

UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERIA

INFORMATICA II  
  
  
PRESENTADO POR:

Cristopher corrales

Tomas Restrepo

Medellín, Colombia

**Análisis del problema:**

Para el ejercicio planteado propone hacer una interfaz en la que un usuario pueda crear una red de estaciones de gasolina con distintas opciones como en el siguiente menú:

1. **Gestión de Red:**

**A.**  Agregar Estación.

**B.** Eliminar Estación.

**C.**  Calcular el Monto Total de Ventas.

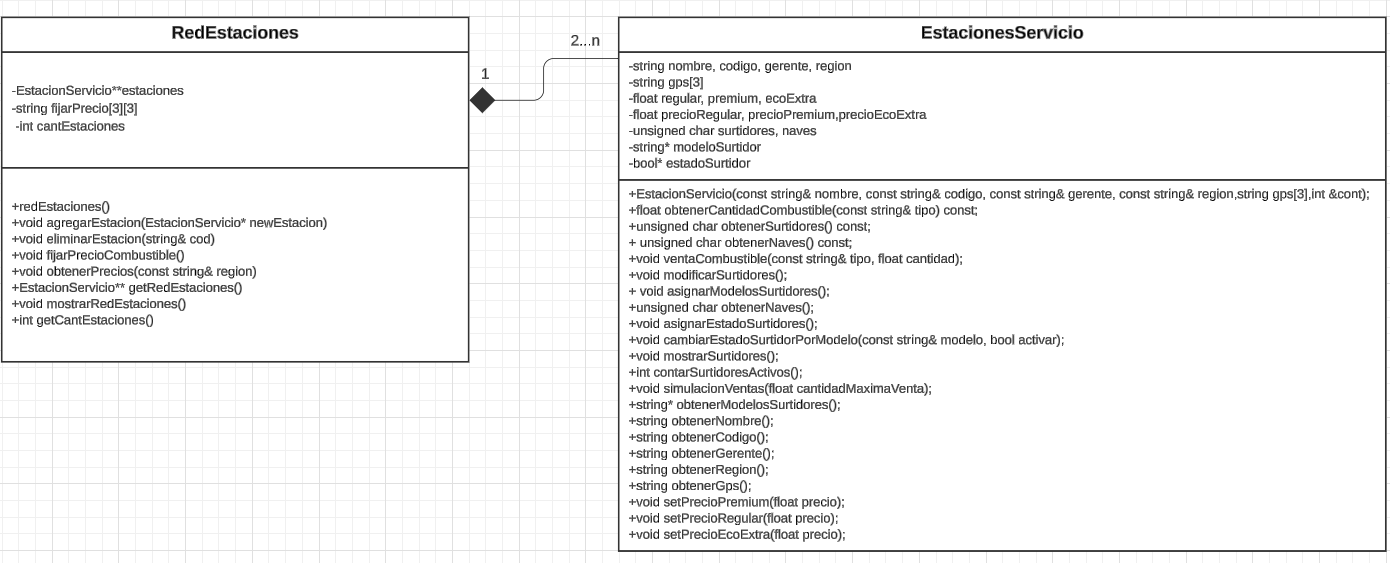
**D.** Mostrar Estaciones.

1. **Gestión de Estaciones de Servicio:**
2. **Sistema Nacional de Verificación de Fugas.**
3. **Simulación de Ventas.**

Las Estaciones de Servicio se manipularán a través de un arreglo dinámico de objetos. Una vez se inicia el programa creamos una estación aleatoria y pediremos que se fijen los precios para las regiones y el tipo de combustible, seguido a esto entraremos en un ciclo mostrando las opciones que tiene el usuario para manipular la Red de Estaciones, este ciclo terminará hasta que el usuario ya no desee seguir modificando la Red Estaciones.

