

**Friadenrich Agustín**

---

Trabajo Práctico Final

**Laboratorio de programación II**

Primer entrega

## Temas (con hipervínculos):

---

<b>EXPLICACIÓN:</b>	<b>3</b>
<b>EXPLICACIÓN DE LAS CLASES PRINCIPALES (UBICACIÓN: PROYECTO “ENTIDADES”):</b>	<b>3</b>
CONTAMINANTES.CS:	3
FÁBRICA.CS:	4
VEHÍCULO.CS:	4
IMPLEMENTACIÓN:	5
<b>FORMULARIOS (UBICACIÓN: PROYECTO “FORMULARIOS”):</b>	<b>5</b>
FORMINICIAL.CS:	5
FORMAGREGARCONTAMINANTES.CS:	6
FORMCARGARARCHIVO.CS:	6
FORMCONTAMINANTES.CS:	7
FORMAGREGARUNAGENTE:	8
FORMANALISISDEDATOS:	8
<b>CLASE 10: EXCEPCIONES. (UBICACIÓN: PROYECTO “EXCEPCIONES”)</b>	<b>9</b>
STRINGINVALIDOEXCEPTION:	9
NUMEROFUERADERANGOEXCEPTION:	9
CONTAMINANTENULOEXCEPTION:	9
TIPOINVALIDOEXCEPTION:	10
<b>CLASE 11: PRUEBAS UNITARIAS. (UBICACIÓN: PROYECTO “TESTUNITARIOS”)</b>	<b>10</b>
FABRICATEST:	10
GENERICLISTTEST:	11
<b>CLASE 12: TIPOS GENÉRICOS. (UBICACIÓN: ENTIDADES.GENERICLIST)</b>	<b>12</b>
GENERICLIST<T> WHERE T : CONTAMINANTES	12
IMPLEMENTACIÓN:	12
<b>CLASE 13: INTERFACES. (UBICACIÓN: ENTIDADES.IARCHIVOS)</b>	<b>13</b>
IMPLEMENTACIÓN:	13
<b>CLASE 14: ARCHIVOS Y SERIALIZACIÓN. (UBICACIÓN: ENTIDADES.GENERICLIST&lt;T&gt;)</b>	<b>13</b>
TRANSFORMARTODOAJSON():	13
GUARDARTODO(String PATH):	13
LEERJSON(String JSON):	13
LEERTODO(String PATH):	13

## Explicación:

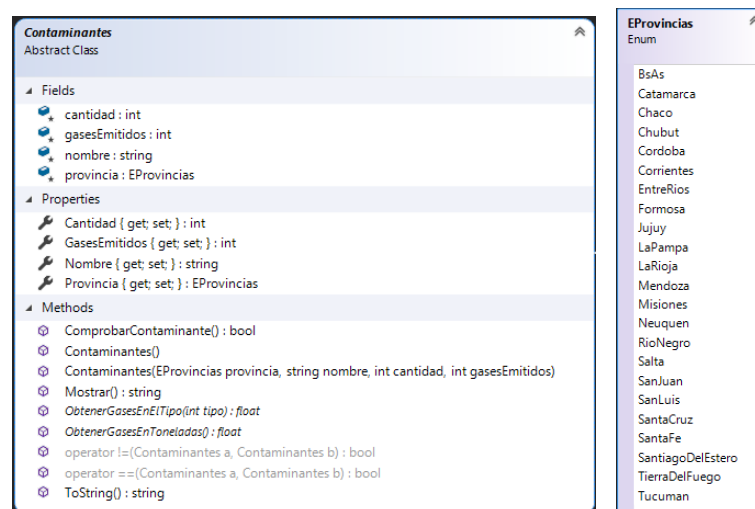
Un proyecto que se encarga del análisis de datos de dos tipos de agentes contaminantes ambientales: vehículos o fábricas que funcionan a base de la quema de combustibles fósiles.

