Análisis y planteamiento de la solución del desafío II

Faith Alvarado Silva
Carla Zapata Valencia

Profesor: Aníbal Guerra
Informática II

2024-2

ÍNDICE

Introducción

El presente informe detalla el desarrollo de un sistema de gestión para una red de estaciones de servicio de combustible. El sistema debe gestionar de forma eficiente las estaciones de servicio, las máquinas surtidoras y las respectivas ventas de combustible.

Cada estación de servicio posee un nombre, un código identificador, un gerente, una región y una ubicación geográfica (expresada en coordenadas GPS). Además, cada estación tiene un tanque central donde se almacenan separadamente las 3 categorías de combustible disponible para vender (Regular, Premium y EcoExtra). A este tanque central se conectan entre 2 a 12 máquinas surtidoras desde las cuales se venderá directamente a los vehículos. La estación de servicio puede estar subdividida en varias islas, en las cuales se agrupan varios puntos surtidores de forma contigua. Cada punto surtidor registra individualmente todas las ventas realizadas en el día, considerando: la fecha, hora, cantidad y categoría de combustible, método de pago (Efectivo, TDebito, TCrédito), número de documento del cliente y la cantidad correspondiente de dinero.

Funcionalidades

El sistema ofrece una variedad de funcionalidades, estas son:

Gestión de la red: La red nacional se encargará de las siguientes funcionalidades.

1. Agregar estaciones de servicio.
2. Eliminar una estación de servicio de la red nacional (solo si las máquinas surtidoras están desactivadas).
3. Calcular el monto total de las ventas en cada estación de servicio del país, discriminado por categoría de combustible (Regular, Premium, EcoExtra).
4. Fijar los precios del combustible para toda la red.

Gestión de estaciones de servicio: Cada una de las estaciones tiene las siguientes funcionalidades.

1. Agregar o eliminar un surtidor a una estación de servicio.
2. Activar o desactivar un surtidor de una estación de servicio.
3. Consultar el histórico de transacciones de cada punto surtidor de la estación de servicio.
4. Reportar la cantidad de litros vendida según cada categoría de combustible.
5. Simular una venta de combustible.
6. Asignar la capacidad del tanque de suministro al configurar la estación, con un valor aleatorio entre 100 y 200 litros para cada una de las categorías.

Sistema nacional de verificación de fugas: El programa debe permitir detectar la existencia de fugas de combustible en cualquiera de las estaciones del país. Para ello, según cada categoría de combustible, debe verificarse que lo vendido más lo almacenado en el tanque de la estación de servicio corresponda a más del 95% de la capacidad original del tanque. Esta verificación opera sobre una estación de servicio específica que sea seleccionada entre todas las de la red.

Simulación de ventas: Dada una estación de servicio, se asigna de forma aleatoria uno de los puntos surtidores activos para que gestione la transacción. Además, para cada venta se solicita de forma aleatoria entre 3 y 20 lt de gasolina. Una vez que se ha bombeado la gasolina deberán mostrarse los datos de la transacción.

Se tendrá en cuenta que el precio por litro de cada categoría de combustible depende de la región a la que pertenece la estación. Por otra parte, sólo se consideran tres regiones: Norte, Centro y Sur. Dicho valor es temporal, y puede variar entre días.

Planteamiento del diagrama de clases

La Figura 1 presenta el diagrama de clases donde se establecen las relaciones entre clases y su respectiva cardinalidad. Cada una de estas clases tiene sus propias responsabilidades.

