UNIVERSIDAD DE ANTIOQUIA
1803

Desafío II

Nombres:

Juan Sebastian Osorio Osorio

Camilo Andrés Caicedo Úsuga

Tema:

Sistema de comercialización de combustible TerMax

Docente:

Augusto Enrique Salazar

Año:

2024

Universidad de Antioquia

Informática II

Informe

Al iniciar el proceso de análisis se opto por crear 4 clases:

Clase Estacion.

En donde se asignan los datos principales de la estación como: nombre, código que identifica a la estación, el nombre del gerente, la región, la ubicación de la estación y la cantidad de surtidores que tiene la estación. Adicional, algunos de los métodos que tiene la clase como:

agregarEstación () - En esta parte se consideró leer un archivo **"datosEstacion.txt"**, en el cual se alojan los datos de cada una de las estaciones. Entonces se lee el archivo y se verifica que al momento de agregar la nueva estación esta no exista.

eliminarEstacion () - Para eliminar la estación primero se debe verificar que la estación existe en el archivo **"datosEstacion.txt"** y segundo, se verifica que la estación tenga surtidores activos o inactivos. Por lo cual, si los surtidores no están activos se realiza el proceso de eliminar la estación, pero si alguno de los surtidores esta activo se le informa al usuario de la situación. Finalmente sobre escribe el archivo, ya que para poder eliminar la estación seleccionada se debe escribir los datos ya existentes, excepto los datos de la estación que se quiere eliminar.

calcularMonto () - En esta parte se calcula la cantidad de dinero del total de las ventas a nivel nacional. Estos datos están alojados en el archivo **"archivoRegistros.txt"**. Para poder realizar el proceso, se leen los datos y se diferencia entre las tres categorías de combustible, sumando cada valor.

mostrarMonto () - Se imprime por consola los datos calculados en el método anterior.

reportarCanLitros () - Se calcula la cantidad de litros vendida según la estación indicada. Estos datos están alojados en un **"archivoRegistros.txt"**. Para esto, se pide al usuario el código de la estación. Finalmente se leen los datos del archivo y solo se toman los datos correspondientes a los litros y se diferencia entre cada categoría de combustible.

~Estacion () - El destructor será usado principalmente para liberar la memoria de algunos arreglos dinámicos.

Clase TanqueCentral.

En esta clase se asigna los datos correspondientes al precio de cada categoría de combustible y los litros de combustible que contiene cada uno de estos. El precio de cada combustible se guardará en tres arrays diferentes, es decir, cada uno guardara el precio de cada categoría pero en diferentes regiones (Sur, Norte, Centro). Finalmente en un array dinámico se guardará la cantidad de litros correspondiente a cada categoría y estación.

Algunos de los métodos:

fijarPreciosCom () - Se pide al usuario la región en la cual se desea realizar los cambios de los precios y los nuevos precios en cada categoría de combustible. Se actualiza el array correspondiente a la región.

asignarLitros () - Se genera un numero aleatorio entre 100 y 200 para cada una de las categorías de combustible. Se actualiza el array dinámico con los nuevos datos.

verificarFugas () - Para verificar las fugas se pide al usuario el código de la estación a verificar. Se leen los datos de un archivo **"archivoRegistros.txt"** y se realiza la suma de los litros vendidos en las diferentes categorías, verificando que estén asociados a la estación seleccionada. Se lee el array dinámico y se suman los litros que restan allí. Finalmente el resultado de la suma se resta a la cantidad de litros correspondiente al tanque central y con una regla de tres se calcula el porcentaje.

Getters y Setters - La clase cuenta con varios getter y setter, los cuales son usados para inicializar las variables de los precios y litros correspondientes a cada categoría de una forma más sencilla.

Clase Surtidor.

