Desafío 2 – Sistema de comercialización de combustible TerMax

Andrés David Durán Quiñones
Ingeniería Electrónica
Universidad de Antioquia
Medellín, Colombia
andres.duranq@udea.edu.co

Daniela Escobar Velandia
Ingeniería Electrónica
Universidad de Antioquia
Medellín, Colombia
daniela.escobarv@udea.edu.co

***Resumen*—El presente trabajo tiene como objetivo modelar y desarrollar un sistema de gestión para una red de estaciones de servicio, utilizando principios de programación orientada a objetos (POO) en C++. Este sistema busca automatizar y optimizar las operaciones de las estaciones, incluyendo la gestión de surtidores y transacciones de venta de combustible, mientras que permite el análisis de problemáticas del mundo real. El trabajo se basa en la empresa TerMax, que lidera el suministro de combustibles en Colombia, y establece un modelo que incluye la gestión eficiente de recursos y la implementación de funciones específicas, como la simulación de ventas y la detección de fugas de combustible.**

I. Análisis del problema y consideraciones para la alternativa de solución propuesta

Para desarrollar este desafío fue necesario realizar un debate entre los integrantes para lograr una mejor comprensión del mismo. Esto permitió plantear el problema de la siguiente manera.

La empresa TerMax tiene una red de estaciones de servicio distribuidas en distintas regiones del país, y busca mejorar la gestión y control de su sistema de comercialización de combustibles. Cada estación de servicio tiene un tanque central que almacena tres tipos de gasolina: Regular, Premium y EcoExtra, pero la capacidad de almacenamiento de cada tipo de gasolina varía de una estación a otra, lo que hace más complejo el manejo del inventario y el control de ventas.

Además, cada estación está conectada a entre 2 y 12 máquinas surtidoras, lo que requiere un sistema que gestione las ventas, registre las transacciones y mantenga actualizada la disponibilidad de combustible en tiempo real. También es necesario que el sistema permita agregar y eliminar estaciones, controlar los surtidores, y asegurar que las estaciones no presenten pérdidas o fugas de combustible que puedan afectar las operaciones.

El principal reto es diseñar un sistema que, además de manejar el inventario y las ventas de combustible, permita detectar posibles fugas en los tanques. Para lograr esto, el sistema deberá verificar que la cantidad de combustible vendido más lo almacenado sea coherente con la capacidad inicial del tanque, manteniendo un margen mínimo del 95 % para evitar sospechas de pérdida o fuga.

Otro desafío clave es la necesidad de simular ventas automáticas, seleccionando aleatoriamente un surtidor activo y generando transacciones de venta de entre 3 y 20 litros. Las ventas deben tener en cuenta los precios del combustible, que varían según la región (Norte, Centro, Sur), y además registrar datos como la categoría de combustible, la cantidad vendida, el método de pago y la identificación del cliente.

Teniendo en cuenta el desafío del problema, es destacable que la programación orientada a objetos (POO) nos ayudará a simplificar las cosas debido a su enfoque modular y estructurado. POO permite organizar el código en clases, representando entidades del mundo real como estaciones de servicio, surtidores y tanques de combustible, lo que facilita la gestión de la complejidad y el mantenimiento del sistema.

I-A. Solución

Cada elemento principal del sistema se representará como una clase. Esto permitirá encapsular la funcionalidad y los datos, facilitando la reutilización del código y el manejo de diferentes componentes del sistema.

En un principio se planteó la solución con 5 clases, pero a lo largo del desarrolló se optó por implementar 3 clases debido a que permiten una organización jerárquica de los elementos del sistema y una simplificación del modelo.

- **Clase `Red_nacional`**: Representa la red nacional de estaciones de servicio. Se encarga de gestionar las estaciones de la red, permitiendo agregar, eliminar y consultar estaciones. También proporciona métodos para simular ventas en todas las estaciones y consultar el estado de los surtidores y tanques a nivel nacional.

- **Clase `Estacion_servicio`**: Modela una estación de servicio individual. Tiene atributos como nombre, gerente y un código identificador. Además, gestiona una colección de surtidores y un tanque central que almacena el combustible disponible para la estación. La clase también incluye métodos para realizar ventas y verificar el estado de los surtidores.

- **Estructura `Tanque`**: Representa un tanque de combustible en la estación de servicio. Contiene información sobre la capacidad máxima del tanque, la cantidad de

combustible disponible actualmente y la cantidad total vendida. Los valores se actualizan cada vez que se realiza una venta.