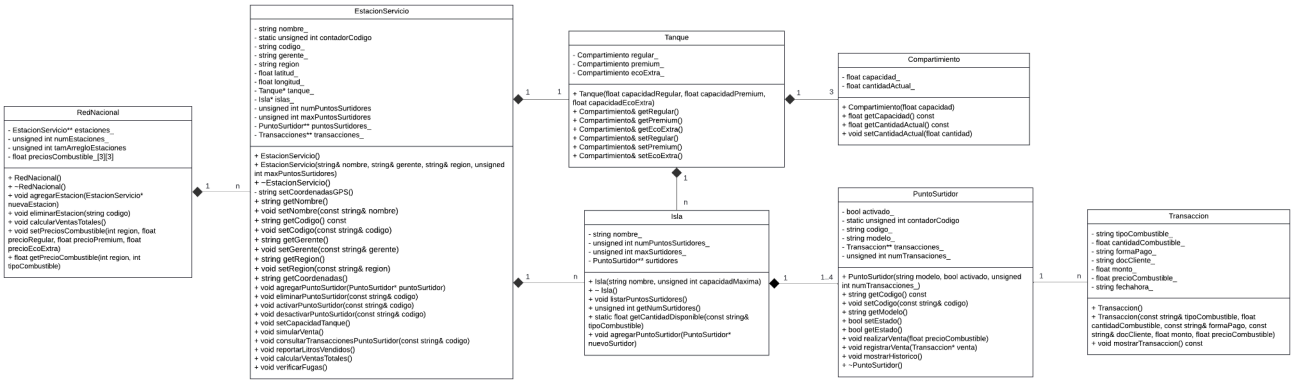

Figura 1. Diagrama de clases

A continuación, de la figura 2 a la figura 8 se muestra con más detalle cada una de las clases del diseño del sistema.

Clase RedNacional

En la figura 2 representa la red de todas las estaciones de servicio gestionadas por la empresa. Esta clase permite agregar y eliminar estaciones, así como gestionar precios y calcular ventas totales.

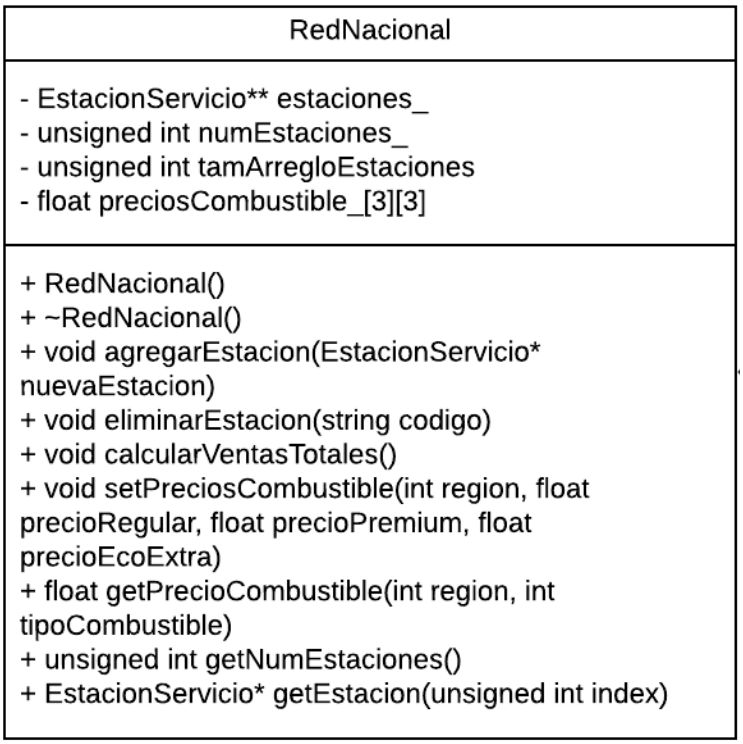

Figura 2. Clase RedNacional

Atributos:

- EstacionServicio** estaciones_: Arreglo dinámico de punteros a objetos EstacionServicio, que almacena todas las estaciones en la red nacional.
- unsigned int numEstaciones_: Contador del número actual de estaciones en la red, que no puede ser negativo.
- unsigned int tamArregloEstaciones: Tamaño actual del arreglo de estaciones. Se inicializa con un valor predeterminado y puede crecer según sea necesario.
- float preciosCombustible_[3][3]: Matriz que almacena los precios de combustible para las tres categorías (Regular, Premium, EcoExtra) en tres regiones (Norte, Centro, Sur).

Métodos:

- RedNacional(): Constructor que inicializa los atributos y establece los precios de combustible por región.

- ~RedNacional(): Destructor que libera la memoria asignada a cada estación y al arreglo de estaciones.
- void agregarEstacion(EstacionServicio* nuevaEstacion): Agrega una nueva estación al arreglo, aumentando su capacidad si es necesario. También genera el código de la nueva estación y asigna capacidad al tanque.
- void eliminarEstacion(string codigo): Busca una estación por su código y la elimina del arreglo, liberando la memoria correspondiente.
- void setPreciosCombustible(int region, float precioRegular, float precioPremium, float precioEcoExtra): Establece los precios de combustible para una región específica.
- float getPrecioCombustible(int region, int tipoCombustible): Obtiene el precio de combustible para una región y tipo específico (no se muestra en el código, pero se infiere que está presente en la declaración de la clase).
- unsigned int getNumEstaciones(): Retorna la cantidad actual de estaciones de servicio en la red.
- EstacionServicio* getEstacion(unsigned int index): Obtiene un puntero a la estación de servicio correspondiente al índice dado, si es válido.

