MostrarMensajeCreacion (Indumentaria zapatillaNueva)
{
    MessageBox.Show($"La indumentaria {zapatillaNueva.ToString()} se ha diseñado con exito");

}
```

Esto permite que al terminar de crear la indumentaria nos avise el nombre del modelo.



Clase 25. Metodos de extension

Cree un metodo de extension para Guid que simplifica su uso en esta aplicación

```
namespace Logica.MetodoExtension

{
    0 referencias
    public static class CodigoUnicoExtension
    {
        1 referencia
        public static string SimplificarCodigoUnico(this Guid codigo)
        {
            return codigo.ToString("N").Substring(0, 5);
        }
    }
}
```

```
0 referencias
protected Indumentaria()
{
    this.codigoUnico = Guid.NewGuid().SimplificarCodigoUnico();
    FabricandoIndumentaria += AltaBajaConsultaListas.AgregarIndumentariaProduccion;
    FabricandoIndumentaria += LeerGuardarBaseDatos.GuardarFabricadoEnDB;
}
```