

**INFORME DESAFÍO II INFORMATICA II**

Jaider Bedoya Carmona

Wilbergt Alexander Osorio Trespalacios

Informe sobre proceso de desarrollo del desafío dos

PROFESORES

Augusto Salazar

 Aníbal Guerra

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería Electrónica

Medellín

2024

**Contextualización**

La empresa TerMax es una compañía líder del suministro de combustibles, tiene estaciones de servicio repartidas a nivel nacional.

Cada estación de servicio posee un nombre, un código identificador, un gerente, una región y una ubicación geográfica (expresada en coordenadas GPS). Además, cada una tiene un tanque central donde se almacenan separadamente las 3 categorías de combustible disponible para vender (Regular, Premium y EcoExtra). La capacidad de almacenamiento de cada categoría no necesariamente es uniforme en el tanque y puede variar entre los tanques de diferentes estaciones de servicio. A este tanque central se conectan entre 2 a 12 máquinas surtidoras desde las cuales se venderá directamente a los vehículos de los clientes. La estación de servicio puede estar físicamente subdividida en varias naves o islas, en las cuales se agrupan físicamente varios surtidores de forma contigua.

Cada surtidor posee un código identificador y un modelo de dicha máquina, y están habilitados para distribuir las tres categorías de combustible. Cada uno registra individualmente todas las ventas realizadas en el día, registrando: la fecha, hora, cantidad y categoría de combustible, método de pago (Efectivo, TDebito, TCrédito), número de documento del cliente y la cantidad de dinero respectiva. Adicionalmente, cada vez que una surtidora vende combustible, se debe actualizar la disponibilidad del mismo en el tanque de la estación. En caso de que una venta requiera más combustible del disponible, se procede con la venta, pero sólo debe cobrarse la cantidad suministrada (además se deben cumplir unas funcionalidades mínimas explicitas en el documento del desafío).

**Análisis del problema**

Dado que debemos usar programación orientada a objetos, tenemos que la propuesta a desarrollar, es implementar 3 clases que interactúan entre ellas y tienen una relación estrecha en sus características, las cuales son:

* Clase Red Nacional 🡪 Busca expresar simplemente una red de varias estaciones también es donde se fijan los precios del combustible de la toda la red diferenciándolos por región y por categoría de combustible.

* Clase Estación 🡪 Estaciones que están repartidas a nivel nacional y cada una cuenta con atributos de diferenciación e implícitamente contiene un numero de surtidores.

* Clase Surtidor 🡪 Permite administrar las características y tareas hechas por un surtidor, puesto que el surtidor es el contacto directo entre el cliente y la empresa.

La clase de surtidores depende de la clase estación y esta depende de la clase Red, no hay herencia, pero la existencia de una depende de la otra; también pensamos implementar archivos tipo “.txt” para guardar datos como las estaciones que hay, los surtidores y las ventas realizadas.