Cada surtidor esta identificado por el código correspondiente a la estación que pertenece. Cuenta con una ubicación dentro de la estación, un modelo que lo diferencia de los demás surtidores y un estado (Activo o Inactivo). Algunos métodos como:

eliminarSurtidor () - Para eliminar un surtidor se pide al usuario el código de la estación y el modelo del surtidor. Se verifican los datos de un archivo **"datosSurtidores.txt"**, allí se encuentra el modelo, el código de la estación a la que pertenece y el estado del surtidor. Si el surtidor existe se realiza la eliminación de este y se actualizan los datos del archivo.

agregarSurtidor () - Se pide al usuario el código de la estación, el modelo y la ubicación dentro de la estación del surtidor. Finalmente el surtidor es agregado al archivo **"datosSurtidores.txt"** y se actualiza el numero de surtidores de la estación.

mostrarHistorial () - Cada una de las ventas que se realice en cada surtidor son registradas en el archivo **"archivoRegistros.txt"** precedido por la clase **transacción**; así pues, estos datos son leídos verificando el modelo del surtidor. Finalmente se imprimen los datos en un archivo.txt de una manera organizada.

Clase Transaccion.

La venta de cada surtidor es registrada en el archivo **"archivoRegistros.txt"** con sus respectivos datos como: el código de la estación, el modelo del surtidor, la cantidad de litros, la categoría del combustible, el dinero recibido, la fecha, la hora y los datos del cliente nombre y documento.

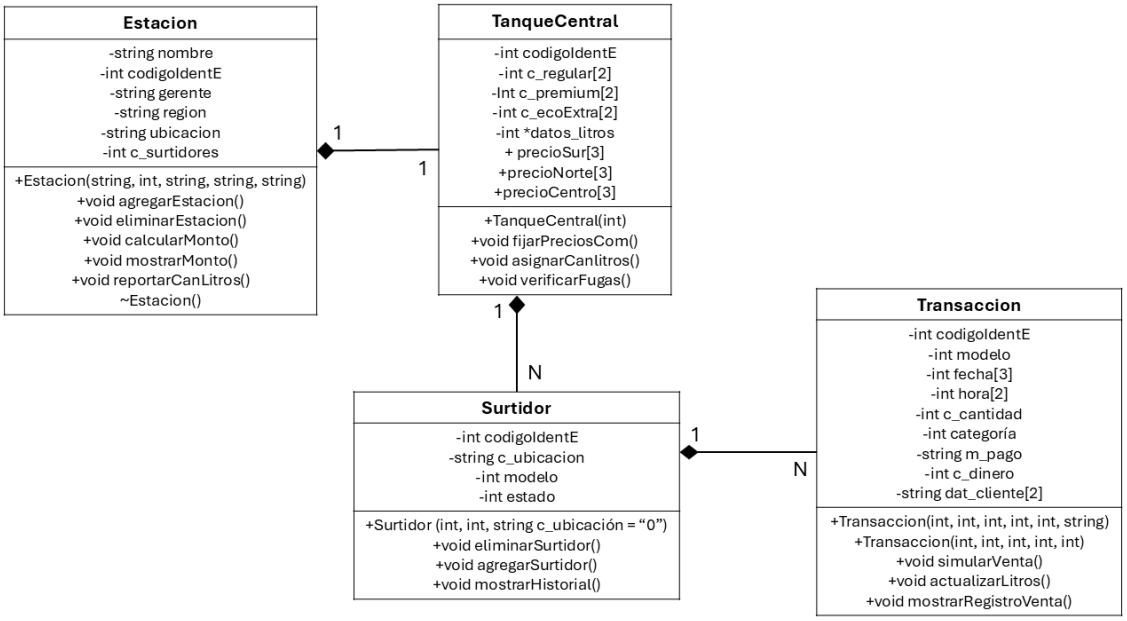
Métodos:

actulizarLitros () - Cada que se realice una transacción la cantidad de litros vendida debe ser restada al array dinámico que contiene los litros de cada estación verificando el código de la estación y categoría de combustible.

simularVenta () - Se asigna una estación al azar y un surtidor al azar de dicha estación. Para la venta de los litros se genera un numero al azar entre 3 y 20 y se pide por consola los datos del usuario. Finalmente con el método actulizarLitros se actualizan los datos.

mostrarRegistroVenta () - Siempre que se realice una transacción es necesario mostrar al usuario el registro de la venta, en este caso los datos del método simularVenta.

Se muestra el diagrama de clases:



Nota: Si el diagrama no es claramente visible, la imagen se encuentra disponible en GitHub.

Organización de los datos.