Encapsulación: Los atributos son privados para proteger la integridad de los datos y solo se puede interactuar con ellos a través de métodos específicos.

Clase EstacionServicio

La clase EstacionServicio de la figura 3 representa una estación de servicio individual que contiene información básica como nombre, código, gerente, región, coordenadas GPS, tanque y un arreglo dinámico de puntos surtidores. Esta clase es responsable de administrar las ventas, los surtidores y el tanque de combustible.

Atributos:

- string nombre_: Almacena el nombre de la estación de servicio.
- static unsigned int contadorCodigo: Contador estático para generar códigos únicos para cada estación de servicio.
- string codigo_: Código identificador único de la estación de servicio.
- string gerente_: Almacena el nombre del gerente de la estación de servicio.
- string region_: Representa la región geográfica donde se ubica la estación de servicio.
- float latitud_: Almacena la latitud de la ubicación geográfica de la estación.
- float longitud_: Almacena la longitud de la ubicación geográfica de la estación.
- Tanque* tanque_: Puntero a un objeto de tipo Tanque, que representa el tanque central de combustible de la estación.
- Isla* islas_: Puntero a un arreglo de islas asociadas a la estación de servicio.

- unsigned int numPuntosSurtidores: Contador del número actual de puntos surtidores en la estación.
- unsigned int maxPuntosSurtidores: Almacena el número máximo permitido de puntos surtidores en la estación.
- PuntoSurtidor** puntosSurtidores_: Puntero a un arreglo dinámico de objetos de tipo PuntoSurtidor, que representan los puntos de venta de combustible.
- Transacciones** transacciones_: Puntero a un arreglo dinámico de objetos de tipo Transaccion, que registran todas las transacciones realizadas en la estación.

Figura 3. Clase EstacionServicio

Métodos:

- EstacionServicio(): Constructor por defecto que inicializa la estación de servicio sin parámetros.

- EstacionServicio(string& nombre, string& gerente, string& region, unsigned int maxPuntosSurtidores): Constructor parametrizado que inicializa los atributos de la estación de servicio con los valores proporcionados.
- ~EstacionServicio(): Destructor que libera la memoria asignada a los objetos de la estación, incluidos tanque_, islas_, puntosSurtidores_ y transacciones_.
- string setCoordenadasGPS(): Genera y establece aleatoriamente las coordenadas GPS de la estación.
- string getNombre(): Devuelve el nombre de la estación de servicio.
- void setNombre(const string& nombre): Establece el nombre de la estación.
- string getCodigo() const: Devuelve el código de la estación de servicio.
- void setCodigo(const string& codigo): Establece el código de la estación.
- string getGerente(): Devuelve el nombre del gerente de la estación.
- void setGerente(const string& gerente): Establece el nombre del gerente.
- string getRegion(): Devuelve la región de la estación.
- void setRegion(const string& region): Establece la región de la estación.
- string getCoordenadas(): Devuelve las coordenadas GPS de la estación.
- void agregarPuntoSurtidor(PuntoSurtidor* puntoSurtidor): Agrega un nuevo surtidor a la estación.
- void eliminarPuntoSurtidor(const string& codigo): Elimina un surtidor según su código.
- void activarPuntoSurtidor(const string& codigo): Activa un surtidor específico.
- void desactivarPuntoSurtidor(const string& codigo): Desactiva un surtidor específico.
- void setCapacidadTanque(): Asigna capacidades aleatorias al tanque de combustible.
- void simularVenta(): Simula la venta de combustible a través de un surtidor activo.
- void consultarTransaccionesPuntoSurtidor(const string& codigo): Consulta las transacciones de un surtidor específico.
- void reportarLitrosVendidos(): Reporta la cantidad de litros vendidos por categoría de combustible.
- void calcularVentasTotales(): Calcula las ventas totales realizadas en la estación.
- void verificarFugas(): Simula la verificación de fugas en el tanque.