El programa permite la carga de dos tipos de listas, ya sea a través de formularios o a través de la carga de archivos JSON: una de fábricas y otra de vehículos, y dónde a cada una se le pueden agregar una cantidad de elementos indeterminada

A cada vehículo/fábrica agregado al programa se le deben asignar los datos para: nombre de la fábrica o marca del vehículo, provincia en la que está ubicado/reside, cantidad del mismo, gases de CO2 que emite anualmente (las fábricas en toneladas anuales y los vehículos en kilogramos anuales, por la diferencia de contaminación), y el tipo.

## Explicación de las clases principales (Ubicación: Proyecto “Entidades”):

### Contaminantes.cs:



La clase abstracta Contaminantes es la clase padre de Fábrica y de Vehículo.

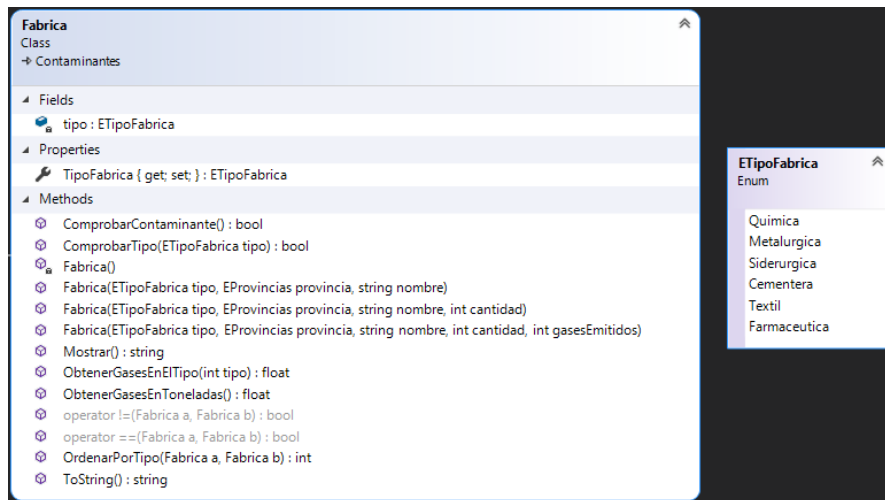
Esta clase le da forma a los agentes contaminantes, ya que establece los atributos para los gases contaminantes que emiten, así como su ubicación, cantidad y nombre.

Posee un método Mostrar() y una sobrecarga del ToString() para mostrar los datos de forma ordenada.

El método ComprobarContaminante comprueba que los datos sean válidos (Fundamentalmente a la hora de cargar archivos JSON).

El método ObtenerGasesEnToneladas() permite calcular la cantidad de gases contaminantes que emite el elemento. Es de tipo abstracto ya que la clase Fábrica cuenta en toneladas por defecto, pero la clase Vehículo cuenta en kilogramos, por lo que la funcionalidad varía en ambas.

## Fábrica.cs:

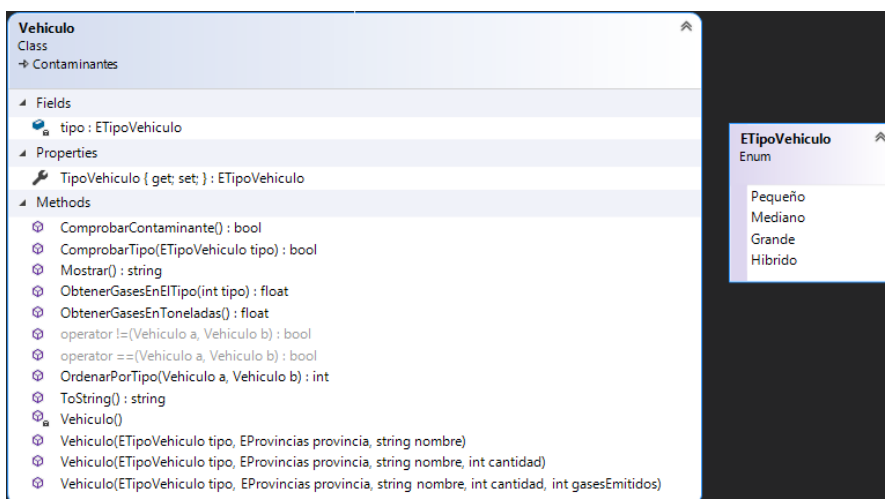


La clase Fábrica implementa los atributos y métodos de la clase Contaminantes, sumando el atributo de “tipo”, de tipo ETipoFabrica, un enumerado que posee diversas categorías de fábricas. Este enumerado, a su vez, contiene valores por defecto para la cantidad de gases contaminantes emitidos por cada componente del enumerado.

