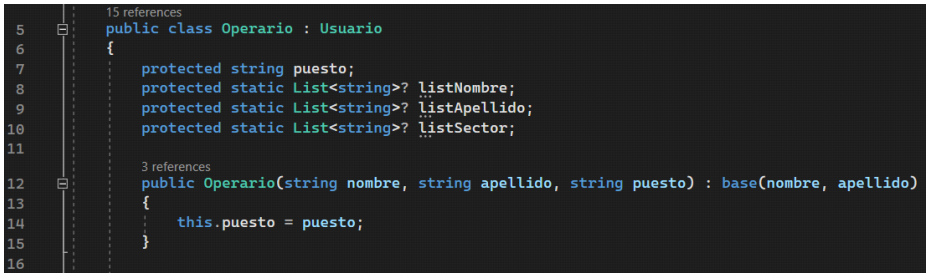
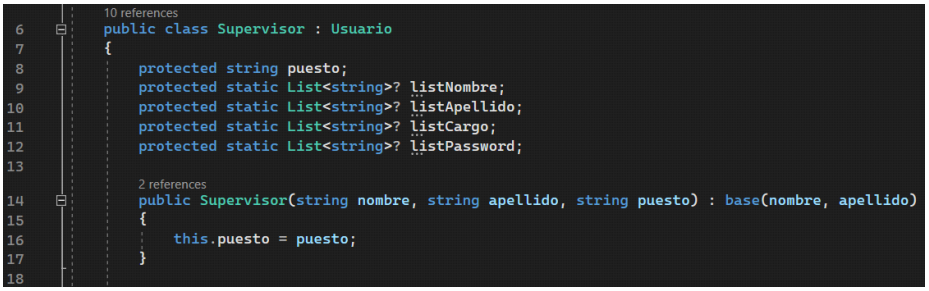
Temas utilizados en el proyecto

+ Herencia:

Aplico herencia en dos casos, uno de ellos es con las clases Operario y Supervisor que heredan de Usuario.

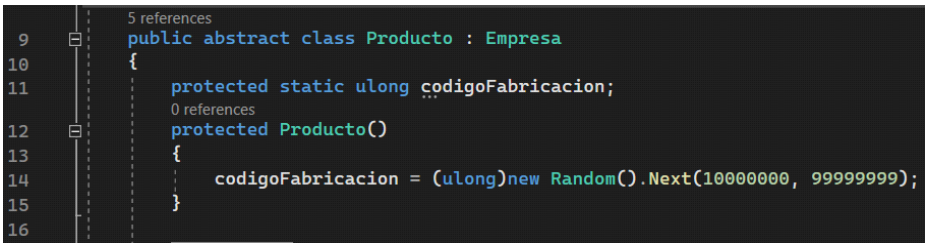


```
15 references
public class Operario : Usuario
{
    protected string puesto;
    protected static List<string>? listNombre;
    protected static List<string>? listApellido;
    protected static List<string>? listSector;

    3 references
    public Operario(string nombre, string apellido, string puesto) : base(nombre, apellido)
    {
        this.puesto = puesto;
    }
```

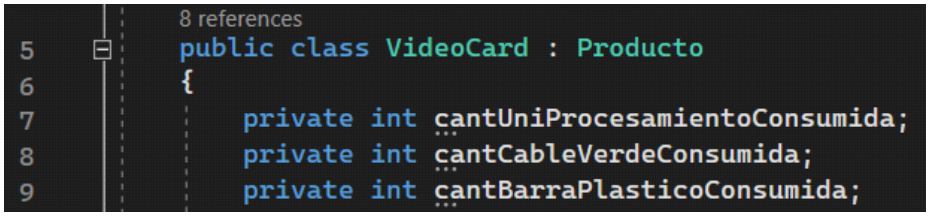


```
10 references
public class Supervisor : Usuario
{
    protected string puesto;
    protected static List<string>? listNombre;
    protected static List<string>? listApellido;
    protected static List<string>? listCargo;
    protected static List<string>? listPassword;

    2 references
    public Supervisor(string nombre, string apellido, string puesto) : base(nombre, apellido)
    {
        this.puesto = puesto;
    }
```

Luego tenemos la clase Producto que hereda de Empresa, y luego VideoCard, Motherboard, Cabinet y Ram que heredan de Producto.



```
5 references
public abstract class Producto : Empresa
{
    protected static ulong codigoFabricacion;
    0 references
    protected Producto()
    {
        codigoFabricacion = (ulong)new Random().Next(10000000, 99999999);
    }
```

