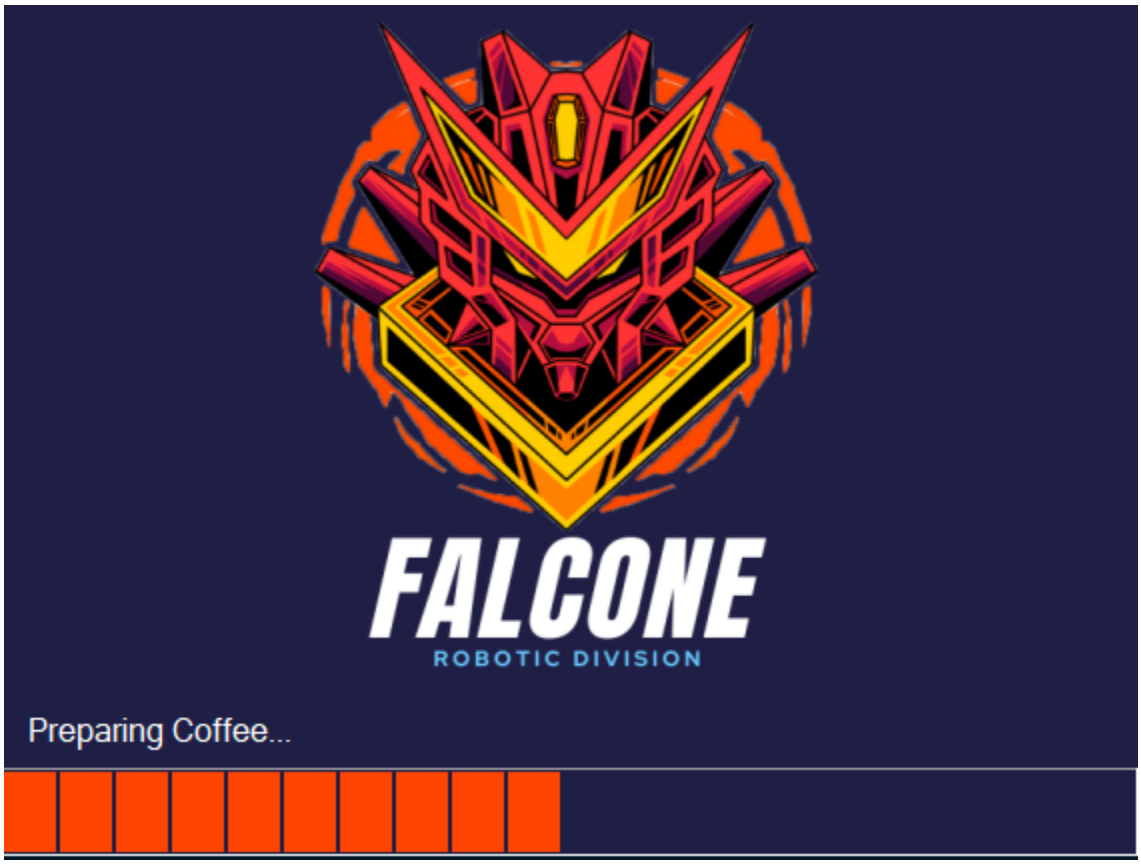
TP3 Final - Laboratorio 2

Amilcar Facundo Falcone

Fabrica de Robots

3 de Junio del 2021



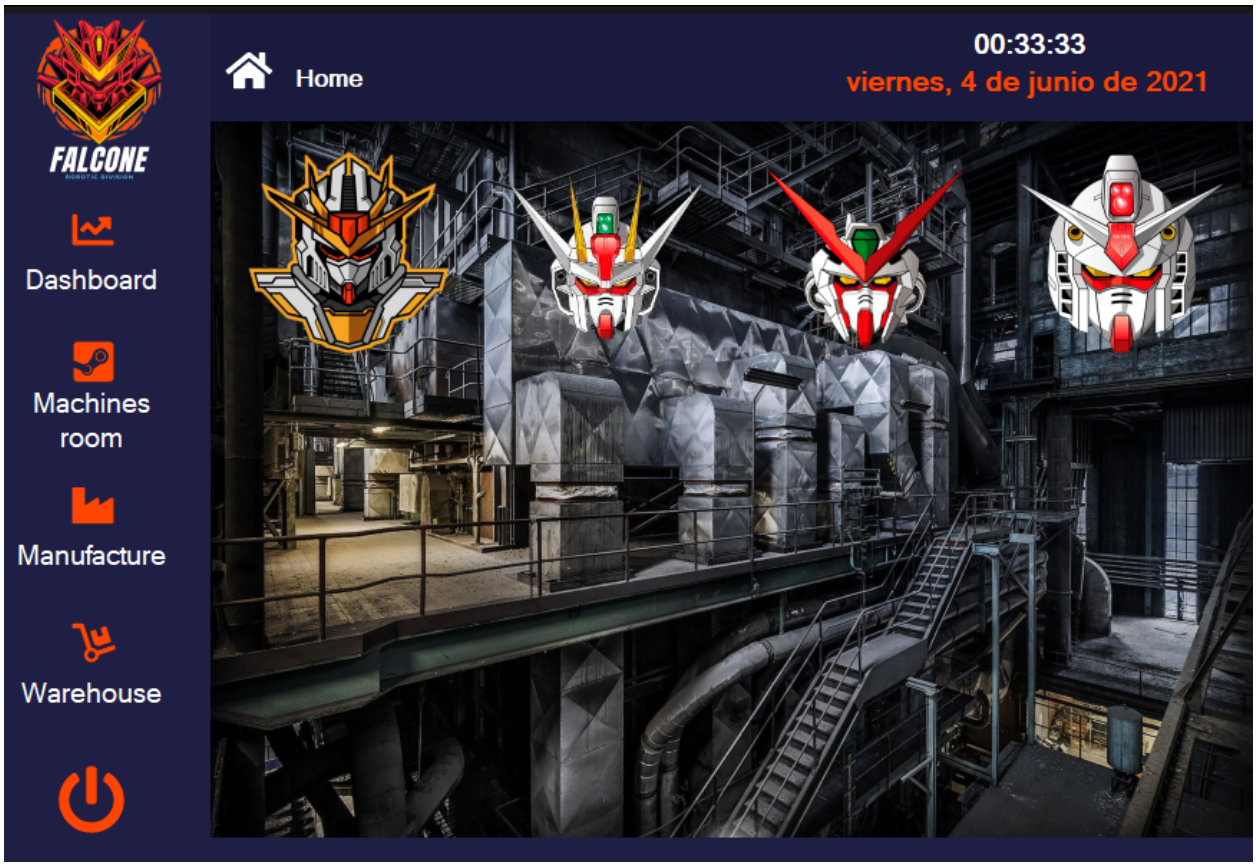
Explicación de la mini app

El siguiente proyecto simula una pequeña fábrica de robots en la que se podrán crear algunos diseños propuestos. Dicho programa para que funcione correctamente deberá inicializar primero su stock de materiales y de ser correcta su cantidad, con las que necesita el modelo robótico, procederá a fabricarlo para luego poner el robot en su almacén. En el almacén o warehouse, se podrá ver los modelos fabricados como así también el stock actual de materiales.

