

Friadenrich Agustín

Trabajo Práctico Final

Laboratorio de programación II

Primer entrega

Temas (con hipervínculos):

EXPLICACIÓN:	4
EXPLICACIÓN DE LAS CLASES PRINCIPALES (UBICACIÓN: PROYECTO “ENTIDADES”):	4
CONTAMINANTES.CS:	4
FÁBRICA.CS:	5
VEHÍCULO.CS:	5
IMPLEMENTACIÓN:	6
FORMULARIOS (UBICACIÓN: PROYECTO “FORMULARIOS”):	6
FORMINICIAL.CS:	6
FORMAGREGARCONTAMINANTES.CS:	7
FORMCARGARARCHIVO.CS:	7
FORMCONTAMINANTES.CS:	8
FORMAGREGARUNAGENTE:	9
FORMMODIFICARUNAGENTE:	10
FORMANALISISDEDATOS:	10
CLASE 10: EXCEPCIONES. (UBICACIÓN: PROYECTO “EXCEPCIONES”)	11
STRINGINVALIDOEXCEPTION:	11
NUMEROFUERADERANGOEXCEPTION:	11
CONTAMINANTENULOEXCEPTION:	11
TIPOINVALIDOEXCEPTION:	11
CLASE 11: PRUEBAS UNITARIAS. (UBICACIÓN: PROYECTO “TESTUNITARIOS”)	12
FABRICATEST:	12
GENERICLISTTEST:	12
CLASE 12: TIPOS GENÉRICOS. (UBICACIÓN: ENTIDADES.GENERICLIST)	14
GENERICLIST<T> WHERE T : CONTAMINANTES	14
IMPLEMENTACIÓN:	14
CLASE 13: INTERFACES. (UBICACIÓN: ENTIDADES.IARCHIVOS)	15
IMPLEMENTACIÓN:	15

<u>CLASE 14: ARCHIVOS Y SERIALIZACIÓN. (UBICACIÓN: ENTIDADES.GENERICLIST<T>)</u>	<u>15</u>
TRANSFORMARTODOAJSON():	15
GUARDARTODO(STRING PATH):	15
LEERJSON(STRING JSON):	15
LEERTODO(STRING PATH):	15
<u>CLASE 15: INTRODUCCIÓN A SQL.</u>	<u>16</u>
<u>CLASE 16: CONEXIÓN A BASES DE DATOS. (UBICACIÓN: ENTIDADES.SQLEXTENSION).</u>	<u>17</u>
<u>CLASE 17: DELEGADOS Y EXPRESIONES LAMBDA. (UBICACIÓN: FORMULARIOS.FORMINICIAL Y FORMULARIOS.FORMCONTAMINANTES)</u>	<u>18</u>
DELEGADOS:	18
EXPRESIONES LAMBDA O FUNCIONES FLECHA:	18
<u>CLASE 18: HILOS (UBICACIÓN: FORMULARIOS.FORMCONTAMINANTES)</u>	<u>18</u>
<u>CLASE 19: EVENTOS (UBICACIÓN: FORMULARIOS.FORMINICIAL Y FORMULARIOS.FORMCARGARARCHIVOS)</u>	<u>19</u>
<u>CLASE 20: MÉTODOS DE EXTENSIÓN (UBICACIÓN: ENTIDADES.SQLEXTENSION)</u>	<u>20</u>

Explicación:

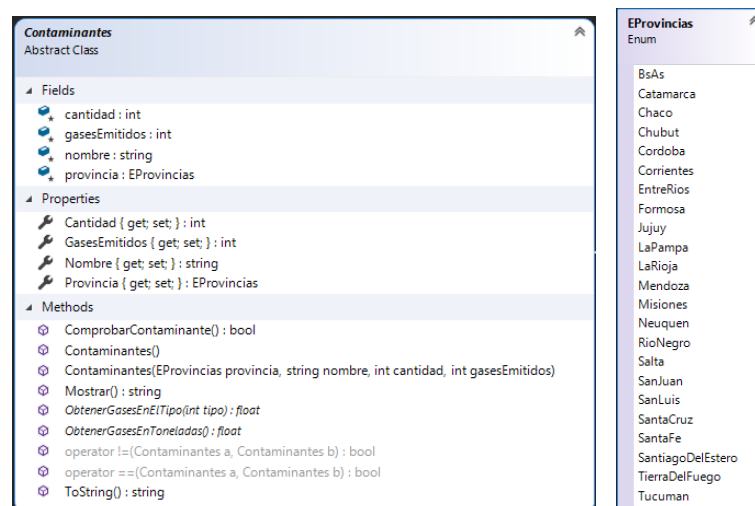
Un proyecto que se encarga del análisis de datos de dos tipos de agentes contaminantes ambientales: vehículos o fábricas que funcionan a base de la quema de combustibles fósiles.

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.

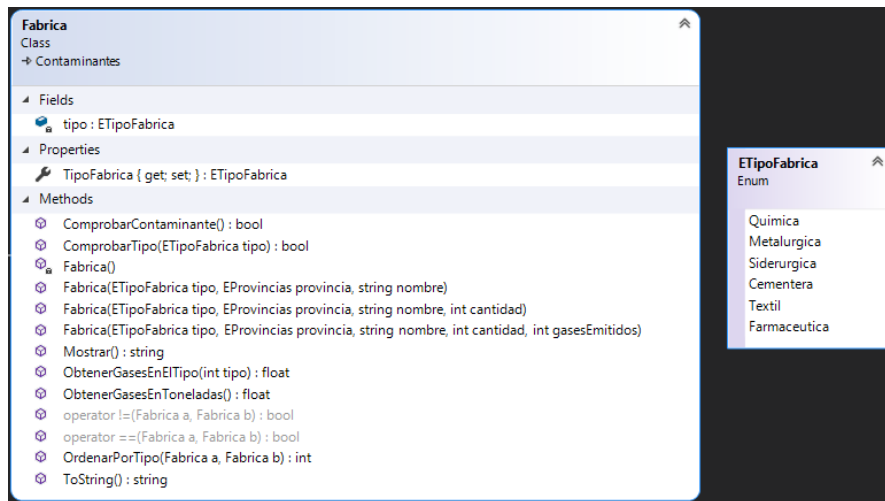
Esta clase le da forma a los agentes contaminantes, ya que establece los atributos para los gases contaminantes que emiten, así como su ubicación, cantidad y nombre.

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:



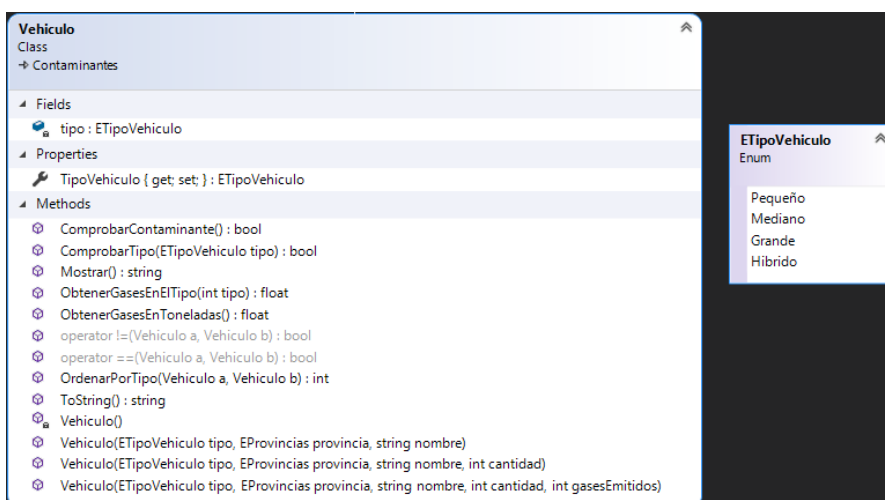
La clase Fábrica implementa los atributos y métodos de la clase Contaminantes, sumando el atributo de “tipo”, de tipo ETipofabrica, un enumerado que posee diversas categorías de fábricas. Este enumerado, a su vez, contiene valores por defecto para la cantidad de gases contaminantes emitidos por cada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