Encapsulación: Los atributos son privados para evitar modificaciones no autorizadas y asegurar que la gestión se haga a través de métodos controlados.

Clase Tanque

En la figura 4 se muestra la clase Tanque que representa el tanque de combustible de una estación de servicio. Este tanque tiene asociado tres compartimentos de la

clase Compartimiento, cada uno para un tipo de combustible (Regular, Premium, EcoExtra).

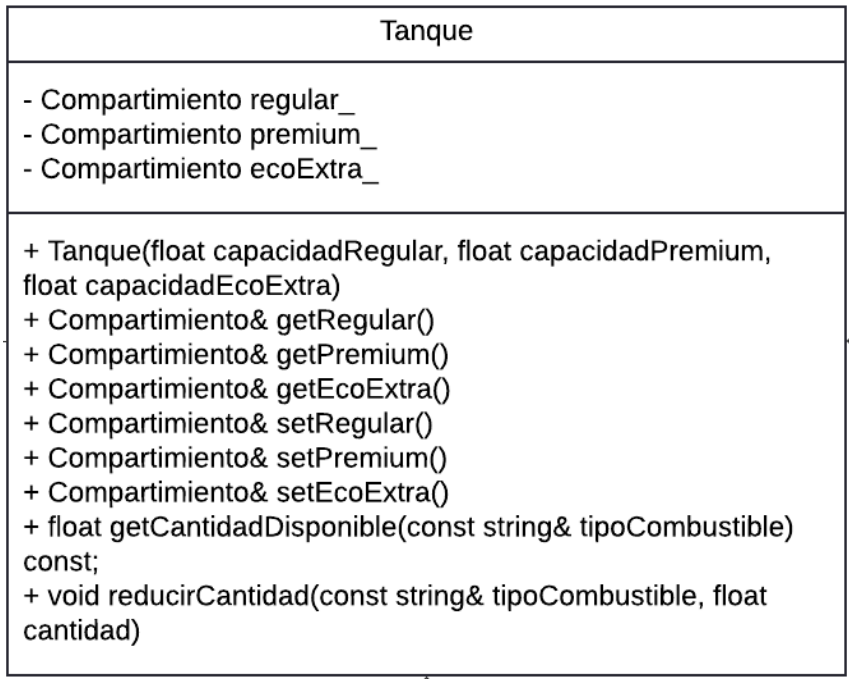

Figura 4. Clase Tanque

Atributos:

- Compartimiento regular_: Almacena un objeto de tipo Compartimiento, que representa el compartimento para combustible regular.
- Compartimiento premium_: Almacena un objeto de tipo Compartimiento, que representa el compartimento para combustible premium.
- Compartimiento ecoExtra_: Almacena un objeto de tipo Compartimiento, que representa el compartimento para combustible EcoExtra.

Métodos:

- Tanque(float capacidadRegular, float capacidadPremium, float capacidadEcoExtra): Constructor que inicializa los compartimientos del tanque con las capacidades especificadas para cada tipo de combustible.
- Compartimiento& getRegular(): Devuelve una referencia al compartimiento de combustible regular.
- Compartimiento& getPremium(): Devuelve una referencia al compartimiento de combustible premium.
- Compartimiento& getEcoExtra(): Devuelve una referencia al compartimiento de combustible EcoExtra.

- Compartimiento& setRegular(): Se espera que configure o actualice el compartimiento regular.
- Compartimiento& setPremium(): similar a setRegular(), destinado a configurar o actualizar el compartimiento premium.
- Compartimiento& setEcoExtra(): tendría el mismo propósito que setRegular() y setPremium(), pero para el compartimiento EcoExtra.

Clase Compartimiento

La clase Compartimiento representa un compartimiento individual del tanque, como se muestra en la figura 5. Tiene una capacidad y una cantidad actual. Es responsable de administrar la cantidad de combustible en el compartimento.