- **Clase `Surtidor`**: Representa un surtidor dentro de una estación de servicio. Cada surtidor está vinculado al tanque de la estación y se encarga de procesar las ventas de combustible, actualizando las cantidades en el tanque correspondiente después de cada transacción. Incluye atributos como un código identificador y un estado que indica si el surtidor está activo o inactivo.

- **Estructura `Venta`**: Registra los detalles de cada venta de combustible. Almacena la fecha y hora de la venta, la cantidad de combustible vendida, la categoría del combustible (Regular, Premium, o EcoExtra), el método de pago utilizado (efectivo, tarjeta de crédito o débito), el identificador del cliente y el total de dinero pagado por la venta.

I-A1. ***Gestión de Transacciones y Ventas****:* Cada surtidor está vinculado a un tanque de combustible, y las ventas se realizan en función de la cantidad de litros disponibles. Cada venta reduce el volumen de combustible en el tanque correspondiente, actualizando el estado del surtidor y del tanque en tiempo real cada vez que se efectúa una venta.

I-A2. ***Simulación de Ventas****:* La solución incluye un método en la clase `Estacion_servicio` para simular una venta. Este método selecciona aleatoriamente un surtidor activo, genera una cantidad aleatoria de litros entre 3 y 20, y ejecuta la venta actualizando tanto el tanque como la transacción registrada. Los precios dependen de la región, y se pueden ajustar dinámicamente en la clase `Red_nacional`.

I-A3. ***Detección de Fugas****:* La clase `EstacionServicio` incluye un método para verificar posibles fugas de combustible. Donde se compara la cantidad inicial almacenada en el tanque con la suma de las ventas y lo que queda disponible, alertando si se ha vendido más del 95 % de la capacidad original del tanque. Esto permite identificar problemas de pérdidas en cualquier estación.

I-A4. ***Menú Interactivo****:* El menú implementado permite al usuario agregar o eliminar estaciones de servicio, activar o desactivar surtidores, consultar ventas por categoría de combustible, fijar precios, verificar fugas y simular ventas. Esto proporciona una interfaz clara y fácil de usar para gestionar toda la red.

I-B. Consideraciones Adicionales

- **Escalabilidad**: La solución permitirá agregar nuevas estaciones y surtidores de manera flexible, manteniendo una estructura clara y ordenada.
- **Manejo de Excepciones**: Uso `try` y `catch` para manejar errores relacionados con la disponibilidad de combustible y la gestión de datos incorrectos o faltantes, mejorando la robustez del sistema.

II. Diagrama de clases de la solución planteada

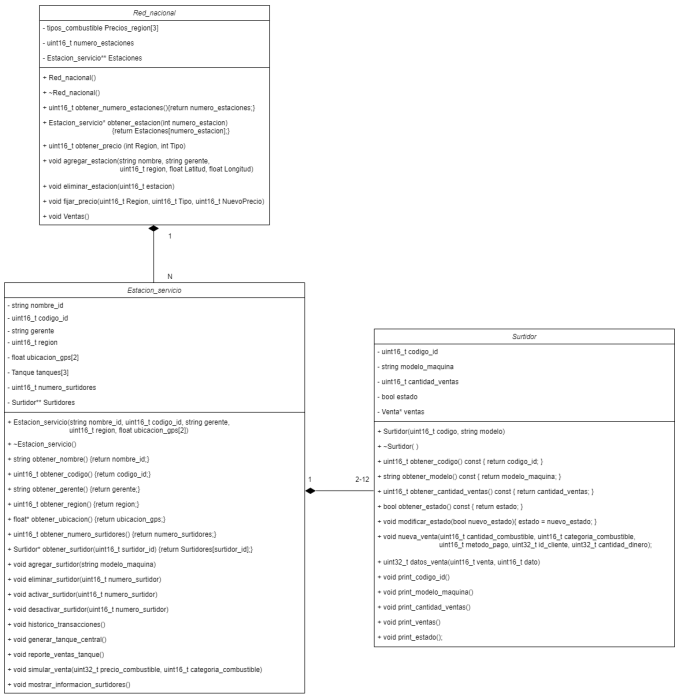

Figura 1. Diagrama de clases implementado.

III. Descripción de alto nivel

III-A. Lógica General del Sistema

El sistema está diseñado para gestionar una Red Nacional de Estaciones de Servicio. Cada estación tiene múltiples surtidores, un tanque con capacidad limitada de tres tipos de combustibles, y permite realizar ventas de combustible, todo gestionado mediante transacciones.