**Diagrama de clases**

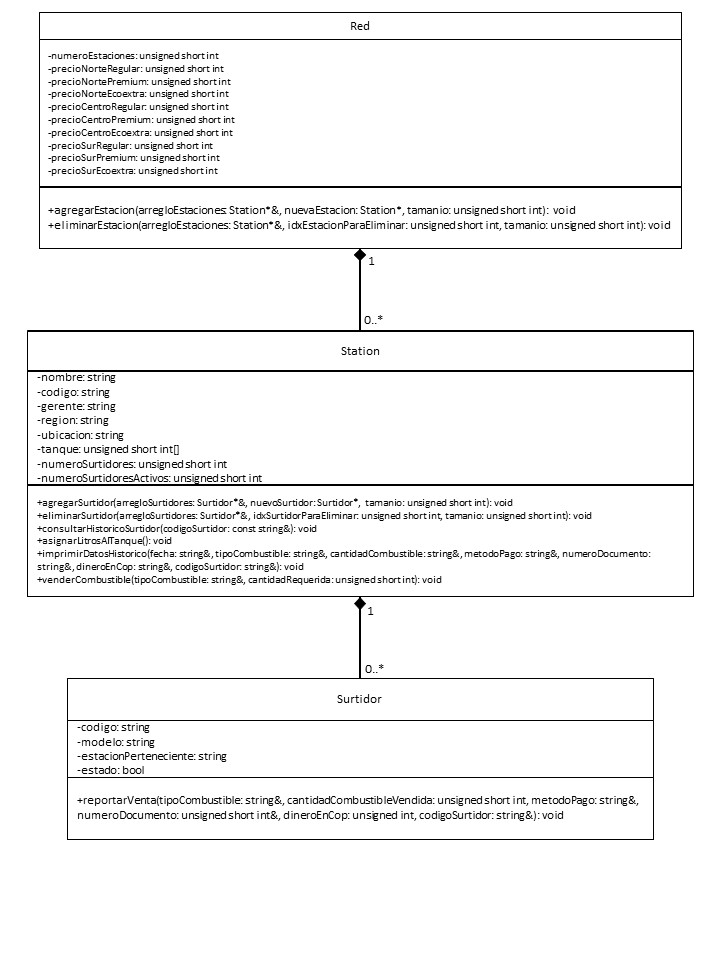
****

Ilustración 1 Diagrama de clases

**Algoritmos implementados**

Algunos algoritmos implementados fuera de cada clase que son importantes en el programa son las siguientes funciones:

**Station leerDatosEstacion(const string datosEstacion){}**

Devuelve una instancia de la clase Station, recibiendo como parámetro un string que contiene toda la información esencial de la estación, para separar y asignar cada atributo a la instancia.

**void llenarArregloEstaciones(Station\* arregloEstaciones, const string& nomArchivo){}**

Llena un arreglo que contiene todas las instancias de las estaciones existentes en el archivo de texto que usamos como “base de datos”, esto con el fin de que los tratamientos de las estaciones sean más rápido y eficiente.

**void menu(){}**

Tiene como objetivo crear una interfaz semigrafica por consola, en la cual el usuario accede a las opciones que están establecidas en los requisitos del desafío, también se encuentra control de excepciones para que el usuario no ingrese opciones invalidas.

**Station\* pedirDatosEstacionNueva(){}**

Le pide al usuario los datos necesarios para la creación de una estación nueva, y retorna un objeto Station que luego se agrega al arreglo de estaciones.

**Surtidor\* pedirDatosSurtidorNuevo(){}**

Le pide al usuario los datos necesarios para la creación de un surtidor nuevo, y retorna un objeto Surtidor y los agrega al arreglo de surtidores.

**string obtenerPosicionEstacionParaEliminar(){}**

busca en el archivo de estaciones para imprimir cada una y espera que el usuario ingrese cual es la estación que desea borrar.

**Unsigned short int obtenerPosicionsurtidorParaEliminar(){}**

Busca en el archivo de surtidores, y los imprime en pantalla para que el usuario escoja cual surtidor desea eliminar y retorna esta opción.

**Void fijarPrecios(){}**

Si el usuario desea cambiar los precios del combustible por cada región será enviado a esta función donde pide estos precios y los pone en los atributos de red donde está cada precio por región y categoría de combustible.

**Metodos de clase Red:**

Unicamente tiene dos funciones que lo que hacen es agregar uno o restar uno a la cantidad de estaciones que tiene en el atributo estaciones, dependiendo de cuantas se agreguen o eliminen.

**Metodos de estación:**

**agregarSurtidor()**

añade el surtidor creado en donde se piden los datos de este, y lo añade al arreglo de surtidores.

**eliminarSurtidor()** tiene la misma lógica, pero lo elimina.

**consultarHistoricoSurtidor()**

busca en el archivo de ventas todas las ventas con el código de surtidor que se le fue entregado y los imprime usando otra función.

**reportarLaCantidadDeCombustiblePorTipo()**

imprime la cantidad de combustible disponible por categoría.