Además agrega el método de clase ComprobarTipo(ETipoFabrica tipo), que comprueba que un tipo esté contenido en el enumerado mencionado anteriormente, principalmente para verificar de forma más eficaz la carga de datos a través de archivos JSON.

Los constructores comprueban que el tipo ingresado sea válido y que la cantidad y los gases emitidos estén por debajo de un límite establecido (100 y 10.000.000 respectivamente).

## Vehículo.cs:



Al igual que su hermana la clase Fábrica, la clase Vehículo implementa los atributos y métodos de la clase Contaminantes, sumando el atributo de “tipo”, de tipo ETipoVehiculo, un enumerado que posee diversas categorías de vehículos. Este enumerado, a su vez, contiene valores por defecto para la cantidad de gases contaminantes emitidos por cada componente del enumerado.

Los constructores comprueban que el tipo ingresado sea válido y que la cantidad y los gases emitidos estén por debajo de un límite establecido (1.000.000 y 5000 respectivamente).

Además agrega el método de clase ComprobarTipo(ETipoVehiculo tipo), que comprueba que un tipo esté contenido en el enumerado mencionado anteriormente, principalmente para verificar de forma más eficaz la carga de datos a través de archivos JSON.

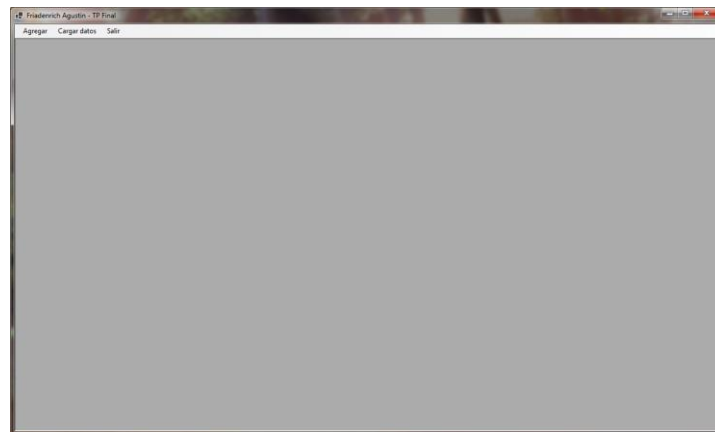
### Implementación:

La clase contaminantes y sus clases hijas están implementadas en:

- La clase GenericList<T>, donde T debe ser alguna de estas clases, y todas las implementaciones de esta clase (ver: “Clase 12: clases genéricas”).
- El test unitario de FabricaTest, donde se ponen a prueba todos los métodos y propiedades de la clase fábrica.

## Formularios (Ubicación: Proyecto “Formularios”):

### FormInicial.cs:



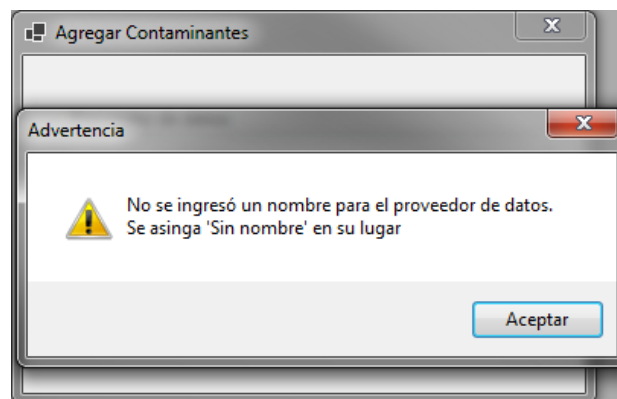
El form inicial es un contenedor MDI que servirá para alojar en él, o al menos ser el punto de partida, del resto de los formularios del programa. Posee un menuStrip que tiene opciones para:

1. Agregar: agregar un FormContaminantes a través del FormAgregarContaminantes.
2. Cargar datos: también para crear un FormContaminantes, pero esta vez con datos proveniente de la carga de archivos (A través del form FormCargarArchivo).
3. Salir de la aplicación.