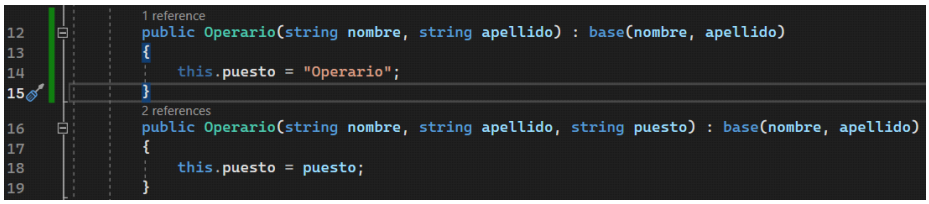


```
8 references
public class VideoCard : Producto
{
    private int cantUniProcesamientoConsumida;
    private int cantCableVerdeConsumida;
    private int cantBarraPlasticoConsumida;
```

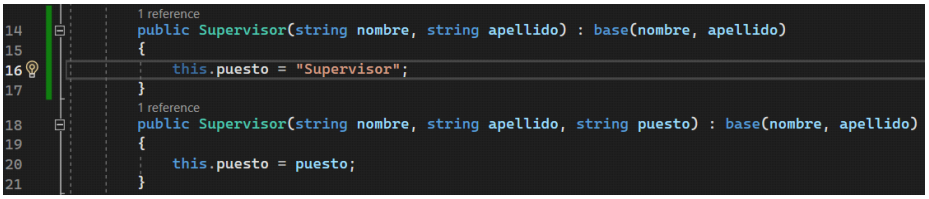
En ambos casos aplico herencia con el fin de obviar escribir tanto código, de esta manera es mas legible y práctico para trabajar con ellas, ya que mis clases derivadas contienen información de sus clases base y de esta forma me evito escribir tanto código innecesario.

+ Sobrecarga:

Aplico sobrecarga de constructores en las clases Operario y Supervisor.



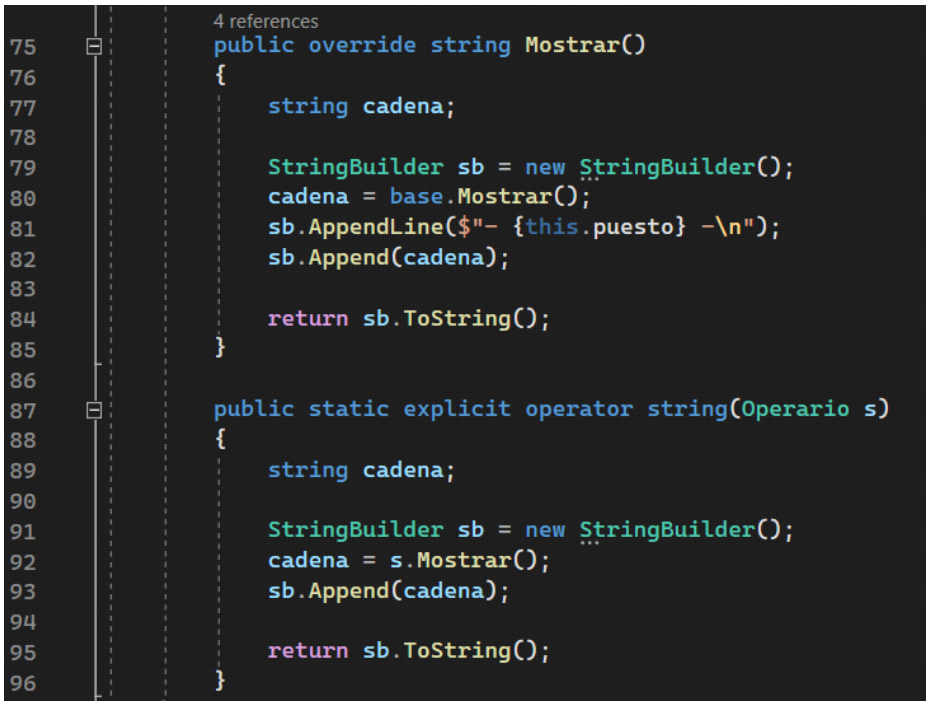
```
1 reference
public Operario(string nombre, string apellido) : base(nombre, apellido)
{
    this.puesto = "Operario";
}
2 references
public Operario(string nombre, string apellido, string puesto) : base(nombre, apellido)
{
    this.puesto = puesto;
}
```



```
1 reference
public Supervisor(string nombre, string apellido) : base(nombre, apellido)
{
    this.puesto = "Supervisor";
}
1 reference
public Supervisor(string nombre, string apellido, string puesto) : base(nombre, apellido)
{
    this.puesto = puesto;
}
```

Al utilizar la sobrecarga en los constructores me permite tener más de una forma de instanciar una clase dándole más o menos parámetros por referencia. De esta forma puedo tener un objeto con más parámetros dados por el usuario o menos ya que tendría algunos datos escritos de "forma default".