Pequeño tour por el TP3 - Youtube

Pequeño Lobby

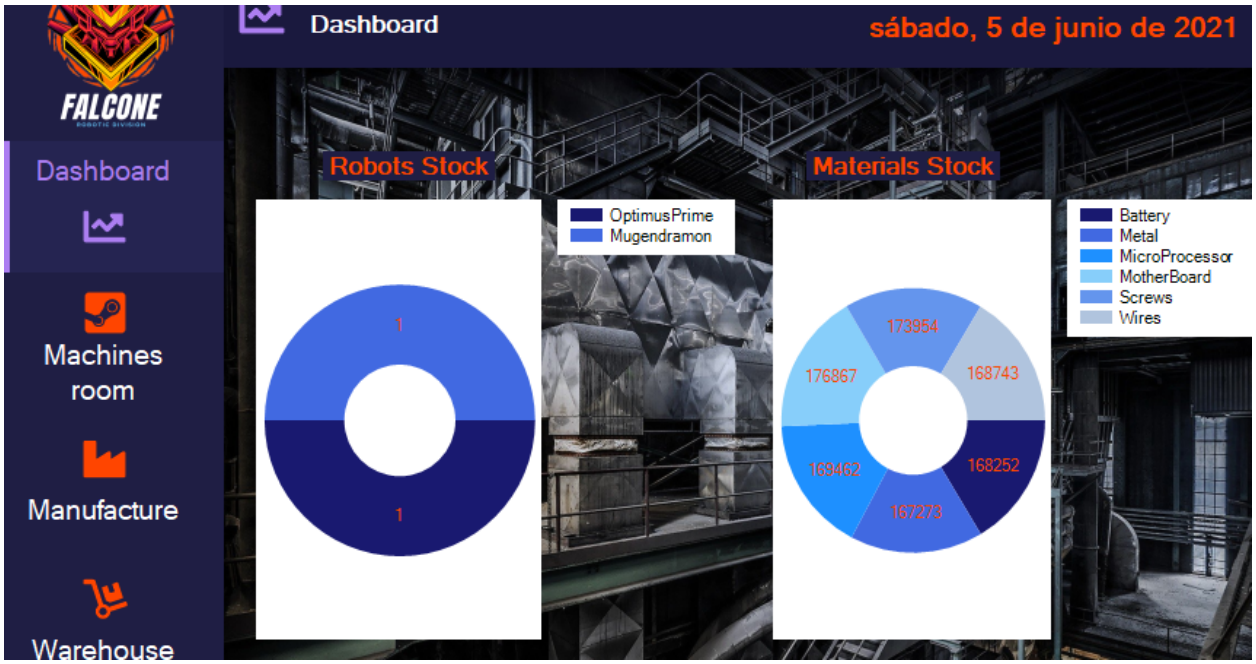
Amilcar Facundo Falcone - 2D



Descripción:

El programa te da la posibilidad de crear stock tanto de materiales como de robots, al consumir esos materiales.