### FormAgregarContaminantes.cs:

The 'Agregar Contaminantes' dialog box contains a text field for 'Proveedor de datos:', a dropdown menu for 'Tipo de agente contaminante:' with 'Ambos' selected, and 'Aceptar' and 'Cancelar' buttons. To the right, the 'EAgentes' Enum is shown with options: 'Ambos', 'Fabricas', and 'Vehiculos'.

Este formulario se instancia al utilizar la opción de “Agregar” del form inicial. Permite escribir el nombre del proveedor de datos, para guardar registro de quién es el que agrega los datos a las listas y/o archivos, al igual que los agentes contaminantes que se van a analizar en el programa, seleccionándolos desde un enumerado para facilitar la diferenciación.



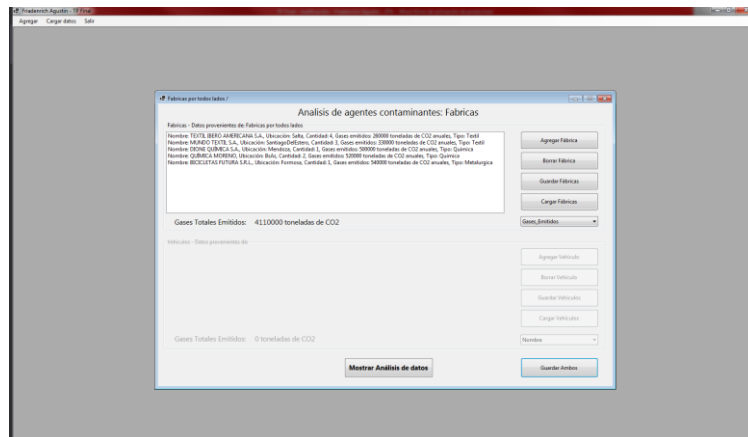
También posee un control de excepciones para corroborar que el texto del proveedor de datos no esté vacío, asignándole 'sin nombre' en ese caso.

### FormCargarArchivo.cs:

The 'CargarArchivo' dialog box features three buttons: 'Cargar Archivo con Fabricas', 'Cargar Archivo con Vehiculos', and 'Cargar Ambos archivos'. To the right, the 'EAgentes' Enum is shown with options: 'Ambos', 'Fabricas', and 'Vehiculos'.

Este formulario permite cargar uno o dos archivos, dependiendo de la selección, utilizados a la hora de instanciar un formulario de tipo FormContaminantes. Este formulario se encargará de obtener la o las listas del tipo seleccionado, junto con los nombres de sus proveedores.

FormContaminantes.cs:



El formulario más importante del programa. Este es el que permite la carga, modificación y guardado de datos, así como manejar el orden en el que se muestran y un botón para mostrar el análisis de los datos.

Los botones para agregar desplegarán el formulario FormAgregarUnAgente, y tendrán un control de excepciones para corroborar de que el resultado no haya sido nulo (Esto está a cargo de la clase GenericList<T>, concretamente en la sobrecarga del operador '+').

Los botones para borrar retirarán el elemento seleccionado en el listBox de la lista correspondiente. Si no hay ninguno seleccionado, simplemente se omitirá la instrucción.

Los botones para guardar permitirán guardar los datos de la lista correspondiente en un archivo de tipo JSON.

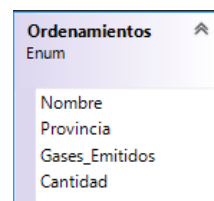
Los botones para cargar permitirán recuperar esos archivos JSON.