El parámetro cantidad se refiere a la cantidad de fábricas que hay con ese nombre en la provincia. Ejemplo: nombre=“Grido” – Provincia = “Chaco” – Cantidad = 2. Entonces, hay dos fábricas de grido en Chaco.

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.

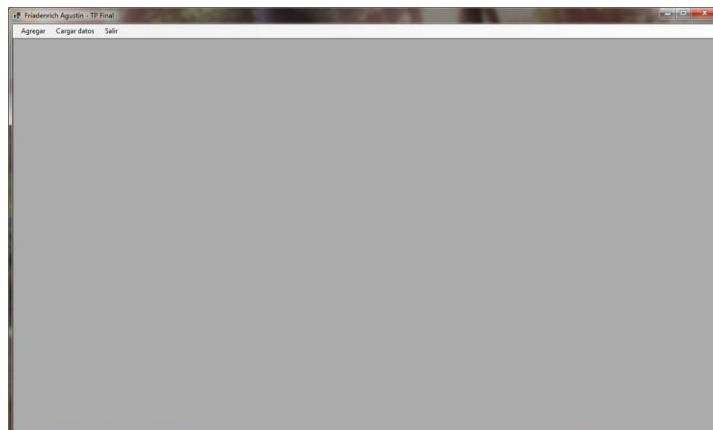
Implementación:

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:

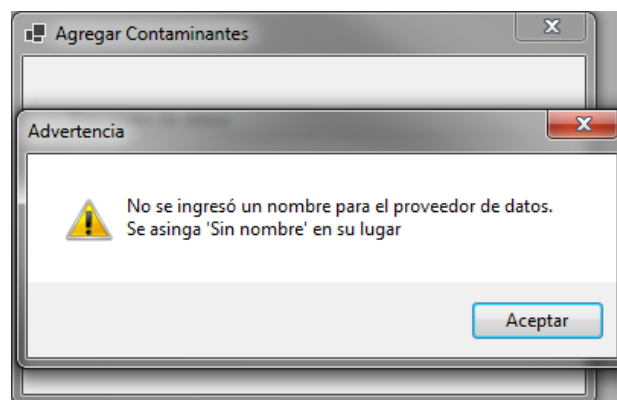
1. Agregar: agregar un `FormContaminantes` a través del `FormAgregarContaminantes`.
2. Cargar datos: también para crear un `FormContaminantes`, pero esta vez con datos proveniente de la carga de archivos (A través del form `FormCargarArchivo`).
3. Salir de la aplicación.

También posee un evento y un manejador de eventos, los cuales se encargan de capturar cuando se está cargando una lista a la base de datos u obteniéndola, y le dice al usuario cuánto puede llegar a tardar esta acción.

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 values: '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.



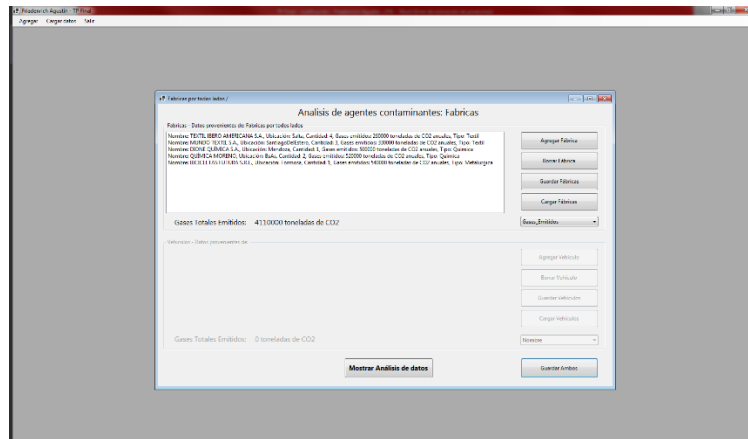
También posee un control de excepciones para corroborar que el texto del proveedor de datos no esté vacío, asignándole 'sin nombre' en ese caso.