Aplico sobrecarga de métodos en las clases Operario y Supervisor - y luego VideoCard, Motherboard, Cabinet y Ram.



```
4 references
public override string Mostrar()
{
    string cadena;

    StringBuilder sb = new StringBuilder();
    cadena = base.Mostrar();
    sb.AppendLine($"- {this.puesto} -\n");
    sb.Append(cadena);

    return sb.ToString();
}

public static explicit operator string(Operario s)
{
    string cadena;

    StringBuilder sb = new StringBuilder();
    cadena = s.Mostrar();
    sb.Append(cadena);

    return sb.ToString();
}
```

Al utilizar la sobrecarga en estos métodos me permite tener mas de una forma de llamar a estos metodos, en este caso mostrando mas o menos datos del objeto.

```
3 references
public override string Mostrar()
{
    StringBuilder sb = new StringBuilder();

    sb.AppendLine($"Empresa: {RazonSocial} - CUIT: {CUIT}");
    sb.AppendLine($"Codigo de fabricacion: {CodigoFabricacionVideocard}");

    return sb.ToString();
}

/// <summary>
/// Sobrecarga del metodo Mostrar
/// </summary>
/// <param name="v">Clase actual</param>
public static explicit operator string(VideoCard v)
{
    StringBuilder sb = new StringBuilder();

    sb.AppendLine(v.Mostrar());
    sb.AppendLine($"Producto *- {TipoProducto} -*");
    sb.AppendLine($"Cantidad de productos fabricados -> {CantidadProducto}");

    return sb.ToString();
}
```

+ Propiedades e Indexadores:

Utilizo propiedades en las clases Usuario, Operario y Supervisor - y luego en Empresa, Producto, VideoCard, Motherboard, Cabinet y Ram.

```
#region Propiedades
4 references
public string Nombre
{
    get { return this.nombre; }
    set { this.nombre = value; }
}

4 references
public string Apellido
{
    get { return this.apellido; }
    set { this.apellido = value; }
}
#endregion
```

```
42          #region Propiedades
            3 references
43          public static int CantidadProducto
44          {
45              get { return contadorProducto; }
46              set { contadorProducto = value; }
47          }
48
            2 references
49          public static string TipoProducto
50          {
51              get
52              {
53                  if (tipoProducto != null)
54                  {
55                      return tipoProducto;
56                  }
57
58                  return "No Type";
59              }
60              set { tipoProducto = value; }
61          }
62
```

Utilizo propiedades para poder obtener o modificar atributos de las clases de forma mas práctica y segura, sin necesidad de tener mis atributos públicos.

Utilizo indexadores en las clases VideoCard, Motherboard, Cabinet y Ram.

```
            1 reference
129         public int this[int index]
130         {
131             get { return listaValores[index]; }
132             set { listaValores[index] = value; }
133         }
134
135         /// <summary>
136         /// Pisamos la lista con valores actualizados
137         /// </summary>
138         /// <param name="cantidadFdabricar">Cantidad de productos a fabricar</param>
139         /// <returns>Retornamos la lista con los valores actualizados</returns>
            1 reference
140         public List<int> PisarLista(int cantidadFdabricar)
141         {
142             for (int i = 0; i < listaValores.Count; i++)
143             {
144                 this[i] = cantidadFdabricar * this.listaCantidades[i];
145             }
146
147             return this.listaValores;
148         }
```

Utilizo indexadores para poder obtener o modificar índices específicos de colecciones, en mi caso, Listas.

+ Colecciones:

Utilizo Listas y Diccionarios.

```
25          private Dictionary<string, int> dictProducto;
26          private List<TextBox> listaTxtBox;
27          private List<int> listaValores;
28
            4 references
29          public FrmProductoFinal(MenuUsuario frmUsuario, int producto)
30          {
31              InitializeComponent();
32              this.frmUsuario = frmUsuario;
33              this.videoCard = new VideoCard();
34              this.motherboard = new Motherboard();
35              this.ram = new Ram();
36              this.cabinet = new Cabinet();
37              this.producto = producto;
38              this.dictProducto = new Dictionary<string, int>();
39              this.listaTxtBox = new List<TextBox>();
40              this.listaValores = new List<int>();
41          }
```

Lo que más utilizo en mi proyecto son listas, ya sea para datos hardcodeados, lista de valores (stock a actualizar), listas de objetos (text box, operarios, etc..) y los diccionarios lo utilizo más que nada para el Stock que voy modificando y mostrando en los formularios.

+ Enumerados:

```
1   public enum TiposProductos { VideoCard, Motherboard, Ram, Cabinet }
        6 references
2   public enum ProcesoProduccion { Soldar, Conectar , Ensamblar, Empaquetar}
```

Utilizo enumerados para trabajar con los tipos de productos y al momento de fabricar productos que estos tengan procesos para la producción.