Los ordenamientos se realizan a través de un enumerado:

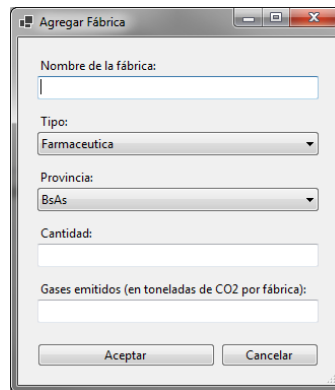
Ordena siempre de forma ascendente en alguno de los siguientes criterios:

Nombre, Provincia, Gases Emitidos, Cantidad y tipo.

Todos los métodos para ordenar están contenidos en la clase GenericList<T> excepto el ordenamiento por tipo, ya que varía dependiendo si el objeto es Fábrica o es Vehículo.



## FormAgregarUnAgente:



Formulario 'Agregar Fábrica' con los siguientes campos:

- Nombre de la fábrica:
- Tipo:
- Provincia:
- Cantidad:
- Gases emitidos (en toneladas de CO2 por fábrica):
- Botones: Aceptar, Cancelar

Este formulario permite la creación de un nuevo objeto de tipo Fabrica o Vehiculo, para agregarlo a la lista genérica correspondiente.

Usa el enumerado de EAgentes para comprobar qué agente es el que debe agregar.

Posee un control de **excepciones** para:

Nombre vacío -> no permite crear el agente

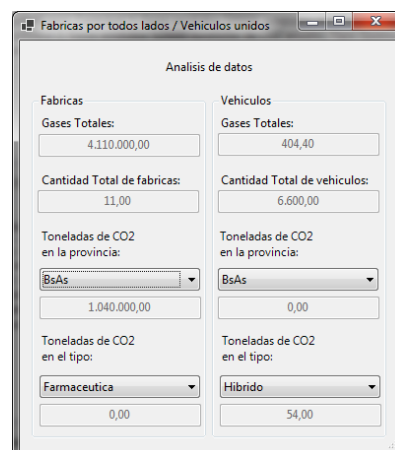
Cantidad menor o igual a 0 -> no permite crear el agente

Gases emitidos -> si es menor a 0 no permite crear el agente (En caso de no haber un número, asignará el valor por defecto dependiendo del tipo seleccionado).

En fábricas -> si la cantidad es mayor a 100 o los gases emitidos son mayores a 10.000.000

En vehículos -> si la cantidad es mayor a 1.000.000 o los gases emitidos son mayores a 5.000.

## FormAnalisisDeDatos:



Formulario 'Analisis de datos' dividido en secciones para Fábricas y Vehículos:

Fábricas	Vehículos
Gases Totales: <input type="text" value="4.110.000,00"/>	Gases Totales: <input type="text" value="404,40"/>
Cantidad Total de fábricas: <input type="text" value="11,00"/>	Cantidad Total de vehículos: <input type="text" value="6.600,00"/>
Toneladas de CO2 en la provincia: <input type="text" value="1.040.000,00"/> <input type="text" value="BsAs"/>	Toneladas de CO2 en la provincia: <input type="text" value="0,00"/> <input type="text" value="BsAs"/>
Toneladas de CO2 en el tipo: <input type="text" value="0,00"/> <input type="text" value="Farmaceutica"/>	Toneladas de CO2 en el tipo: <input type="text" value="54,00"/> <input type="text" value="Hibrido"/>

Este formulario sirve para mostrar todos los datos en un solo lugar. Calcula todo a través de los métodos y propiedades de la clase GenericList<T>.

Las propiedades CantidadTotal (get;) y GasesTotales (get;) calculan el total de esos atributos en cada lista. GasesTotales utiliza el método ObtenerGasesEnToneladas() de las clases Vehículo y Fabrica.



Los métodos `ContaminacionEnLaProvincia(EProvincia)` y `ContaminacionPorTipo(int tipo)` utilizan los métodos `ObtenerGasesEnToneladas()` y `ObtenerGasesEnTipo(tipo)` de las clases `Vehiculo` y `Fabrica` para calcular la cantidad de gases emitidos en la provincia o tipo seleccionados.