FormCargarArchivo.cs:

The 'CargarArchivo' dialog box features three buttons: 'Cargar Archivo con Fabricas', 'Cargar Archivo con Vehiculos', and 'Cargar Ambos archivos'. An exception message dialog box is overlaid on top, stating: 'Agregue un archivo que contenga vehiculos.' with an 'Aceptar' button.

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 '+'). Solo se agregará un agente si no existe en la lista con el mismo nombre y provincia. A su vez, cargará el agente a la base de datos SQL, a su tabla correspondiente, si y solo si no existe un agente en ella que tenga el mismo nombre, provincia y nombre del proveedor (El proveedor = Titulo de la lista genérica).

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. También borrará el elemento de la base de datos, si es que este estaba cargado en ella.

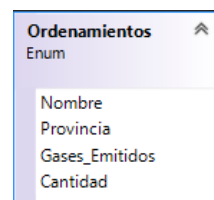
Los botones para modificar desplegarán el formulario FormModificarUnAgente, el cual permitirá modificar los datos del agente seleccionado en la listBox correspondiente. Solo modificará los datos del tipo, cantidad y gasesEmitidos. El nombre y la provincia se quedan iguales. Esto también sirve para que, al momento de realizar el update del agente, solo se modifique aquel elemento que en la base de datos tenga el mismo nombre, provincia y proveedor.

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, y a su vez enviar los datos recibidos al servidor JSON.

Los ordenamientos se realizan a través de un enumerado:

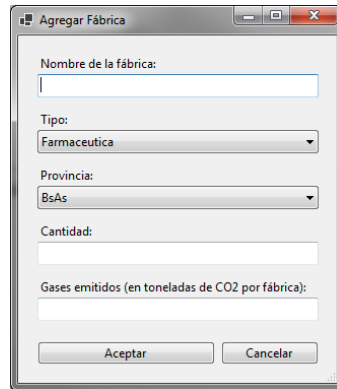
Ordena siempre de forma ascendente en alguno de los siguientes criterios:



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:



Este formulario permite la creación de un nuevo objeto de tipo Fábrica o Vehículo, 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

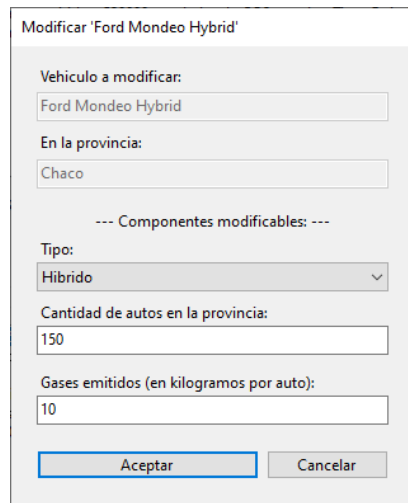
Cantidad menor o igual a 0 -> no permite crear el agente

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.

