**Informe del proyecto sobre desarrollo de un sistema que gestiona una red de estaciones de combustible.**

**Gestión dinámica a la red y sus estaciones utilizando el paradigma de programación orientada a objetos (POO).**

Autor: Martín Santiago Rodríguez Forero

1. **Análisis del problema:**

La empresa TerMax, lider en la industria de combustibles en Colombia requiere un sistema que permita una gestión eficiente de sus estaciones de combustible teniendo en cuenta diversos aspectos de las mismas que se mencionarán a continuación.

1. **Consideraciones para la alternativa de solución propuesta:**

* **Relaciones entre los objetos a gestionar:** Se evidencia que la empresa menciona una relación muy estrecha entre distintos elementos que deben ser tenidos en cuenta, tanto a la hora de realizar la codificación como para realizar los diagramas y esquemas solicitados.
* **Uso de POO:** Este permitirá una gestión más eficiente y cercana al mundo real para la solución de la problemática.
* **Uso de memoria dinámica:** Para la actualización del combustible dependiendo del caso, las modificaciones a los precios del combustible, la activación de surtidores de forma eficiente, entre otros.
* **Funcionalidades esenciales:** TerMax menciona varias funcionalidades que debe tener el programa bajo un menú con el que los usuarios puedan interactuar para realizar las diversas modificaciones que se requieran, las cuales son las siguientes:
* **Gestión de la red:**

- Agregar estaciones de servicio.

- Eliminar una E/S de la red nacional (sólo si no posee surtidores activos).

- Calcular el monto total de las ventas en cada E/S del país, discriminado por categoría de combustible.

- Fijar los precios del combustible para toda la red.

* **Gestión de estaciones de servicio:**

- Agregar/eliminar un surtidor a una E/S.

- Activar/desactivar un surtidor de una E/S.

- Consultar el histórico de transacciones de cada surtidor de la E/S.

- Reportar la cantidad de litros vendida según cada categoría de combustible.

- Simular una venta de combustible.

- Asignar la capacidad del tanque de suministro, con un valor aleatorio entre 100 y 200 litros para cada una de las categorías

* **Sistema nacional de verificación de fugas:**

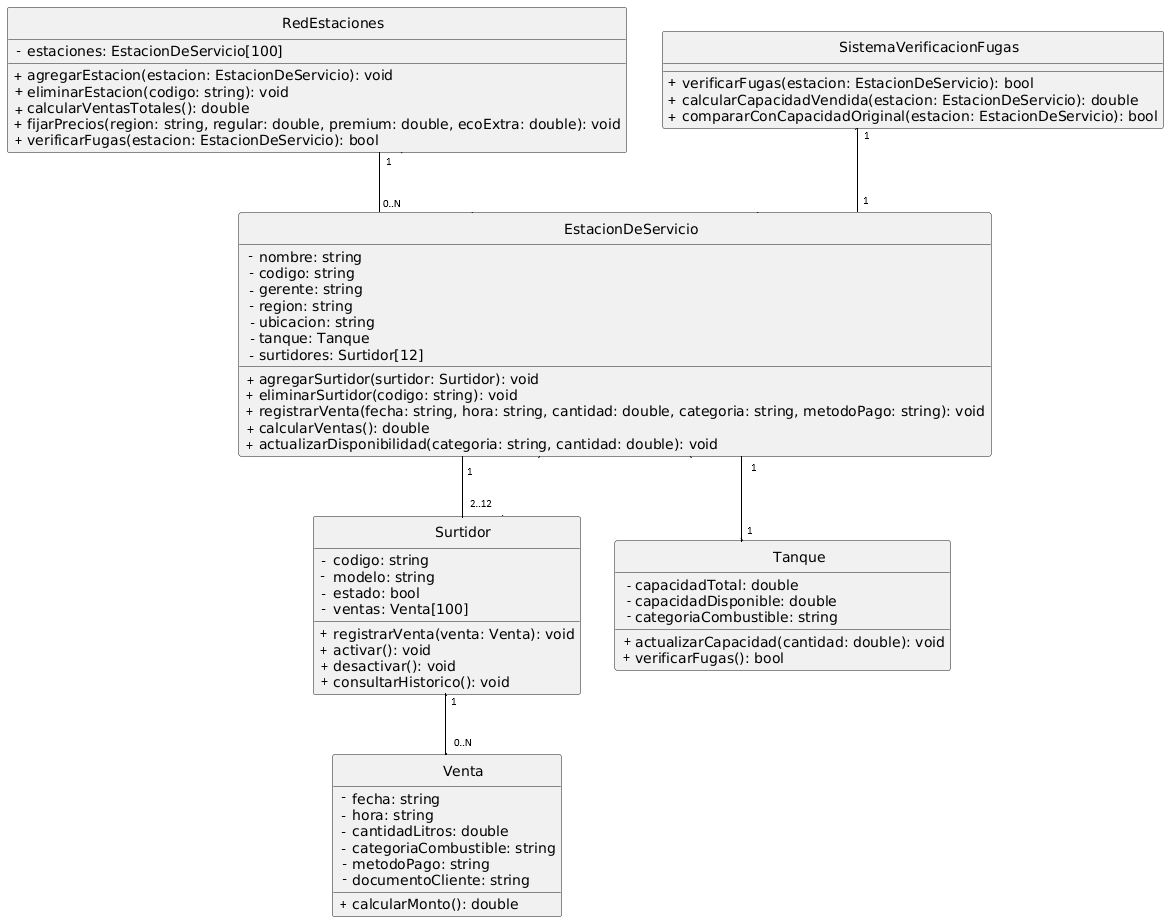
-El programa debe permitir detectar la existencia de fugas de combustible en cualquiera de las estaciones del país. Para ello, según cada categoría de combustible, debe verificarse que lo vendido más lo almacenado en el tanque de la E/S corresponda a más del 95% de la capacidad original del tanque. Esta verificación opera sobre una estación de servicio específica que sea seleccionada entre todas las del país.

