**Sistema de administración de estaciones de gasolina**

Juan David Naranjo Jara y José Alejandro Pabón Leal  
  
  
  
Informatica2  
  
  
  
12/10/24

1. Introducción

El presente informe describe el desarrollo del proyecto de gestión de combustibles para estaciones de gasolina. Este sistema permite gestionar diferentes estaciones de gasolina con una cantidad n de surtidores y controlar la disponibilidad y venta de tres tipos de combustibles: Regular, Premium y Ecomax.

El objetivo principal es ofrecer una solución que permita realizar ventas según la disponibilidad de los combustibles en cada estación y gestionar los surtidores asociados.

2. Análisis del Problema

2.1. Descripción del Problema

Las estaciones de gasolina deben administrar varios tipos de combustibles que pueden o no estar disponibles en función de su inventario. Cada surtidor puede vender uno o varios tipos de combustibles, pero es necesario verificar que estos se encuentren activos antes de realizar la venta, también cada surtidor se debe hacer un registro de venta detallado.

Con este Análisis decidimos gestionar las estaciones como objetos de una clase y los tanques como objetos de otra clase, con las cuales, las estaciones nos permitirán gestionar de manera eficiente los surtidores y las ventas, mientras que, los tanques permitirán almacenar el combustible y permitir el uso de este.

2.2. Consideraciones para la Alternativa de Solución

Otra posible solución es crear objetos en la clase estación referentes a los surtidos para gestionarlos de manera individual a los objetos creados como estaciones, así se puede gestionar métodos de ventas para los surtidores y registrarlos de manera mas clara y sencilla  
Otra posible solución es crear una clase estación de arreglos, que permitan en los arreglos guardar los objetos de tipo, estación, surtidor y tanque de una manera mas eficiente sencilla y practica para manejar los problemas que abordamos.

4. Diagrama   
<https://lucid.app/lucidchart/979dfd5b-f477-46d3-a05f-acb5d1f6685f/edit?viewport_loc=-1316%2C-786%2C1897%2C800%2C0_0&invitationId=inv_74913bb7-07a6-4582-9751-6de50fb79035>  
  
5. Algoritmos implementados  
5.1 Clase estacion  
 Constructor por defecto: inicializa los atributos sin valores específicos.   
estacion();   
Constructor sobrecargado: permite inicializar los atributos de la estación con datos específicos.   
estacion(string nombre, unsigned int id, string gerente, char region, double latitud, double longitud, string maquina, unsigned short int isla, unsigned short int activo);   
 Destructor: libera la memoria dinámica asignada para las ventas.   
~estacion();   
Getters: devuelven los valores de los atributos privados.   
Setter: para cambiar el estado activo de la estación.   
Métodos para gestionar surtidores:  
void agregarSurtidor(unsigned short int idSurtidor, bool activo, unsigned short int ventas).  
void eliminarSurtidor(unsigned short int idSurtidor).  
 void activarSurtidor(unsigned short int indice, bool estado).  
void mostrarEstadoSurtidores() const. Muestra la actividad y ventas de un surtidor.  
void registrarVentaSurtidor(unsigned short int id).  
void mostrarInfo() const. Muestra el estado actual del combustible.   
específico. void venderCombustible(string tipo) const.  
Métodos para registrar ventas:  
void registrarVenta(double cantidad, string categoria, string metodoPago, string documentoCliente, double monto, unsigned short int id, unsigned short int est).   
bool guardarVentasEnArchivo(). Guarda todas las ventas realizadas.   
void mostrarVentas(). Métodos para manejar estaciones desde archivos.   
Métodso para guardar estaciones desde archivos:  
static unsigned int contadorlineas(const string& rutaArchivo).   
void guardarTXT(const string& rutaArchivo). Guarda objetos de la clase “estacion” en un archivo.  
void guardarSurtidoresTXT(const string& rutaArchivo). Guarda surtidores de una estacion en un archivo.  
 static estacion\* TXTobj(const string& rutaArchivo, const string& rutaSurtidores). Guarda en un array objetos de la clase “estacion”.  
 static surtidor\* TXTsurtidor(const string& rutaArchivo, unsigned short& count). Complementa TXTobj.  
5.2 Clase tank  
Constructor por defecto: inicializa los valores sin datos específicos.   
tank();   
Constructor sobrecargado:   
Tank(unsigned int id\_, float regular\_, float premiun\_, float ecoextra\_, float pr, float pp, float peco, unsigned short int activo\_). permite inicializar un tanque con identificador, cantidades de combustible, precios y estado.  
Constructor sobrecargado:   
tank(unsigned int id\_, unsigned short int activo\_). solo inicializa con el id y el estado.  
Destructor:

~tank();   
Getters.  
Setters.  
Métodos para guardar tanques:

void GuardarTank(). Guarda los datos del tanque en un archivo de texto.  
void setprecios(float pr, float pp, float peco). pr: Precio del combustible regular. pp: Precio del combustible premium.. peco: Precio del combustible eco.   
static unsigned int contadorlineas(); Convierte los datos de un archivo de texto a un objeto de clase `tank`. static tank\* TXTobj(const string& rutaArchivo). Guarda un objeto de tipo `tank` en un archivo.   
static void Saveobj(tank\* array, tank obj, const string& archivo). Guarda en una array desde un archivo objetos de la clase “tank”.  
  
6. Problemas de desarrollo afróntados

Debido a los diferentes acciones que debe realizar una estacion de gasolina, afrontamos diferentes desafios para lograr gestionarlas de manera adecuada y eficiente.   
Los principales problemas fueron lograr guardar nuestro objetos para evitar que al momento de cerrar el programa las estaciones creadas se perdieran y al momento de iniciar el progama nuestras estaciones y los cambios que realiamos en ellas permanecieran intactas.  
Otro problema principal fue lograr un registro presivo y detallado de las ventas que se realizan en nuestros surtidores , debido a que, dichos surtidores tienen caracteristicas especificas que se deben tener en cuenta a la hora de simular una venta de combustible.  
Otro problema que afrontamos fue el manejo adecuado de los tanques que permiten sumistrar los 3 tipos de combistible en las estaciones y que estos cambien su contenido.  
  
7. Evolución de la solución y consideraciones para tener en cuenta en la implementación.

7.1 Evolución  
Las ideas principales en nuestro código por supuesto tuvieron que evolucionar para adaptarse a las difretens problematicas que afrontamos sin dejar de lado la eficiencia y el funcionamiento adecuado de nuestras estaciones de galosiona, como recurrir a alternativas para guardar correctamente nuestros objetos y nuestras de manera que el algoritmo realice la gestión idonia de las estaciones de gasolina. Se evolucionó en la estrucuta de datos para tener un orden adecuado en los diferentes atributos de cada objeto, su correcto almanecamiento y/o carga en/desde archivos, como también el registro eficiente en las ventas y en general las decidimos que tomamos para tener una implementacion eficiente y estructura de las estaciones de gasolina.  
7.2 Consideraciones para la implementación.  
Nuestra decision fue tener uso responsable de archivos.txt que permitan gestionar adecuadamente el almacenamiento de los objetos y el registro de ventas que nos ayuda a guardar ordenamente los objetos teniendo en cuenta por supuesto la eficiencia en dicho proceso para garantizar la optimizacion y menor gasto de recuros para cumplir con el requisitos deseados por el cliente. También consideramos ser cuidadosos con nuestros tipos de datos sabiendo que los objetos por naturaleza consumen mas recursos que cualquier otro tipo de dato, por lo que con este nivel de detalle lograramos minimizar los gatos permitiendonos sacar ventaja de los objetos.