FormModificarUnAgente:



Modificar 'Ford Mondeo Hybrid'

Vehículo a modificar:
Ford Mondeo Hybrid

En la provincia:
Chaco

--- Componentes modificables: ---

Tipo:
Hibrido

Cantidad de autos en la provincia:
150

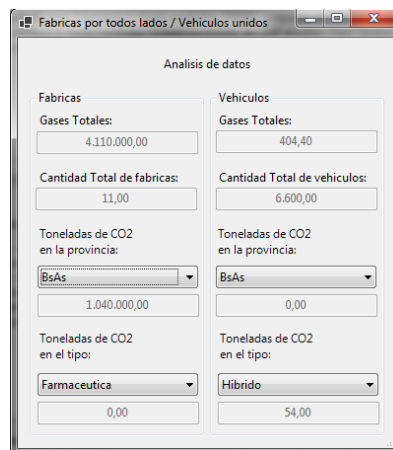
Gases emitidos (en kilogramos por auto):
10

Aceptar Cancelar

Permite modificar los datos de un agente, excepto el nombre y la provincia (Ya que estos dos son los que lo hacen único). Los datos del formulario se cargarán dinámicamente dependiendo de si el agente es una fábrica o un vehículo.

Una vez que se le da al botón aceptar, la propiedad de contaminante asignará los nuevos valores para poder ser recuperados en el FormContaminatnes, y cargarlos a las listas y a la DB.

FormAnalisisDeDatos:



Fabricas por todos lados / Vehiculos unidos

Analisis de datos

Fabricas	Vehiculos
Gases Totales: 4.110.000,00	Gases Totales: 404,40
Cantidad Total de fabricas: 11,00	Cantidad Total de vehiculos: 6.600,00
Toneladas de CO2 en la provincia: BsAs 1.040.000,00	Toneladas de CO2 en la provincia: BsAs 0,00
Toneladas de CO2 en el tipo: Farmaceutica 0,00	Toneladas de CO2 en el tipo: Hibrido 54,00

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 `Vehículo` 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.

TipoInvalidoException:

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 (nombre y provincia son iguales).
- FabricasNombreDistinto: Comprueba que el operador != devuelva true cuando solo el nombre es distinto.
- FabricasProvinciaDistinta(): Comprueba que el operador != devuelva true 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 “TipoInvalidoException”.
- 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.

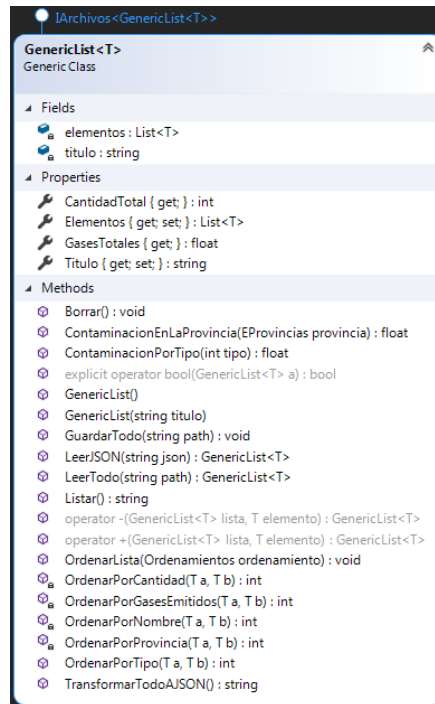
GenericListTest:

- 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



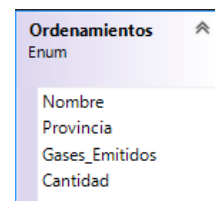
Esta clase genérica admite solo objetos del tipo Contaminantes y sus clases heredadas.

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.

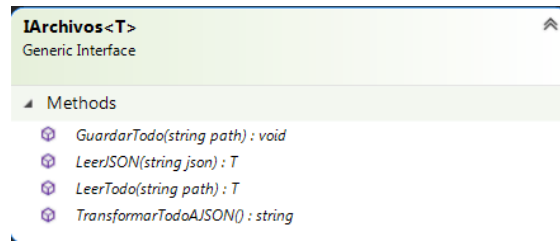
Un método de clases OrdenarLista, que recibe como parámetro un elemento del enumerado de ordenamientos. A este método se le acoplan otros cuatro, los cuales se encargan de los ordenamientos dependiendo de cuál sea el parámetro (Cantidad, Gases Emitidos, Nombre, Provincia).