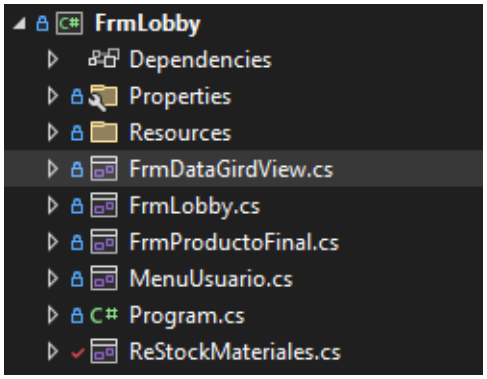
+ Casteo Explicito:

Convertimos de forma explícita un tipo decimal a int

```
190     /// <summary>
191     /// Casteo de dato de decimal a entero
192     /// </summary>
193     /// <param name="dato"></param>
194     /// <returns>Retorno el dato casteado</returns>
        3 references
195     public static int CasteoExplicito(decimal dato)
196     {
197         int valor = 0;
198
199         if (dato > -1)
200         {
201             valor = (int)dato;
202         }
203         return valor;
204     }
205
206     /// <summary>
207     /// Casteo Explicito de una Lista de decimal a entero
208     /// </summary>
209     /// <param name="listaDecimal"></param>
210     /// <returns>Retornamos la lista casteada</returns>
        1 reference
211     public static List<int> CasteoExplicitoLista(List<decimal> listaDecimal)...
```

+ Formularios modales:

En el proyecto utilizo cinco formularios



1.Lobby -> ingreso del usuario

2.Menu Usuario -> tiene la información y acceso a los sectores de producción.

3.Sector Producción -> dependiendo con que botón interactúes en el menu anterior ingresaras al mismo form con sus diferencias de información para la producción de cierto producto.

4.Ingreso Stock -> siendo supervisor puede ingresar al formulario para modificar el sotck

5.Registro Operarios -> listado de los operarios ..

+ Clases estáticas:

Hay dos clases estáticas en el proyecto Stock y Procesos

```
5   public static class Stock
6   {
7       private static int cantCircuitoElect;
8       private static int cantCircuitoElectAvanzado;
9       private static int cantUnidadProcesamiento;
10      private static int cantBarraPlastico;
11      private static int cantCableVerde;
12      private static int cantCableRojo;
13      private static int cantBaraHierro;
14      private static int cantEngranajeHierro;
15      private static int cantFibrasVidrio;
16      private static int cantCondensador;
17      private static int cantVentilador;
18      private static List<int> listaStock;
19
```

La que más "fuerza" tiene en el código es Stock que la utilizamos en gran parte del código ya que estamos interactuando con nuestro stock constantemente.

+ Polimorfismo:

Clases abstractas y métodos abstractos

```
2 references
9   public abstract class Empresa
10  {
11      protected static string? razonSocial;
12      protected static string? cuit;
        0 references
13      protected Empresa()
14      {
15          razonSocial = "Factory.IO";
16          cuit = "16-56433112-2";
17      }
18
        4 references
19      protected string CUIT ...
        4 references
34      protected string RazonSocial ...
49
```

A la clase Empresa no hay necesidad de instanciarla en ningún momento y la clase Producto tampoco, pero si creamos dos métodos abstractos, uno para mostrar información y otro para fabricar producto.

```
5 references
public abstract class Producto : Empresa
{
    protected static ulong codigoFabricacion;
    0 references
    protected Producto()
    {
        codigoFabricacion = (ulong)new Random().Next(10000000, 99999999);
    }

    0 references
    protected ulong CodigoFabricacion
    {
        get { return codigoFabricacion; }
        set { codigoFabricacion = value; }
    }

    8 references
    public abstract bool FabricarProducto(int valor, List<int> lista);

    12 references
    public abstract string Mostrar();
}
```

Método virtual

```
6 references
public virtual string Mostrar()
{
    StringBuilder sb = new StringBuilder();

    sb.AppendLine($"Nombre: {this.nombre}");
    sb.AppendLine($"Apellido: {this.apellido}");

    return sb.ToString();
}
```

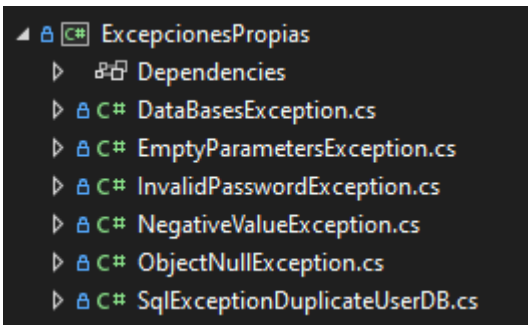
En la clase Usuario le creamos un método virtual con el fin de sobrescribirlo en las clases Supervisor y Operario, teniendo así un método donde mostramos mas información que nuestra clase base.

