**Informe desafio ll**

**Samuel villa Carmona**

**Augusto Salazar**

**Aníbal Guerra**

**Docentes**

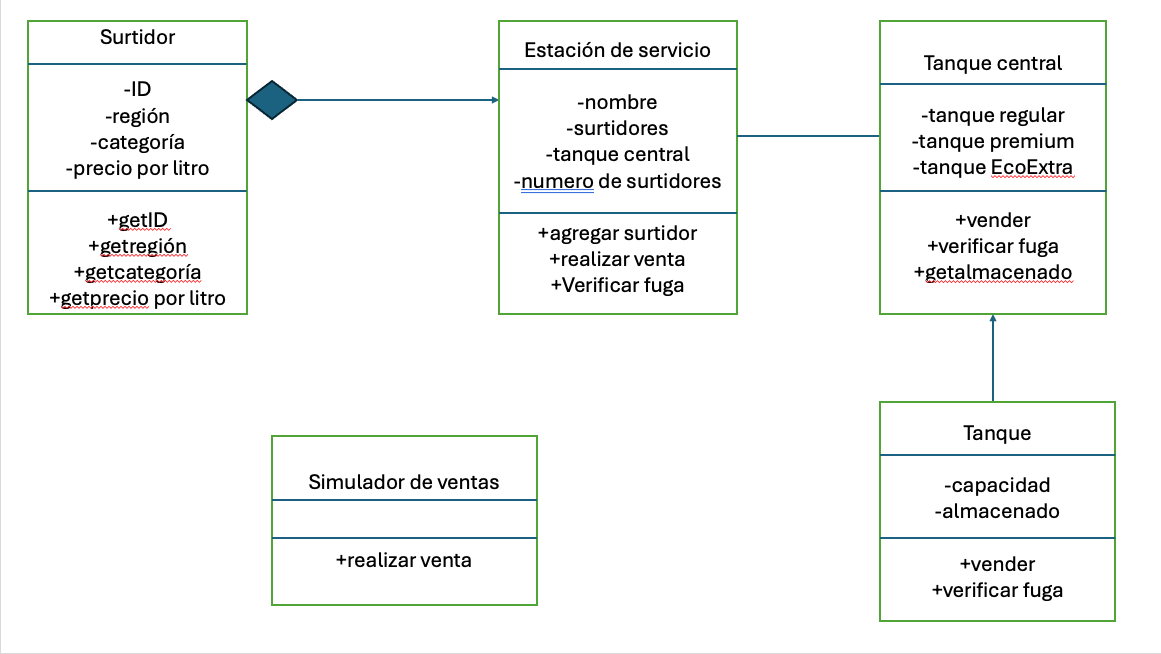
**Universidad de Antioquia**

**Informática ll**

**15 de oct. de 24**

**Informe**

1. El problema planteado consiste en diseñar un sistema que gestione una red nacional de estaciones de servicio para TerMax, permitiendo la gestión de estaciones, surtidores, y ventas de combustible. Esta red debe ser eficiente y flexible, considerando las diversas operaciones que ocurren en las estaciones de servicio, como la venta diaria de combustible, la actualización de inventarios, y la verificación de fugas.



El programa empieza definiendo las clases: Surtidor quien es el que atiende y vende la gasolina, Estación de servicio la cual sería como la bomba donde está la gasolina, el surtidor y además verifica la fuga de gasolina, Tanque central el cual tiene almacenado los 3 tanques de gasolina (regular, premium y EcoExtra), Tanque el cual es el que almacena solo un tipo de gasolina y tiene una capacidad de almacenamiento, y por ultimo Simulador de ventas, el cual simula una venta.

#include <iostream>

#include <string>

#include <cstdlib>

#include <ctime>

using namespace std;

class Surtidor {

private:

int id;

string region;

string categoria;

float precioPorLitro;

public:

Surtidor(int id, const string& region, const string& categoria, float precio)

: id(id), region(region), categoria(categoria), precioPorLitro(precio) {}

float getPrecioPorLitro() const {

return precioPorLitro;

}

int getId() const {

return id;

}

string getRegion() const {

return region;

}

string getCategoria() const {

return categoria;

}

};

class Tanque {

private:

float capacidad;

float almacenado;

public:

Tanque(float capacidad) : capacidad(capacidad), almacenado(capacidad) {}

void vender(float litros) {

almacenado -= litros;

}

float getAlmacenado() const {

return almacenado;

}

float getCapacidad() const {

return capacidad;

}

bool verificarFuga(float litrosVendidos) const {

return (almacenado + litrosVendidos) >= (0.95 \* capacidad);

}

};

class TanqueCentral {

private:

Tanque\* tanqueRegular;

Tanque\* tanquePremium;

Tanque\* tanqueEcoExtra;

public:

TanqueCentral(float capacidadRegular, float capacidadPremium, float capacidadEcoExtra) {

tanqueRegular = new Tanque(capacidadRegular);

tanquePremium = new Tanque(capacidadPremium);

tanqueEcoExtra = new Tanque(capacidadEcoExtra);