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 indentada 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.

Clase 15: Introducción a SQL.

La base de datos creada para este programa se llama “FriadenrichAgustin” para facilitar la corrección.

Posee dos tablas: “dbo.fabricas” y “dbo.vehiculos”.

Ambas tienen las mismas columnas de datos, pero están separadas para poder diferenciar ambos agentes al momento de leerlas.

	Column Name	Data Type	Allow Nulls
►	tipo	int	<input type="checkbox"/>
	provincia	int	<input type="checkbox"/>
	nombre	varchar(100)	<input type="checkbox"/>
	cantidad	int	<input type="checkbox"/>
	gasesEmitidos	int	<input type="checkbox"/>
	proveedor	varchar(100)	<input type="checkbox"/>
			<input type="checkbox"/>

Los archivos de la base de datos están junto a este archivo en la carpeta principal del proyecto. Ante cualquier problema, dejo escritos los queries para crearlas.

```
USE [FriadenrichAgustin]
GO
```

```
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE TABLE [dbo].[fabricas](
    [tipo] [int] NOT NULL,
    [provincia] [int] NOT NULL,
    [nombre] [varchar](100) NOT NULL,
    [cantidad] [int] NOT NULL,
    [gasesEmitidos] [int] NOT NULL,
    [proveedor] [varchar](100) NOT NULL
) ON [PRIMARY]
GO
```

```
CREATE TABLE [dbo].[vehiculos](
    [tipo] [int] NOT NULL,
    [provincia] [int] NOT NULL,
    [nombre] [varchar](100) NOT NULL,
    [cantidad] [int] NOT NULL,
    [gasesEmitidos] [int] NOT NULL,
    [proveedor] [varchar](100) NOT NULL
) ON [PRIMARY]
GO
```


Dejo también el query para insertar algunas fabricas de ejemplo.

```
USE [FriadenrichAgustin]  
GO
```

```
INSERT INTO dbo.fabricas (tipo, provincia, nombre, cantidad, gasesEmitidos,  
proveedor) VALUES (650000, 7, 'BICICLETAS FUTURA S.R.L.', 1, 540000,  
'Analisis General')  
INSERT INTO dbo.fabricas (tipo, provincia, nombre, cantidad, gasesEmitidos,  
proveedor) VALUES (500000, 11, 'DIONE QUÍMICA S.A.', 1, 500000, 'Analisis  
General')  
INSERT INTO dbo.fabricas (tipo, provincia, nombre, cantidad, gasesEmitidos,  
proveedor) VALUES (200000, 20, 'MUNDO TEXTIL S.A.', 3, 330000, 'Analisis  
General')  
INSERT INTO dbo.fabricas (tipo, provincia, nombre, cantidad, gasesEmitidos,  
proveedor) VALUES (500000, 0, 'QUÍMICA MORENO', 2, 520000, 'Analisis  
General')  
INSERT INTO dbo.fabricas (tipo, provincia, nombre, cantidad, gasesEmitidos,  
proveedor) VALUES (200000, 15, 'TEXTIL IBERO AMERICANA S.A.', 6, 260000,  
'Analisis General')
```

Al momento de ejecutar el programa, al ir al a opción de “Agregar contaminante”, con escribir “Analisis General” (texto por defecto) recibirá en la aplicación estos ejemplos.

También, si usa alguna de las opciones de cargar archivos, y entra a la carpeta “Archivos de muestra” y elige uno de los archivos validos de ejemplo, estos se cargarán al programa y a la base de datos, para no tener que cargarlos uno a uno.

Clase 16: Conexión a bases de datos.

(Ubicación: Entidades.SQLExtension).

Esta clase posee el CRUD para elementos de tipo Contaminantes (Fábricas y Vehículos) y algunos métodos auxiliares, principalmente para evitar la repetición de código. Posee también 4 métodos de extensión a la clase Contaminante.