Para el guardado de los datos se consideró usar el manejo de archivos. Se muestra un ejemplo de cómo se verían los datos de una Estación considerando los siguientes identificadores:

Datos de cada estación:

```
#
-estacion:Juana
--codigo:131
---gerente:Juan
----region:Norte
-----surtidores:5
------ubicacion:122ø98495gh
#
```

Archivo **datosEstacion.txt**

(#) Indica que en la próxima línea inician cada uno de los datos de la estación.

(-) Indica que tipo de dato es el que corresponde. Un solo (-) indicara que corresponde al nombre de la estación, dos (--) indicara que corresponde al código de la estación es decir, el número de guiones indica un dato diferente de la estación.

(:) Indica que después de los dos puntos se encuentran cada uno de los valores.

Datos transacciones:

```
#123/1234f/Premium/9/20000/Efectivo/Sebastian/10075689/1-3-24/12:40/
#131/1234/Regular/15/34000/TDebito/Jorge/1223345/16-5-24/01:20/
#101/12345/Premium/10/28000/TCredito/Maria/2334577/23-8-24/03:10/
#567/12345/ecoExtra/13/45000/Efectivo/Santiago/234556789/10-4-24/13:23/
```

Archivo **archivoRegistros.txt**

(#) Indica que ha iniciado una nueva línea.

(/) Separa cada uno de los datos en el siguiente orden: código de la estación, modelo del surtidor, tipo de gasolina, litros surtidos, dinero cobrado, método de pago, nombre del cliente, documento del cliente, fecha y hora.

Datos surtidores:

```
101/3781/4567/activo
345/3710/78/activo
567/7812/9073/inactivo
```

Archivo **datosSurtidores.txt**

(/) Separa cada uno de los datos en el siguiente orden: código de la estación, modelo del surtidor, ubicación en la estación y estado.

Array litros categoría

Por otro lado el array dinámico el cual contiene la cantidad de litros de cada categoría de combustible se vería de la siguiente forma:

{323, 120, 160, 200, 456, 123, 167, 100, 342, 105, 200, 189, …}

Los datos de cada estación serán tomados por 4 valores del array por ejemplo**: {323, 120, 160, 200}** el primer valor 323 corresponde al código de la estación, el segundo valor 120 corresponde a la cantidad de combustible Regular, el tercero 160 corresponde a la cantidad de combustible Premium y el cuarto 200 a ecoExtra. Así, al encontrar un valor correspondiente al código de una estación, se sabe que los tres valores siguientes corresponden a la cantidad de litros de cada categoría.

Cabe resaltar que cada uno de los valores de los litros será un numero aleatorio entre 100 y 200. No obstante, al agregar una nueva estación solo es actualizar el array con el código nuevo y los 3 valores de los litros de cada categoría.

Problemas

La cantidad de datos que el programa debe procesar es realmente grande, por lo que pueden surgir problemas de consumo de memoria y al plantear la función que diferencie entre los caracteres para capturar los datos puede ser muy extensa. El archivo archivoRegistros.txt es donde se almacenan cada una de las transacciones y esto puede tener una gran cantidad de registros, ya que muchos clientes pueden llegar en un solo día. Por otro lado, para calcular la cantidad de litros vendidos, se recorre el archivo por completo y los datos correspondientes a los litros se almacenan en un array lo que puede ocasionar un déficit de memoria.

Ahora bien, podemos usar un array dinámico donde se guarden estos valores y liberar la memoria cuando la operación de calcular los litros haya finalizado. Y así, se podría hacer con los demás archivos.

No obstante, el hecho de leer y modificar archivos constantemente consume memoria. Teniendo en cuenta esto, se opto por usar un arreglo dinámico para capturar la cantidad de combustible de cada categoría y estación y liberar esa memoria cuando se requiera. Con esto se evitaría el uso de tantos archivos posibles para almacenar los datos y poder hacer cuentas un poco más fáciles. Sin embargo, el programa en algún momento puede cerrar por autorización del usuario, lo que para los datos temporales como los arreglos se perderían completamente y al momento de volver a iniciar el programa estos valores serán diferentes a los anteriores.

Nota: Al momento de avanzar en la codificación del programa pueden surgir otros problemas que requieran cambiar algún diseño de las clases o funcionalidades. También, se podrían hacer cambios en la lectura de los datos para mejorar el programa. Por lo que, las soluciones planteadas están sujetas a cambios.