## Clase 10: Excepciones. (Ubicación: Proyecto “Excepciones”)

### `StringInvalidoException`:

Este tipo de excepcion es utilizada para corroborar que una cadena no esté vacía.

Esta excepción está implementada en:

- (throw) El constructor de la clase `Contaminantes`, donde valida que el atributo `nombre` no esté vacío.
- (Catch) En el `FormAgregarContaminantes` atrapa la excepción causada por dejar vacío el `textBox` Para el nombre y coloca en su lugar el texto "Sin nombre".
- (Catch) En el `FormAgregarAgente` atrapa la excepción causada por dejar vacío el `textBox` Para el nombre y evita que se cree un agente con el nombre vacío.
- (Catch) En el proyecto Test, en el `program`, donde se crea a propósito un `vehiculo` sin nombre para mostrar el manejo de excepciones.

### `NumeroFueraDeRangoException`:

Este tipo de excepcion es utilizada para corroborar que un dato numérico esté dentro de un rango válido.

Esta excepción está implementada en:

- (throw) El constructor de las clase `Contaminantes`, donde se lanza la excepción si el atributo `gasesEmitidos` es menor a 0 o el atributo `cantidad` es menor o igual a 0.
- En los constructores de las clases `vehiculo` y `fábrica`, donde se lanza la excepción si el atributo `gasesEmitidos` o el atributo `cantidad` superan un limite máximo determinado para cada clase.
- (Catch) Al agregar un contaminante en el `FormAgregarUnAgente`, para evitar que se creen elementos no válidos.
- (Catch) En el proyecto Test Unitario, como excepciones esperadas en algunos métodos de los test de `Fábrica` y `GenericList`.

### `ContaminanteNuloException`:

Este tipo de excepcion es utilizada para corroborar que los `Contaminantes` no sean iguales a `null`.

Esta excepción está implementada en:

- (throw) Los constructores de las clases `Contaminantes`, `Fabrica` y `Vehiculo`.
- (throw y catch) El método `ComprobarContaminante()` de las clases ya mencionadas
- (throw y catch) Sobrecarga de operadores de igualdad de las clases ya mencionadas y la sobrecarga del operador `+` en la clase `GenericList`
- (throw y catch) En la clase `GenericList`, al intentar leer un archivo que no tiene contenido.

### TipInvalidoException:

Se lanza si el tipo asignado a un contaminante no coincide con ninguno de los valores del enumerado para su tipo correspondiente

Esta excepción está implementada en:

- (Throw) En los constructores de las clases Vehículo y Fabrica, luego de comprobar si el tipo ingresado pertenece al enumerado
- (Catch) Se implementa en algunos Tests Unitarios, tanto en FabricaTest como en GenericListTest, al forzar el ingreso de un tipo no valido. Esto no sucede en el proyecto de formularios ya que siempre se asignan los tipos mediante los enumerados.

