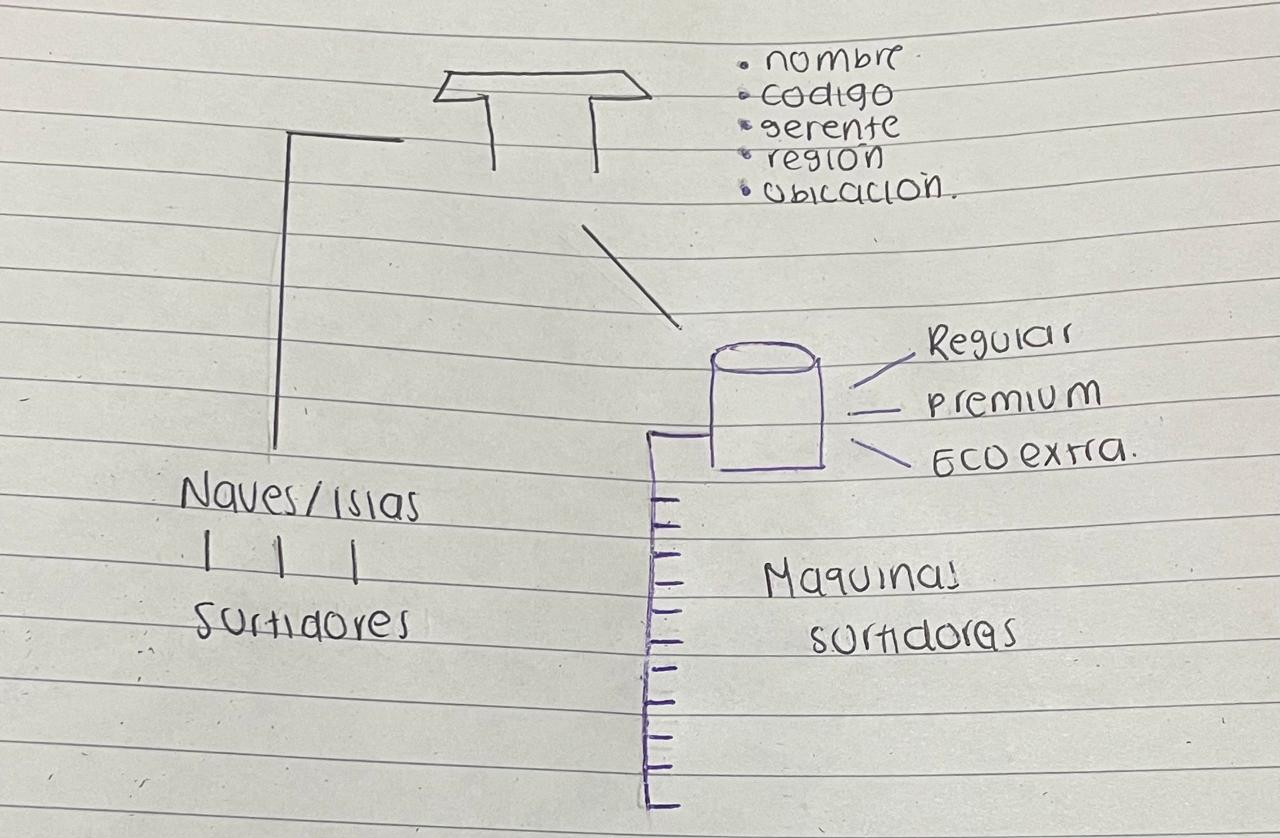
**INFORME DESAFIO 2**

**Juan Arbeláez María Alvarino**

1. **Contextualización del problema:**

El objetivo de este desafío es programar un sistema que simule las ventas y la gestión en una red de estaciones de combustible, cada estación tiene un nombre, un código identificador, un gerente, una región y una ubicación geográfica, cada estación cuenta con un tanque que se puede conectar de 2 a 12 maquinas surtidoras y a su vez las estaciones se pueden dividir en varias islas que contienen diferentes surtidores.

Para mejorar nuestro análisis graficamos una idea de lo que podía ser la distribución:

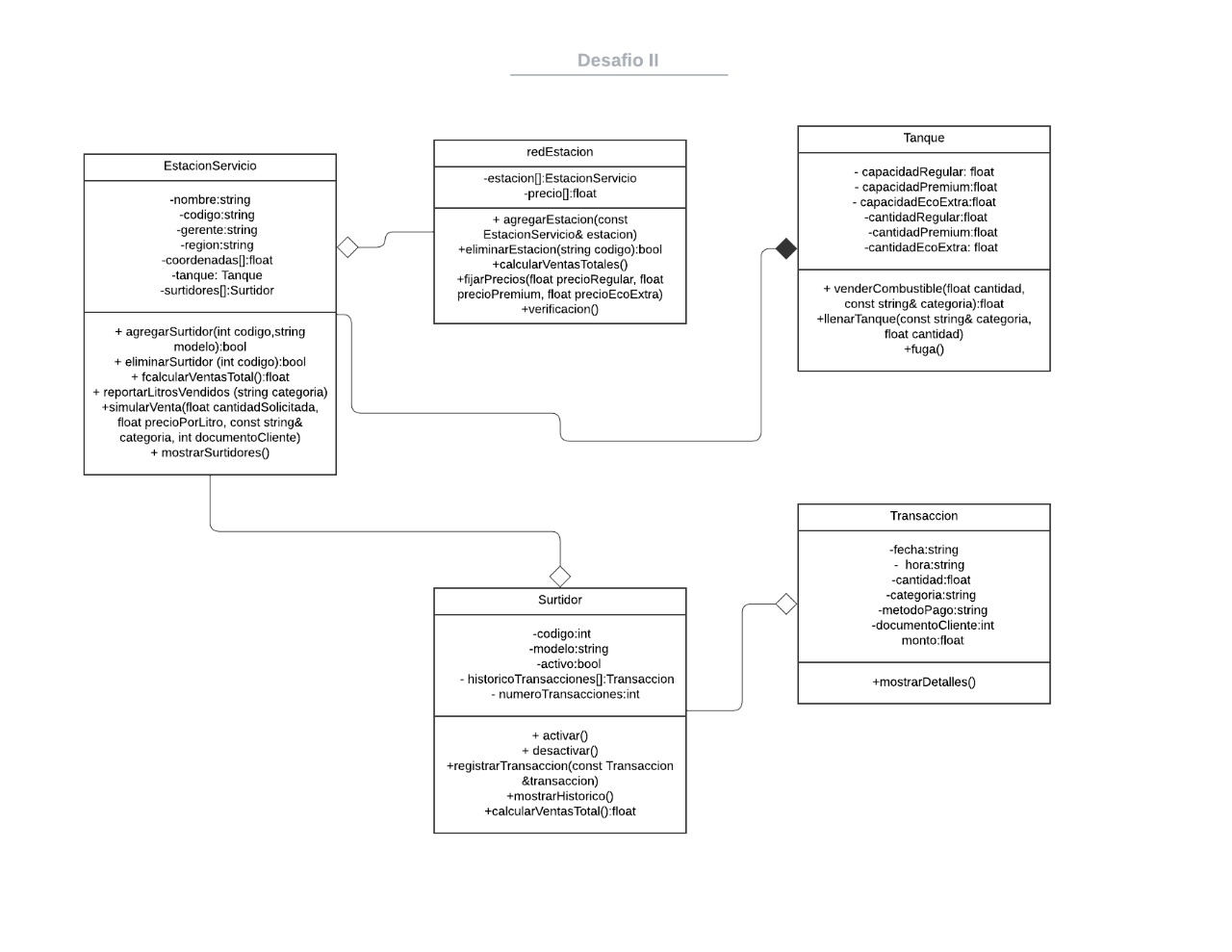


Cada surtidor maneja las ventas del día registrando diferentes datos además actualiza la cantidad de combustible en el tanque

1. **Consideraciones para tener en cuenta:**

Se debe contar con un menú para realizar diferentes acciones en dos partes: la red y las estaciones, en la red se podrá agregar estaciones, eliminar si no posee surtidores, calcular el total de ventas haciendo diferencia del tipo de combustible y fijar los precios de combustibles en las estaciones se podrá agregar o elimina un surtidor, activar o desactivar, conocer la cantidad de transacciones en cada surtidor, reportar la cantidad de litros vendida, simular una venta y asignar la capacidad del tanque con un valor aleatorio entre 100 y 200 litro, debe haber una comprobación de fugas en cada estación y por ultimo se debe generar transacciones dependiendo del precio de la región.

1. **Diagrama de clases:**



1. **Diseño de funciones, clases y algoritmos:**

Vamos a definir cinco clases:

1. Clase redEstacion, será la encargada de gestionar las estaciones, en donde manejaremos la suma y resta de estaciones, la fijación de los precios y la posible verificación de fugas, esta clase tendrá dos atributos: estacones, será un arreglo que almacenará las estaciones registradas y precios, otro arreglo que tendrá los precios del combustible en las regiones.
2. Clase EstacionServicio, representa a las estaciones en la red de forma individual, los surtidores y sus transacciones, aquí asignaremos el nombre, el código, el gerente, la región, los surtidores y el tanque como atributos.
3. Clase Surtidor, esta clase es para manejar las ventas y almacenar el histórico de las transacciones en los surtidores para eso asignamos cinco atributos: código, transacciones, activo, histórico de transacciones y numero de transacciones.
4. Clase Tanque es responsable de manejar y actualizar las reservas de combustible en los tanques cada que se hace una transacción, cada tanque almacena tres tipos de combustible, usamos seis atributos tres de ellos almacenan la cantidad del cada tipo y los otros tres la capacidad en el tanque.
5. Clase Transacción, aquí se almacenan los detalles de las ventas de combustible realizadas en los surtidores, maneja los datos como la fecha, la hora, la cantidad de litros vendidos y el método de pago usando siete atributos.