+ Excepciones:

Algunas de las excepciones utilizadas para evitar que el programa cierre de forma abrupta.

```
catch (EmptyParametersException ex)
{
    this.manejadorArchivosTXT.EscribirArchivo(this.path
    mostrarError(ex.Message, "Parametros Vacios");
}
catch (FormatException ex)
{
    this.manejadorArchivosTXT.EscribirArchivo(this.path
    mostrarError(ex.Message, "Tipo de dato Incorrecto");
}
catch (InvalidPasswordException ex)
{
    this.manejadorArchivosTXT.EscribirArchivo(this.path
    mostrarError(ex.Message, "ID incorrecto");
}
catch (DataBasesException ex)
{
    this.manejadorArchivosTXT.EscribirArchivo(this.path
    mostrarError(ex.Message, "Error con DB");
}
catch (NegativeValueException ex)
{
    this.manejadorArchivosTXT.EscribirArchivo(this.path
    mostrarError(ex.Message, "El valor en Stock no puede
}
catch (Exception ex)
{
    this.manejadorArchivosTXT.EscribirArchivo(this.path
    mostrarError(ex.Message, "Error Inesperado");
}
```

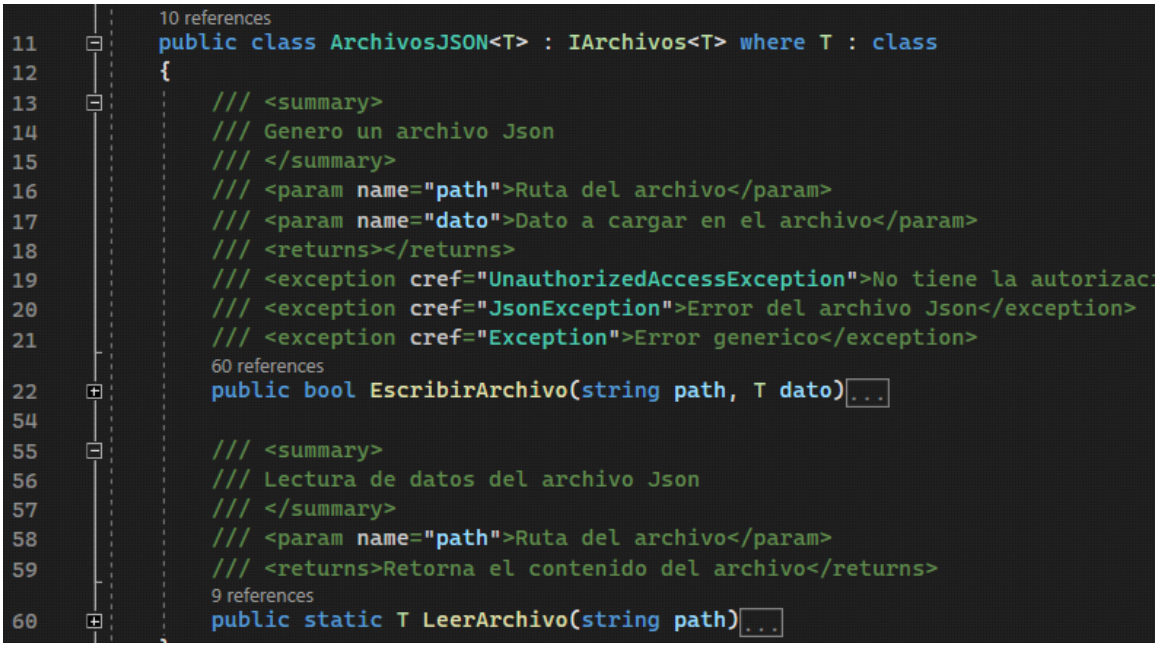
Excepciones propias.



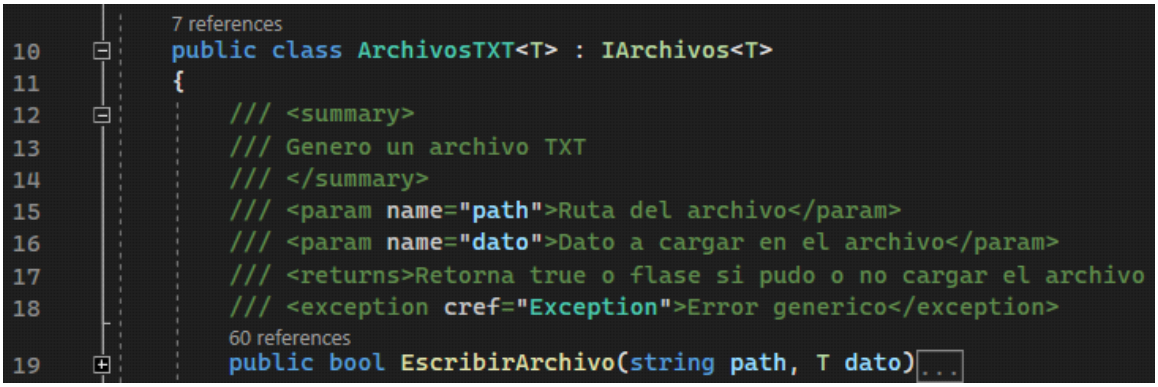
+ Archivos y serializacion

Utilizamos tres archivos distintos, JSON, TXT y XML.