## Clase 11: Pruebas Unitarias. (Ubicación: Proyecto “TestUnitarios”)

### FabricaTest:

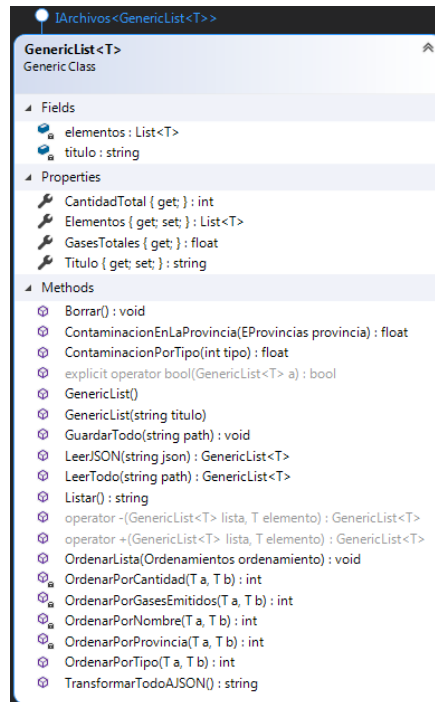
- FabricasSonIguales(): Comprueba que el operador == funcione correctamente.
- FabricasNombreDistinto(): Comprueba que el operador != devuelva false cuando solo el nombre es distinto.
- FabricasProvinciaDistinta(): Comprueba que el operador != devuelva false cuando solo la provincia es distinta.
- FabricasCantidadDistinta(): Comprueba que el operador != devuelva false cuando solo la cantidad es distinta.
- FabricasTipoDistinto(): Comprueba que el operador != devuelva false cuando solo el tipo es distinto.
- FabricasGasesEmitidosDistintos(): Comprueba que el operador != devuelva false cuando solo los gases emitidos son distintos.
- MostrarFabrica(): Comprueba que el método mostrar() cree el string correctamente.
- FabricaToString(): Comprueba que el método ToString() cree el string correctamente.
- VerificaFabricaCorrecta(): Verifica que el método ComprobarContaminante() retorne true si todos los parámetros son correctos.
- VerificarFabricaNula(): Intenta verificar con el método ComprobarContaminante() una fábrica nula, pero la excepción esperada es “NullReferneceException”, que es lanzada antes de invocar al método.
- VerificarFabricaErronea(): Intenta crear una fábrica con el tipo en 0, pero la excepción esperada es “TipInvalidoException”.
- FabricaCantidadInvalida(): Intenta crear una fábrica con un número negativo por cantidad, esto devuelve la excepción esperada “NumeroFueraDeRangoException”
- FabricaGasesEmitidosInvalidos(): Intenta crear una fábrica con un número negativo por gases emitidos, esto devuelve la excepción esperada “NumeroFueraDeRangoException”.
- ComprobarTipoValido(): Comprueba que diversos tipos sean válidos utilizando el método de clase ComprobarTipo(ETipoFabrica tipo).
- PropiedadGetTipoFabrica(): Verifica que la propiedad TipoFabrica devuelva el valor correctamente.
- PropiedadSetTipoFabrica(): Verifica que la propiedad TipoFabrica escriba el valor correctamente.

### GenericListTest:

- `AgregarElementoCorrecto()`: Comprueba que una lista contenga un elemento agregado a través del operador `+=`.
- `AgregarElementoConTipoInvalido()`: Intenta agregar un elemento a la lista que fue creado con un tipo no valido, esto espera recibir una excepción de tipo `"TipoInvalidoException"`.
- `AgregarElementoCantidadInvalida()`: Intenta agregar un elemento a la lista que fue creado con una cantidad menor o igual a 0, esto espera recibir una excepción de tipo `"NumeroFueraDeRangoException"`.
- `AgregarElementoGasesEmitidosInvalidos()`: Intenta agregar un elemento a la lista que fue creado con unos gases emitidos menores o iguales a 0, esto espera recibir una excepción de tipo `"NumeroFueraDeRangoException"`.
- `EliminarElementoValido()`: intenta eliminar de una lista elementos que no están contenidos en ella, por lo que los elementos que ya estaban contenidos seguirán ahí.
- `EliminarElementoNoValido()`: Elimina elementos de una lista que si están contenidos en ella y comprueba que ya no estén contenidos (a través de la sobrecarga de `-=`).
- `EliminarTodo()`: Verifica que todos los elementos de la lista hayan sido borrados correctamente con el método `borrar()`.
- `GuardarArchivoCorrecto()`: Verifica que un archivo se guarde correctamente, al leerlo y comprar el resultado con el string JSON de la lista.
- `GuardarYLeerArchivoCorrecto()`: Verifica que al guardar y volver a leer una lista, esta siga estando en el mismo estado que en el estado anterior a ser guardada.

## Clase 12: Tipos Genéricos. (Ubicación: Entidades.GenericList)

GenericList<T> where T : Contaminantes



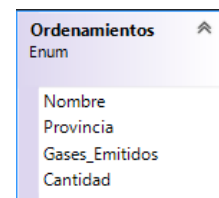
Esta clase genérica admite solo objetos del tipo Contaminantes y sus clases heredadas.