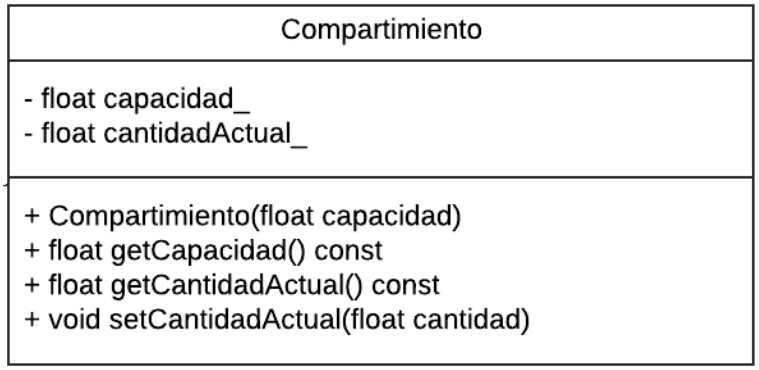

Figura 5. Clase Compartimiento

Atributos:

- float capacidad_: Capacidad total del compartimento.
- float cantidadActual_: Cantidad actual de combustible en el compartimento.

Métodos: Manejo de la capacidad y la cantidad actual de combustible.

Encapsulación: Atributos privados para evitar manipulaciones directas y asegurar que solo se modifiquen a través de métodos.

Clase Isla

A continuación se muestra la clase Isla, la cual representa un conjunto de surtidores de combustible de una estación de servicio.

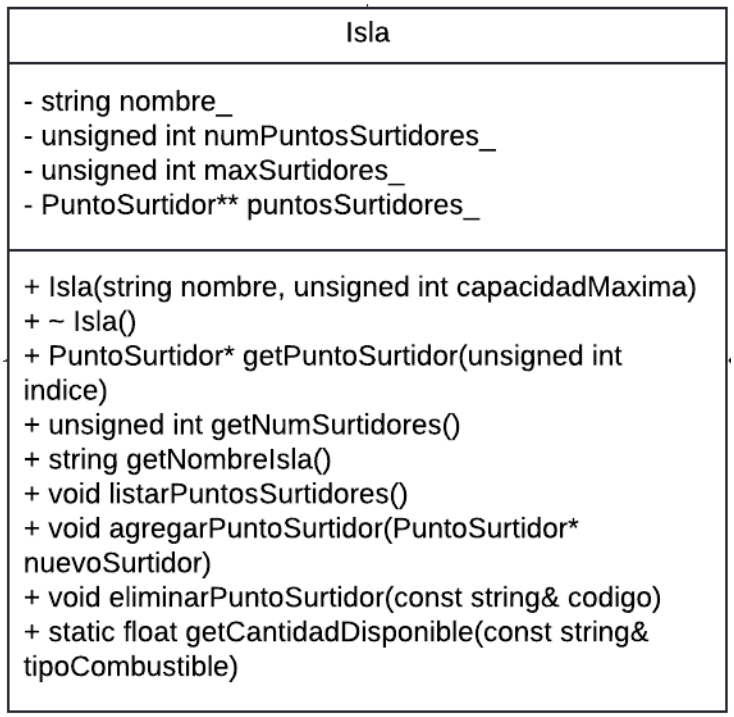

Figura 6. Clase Isla

Atributos:
Contiene un nombre y una lista de PuntoSurtidor.

- string nombre_:
- unsigned int numSurtidores:
- unsigned int maxSurtidores:
- PuntoSurtidor** surtidores:

Métodos: Permite listar surtidores y calcular la cantidad de combustible disponible en la isla.

Clase PuntoSurtidor

La clase PuntoSurtidor, representada en la figura 7, representa un surtidor de combustible individual. Tiene una cantidad de combustible disponible, un estado de activado o desactivado, una colección de ventas. Es responsable de administrar las ventas de combustible y el estado de la máquina.