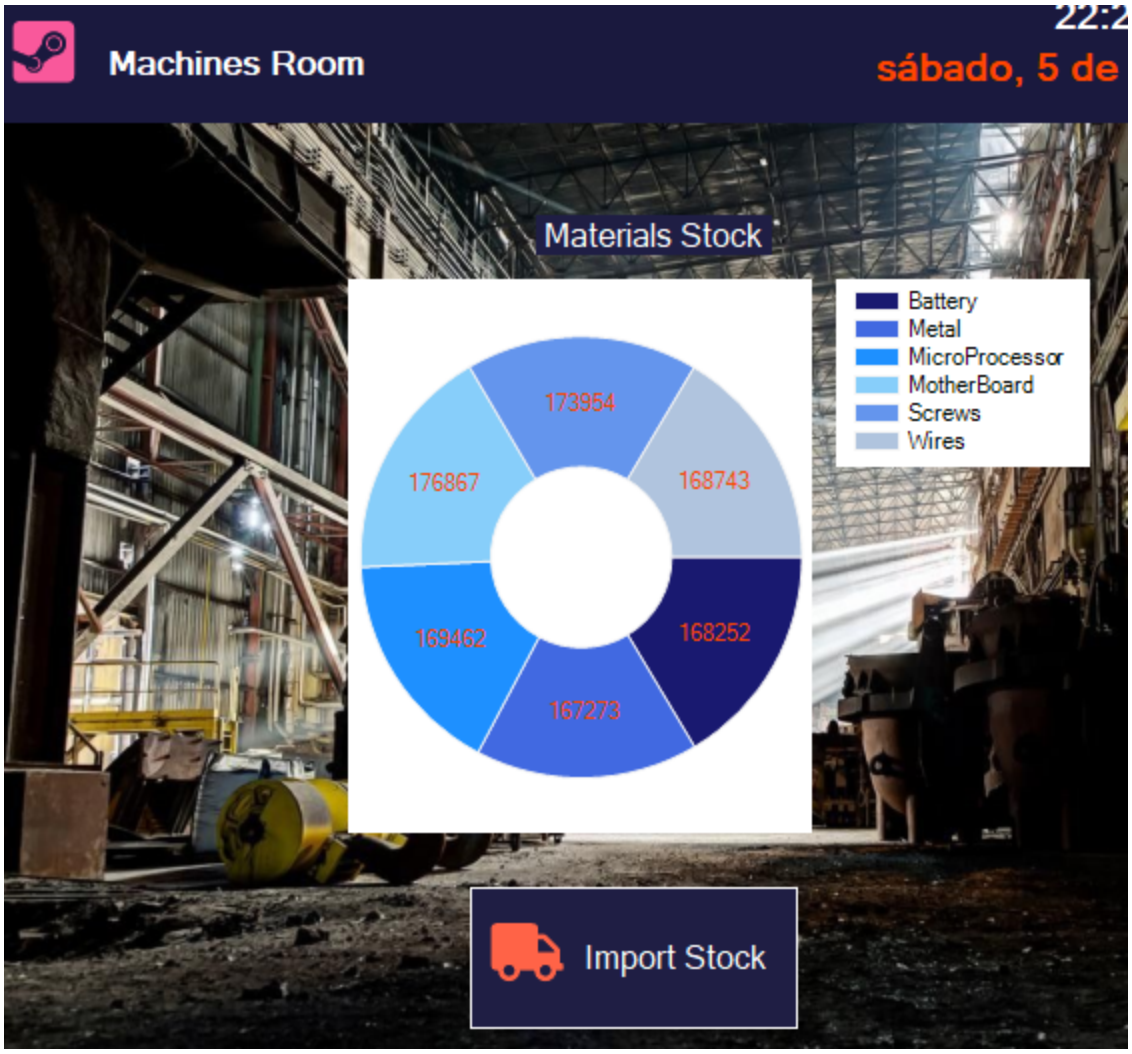
En el botón **'Dashboard'** podrán ver dos gráficos de dona con la cantidad y porcentaje de materiales y robots que hay en el stock de la fábrica, en caso de no haber robots y/o stock de materiales, no se visualizarán los gráficos.



Si es la primera vez que ejecutas el programa, Empeza por aca

En el botón '**Machines Room**' podrá inicializar la fábrica e importar materiales a la misma usando números random para aumentar el stock. La primera vez que ejecutes el programa, Acá aparecerán dos botones, uno de ellos será para inicializar la fábrica, al darle clic el botón desaparecerá para nunca más volver. Con el otro botón se podrá importar materia prima a la fábrica (se generan de manera random) y los valores se podrán visualizar en el gráfico de dona. Cada vez que se apriete el botón de '**Import Stock**', los valores del gráfico se actualizarán. Al iniciar el programa nuevamente (Con stock generado), El botón '**Initialize Factory**' no estará visible, ya que esta acción solo se puede hacer una vez. Se cargará un archivo serializado en el

que sus registros serán cargados en la memoria del programa.



En el botón **'Manufacture'** podrán crear hasta 9 tipos distintos de robots (en caso de que la cantidad de materiales del almacén sean suficientes, caso contrario tirara una excepción

mediante un formulario).