**asignarLitrosAlTanque()**

asigna de forma aleatoria una cantidad de litros a los tanques entre 100 y 200 litros.

**venderCombustible()**

cambia el valor de la cantidad del tanque del combustible que fue vendido.

**Metodos de Surtidor:**

**reportarVenta()**

sube todos los datos de la venta a un archivo.

**otras funciones auxiliares:**

**unsigned short int contarLineasEnArchivo(const string& nombreArchivo){}**

**unsigned short int contarCaracteresEnString(const string cadenaCaracteres, const char caracterParaContar){}**

Estas funciones nos ayudan en la implementación ya que hacen tareas muy sencillas, las cuales deben ser realizadas en numerosas ocasiones. así seguimos usando varias funciones que nos ayudaban en tareas repetitivas que no vimos necesario escribirlas aquí.

**Problemas afrontados**

El primer problema en el desarrollo fue la idea de correlación entre las clases y pensar en la definición de cada atributo y método, puesto que también hay que tener en cuenta la abstracción para no sobre objetivizar las entidades.

El tener criterio para decidir cómo distribuir los reportes de venta, información de estaciones y surtidores, y demás cosas que se necesitan guardar, resulta un problema que le dimos solución con criterios de eficiencia, facilidad y organización, siempre con argumentos para cada decisión.

En algunas funciones que utilizaban pasos por referencia a objetos o variables, ocurrían fallos de ámbito por lo que se tuvo que modificar la forma en la que accedían estas funciones a estos valores.

Al usar metodos y funciones que dependían de otros, ambas ramas de desarrollo tenían partes que dependian de otras, sobre todo el menú principal que es el que interactúa con el usuario y despliega sus respectivas funcionalidades, por lo que en algunos casos no coincidían ambas partes y se debían hacer ajustes que en algunos casos fueron cambios grandes en alguna de las partes.

El menú principal es una función que interactúa con el usuario y redirecciona a este hacia los demás menús y partes del programa, cuando se creó la función que agrega una nueva estación daba problemas al salir de esta nuevamente hacia el menú, ya que es un ciclo donde se estaban usando en un inicio condicionales de la forma “switch”, estos últimos son presuntamente el problema, por lo que se tuvo que cambiar la forma en la que el menú ingresa a las diferentes funciones.

Se hizo toda la implementación del programa en GDB online debugger, y cuando pudimos pasar el programa a un archivo en Qt, este no funcionó correctamente siendo exactamente el mismo programa.

**Evolución de la solución**

En la primera sesión simplemente nos limitamos a analizar muy bien el problema y pensar en la solución, teniendo en cuenta todos los requerimientos para poder ir pensando en la implementación que llevaremos luego.

Empezando con la implementación del programa, se dividió equitativamente el desarrollo del mismo, cada uno tomando 2 tareas grandes, ya sea una o dos clases, o funciones anexas.

Para el desarrollo declaramos 3 clases como lo planeamos, divididos respectivamente en archivos .h y .cpp, para declaración e implementación de cada componente, usando la programación modular y buenas prácticas para el orden del proyecto.

Se llevó a cabo el desarrollo satisfactoriamente de la clase surtidor, estación y red, además de funciones anexas que nos ayudan al programa, así como el menú principal, hubo bastante trabajo colaborativo usando la herramienta de GitHub como principal fuente de transferencia y compartición de los archivos desarrollados.

Se desarrollaron varias funciones para diferentes partes del programa como lo son, crear y borrar estaciones, asignar los precios de toda la red según la región, también crear y borrar surtidores como activar y desactivar estos mismos.

El menú principal tuvo que ser parcialmente re diseñado por los problemas que estaba dando, al entrar en las diferentes funciones este fue dando errores de ámbito por los pasos por referencia de diferentes tipos de datos los cuales no tuvieron una solución individual, sino que se tuvo que cambiar desde le menú donde se accedía estas funciones, para que hubiese un mejor manejo de los datos por todo el programa.

Se implementó correctamente la función de vender y así mismo inscribir esta venta en el archivo de ventas.