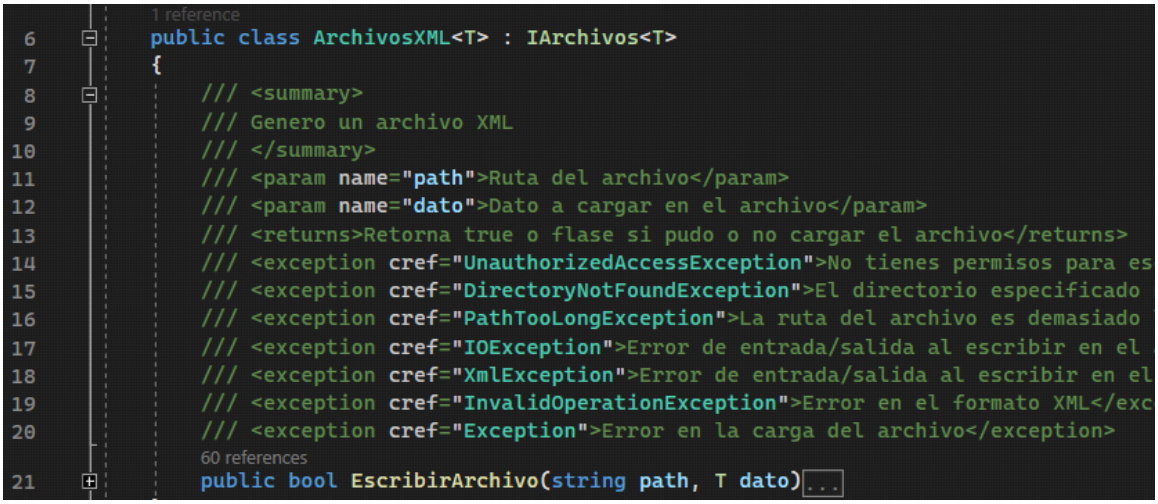
En el archivo JSON, guardamos las imagenes que cargamos en nuestros formularios.



En el archivo TXT tenemos un Log de errores (excepciones) con un formato especifico pedido por el enunciado del proyecto.

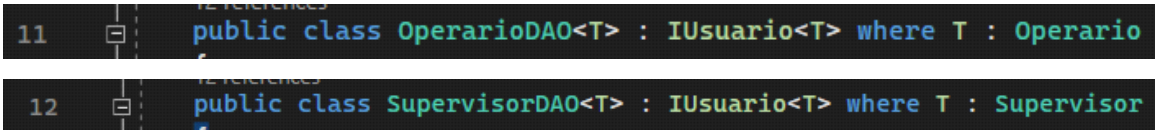


En el archivo XML guardamos la cantidad de materiales que se encuentran en el Stock, y son actualizadas cada vez que se modifica su cantidad.



+ Generics

Aplicamos Generics en las clases SupervisorDAO y OperarioDAO, para poder trabajar con las funciones con datos Genericos en sus respectivos parametros o retornos de los propios metodos.

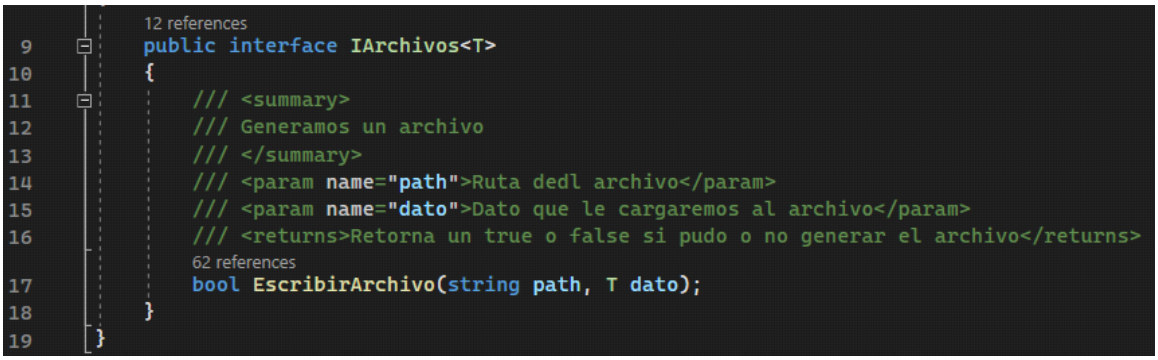


Y ademas en los Archivos tambien aplicamos Generics.

+ Interfaces

Aplicamos tres tipos de Interfaces: IArchivos, IMateriales y IUsuarios.

IArchivos estan implementados en las clases ArchivosJSON, ArchivosTXT y ArchivosXML.