En el botón **‘Warehouse’** se podrá ver el stock de la fábrica, tanto de materiales como de robots, como así también una breve información sobre cada robot en el almacén.
El Logotipo ubicado en la esquina superior izquierda es un botón oculto que te lleva al ‘home’ de la app.

Temas Utilizados para el programa:

- *Excepciones*
- *Test Unitarios*
- *Tipos Genericos*
- *Interfaces*
- *Archivos*
- *Serialización*

Excepciones:

Implementado en varias secciones de los formularios, por ejemplo una de las implementadas es

```
14 referencias
public class InsufficientMaterialsException : Exception {

    #region Builders

    /// <summary>
    /// Creates an exception with the message.
    /// </summary>
    /// <param name="message">Message to show in the exception.</param>
    2 referencias
    public InsufficientMaterialsException(string message) : this(message, null) { }

    /// <summary>
    /// Creates an exception with the message and its innerException.
    /// </summary>
    /// <param name="message">Message to show in the exception.</param>
    /// <param name="innerException">InnerException of the exception.</param>
    1 referencia
    public InsufficientMaterialsException(string message, Exception innerException) : base(
```

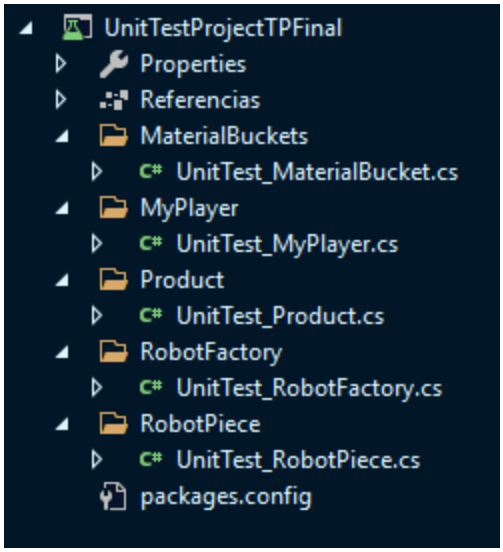
La cual será lanzada al querer fabricar un robot sin tener la cantidad necesaria de materiales en: frmManufacture/ConfigureRobotForBuild()

```
    prototype = RobotFactory.CreateMultiplePiecesAmdAddToStock(metalType, origin, modelName, amountOfMaterials, an
    filename = prototype.Model.ToString();
    absBioPath = $"{pathBiography}{filename}.bin";
    prototype.LoadBioFile(absBioPath);

    if (RobotFactory.QualityControl(prototype, amountPieces)) {
        if (RobotFactory.AddRobotToWarehouse(prototype)) {
            MyPlayer.Play($"BuildingForm", false);
            frmLobby.FormShowDialogHandler(new frmBuilding());
            RobotFactory.SaveDataOfFactory();
            MyPlayer.Play($"Create{prototype.Model}", false);
            frmLobby.FormShowDialogHandler(new frmISOCertified(prototype.Model.ToString()));
        }
    } else {
        if (RobotFactory.DissasembleRobot(prototype)) {
            throw new QualityControlFailedException("There have a difference in the amount of pieces, Quality cont
        }
    }
    prototype = null;
} else {
    throw new InsufficientMaterialsException("Insufficients Materials to build this model of robot, you need to in
}
```

Test Unitarios:

En el proyecto 'UnitTestProjectTPFinal' hay test unitarios para la mayoría de las clases, donde se testean métodos que hagan lo correspondiente.



Tipos Genericos:

Implementados en interfaces por un tema de practicidad, se podrán encontrar en el proyecto bibliotecas de clases 'Models/Classes/Interfaces'

```
1 referencia
public interface IFileManager<T, U> {

    /// <summary>
    /// Saves the data from the list of robots into a file in the path passed by
    /// </summary>
    /// <param name="path">Path to save the file.</param>
    /// <param name="file">List to save in the file.</param>
    /// <returns>True if can write the file, otherwise returns false.</returns>
    1 referencia
    bool SaveRobotFile(string path, List<T> file);

    /// <summary>
    /// Saves the data from the list of buckets into a file in the path passed b
    /// </summary>
```

Implementación en clase 'FileManager'.

