DESAFÍO II: SISTEMA DE COMERCIALIZACIÓN DE COMBUSTIBLE TERMAX

INFORMÁTICA II 2024-2

UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA

JUAN SEBASTIAN GALEANO ARISTIZABAL 1006110753
MARIA ADELAIDA ANGEL MONTOYA 1001367810

Análisis del problema:

El problema planteado consistía en la construcción de un programa que permitiera a la empresa de combustible TerMax, gestionar sus puntos de venta (estaciones de servicio) a nivel nacional, teniendo cada una de estas características propias como nombre, código identificador, gerente y ubicación geográfica y conteniendo un tanque central y un numero de máquinas surtidoras de combustible encargadas de la venta de forma particular y estas a su vez poseen atributos propios como código identificador y modelo, con la capacidad de surtir los tres tipos de combustible (Regular, Premium o Ecoextra). Principalmente se pide que el programa contenga un menú principal con opciones como agregar/eliminar una estación, agregar/eliminar un surtidor, establecer los precios para cada combustible según la región donde se encuentre la estación (Norte, Sur y Centro), simular una venta y la posibilidad de verificar si hubo fugas, entre otras cosas.

Solución propuesta:

En un primer momento, se consideró implementar seis clases, una red nacional, estaciones de servicio, surtidores, tanque central, transacciones y clientes. Se hicieron varios esquemas a modo de guía para desglosar el problema de la forma adecuada, llegando a la conclusión de que lo mas óptimo era tener tres clases principales, Termax, Estación de servicio y surtidores, pensando en la clase Termax como la que contenga la información de las otras, por ende, la mayoría de los datos de salida provienen de esta clase, donde se tienen los metódos que darán base al menú pedido.(La solución implementada se basó en la programación orientada a objetos).

. **Diagrama de clases**

Se dispuso de un diagrama de clases con notación UML para registrar debidamente la solución planteada.

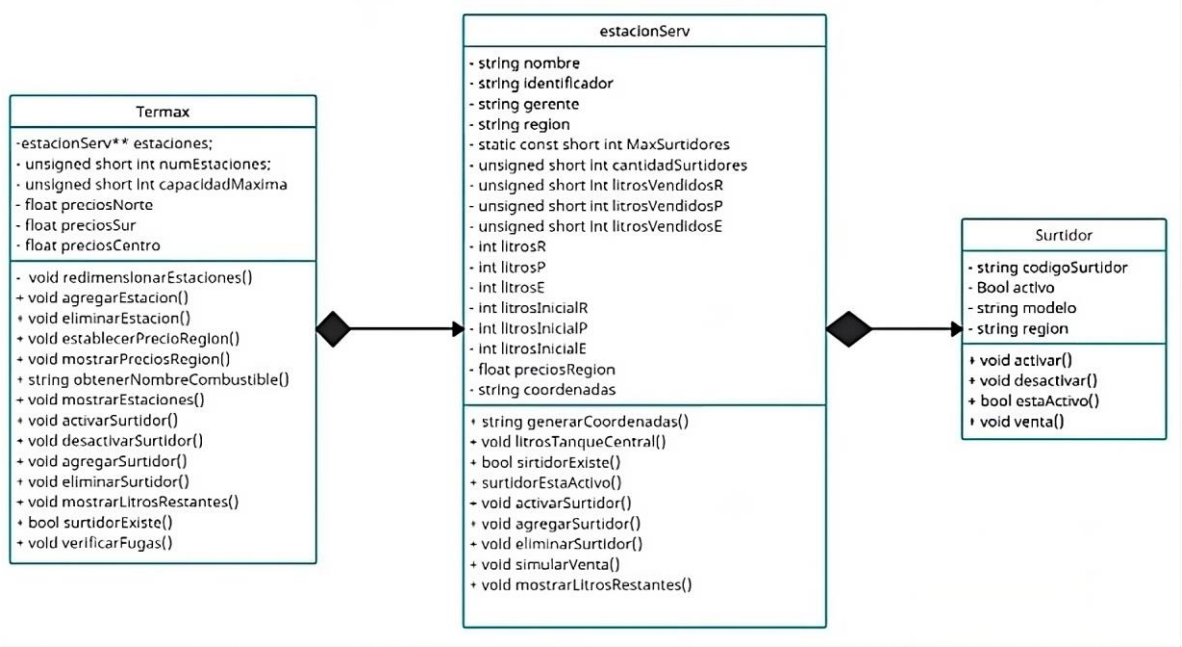

Fig 1. Diagrama de clases notación UML

Algoritmos implementados:

Métodos fundamentales

```
void venta(int cantidad, char tipoCombustible, int &litrosR, int &litrosP, int &litrosE,
           short unsigned int &vendidosR, short unsigned int &vendidosP, short unsigned int &vendidosE,
           float &costo,
           const float preciosR[3]) {

    int tipoCombustibleI;
    if (tipoCombustible == 'R') tipoCombustibleI = 0;
    else if (tipoCombustible == 'P') tipoCombustibleI = 1;
    else if (tipoCombustible == 'E') tipoCombustibleI= 2;
    else {
        cout << "Tipo de combustible no valido." << endl;
        return;
    }
    // Calcular litros disponibles
    int &litrosDisponibles = (tipoCombustible == 'R') ? litrosR : (tipoCombustible == 'P') ? litrosP : litrosE;
    int litrosAUsar = min(litrosDisponibles, cantidad); // Usar lo que haya disponible

    if (litrosAUsar < cantidad) {
        cout << "No hay suficientes litros. Se venderan solo " << litrosAUsar << " litros.\n";
    }

    // Calcular costo
    costo += litrosAUsar * preciosR[tipoCombustibleI];

    // Actualizar litros vendidos
    if (tipoCombustible == 'R') {
        vendidosR += litrosAUsar;
    } else if (tipoCombustible == 'P') {
        vendidosP += litrosAUsar;
    } else if (tipoCombustible == 'E') {
        vendidosE += litrosAUsar;
    }

    // Actualizar litros restantes
    litrosDisponibles -= litrosAUsar; // Actualiza litros restantes

    cout << "Venta realizada de " << litrosAUsar << " litros de " << tipoCombustible << "." << endl;
    cout << "Costo total de la venta: " << costo << endl;
}
```

Método de la clase “surtidor”, lleva a cabo el proceso de la venta, tomando los precios según tipo de combustible y actualizando la cantidad de litros vendidos discriminando por tipo, también calcula el precio final.

```
void simularVenta(float* PreciosR) {
    if (cantidadSurtidores == 0) {
        cout << "No hay surtidores disponibles para la venta.\n";
        return;
    }
    if(surtidorestaActivo() == false){
        cout << "No tiene surtidores activos para la venta.\n";
        return;
    }
    for (int i = 0; i < 3; i++) {
        preciosRegion[i] = PreciosR[i]; // Copia los precios
    }

    int surtidorElegido;
    do {
        surtidorElegido = rand() % cantidadSurtidores;
    } while (!surtidores[surtidorElegido].estaActivo());

    unsigned short int cantidad = 3 + rand() % 18; // Genera una cantidad aleatoria
    char tipoCombustible;
    cout << "Simulando venta en surtidor " << surtidores[surtidorElegido].getCodigo() << endl;
    cout << "Ingrese el tipo de combustible (R para Regular, P para Premium, E para EcoExtra): ";
    cin >> tipoCombustible;

    float costo = 0.0;  // Inicializa costo

    // Llama a la funcion de venta litrosVendidosR;
    surtidores[surtidorElegido].venta(cantidad, tipoCombustible, litrosR, litrosP, litrosE,
    litrosVendidosR, litrosVendidosP, litrosVendidosE, costo, PreciosR);
}
```

Método parte de la clase “estacionServ” que se encarga de las principales condiciones para poder simular la venta, como elegir aleatoriamente uno de los surtidores asociados a la estación y verificar si este está activo, buscar el precio del combustible solicitado e invocar el método “venta” perteneciente a la clase surtidores.

```
void redimensionarEstaciones() {
    unsigned short int nuevaCapacidad = capacidadMaxima * 2;
    estacionServ** nuevoArreglo = new estacionServ*[nuevaCapacidad];

    for (int i = 0; i < numEstaciones; i++) {
        nuevoArreglo[i] = estaciones[i];
    }

    delete[] estaciones;
    estaciones = nuevoArreglo;
    capacidadMaxima = nuevaCapacidad;
}
```

Al estar las estaciones de servicio almacenadas en un arreglo dinamico, si se desa agregar una nueva este arreglo debe ser ampliado, esta función realiza una copia y elimina el arreglo para duplicar su tamaño, luego regresa la copia a la variable original.

```
void eliminarEstacion(string id) {
    for (int i = 0; i < numEstaciones; i++) {
        if (estaciones[i]->getIdentificador() == id) {
            if (estaciones[i]->surtidorestaActivo()){
                cout << "La estacion no puede ser eliminada porque tiene surtidores activos. \n";
                return;
            }
            else {
                delete estaciones[i];
                for (int j = i; j < numEstaciones - 1; j++) {
                    estaciones[j] = estaciones[j + 1];
                }
                numEstaciones--;
                cout << "Estacion con ID " << id << " eliminada.\n";
                return;
            }

        }
    }
    cout << "La estacion no fue encontrada.\n";
}
```