Para poder acceder a la información con la que trabajamos, realizaremos un menú.

**Funciones para diseñar:**

***Para la Red Nacional:***

**void agregarEstacion (const EstacionServicio& estacion);**

/\* Esta función agrega una nueva estación a la red de estaciones.

Retorna: void.

\*/

**bool eliminarEstacion(string codigo)**

/\* Elimina una estación de la red que cuente con al menos un surtidor para eso retorna true si la función pudo eliminar la estación y false si no fue posible.

Retorna: bool:: true or false.

\*/

**Void fijarPrecios(float precioRegular, float precioPremium, float precioEcoExtra)**

/\* Establece los precios del combustible a cada tipo y todas las estaciones.

//funcion en proceso

\* Retorna: void.

\*/

**Void verificarFugas());**

/\* Verifica si hay fugas de combustible en los tanques imprimiendo el resultado.

//Función en proceso

\* Retorna: void

\*/

***Para la EstacionServicio:***

**bool agregarSurtidor(int codigo, string modelo);**

/\* Agrega surtidores siempre que no hayan más de 12 retorna true si se puede agregar el surtidor o false si no.

\* Retorna: bool :: true o false

\*/

**bool eliminarSurtidor(int codigo);**

/\*Su función es eliminar surtidores verificando que el deseado este en el arreglo estático y remplazar en el mismo desplazando hacia la izquierda, retorna true o false si se puede eliminar el surtidor o no .

\* Retorna: bool :: true o false.

\*/

**Float calcularVentasTotal(string categoria);**

/\* Calcula el total de ventas sumando las ventas de todos los surtidores de esa estación .

\* Retorna: float

\*/

**Void reportarLitrosVendidos (string categoria);**

/\* Esta función reporta el total de litros vendidos para una categoría específica de combustible (Regular, Premium, EcoExtra) en todos los surtidores de la estación.

\* Retorna: void

\*/

**void simularVenta(float cantidadSolicitada, float precioPorLitro, const string& categoria, int documentoCliente);**

/\* simula una venta de combustible, maneja todo el proceso de verificar la cantidad disponible, registra la venta y genera la transacción asociada, para eso se busca si hay surtidores disponibles, llama a otra función para determinar la cantidad de combustible en la categoría solicitada y hace operaciones para conocer el precio final.

\* Retorna: void

\*/

**void mostrarSurtidores();**

/\* muestra el historial de transacciones de cada surtidor en la estación de servicio, llama a otra función para recorrer las transacciones realizadas.

\* Retorna: void.

\*/

***Para Surtidor:***

**Void registrarTransaccion(const Transaccion &transaccion);**

/\* Verifica que el número de transacciones registradas no haya superado el límite de 100. Si el número de transacciones es menor a 100, guarda la nueva transacción en el arreglo historicoTransacciones. Si ya se alcanzó el límite de 100 transacciones, muestra que se ha alcanzado el límite.

\* Retorna:void.

\*/

**void mostrarHistorico();**

/\* Muestra el historial de transacciones realizadas en el surtidor, recorre todas las transacciones registradas y llama a la función mostrarDetalles de cada transacción para imprimir sus detalles.

\* Retorna: void.

\*/

**Float calcularVentasTotal();**

/\* Calcula el total de las ventas realizadas en ese surtidor, recorre todas las transacciones registradas en historicoTransacciones y suma el monto de cada transacción usando get al valor acumulado de total.

\* Retorna: float.

\*/

***Para Tanque:***

**float venderCombustible(float cantidad, const string& categoria);**

/\* Este método vende combustible de una categoría específica y actualiza la cantidad de combustible restante en el tanque de la estación, para esto compara y actualiza variables.

\* Retorna: float.

\*/

**Void llenarTanque(const string& categoria, float cantidad);**

/\* Esta función actualiza los tanques sumando cantidad sin superar el tope máximo, para eso Verifica si la cantidad a agregar junto con la cantidad existente no excede la capacidad máxima del tanque de Regular, si no excede la capacidad, añade la cantidad solicitada a una variable, si por el contrario excede la máxima, llena el tanque hasta su capacidad máxima.

\* Retorna: void.

\*/

***Para la Transaccion:***

**void mostrarDetalles()**

/\* Esta función muestra en pantalla los datos necesarios para hacer una transacción.

\* Retorna: void .

\*/

1. **Observaciones del desarrollo y conclusiones:**
2. Nos enfrentamos