}

~TanqueCentral() {

delete tanqueRegular;

delete tanquePremium;

delete tanqueEcoExtra;

}

void vender(const string& categoria, float litros) {

if (categoria == "Regular") {

tanqueRegular->vender(litros);

} else if (categoria == "Premium") {

tanquePremium->vender(litros);

} else if (categoria == "EcoExtra") {

tanqueEcoExtra->vender(litros);

}

}

bool verificarFugas(const string& categoria, float litrosVendidos) {

if (categoria == "Regular") {

return tanqueRegular->verificarFuga(litrosVendidos);

} else if (categoria == "Premium") {

return tanquePremium->verificarFuga(litrosVendidos);

} else if (categoria == "EcoExtra") {

return tanqueEcoExtra->verificarFuga(litrosVendidos);

}

return true;

}

float getAlmacenado(const string& categoria) const {

if (categoria == "Regular") {

return tanqueRegular->getAlmacenado();

} else if (categoria == "Premium") {

return tanquePremium->getAlmacenado();

} else if (categoria == "EcoExtra") {

return tanqueEcoExtra->getAlmacenado();

}

return 0;

}

};

class EstacionServicio {

private:

string nombre;

Surtidor\*\* surtidores;

TanqueCentral\* tanqueCentral;

int numSurtidores;

public:

EstacionServicio(const string& nombre, int numSurtidores, float capacidadRegular, float capacidadPremium, float capacidadEcoExtra)

: nombre(nombre), numSurtidores(numSurtidores) {

surtidores = new Surtidor\*[numSurtidores];

tanqueCentral = new TanqueCentral(capacidadRegular, capacidadPremium, capacidadEcoExtra);

}

void agregarSurtidor(int index, Surtidor\* surtidor) {

if (index >= 0 && index < numSurtidores) {

surtidores[index] = surtidor;

}

}

Surtidor\* obtenerSurtidor(int index) {

if (index >= 0 && index < numSurtidores) {

return surtidores[index];

}

return nullptr;

}

Surtidor\* obtenerSurtidorAleatorio() {

int index = rand() % numSurtidores;

return obtenerSurtidor(index);

}

void realizarVenta(const string& metodoPago) {

Surtidor\* surtidor = obtenerSurtidorAleatorio();

if (!surtidor) {

cout << "No se pudo obtener un surtidor." << endl;

return;

}

int litros = rand() % 18 + 3;

float total = litros \* surtidor->getPrecioPorLitro();

tanqueCentral->vender(surtidor->getCategoria(), litros);

cout << "Transaccion realizada en " << nombre << endl;

cout << "Surtidor ID: " << surtidor->getId() << endl;

cout << "Region: " << surtidor->getRegion() << endl;

cout << "Categoria: " << surtidor->getCategoria() << endl;

cout << "Litros vendidos: " << litros << endl;

cout << "Total a pagar: " << total << endl;

cout << "Metodo de pago: " << metodoPago << endl;

}

bool verificarFugas(const string& categoria, int litrosVendidos) {

return tanqueCentral->verificarFugas(categoria, litrosVendidos);

}

string getNombre() const {

return nombre;

}

~EstacionServicio() {

delete[] surtidores;

delete tanqueCentral;

}

};

class SimuladorVentas {

public:

static void realizarVenta(EstacionServicio& estacion) {

string metodosPago[] = {"Efectivo", "Tarjeta de Debito", "Tarjeta de Credito"};

string metodoPago = metodosPago[rand() % 3];

estacion.realizarVenta(metodoPago);

}

};

int main() {

srand(static\_cast<unsigned int>(time(0)));

EstacionServicio estacion("Estacion Central", 6, 1000.0, 1000.0, 1000.0);

estacion.agregarSurtidor(0, new Surtidor(1, "Norte", "Regular", 6.000));

estacion.agregarSurtidor(1, new Surtidor(2, "Centro", "Premium", 5.000));

estacion.agregarSurtidor(2, new Surtidor(3, "Sur", "EcoExtra", 5.500));

estacion.agregarSurtidor(3, new Surtidor(4, "Norte", "Premium", 5.300));

estacion.agregarSurtidor(4, new Surtidor(5, "Centro", "EcoExtra", 5.600));

estacion.agregarSurtidor(5, new Surtidor(6, "Sur", "Regular", 6.100));

estacion.agregarSurtidor(6, new Surtidor(7, "Norte", "EcoExtra", 5.400));

estacion.agregarSurtidor(7, new Surtidor(8, "Centro", "Regula", 5.900));

estacion.agregarSurtidor(8, new Surtidor(9, "Sur", "Premium", 5.100));

SimuladorVentas::realizarVenta(estacion);

if (!estacion.verificarFugas("Regular", 10)) {

cout << "Se detecto una fuga en la estacion: " << estacion.getNombre() << endl;

} else {

cout << "No se detectaron fugas en la estacion: " << estacion.getNombre() << endl;

}

return 0;

}

1. Un problema que tuve con el desarrollo del desafio fue con la comprobación de fuga.