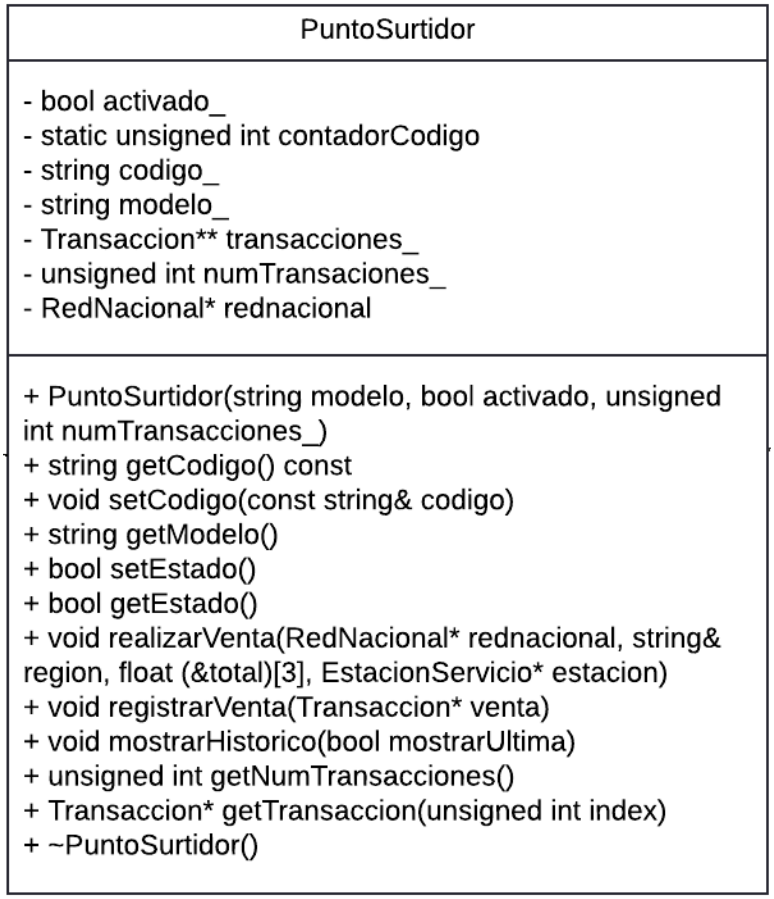

Figura 7. Clase PuntoSurtidor

Atributos:
Estado (activo/inactivo), modelo, código, número de transacciones, y un arreglo de punteros a Transaccion.

- bool activado_: Estado del surtidor.
- string codigo_: Código identificador del surtidor.
- string modelo_: Modelo del surtidor.
- unsigned int numTransacciones_: Contador de transacciones realizadas.
- Transaccion** transacciones_:

Métodos: Gestión de ventas y registro de transacciones.

Encapsulación: Mantiene atributos privados para manejar el estado y las operaciones sin acceso directo externo.

Clase Transaccion

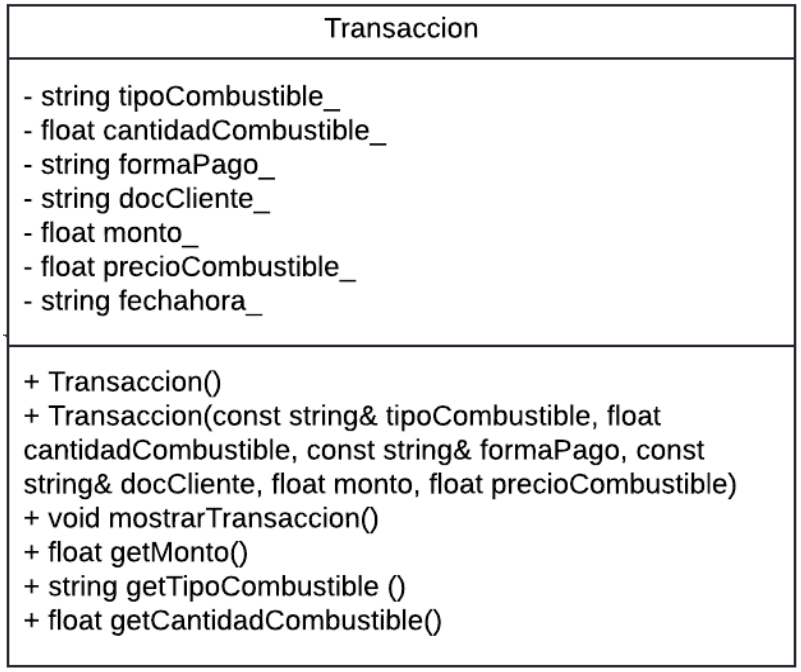

Figura 8. Clase Transaccion

La clase Transaccion mostrada en el figura 8Modela cada venta realizada en un punto surtidor.

Atributos:
Información de la venta: tipo de combustible, cantidad, forma de pago, datos del cliente, monto, precio y hora.

- string tipoCombustible_: Tipo de combustible vendido.
- float cantidadCombustible_: Cantidad de combustible en litros.
- string formaPago_: Método de pago.
- string docCliente_: Documento del cliente.
- float monto_: Monto total de la transacción.
- float precioCombustible_:
- string fechaHora_: Fecha y hora de la transacción.