AgregarASQL(this Contaminante contaminante, string proveedor):

Este método sirve para insertar en la base de datos un contaminante junto con el nombre de su proveedor de datos (este parámetro siempre será dado por el título de la lista genérica que contiene al contaminante. Lo primero que hace este método es comprobar que el contaminante a agregar no exista en la base de datos a través del método ComprobarQueNoExista(string). Si no existe en la base de datos, entonces lo agrega. Cabe destacar que para que un contaminante “exista” debe tener el mismo nombre, provincia y proveedor.

ModificarASQL(this Contaminante contaminante, string proveedor):

Modifica en la base de datos al contaminante pasado como parámetro. Un contaminante con el mismo nombre, provincia y proveedor debe existir en la DB, entonces, se modificarán el resto de los datos (tipo, cantidad y gasesEmitidos).

EliminarASQL(this Contaminante contaminante, string proveedor):

Elimina de la DB el contaminante que tenga el mismo nombre, provincia y proveedor que el que llama a la función.

AgregarParametros(Contaminante contaminante, string proveedor, out string tabla):

Agrega todos los atributos del contaminante más el nombre del proveedor a los parámetros del SqlCommand de la clase. También, escoge a qué tabla irán los datos (dbo.fabricas o dbo.vehiculos) en función del tipo de dato que sea contaminante. A su vez, comprueba que sea un contaminante válido, para evitar agregar algún elemento no válido a la DB.

EjecutarQuery(string query, out bool resultado):

Abre y cierra la conexión a la base de datos, luego ejecuta el query, y por último borra los parámetros para evitar errores. El resultado se determina en función si logró o no modificar alguna fila.

ComprobarQueNoExista(this Contaminante contaminante, string proveedor):

Como fue explicado previamente, comprueba que el contaminante que llama a la función exista o no en la DB. Solo se revisará que coincidan el nombre, la provincia y el proveedor de datos.

Clase 17: Delegados y expresiones lambda. (Ubicación: Formularios.FormInicial y Formularios.FormContaminantes)

Delegados:

Hay un delegado en la clase FormInicial:

```
public delegate void EsperarConexion(object sender, TiempoAEsperarArgs args);
```

Este se encarga de contener al evento, también de la clase FormInicial:

```
void ConectandoConElServidorEvento(object sender, TiempoAEsperarArgs args)
```

Para la explicación del evento, ver Clase 19: Eventos.

Expresiones lambda o funciones flecha:

En FormContaminantes, línea 65, se lanza un Task que llama a la función ObtenerLista() Mediante una función flecha.

Otro ejemplo más claro es en las funciones GetListaFabricas y GetListaVehiculos del mismo form (Líneas 388 y 422), que poseen tasks que llaman a una función flecha, la cual agrega los elementos de la lista obtenida a la base de datos.

Clase 18: Hilos (Ubicación: Formularios.FormContaminantes)

Hay 3 tareas ejecutadas en hilos en el formulario FormContaminantes, que ejecutan las tareas mencionadas en las expresiones Lambda.

Línea 65: Task.Run(() => ObtenerLista()); Línea 92: task.Wait();

Funcionalidad: Llama a la función Obtener lista al principio de la función Load del formulario, y detiene el hilo principal si no terminó para el final de Load, antes de llamar a la función que Actualiza los datos (Ya que debería actualizarlos al tener la lista).

La función ObtenerLista hace lo siguiente:

1. Asignar el evento `ConcentandoConElServidorEvento` al delegado del `FormInicial`.
2. Dependiendo el tipo de agente, traerá las listas de vehículos y/o fabricas correspondientes de la base de datos SQL.
3. Llamará al manejador de eventos correspondiente de la clase `FormInicial`
4. Mostrará un mensaje con la cantidad de elementos obtenidos.