* **Simulación de ventas:**

-Dada una E/S, asigne de forma aleatoria uno de los surtidores activos para que gestione la transacción. Suponga que en una venta se solicita en forma aleatoria entre 3 y 20 litros de gasolina. Una vez que se ha bombeado la gasolina deberán desplegarse los datos de la transacción. El precio por litro de cada categoría de combustible depende de la región a la que pertenece la estación. Sólo se consideran tres regiones: Norte, Centro y Sur. Dicho valor es temporal, pudiendo variar entre días.

* Por último se deben considerar subprogramas que gestionen todos los códigos de forma automática sin que estos impidan la interacción con los menús.

1. **Diagrama de clases de la solución planteada.**



### 3. Algoritmos Implementados

A continuación se describen los principales algoritmos implementados en cada clase, con la documentación adecuada para cada uno.

#### Clase EstacionDeServicio:

#### EstacionDeServicio::EstacionDeServicio(const std::string& nombre, const std::string& codigo, const std::string& gerente, const std::string& region, const std::string& ubicacion)

* **Descripción**: Constructor de la clase. Inicializa una nueva estación de servicio con los parámetros proporcionados.

- void EstacionDeServicio::agregarSurtidor(Surtidor\* surtidor):

* **Descripción**: Agrega un nuevo surtidor a la estación. Verifica si hay espacio para más surtidores.
* **Funcionamiento:**
  1. Comprobar si el número actual de surtidores es menor que el máximo permitido.
  2. Si es posible, agregar el surtidor y aumentar el contador.

- void EstacionDeServicio::eliminarSurtidor(const std::string& codigo)

* **Funcionamiento:**
* Iterar sobre el arreglo de surtidores.
* Comparar el código de cada surtidor con el código proporcionado.
* Si se encuentra, liberar la memoria y desplazar los elementos restantes.

- Surtidor\* EstacionDeServicio::buscarSurtidor(const std::string& codigo)

* **Descripción**: Busca un surtidor por su código.
* **Algoritmo**:
* Iterar sobre el array de surtidores.
* Comparar el código de cada surtidor con el código proporcionado.
* Retornar el surtidor si se encuentra; si no, retornar nullptr.

#### 3.2 Clase Surtidor

- Surtidor::Surtidor(const std::string& codigo, const std::string& modelo)

* **Descripción**: Constructor de la clase Surtidor. Inicializa un nuevo surtidor.

- void Surtidor::consultarHistorico()

* **Descripción**: Método para consultar el histórico de ventas del surtidor.

- bool Surtidor::verificarFugas()

* **Descripción**: Verifica si el surtidor tiene fugas.

#### 3.3 Clase RedEstaciones

- void RedEstaciones::agregarEstacion(EstacionDeServicio\* nuevaEstacion)

* **Descripción**: Agrega una nueva estación a la red de estaciones.
* **Algoritmo**:
* Comprobar si el número actual de estaciones es menor que el máximo permitido.
* Si es posible, agregar la estación y aumentar el contador.

- Surtidor\* RedEstaciones::buscarSurtidor(const std::string& codigo)

* **Descripción**: Busca un surtidor en todas las estaciones de servicio.
* **Algoritmo**:
* Iterar sobre cada estación en la red.
* Llamar a buscarSurtidor en cada estación.
* Retornar el surtidor si se encuentra; si no, retornar nullptr.

Es necesario aclarar que tanto el diagrama de clases como los algoritmos implementados en este primer informe pueden tener modificaciones tras la codificación y serán incluidos en el informe final.

También se añadirán dos capítulos más al informe final: Problemas de desarrollo afrontados junto a sus respectivas soluciones, y consideraciones a tener en cuenta para la implementación (Este último basado en el código final).