Métodos: Constructor para registrar ventas y método para mostrar los detalles.

Encapsulación: Datos privados para proteger la confidencialidad de la información.

Relación entre RedNacional y EstacionServicio

Tipo de relación: Composición.

Cardinalidad: 1 a *n* (una instancia de RedNacional tiene múltiples EstacionServicio).

Relación entre EstacionServicio y Tanque

Tipo de relación: Composición (dado que Tanque es parte fundamental de EstacionServicio).
Cardinalidad: 1 a 1 (cada EstacionServicio tiene un único tanque central).

Relación entre Tanque y Compartimiento:

Tipo de relación: Composición (los compartimientos están dentro del tanque y no existen por separado).
Cardinalidad: 1 a 3 (cada Tanque tiene tres compartimientos, uno para cada tipo de combustible).

Relación entre EstacionServicio e Isla:

Tipo de relación: Composición (una estación está compuesta por hasta 3 islas).
Cardinalidad: 1 a *n* (cada EstacionServicio puede tener múltiples Islas).

Relación entre Isla y PuntoSurtidor:

Tipo de relación: Composición (los puntos surtidores son parte integral de la Isla, se destruyen si la isla desaparece).
Cardinalidad: 1 a 1..4 (cada Isla puede tener entre 1 a 4 puntos surtidores).

Relación entre PuntoSurtidor y Transaccion:

Tipo de relación: Asociación (una venta es una transacción independiente que se relaciona con el PuntoSurtidor).
Cardinalidad: 1 a *n* (un PuntoSurtidor puede tener múltiples ventas asociadas).

Justificación del uso de las variables

Las variables son fundamentales en cualquier sistema de programación, ya que permiten almacenar, manipular y gestionar datos de manera eficiente. En el diseño del desafío, se ha prestado especial atención a la elección de tipos de datos para cada variable, garantizando así la eficiencia y la claridad en el manejo de la información.

Tipos de datos

string: Este tipo de dato se utiliza para almacenar texto, lo que resulta esencial para manejar nombres, códigos y descripciones en el sistema. Al utilizar string, se garantiza flexibilidad en la longitud y el contenido, permitiendo que se puedan manejar fácilmente caracteres alfanuméricos, como códigos de estaciones y nombres de combustible.

float: Para los atributos que representan cantidades o precios (como la capacidad de los tanques y el monto de las transacciones), se elige el tipo float. Este tipo de dato permite manejar valores decimales, lo cual es crucial para la precisión en un sistema que gestiona ventas de combustible y transacciones monetarias.

unsigned int: En el caso de contadores, como el número de estaciones y transacciones, se utiliza unsigned int. Esto es eficiente porque se garantiza que estos valores nunca serán negativos, lo cual es lógico en el contexto de contadores. Al ser un tipo sin signo, se maximiza el rango de valores positivos que se pueden almacenar, lo cual es ventajoso cuando se espera un gran número de entradas.

bool: Para atributos que representan estados binarios, como si un surtidor está activado o desactivado, se utiliza el tipo bool. Este tipo de dato es ideal para condiciones lógicas y permite una gestión sencilla y clara de los estados dentro del sistema.

Encapsulación y seguridad

En el diseño del sistema, se ha optado por encapsular los atributos de las clases (definiéndolos como privados) para proteger la integridad de los datos. Esto significa que las variables no pueden ser accedidas directamente desde fuera de la clase, lo que evita modificaciones no controladas. En su lugar, se proporcionan métodos públicos (getters y setters) que permiten acceder y modificar los atributos de forma controlada.

Relaciones entre clases

El uso de punteros y referencias para gestionar relaciones entre clases (por ejemplo, entre EstacionServicio y Tanque) permite una mayor flexibilidad en la manipulación de objetos. Esto facilita la creación de estructuras dinámicas, como

listas de islas y surtidores, optimizando la gestión de la red de estaciones de servicio.

Eficiencia en el manejo de datos
La elección cuidadosa de tipos de datos permite realizar operaciones de manera eficiente. Por ejemplo, al utilizar arreglos para almacenar las estaciones y precios de combustible, se facilita el acceso rápido a la información, lo que es crítico para el cálculo de ventas y la gestión de transacciones en tiempo real.