Para eliminar una estación primero se debe verificar que no tenga surtidores activos adscritos a ella, este método recorre el arreglo de estaciones, al hallarla recurre a un método de otra clase para saber si tiene surtidores activos, de lo contrario se elimina la estación.

```
void establecerPrecioRegion(string region, int tipoCombustible, float precio) {
    if (region == "Norte") {
        preciosNorte[tipoCombustible] = precio;
        cout << "Precio de " << obtenerNombreCombustible(tipoCombustible) << " en la region Norte establecido en: " << precio << endl;
    } else if (region == "Sur") {
        preciosSur[tipoCombustible] = precio;
        cout << "Precio de " << obtenerNombreCombustible(tipoCombustible) << " en la region Sur establecido en: " << precio << endl;
    } else if (region == "Centro") {
        preciosCentro[tipoCombustible] = precio;
        cout << "Precio de " << obtenerNombreCombustible(tipoCombustible) << " en la region Centro establecido en: " << precio << endl;
    } else {
        cout << "Region no valida." << endl;
    }
}
```

Los precios en la red nacional se dividen según la región donde se encuentre ubicada la estación, Norte, Sur o Centro, este método recibe la región y según esto busca el precio establecido para ella, contiene validación para el ingreso de algo diferente a una de las regiones indicadas.

```
void mostrarPreciosRegion(string region) {
    if (region == "Norte") {
        cout << "Precios en la region Norte:\n";
        for (int i = 0; i < 3; i++) {
            cout << obtenerNombreCombustible(i) << ": " << preciosNorte[i] << " por litro\n";
        }
    } else if (region == "Sur") {
        cout << "Precios en la region Sur:\n";
        for (int i = 0; i < 3; i++) {
            cout << obtenerNombreCombustible(i) << ": " << preciosSur[i] << " por litro\n";
        }
    } else if (region == "Centro") {
        cout << "Precios en la region Centro:\n";
        for (int i = 0; i < 3; i++) {
            cout << obtenerNombreCombustible(i) << ": " << preciosCentro[i] << " por litro\n";
        }
    } else {
        cout << "Region no valida." << endl;
    }
}
```

El precio del combustible puede ser modificado en cualquier momento, este método se encarga de, según la región en la que se desean modificar los precios, modifica en los arreglos correspondientes los precios para las tres categorías de combustible dadas.

```
void verificarFugas() {
    for (int i = 0; i < numEstaciones; i++) {
        unsigned short int litrosE = estaciones[i]->getLitrosE();
        unsigned short int litrosR = estaciones[i]->getLitrosR();
        unsigned short int litrosP = estaciones[i]->getLitrosP();
        unsigned short int litrosVendidosE = estaciones[i]->getLitrosVendidosE();
        unsigned short int litrosVendidosR = estaciones[i]->getLitrosVendidosR();
        unsigned short int litrosVendidosP = estaciones[i]->getLitrosVendidosP();
        unsigned short int litrosInicialE = estaciones[i]->getLitrosInicialE();
        unsigned short int litrosInicialR = estaciones[i]->getLitrosInicialR();
        unsigned short int litrosInicialP = estaciones[i]->getLitrosInicialP();

        // Calculamos el total restante más el vendido para cada tipo de combustible
        float totalRegular = litrosR + litrosVendidosR;
        float totalPremium = litrosP + litrosVendidosP;
        float totalEcoExtra = litrosE + litrosVendidosE;

        // Comprobamos si es al menos el 95% de los litros iniciales
        bool fugaRegular = totalRegular < 0.95 * litrosInicialR;
        bool fugaPremium = totalPremium < 0.95 * litrosInicialP;
        bool fugaEcoExtra = totalEcoExtra < 0.95 * litrosInicialE;


        // Imprimimos resultados
        cout << "Verificacion de fugas para estacion:"<< estaciones[i]->getNombre() << ".\n";
        ;
        cout << "Regular: " << (fugaRegular ? "FUGA DETECTADA" : "SIN FUGA.") << endl;
        cout << "Premium: " << (fugaPremium ? "FUGA DETECTADA" : "SIN FUGA.") << endl;
        cout << "EcoExtra: " << (fugaEcoExtra ? "FUGA DETECTADA" : "SIN FUGA.\n") << endl;
    }
}
```

El método para la verificación de fugas es de gran importancia debido a que forma parte directa del menú, este método se encarga de, según los litros de combustible fijados

inicialmente para cada estación, rectifica que los litros vendidos más los litros restantes sean al menos el 95% de los primeramente mencionados.