DESAFIO II

INFORMATICA II

ALFONSO DIAZ LINA YOJANA  
RINCON SALDARRIAGA SEBASTIAN

GUERRA ANIBAL JOSE

UNIVERSIDAD DE ANTIOQUIA

2024-2

1. **Análisis del problema y consideraciones para la alternativa de solución propuesta.**

Se presenta un análisis y diseño de un sistema para la gestión eficiente de una red nacional de estaciones de servicio de combustible. El análisis incluye la implementación de funciones clave que permiten la simulación de ventas y la verificación automática de problemas, lo que justifica su importancia en la mejora de la gestión operativa de la red, siendo estas las siguientes:

1. **Clase RedEstaciones**

* **Atributos**:
* estaciones: Lista de todas las estaciones de la red.
* **Métodos**:
* agregar\_estacion(nombre de estacion): Agrega una nueva estación a la red.
* eliminar\_estacion(con codigo): Elimina una estación si no tiene surtidores activos.
* fijar\_precio\_combustible(region, categoria, precio): Fija los precios de combustible para una región.
* ventas\_nacional: Calcula las ventas de todas las estaciones por categoría.

La red de estaciones de servicio está compuesta por múltiples estaciones, por lo que su administración requiere almacenar una lista de todas las estaciones. Agrupar esta información en una clase facilita su organización y coherencia. El atributo (privado) contiene todas las estaciones de la red, y los métodos (públicos) gestionan las operaciones asociadas a esta red.

1. **Clase EstacionDeServicio**

* **Atributos**:
  + nombre: Nombre de la estación.
  + codigo: Código identificador.
  + gerente: Nombre del gerente.
  + region: Región (Norte, Centro, Sur).
  + ubicacion: Coordenadas GPS.
  + tanque: Capacidad del tanque por las 3 categorías de combustible.
  + surtidores: Lista de surtidores en la estación.
  + ventas: Historial de ventas (fecha, hora, cantidad, categoría, método de pago, documento cliente, dinero).
* **Métodos**:
* agregar\_surtidor(surtidor): Agrega un surtidor a la estación.
* eliminar\_surtidor(codigo\_surtidor): Elimina un surtidor si no está activo.

La estación tiene varias características y en lugar de manejar estos datos por separado, se agrupan dentro de una clase para mantener coherencia y organización.   
Los atributos (privados) representan los datos específicos de cada estación, mientras que los métodos (públicos) son las acciones que la estación puede realizar, como agregar o eliminar surtidores. Cada estación de servicio es un objeto de la clase, lo que permite crear varias estaciones con la misma estructura y comportamiento. La clase define la plantilla, y cada objeto será una instancia de esa plantilla con datos específicos.

1. **Clase Surtidor**

* **Atributos**:
* codigo: Código identificador.
* modelo: Modelo del surtidor.
* activo: Estado del surtidor (activo/inactivo).
* **Métodos**:
* registrar\_venta(cantidad, combustible, metodo\_pago, documento\_cliente): Registra una venta y actualiza el tanque.
* activar(): Activa el surtidor.
* desactivar(): Desactiva el surtidor.
* consultar\_historial: Devuelve el historial de transacciones.

El surtidor tiene varias características propias y en lugar de manejar estos datos de forma independiente, se agrupan dentro de la clase para mantener la **coherencia y organización.** Los **atributos** (privados) representan información específica del surtidor, mientras que los **métodos** (públicos) corresponden a las acciones que puede realizar cada surtidor. Cada surtidor es un **objeto de la clase**, lo que permite crear múltiples surtidores con la misma estructura y comportamiento.

1. **Funciones :**

* generar\_codigo: Genera código único para estaciones y surtidores.
* simular\_venta: Selecciona un surtidor y gestionar una venta.
* menu\_principal: Muestra las opciones de gestión de la red y las estaciones.
* Sistema de gestión de códigos: Asegura que los códigos de surtidores y estaciones sean únicos.
* Simulación automática de ventas: Función que gestiona ventas aleatorias en la red.
* Reporte de litros vendidos: Una función que permite consultar la cantidad de litros vendidos por categoría.
* actualizar\_disponibilidad(combustible, cantidad): Actualiza cantidad de combustible en el tanque.
* calcular\_monto\_ventas(): Calcula el monto total de ventas por categoría.
* verificar\_fugas(): Detecta si hay una fuga en la estación (lo vendido + almacenado < 95% del tanque inicial).

Estas funciones realizan tareas comunes que no dependen del estado interno de ningún objeto, por tanto se separan tareas y es mas eficiente y orgnizado el código.

1. **Diagrama de clases de la solución planteada. Adicionalmente, describa en alto nivel la lógica de las tareas que usted definió para aquellos subprogramas cuya solución no sea trivial.**

|  |
| --- |
| EstacionDeServicio |
| * Nombre * Código * Gerente * Ubicación * Tanque * Surtidores * ventas |
| + agregar\_surtidor  + eliminar\_surtidor |

|  |
| --- |
| RedEstaciones |
| * estaciones |
| + agregar\_estacion  + eliminar\_estacion  + fijar\_precio\_combustible  + ventas\_nacional |

**1…N**

**1**

**N**

|  |
| --- |
| Surtidor  **1…N** |
| * código * modelo * activo |
| + registrar\_venta  + activar  + desactivar  + consultar\_historial |

1. **Algoritmos implementados debidamente intra-documentados. (diagrama de clase)**
2. **Problemas de desarrollo que afrontó.**
3. **Evolución de la solución y consideraciones para tener en cuenta en la implementación.**