Posee la implementación de la interfaz IArchivos, la cual la utiliza para transformar a una cadena JSON sus datos, guardarlos y recuperarlos a través de la deserialización (Ver Clase 14: archivos).

Posee propiedades y métodos para recuperar datos generales de los objetos de la lista, como la cantidad total de gases emitidos (propiedad: GasesTotales {get;}), la cantidad total de contaminantes a través de su atributo cantidad (propiedad: CantidadTotal{get;}) y los gases emitidos totales de una provincia seleccionada (Método: ContaminaciónEnLaProvincia(EProvincias provincia)).

Un método borrar, que simplemente vacía los elementos de la lista genérica.

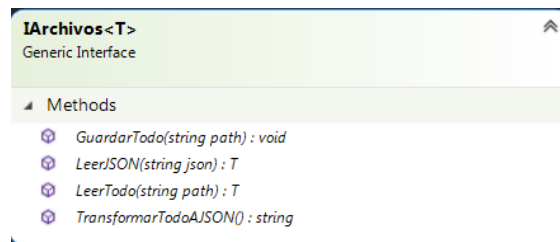
Un método de clases OrdenarLista, que recibe como parámetro un elemento del enumerado de ordenamientos. A este método se le acoplan otros cuatro, los cuales se encargan de los ordenamientos dependiendo de cuál sea el parámetro (Cantidad, Gases Emitidos, Nombre, Provincia).



### Implementación:

- En el proyecto Formularios, es uno de los componentes principales y está contenido en casi todos los formularios. Se encarga de contener, mostrar, ordenar y calcular los datos que serán mostrados al usuario en listBoxs, labels o textBoxs.
- En el proyecto Test, para probar las funcionalidades de su clase y mostrarlas por consola.
- En el proyecto TestUnitarios, en el archivo GenericListTest, donde se ponen a prueba las funcionalidades de la mayoría de sus métodos y propiedades.

## Clase 13: Interfaces. (Ubicación: Entidades.IArchivos)



Es una interfaz sencilla destinada al guardado y lectura de archivos JSON.

TransformarTodoAJSON(): es un método pensado para crear una cadena JSON del objeto que llama a la función.

GuardarTodo(string path): este método está pensado para guardar en el path indicado los datos del objeto (llamando a la función TransformarTodoAJSON() para la conversión).

LeerJSON(string json): está pensado para encargarse de leer una cadena JSON y transformarla en un objeto del tipo asignado.

LeerTodo(string path): está pensado para encargarse de leer un archivo ubicado en el path seleccionado, utilizando el método LeerJSON(string json).

### Implementación:

Esta interfaz está implementada al final de la clase GenericList<T>, donde está el código para serializar y de serializar la lista genérica en archivos JSON.

## Clase 14: Archivos y serialización. (Ubicación: Entidades.GenericList<T>)

### TransformarTodoAJSON():

Utiliza la clase JsonConvert(), junto con su método SerializeObject para serializar de forma idéntica el objeto que llama a la función. Luego, devuelve el string generado.

### GuardarTodo(string path):

A través del StreamWriter crea o sobrescribe el archivo pasado en el path, en el que escribe el resultado de la función TransformarTodoAJSON().

### LeerJSON(string json):

Utiliza la clase JsonConvert(), junto con su método DeserializeObject para recuperar la lista de un string json. Lanza una excepción de tipo ContaminanteNuloException si el json es una cadena vacía.

### LeerTodo(string path):

A través de un StreamReader lee el archivo indicado en el path (siempre que no sea nulo) y deserializa la lista utilizando el método LeerJSON. Tiene un control de excepciones en caso de que LeerJSON lance la excepción de ContaminanteNuloException. En caso de que haya un error o el archivo no contenga una lista válida, borrará lo que haya leído y devolverá una lista vacía.

**Aclaración:** en caso de que sea necesario, hay archivos de ejemplo, válidos y no válidos en la carpeta “Archivos de ejemplo”, ubicadas tanto en la carpeta principal del proyecto.