IMateriales estan implementados en las clases ProductosDAO y StockDAO.

```
12 references
public interface IMateriales
{
    /// <summary>
    /// Materiales que se modificaran en la DB
    /// </summary>
    /// <param name="material">Material a modificar en la DB</param>
    /// <param name="cantidadAgregar">Cantidad a modificar</param>
    /// <param name="id">ID que necesito para poder modificar datos en la DB</param>
    /// <returns>Retorna un true o flase si pudo o no modificar los datos</returns>
    30 references | 11/11 passing
    bool Modificar(string material, int cantidadAgregar, int id);

    /// <summary>
    /// Lectura de datos de la DB
    /// </summary>
    /// <param name="id">ID para poder interactuar con la DB</param>
    /// <param name="material">Material que voy a leer</param>
    /// <returns>Retorna un int, la cantidad leida</returns>
    33 references
    int LeerPorMaterial(int id, string material);
```

IUsuarios estan implementados en las clases OperarioDAO y SupervisorDAO.

```
11 references
public interface IUsuario<T>
    where T : Usuario
{
    /// <summary>
    /// Registrar un Usuario en la DB
    /// </summary>
    /// <param name="usuario">Usuario a registrar</param>
    /// <returns>Retorna un true o flase si pudo o no registrar correctamente un Usuario en la DB</returns>
    6 references | 2/2 passing
    bool GuardarRegistro(T usuario);

    /// <summary>
    /// Leo un Usuario por ID de la DB
    /// </summary>
    /// <param name="id">ID necesario para saber a que Usuario visualizare</param>
    /// <returns>Retorno un Usuario</returns>
    7 references
    T LeerPorID(int id);

    /// <summary>
    /// Leo un Usuario por DNI de la DB
    /// </summary>
    /// <param name="dni">DNI necesario para saber a que Usuario visualizare</param>
    /// <returns>Retorno un Usuario</returns>
    4 references
    T LeerPorDNI(long dni);
```

+ Test Unitario

Aplicamos 23 Test Unitarios

```
[TestClass]
0 references
public class Pruebas
{
    [TestMethod]
    [ExpectedException(typeof(SqlExceptionDuplicateUserDB))]
    | 0 references
    public void GuardarRegistro_CuandoRegistroUnOperarioConUnDNIExistente_DeberiaLanzarSqlExceptionDuplicateUserDB()...

    [TestMethod]
    [ExpectedException(typeof(SqlExceptionDuplicateUserDB))]
    | 0 references
    public void GuardarRegistro_CuandoRegistroUnSupervisorConUnDNIExistente_DeberiaLanzarSqlExceptionDuplicateUserDB()...

    [TestMethod]
    | 0 references
    public void VerificarExisteSupervisor_CuandoIngresoUnSupervisorEnElSistema_DeberiaDevolverUnTrue()...
```

+ Base de Datos

Interacción con la Base de Datos, para la modificacion de Materiales y Productos.

Hay un C.R.U.D. en donde se puede generar Supervisores y Operarios, lectura de datos tanto de Supervisor como de Operario, modificar y eliminar datos del Operario.

Generar Usuario

```
/// <summary>
/// Guardamos / agregamos un Usuario en la base de datos
/// </summary>
/// <param name="usuario">Supervisor que agregaremos a la DB</param>
/// <returns>Retorna un true o false para ver si se pudo o no agregar a la DB</returns>
/// <exception cref="SqlExceptionDuplicateUserDB">Lanzara la excepcion en caso de que el DNI ingresado EXISTA</exception>
/// <exception cref="DataBasesException">Lanzara la excepcion en caso de que haya un error con la DB</exception>
5 references | 2/2 passing
public bool GuardarRegistro(T usuario)...
```

Leer datos del Supervisor y Operario

```
/// <summary>
/// Leemos y devolvemos un Supervisor por un ID
/// </summary>
/// <param name="id">ID que intentaremos de encontrar en la DB</param>
/// <returns>Retorna el Supervisor encontrado por ID</returns>
/// <exception cref="DataBasesException">Lanzara la excepcion en caso de que haya un error con la DB</exception>
6 references
public T LeerPorID(int id)...
```

```
/// <summary>
/// Leemos y devolvemos una Lista de los Supervisores (con ciertos datos) que se encuentren en la DB
/// </summary>
/// <param name="dato">Cargo del Usuario</param>
/// <returns>Retorna la Lista</returns>
/// <exception cref="DataBasesException">Lanzara la excepcion en caso de que haya un error con la DB</exception>
3 references | 1/1 passing
public static List<Supervisor> LeerSupervisores(string dato)...
```

Modificar datos de Usuario

```
/// <summary>
/// Modificamos datos del Usuario seleccionado
/// </summary>
/// <param name="operario">Operario a modificar</param>
/// <returns>Retorna un true o false para verificar que se modifico el Usuario con EXITO</returns>
/// <exception cref="DataBasesException">Lanzara la excepcion en caso de que haya un error con la DB</exception>
2 references | 1/1 passing
public static bool ModificarUsuario(Operario usuario)...
```