Clase 19: Eventos (Ubicación: Formularios.FormInicial y Formularios.FormCargarArchivos)

El evento creado en el `FormInicial` es el encargado de avisar al usuario que se están cargando u obteniendo listas completas de la base de datos SQL.

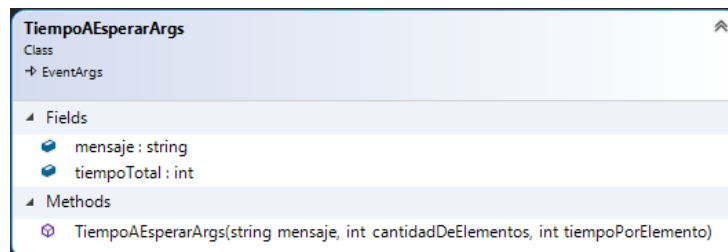
Funciona a través de un delegado que está en el form `FormInicial`:

```
public delegate void EsperarConexion(object sender, TiempoAEsperarArgs args);
```

Que es almacenado en una variable estática:

```
public static EsperarConexion esperarConexionDelegada;
```

El segundo parámetro del evento es de tipo `TipoAEsperarArgs`, que es una clase ubicada en el proyecto de Entidades y deriva de `EventArgs`.



Esta posee dos atributos: `mensaje` y `tiempoTotal`. El tiempo total refiere al tiempo máximo que puede tardar el programa en hacer la consulta a la base de datos. Al momento de enviar una lista al servidor, el tiempo máximo que puede demorar son 5 segundos por elemento, ya que eso es lo que tarda la conexión en esperar una respuesta del servidor.

Entonces, al constructor de la clase se lo llama con 3 parámetros:

El mensaje, el cuál puede ser algo como “Se obtendrán los vehículos de la base de datos en segundo plano” o “Se agregaran los vehículos del archivo a la base de datos en segundo plano.”. (Pone “en segundo plano” ya que estas acciones son ejecutadas desde `Tasks` para evitar interferir lo máximo posible con la ejecución del hilo principal).

El parámetro `cantidadDeElementos` refiere a los elementos de la o las listas que se cargaran a la base de datos. Ej: `cantidadDeElementos = 8`; si son 4 fábricas y 4 vehículos.

El parámetro `tiempoPorElemento` está para poder configurar cuánto tiempo es el aproximado que tardará la acción.

Entonces, `tiempoTotal` va a ser la cantidad de elementos * el tiempo por elemento.

El manejador de eventos es un método estático en la clase FormInical que tiene la siguiente firma: `public static void` ManejadorEsperarConexion(`object` sender, `string` mensaje, `int` cantidadDeElementos, `int` tiempoPorElemento)

Este se encarga de instanciar un objeto de tipo TiempoAEsperarArgs e invocar el evento ConectandoConElServidorEvento utilizando el objeto sender y el objeto mencionado anteriormente como parámetros, solo si es que el delegado no es nulo.

Por último, el evento `public static void` ConectandoConElServidorEvento(`object` sender, TiempoAEsperarArgs args)

El cuál mostrará el mensaje y el tiempo estimado a través de un message box. Si el tiempo estimado es 0, el mensaje mostrará que es “indeterminado”. (Ya que no se puede saber cuánto tardará en traer los objetos de la base de datos).

Clase 20: Métodos de extensión

(Ubicación: Entidades.SQLExtension)

Los métodos AgregarASQL AgregarSQL(this Contaminante contaminante, string proveedor),

ModificarASQL(this Contaminante contaminante, string proveedor),

EliminarASQL(this Contaminante contaminante, string proveedor) y

ComprobarQueNoExista(this Contaminante contaminante, string proveedor).

Todos de la clase Entidades.SQLExtension son métodos de extensión para la clase abstracta Contaminantes y sus derivadas.

(Ver Clase 16 – Conexión a bases de datos para leer su implementación).