Diagrama

Descripción generada automáticamente



**ALGORITMOS IMPLEMENTADOS:**

En la clase EstacionServicio se usaron los siguientes métodos:

|  |
| --- |
| 1. float obtenerCantidadCombustible(const string& tipo) const: Metodo que tiene como finalidad obtener la cantidad de combustible de las estaciones de servicio, teniendo en cuenta los parámetros iniciales. 2. float obtenerCantidadCombustible(const string& tipo) const: Este método inicializa los valores en litros de cada tipo de gasolina, de manera que se cumpla el parámetro de que haya entre 100 y 200 litros de cada gasolina. Se le pasan como parámetro los tipos de gasolina para que etse los asigne respectivamente. 3. unsigned char obtenerSurtidores() const: Solo tiene la tarea de obtener el valor que le otorga el constructor a los surtidores. 4. void modificarSurtidores(): Este método únicamente modifica la cantidad de surtidores a través de una interfaz de usuario para facilitar las tareas en el main. 5. unsigned char obtenerNaves(): Se tiene como condición inicial que cada nave contenga dos surtidores, de manera que este método a partir de la cantidad de naves que haya la divide entre 2 y genera la cantidad de naves, en caso de que la cantidad de surtidores sea un numero impar agrega una nave. 6. void asignarModelosSurtidores(): A partir de la cantidad de surtidores existentes les asigna un modelo al azar entre S, PMD y PHR. (estos son algunos modelos de surtidores que se encuentran en el mercado en la vida real). 7. void asignarEstadoSurtidores(): A partir de la cantidad de surtidores que esten en el momento en la estación les asigna el estado de activos a todos. Lo que implica que todos son usables. 8. void cambiarEstadoSurtidorPorModelo(const string& modelo, bool activar): Este método recibe como parámetros los modelos de cada surtidor y el booleano activar que es lo que permite activar o desactivar un surtidor. Este método a partir del modelo de surtidor los activa o desactiva. 9. void mostrarSurtidores(): Muestra la cantidad de surtidores, asociados con su estado y su modelo. 10. void simulacionVentas(float cantidadMaximaVenta): Hace una simulación de las ventas del dia, y tiene como parámetro la máxima venta posible, de manera que el usuario defina en la simulación cuanto es lo máximo que se puede vender por venta. (esta simulación es de la estación completa, no de surtidor) 11. string\* obtenerModelosSurtidores():Devuelve un puntero al arreglo dinámico de string que contiene los modelos de surtidores. Esto permite acceder a los modelos de surtidores desde otras partes del código. 12. string obtenerNombre():Se utiliza para obtener el nombre de la estación cuando sea necesario, por ejemplo, para mostrarlo en una interfaz de usuario o en un reporte. 13. string obtenerGerente():Devuelve el nombre del gerente de la estación de servicio como un string. 14. string obtenerRegion():Devuelve la región donde se ubica la estación de servicio como un string. 15. string obtenerCodigo():Devuelve el código único que identifica a la estación de servicio como un string. 16. string obtenerGps(): Devuelve las coordenadas GPS de la estación de servicio como un string, concatenando las tres coordenadas con comas. 17. void setPrecioPremium(float precio): Establece el precio del combustible Premium en la estación de servicio. Recibe un parámetro precio de tipo float. 18. void EstacionServicio::setPrecioRegular(float precio): Establece el precio del combustible Regular en la estación de servicio. Recibe un parámetro precio de tipo float. 19. void EstacionServicio::setPrecioEcoExtra(float precio): Establece el precio del combustible EcoExtra en la estación de servicio. Recibe un parámetro precio de tipo float. |

En la clase RedEstaciones se usaron los siguientes métodos:

|  |
| --- |
| 1. void agregarEstacion (EstacionServicio\* nuevaEstacion): Método que tiene como parámetro nueva estación que es un puntero de tipo EstacionServicio, el cual espera un objeto de EstacionServicio, siguiente a esto se guarda en la redEstaciones. 2. void eliminarEstacion (string& cod): Método que tiene como parámetro la dirección de memoria de un codigo el cual se busca eliminar, este cod se busca en redEstaciones para eliminarlo. 3. void fijarPrecioCombustible (): Método que sirve para fijar el precio del combustible depende la region y tipo de combustible. 4. void obtenerPrecios (const string& region): Método que recibe una region y buscaremos el precio de los tipos de combustible de la region. 5. EstacionServicio\*\* getRedEstaciones (): Método que sirve para retornar las estaciones. 6. void mostrarRedEstaciones (): Método que sirve para mostrar los valores de las estaciones de la red. 7. int getCantEstaciones (): Método que retorna el total de estaciones que hay en la red. |

**PROBLEMAS DURANTE EL DESARROLLO DEL PROGRAMA:**

**Los principales problemas que hemos tenido a la hora de realizar el programa es asociar la clase red con la clase de las estaciones, ya que el manejo de la información en cada una es la mayor complicación al ser demasiado estrica la lógica para el funcionamiento.**