Eliminar Usuario

```
286     /// <summary>
287     /// Eliminar un Usuario de la DB a travez de su ID y Cargo
288     /// </summary>
289     /// <param name="id">ID del Usuario</param>
290     /// <param name="dato">Cargo del Usuario</param>
291     /// <returns>Retorna un true o false para verificar que se elimino el Usuario con EXITO</returns>
292     /// <exception cref="DataBasesException">Lanzara la excepcion en caso de que haya un error con la DB</exception>
        2 references | 3/3 passing
293     public static bool Eliminar(string dato, int id)...
```

+ Delegados

```
14      public delegate void Mostrar(string texto, string titulo);
```

Utilizamos un delegado para poder pasar informacion por un metodo de un formulario a otro y poder mostrar un mensaje por pantalla, siendo una muetra de Informacion o Error.

```
201         Mostrar mostrarError = new Mostrar(FrmLobby.MostrarError);
202         Mostrar mostrarInformacion = new Mostrar(FrmLobby.MostrarInformacion);
```

Metodos que realizan las tareas

```
273     /// <summary>
274     /// Mostramos un mensaje de Error con ciertos datos
275     /// </summary>
276     /// <param name="texto">Texto que tendra el mensaje</param>
277     /// <param name="titulo">Titulo que tendra el mensaje</param>
        13 references
278     public static void MostrarError(string texto, string titulo)...
282
283     /// <summary>
284     /// Mostramos un mensaje de Informacion con ciertos datos
285     /// </summary>
286     /// <param name="texto">Texto que tendra el mensaje</param>
287     /// <param name="titulo">Titulo que tendra el mensaje</param>
        22 references
288     public static void MostrarInformacion(string texto, string titulo)...
```

+ Eventos

Tenemos eventos propios creados

```
23      public delegate void ManejadorEventos(int cantidadAgregar);
24      public event ManejadorEventos manejadorEventos;
```

Edonde cada vez que interactuo con la base de datos, ya sea generando un usuario, modificando o eliminando un usuario, modificando los productos o stock, cargo en un log de DB para tener un seguimiento de las interacciones que tengo con la base de datos.

```
448     /// <summary>
449     /// Invocamos el evento para modificar la cantidad de materiales en la DB
450     /// </summary>
451     /// <param name="cantidadAgregar">Cantidad a modificar</param>
        1 reference
452     public void ModificarCantidad(int cantidadAgregar)
453     {
454         Mostrar mostrarError = new Mostrar(FrmLobby.MostrarError);
455
456         if (this.manejadorProductos.Modificar(this.productoDB, cantidadAgregar, Stock.CasteoExplicito(t
457         {
458             FrmProcesos frmProcesos = new FrmProcesos((int)this.numFabricar.Value);
459             frmProcesos.Show();
460         }
461         else
462         {
463             mostrarError("No se pudo modificar los datos del material ingresado", "Error de carga");
464         }
465     }
```

Hilos, tenemos procesos en donde se muestra los primeros tres procesos Soldado, Ensamblado y Conectado de piezas para luego poder continuar al proceso final, Empaquetado del producto. Mostramos barras de progreso para ver el avance de los procesos.

```
47      private void IniciarHilos()
48      {
49          this.cancellationTokenSource = new CancellationTokenSource();
50          CancellationToken cancellationToken = this.cancellationTokenSource.Token;
51
52          Task primerTask = Task.Run(() => IniciarProceso(this.progressBarSoldado, this.lblSoldado, "Soldado de piezas"), ca
53          Task segundoTask = Task.Run(() => IniciarProceso(this.progressBarEnsamblado, this.lblEnsamblado, "Ensamblado de pi
54          Task tercerTask = Task.Run(() => IniciarProceso(this.progressBarConectado, this.lblConectado, "Conexion de compone
55
56          this.tasks.Add(primerTask);
57          this.tasks.Add(segundoTask);
58          this.tasks.Add(tercerTask);
59
60          Task.Run(() => Task.WaitAll(this.tasks.ToArray())).ContinueWith(_ =>
61          {
62              if (!cancellationTokenSource.IsCancellationRequested)
63              {
64                  Task cuartoTask = Task.Run(() => IniciarProceso(this.progressBarEmpaquetado, this.lblProcesoFinal, "Empaqu
65                  Task.Run(() => Task.WaitAll(cuartoTask)).ContinueWith(_ =>
66                  {
67                      MessageBox.Show("Procesos finalizados", "Finish", MessageBoxButtons.OK, MessageBoxIcon.Information);
68
69                      // Asegurarse de cerrar el formulario en el hilo de la interfaz de usuario
70                      this.Invoke((MethodInvoker)delegate
71                      {
72                          this.Close();
73                      });
74                  });
75              }
76          });
77      }
```