1. Red Nacional de Estaciones de Servicio

- La red maneja una colección de estaciones de servicio. Se permite agregar y eliminar estaciones, así como gestionar surtidores y transacciones de manera individual para cada estación.
- Se tiene una clase Red_nacional, que contiene un vector de objetos Estacion_servicio. A través de funciones de acceso, el sistema permite la manipulación de las estaciones de servicio de la red y realizar operaciones específicas sobre ellas, como obtener el total de ventas por tipo de combustible o fijar precios.

2. Estaciones de Servicio

- Cada estación de servicio almacena datos como su nombre, código, gerente, ubicación y tanque de almacenamiento de combustible. También gestiona las máquinas surtidoras que permiten realizar ventas.
- La clase Estacion_servicio contiene un vector de objetos Surtidor, que representan las máquinas que realizan las transacciones de venta.

- Tiene un tanque que almacena combustible en tres categorías: Regular, Premium y EcoExtra, con una capacidad inicial aleatoria (entre 100 y 200 litros por categoría).
- La estación permite realizar operaciones como agregar o eliminar surtidores, consultar el histórico de ventas, y reportar las cantidades de combustible vendidas por tipo.

3. Surtidor

- Cada surtidor es una máquina individual que realiza ventas de combustible. Lleva el registro de transacciones, las cuales incluyen detalles como la cantidad de litros vendidos, la categoría de combustible y el método de pago.
- La clase Surtidor mantiene un historial de transacciones, que es un vector de objetos Transaccion. Cada vez que se realiza una venta, se actualiza el tanque de la estación correspondiente para reflejar la cantidad de combustible disponible. Además, se asegura que no se venda más combustible del disponible en el tanque.

IV. Problemas de desarrollo que afrontó

Al desarrollar un sistema con múltiples estructuras de datos, como en el caso de las clases, es importante gestionar adecuadamente la memoria para prevenir fugas. La asignación dinámica de objetos requiere que implementemos destructores que liberen la memoria cuando ya no se necesite, garantizando así el correcto funcionamiento del sistema y evitando un consumo innecesario de recursos. Además, esto facilita la escalabilidad y mantenibilidad del software, permitiendo incorporar nuevas funcionalidades sin comprometer el rendimiento.

Nos enfocamos en la elección de los tipos de datos porque son fundamentales para la estructura y eficiencia del sistema. Al seleccionar cuidadosamente los tipos de datos, garantizamos un uso eficiente de la memoria y una adecuada representación de los valores en nuestro contexto específico. Además, al definir tipos de datos personalizados, facilitamos la organización y manipulación de la información relacionada con las estaciones de servicio, lo que a su vez mejora la legibilidad y mantenibilidad del código.

V. Evolución de la solución y consideraciones

V-A. Evolución de la solución

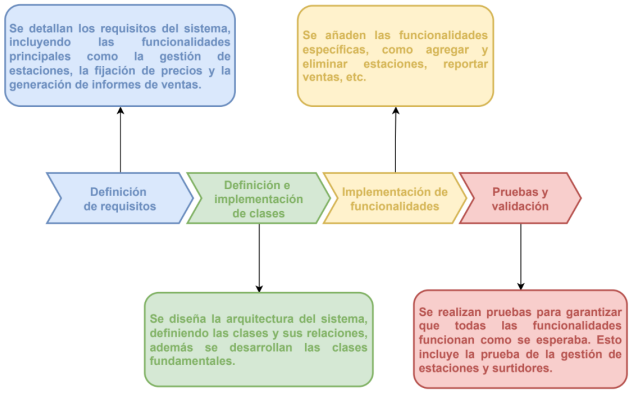

Figura 2. Evolución de la Solución.

V-B. Consideraciones

A continuación se presentan algunas consideraciones que se pueden tener en cuenta para realizarle mejoras al sistema:

- Diseñar el sistema de manera modular, lo que facilita la adición de nuevas características o módulos sin afectar el resto del sistema.
- Diseñar una interfaz de usuario intuitiva que permita una buena experiencia.
- Utilizar una base de datos para almacenar la información de las estaciones y las ventas, lo que facilitará la gestión y el análisis de datos a largo plazo. También se puede tener en cuenta la implementación de herramientas de análisis para monitorear el rendimiento de las estaciones, la demanda de combustibles y otros datos importantes.
- Implementación de mecanismos de autenticación y autorización para proteger la información sensible y garantizar que solo los usuarios autorizados tengan acceso a ciertas funcionalidades. Considerar el cifrado de datos, como la comunicación entre el cliente y el servidor.