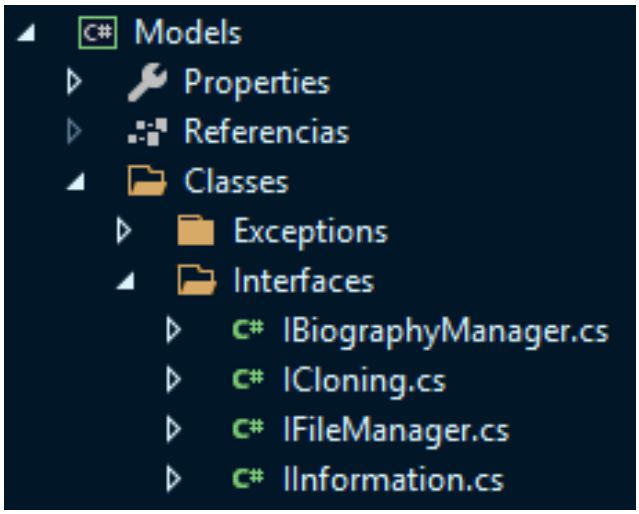
```
13 referencias
public class FileManager : IFileManager<Robot, MaterialBucket> {

    #region Builder

    /// <summary>
    /// Creates the entity with de the default constructor.
    /// </summary>
    5 referencias
    public FileManager() { }
```

Interfaces:

se podrán encontrar en el proyecto bibliotecas de clases ‘Models/Classes/Interfaces’



La gran mayoría de las interfaces son genéricas, por ejemplo:

```
1 referencia
public interface ICloning<T>
    where T : RobotPiece {

    /// <summary>
    /// Clones the list passed by parameter and returns the clone.
    /// </summary>
    /// <param name="listToClone">List to clone.</param>
    /// <returns>The list clonned.</returns>
    2 referencias
    List<T> CloneList(List<T> listToClone);

}
```

Implementadas en clases como ‘Robot’

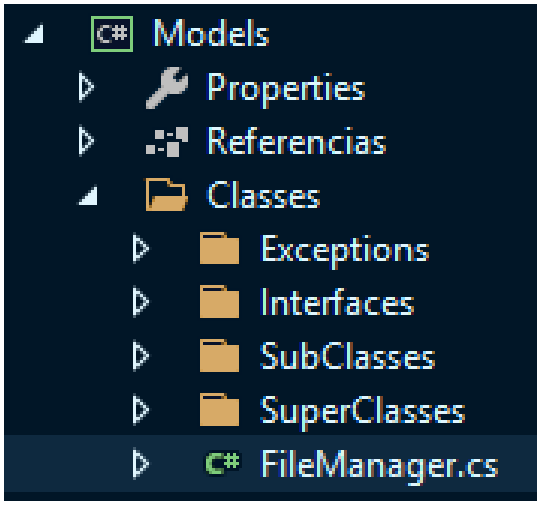
```
43 referencias
public class Robot : IInformation, ICloning<RobotPiece>, IBiographyManager {

    #region Attributes

    private int serialNumber;
    private EOrigin origin;
    private EModelName modelName;
    private bool isRideable;
    private string biography;
    private List<RobotPiece> pieces;
```

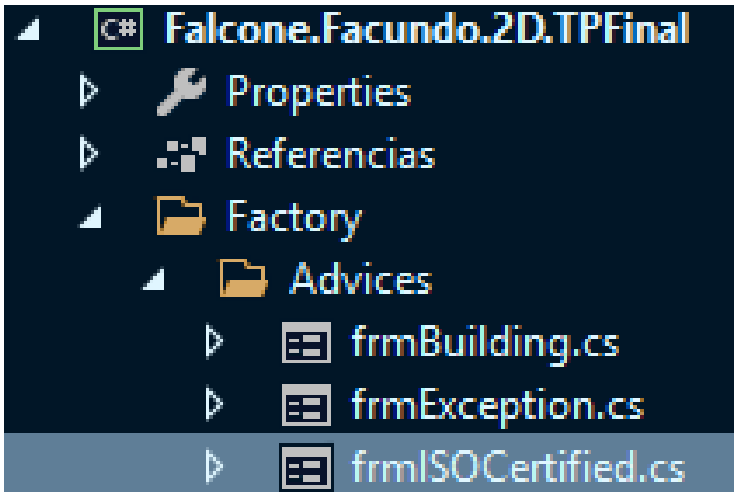
Archivos:

Usados para escribir en txt el historial de fabricación de robots, como así también los 'logs' de las excepciones. Se implementa una interfaz en una clase de administrador de archivos




```
/// <summary>
/// Saves the data from the Text into a file in the path passed by parameter.
/// </summary>
/// <param name="path">Path to save the file.</param>
/// <param name="filename">Name of the file with its extension '.txt'.</param>
/// <param name="text">Text to write in the file.</param>
/// <returns>True if can write the file, otherwise returns false.</returns>
3 referencias
public bool SaveDocument(string path, string filename, string text) {
    StringBuilder data = new StringBuilder();
    data.AppendLine($"{DateTime.Now} - {text}\n----------------------------");
    if (!Directory.Exists(path)) {
        Directory.CreateDirectory(path);
    }
    using (StreamWriter sw = File.AppendText($"{path}\\{filename}")) {
        sw.WriteLine(data);
        return true;
    }
}
```

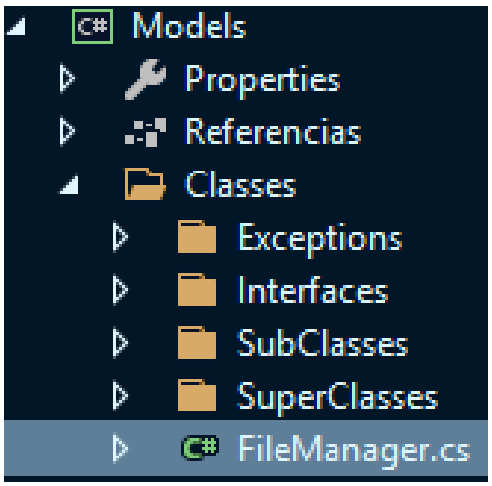
usados en formularios al realizar ciertas acciones




```
private void frmISOCertified_Load(object sender, EventArgs e) {
    this.Opacity = 0.0;
    this.pbCertified.Value = 0;
    this.pbCertified.Minimum = 0;
    this.pbCertified.Maximum = 100;
    this.timeFadeIn.Start();
    ipbImage.BackgroundImageLayout = ImageLayout.Zoom;
    ipbImage.BackgroundImage = Image.FromFile($"{this.systemImagePath}\\{this.productName}.png");
    warrantyMessage = WarrantyMessage();
    rtbWarrantyMessage.Text = warrantyMessage;
    fm.SaveDocument(path, filename, warrantyMessage);
}
```

Serialización:

Usados en Interfaces e implementado en la clase FileManager para serializar y deserializar archivos y listas.




```
/// <summary>
/// Reads the data from the path passed by parameter.
/// </summary>
/// <param name="path">Path to read the file.</param>
/// <returns>True if can read the file, otherwise returns false.</returns>
2 referencias
public List<MaterialBucket> ReadBucketsFile(string path) {
    List<MaterialBucket> aux;
    using (XmlTextReader reader = new XmlTextReader(path)) {
        XmlSerializer serial = new XmlSerializer(typeof(List<MaterialBucket>));
        aux = (List<MaterialBucket>)serial.Deserialize(reader);
    }

    return aux;
}
```