UNIVERSIDAD DE ANTIOQUIA
FACULTA DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE TELECOMUNICACIONES
SEMESTRE II

Desafío 2
Sistema de comercialización de combustible TerMax
Paradigma programación orientada a objetos - POO
Informe detallado del proyecto

Asignatura:
Informática II Teoría

Autores:
Sandra Milena Hoyos Muñoz
Oscar Miguel López Peñata

Tutores:
Anibal Guerra
Augusto Salazar

Medellín, Octubre de 2024

INTRODUCCIÓN

El presente informe aborda el desarrollo de un sistema de gestión para una red de estaciones de servicio de combustible, utilizando los principios de la programación orientada a objetos (POO). El objetivo es modelar un sistema realista que permita la administración eficiente de las estaciones, los surtidores de combustible y las transacciones de venta, asegurando un control preciso sobre los recursos disponibles y las ventas realizadas. La compañía TerMax, una de las principales proveedoras de combustible en Colombia, ha establecido una red de estaciones de servicio a nivel nacional, lo que implica la necesidad de un sistema robusto y flexible que facilite la supervisión y operación de cada estación.

En este contexto, cada estación de servicio cuenta con tanques de almacenamiento que distribuyen tres tipos de combustibles: Regular, Premium y EcoExtra. Los surtidores conectados a estos tanques se encargan de suministrar el combustible a los vehículos, registrando detalladamente las transacciones que incluyen información sobre la cantidad de combustible despachada, la categoría del producto, el método de pago y los datos del cliente. Adicionalmente, el sistema debe gestionar la capacidad de los tanques, calcular el monto total de las ventas discriminado por tipo de combustible y permitir la detección de posibles fugas, contribuyendo así a un control exhaustivo de los recursos.

El enfoque propuesto en este sistema incluye la implementación de múltiples funcionalidades basadas en los principios de abstracción, encapsulación y modularidad, características esenciales de la programación orientada a objetos. Entre las funcionalidades más relevantes se encuentran la gestión de la red nacional de estaciones, la activación y desactivación de surtidores, la simulación de ventas y la verificación de fugas de combustible. Además, el sistema permitirá la creación y eliminación de estaciones de servicio y surtidores, así como la asignación de precios del combustible según la región geográfica.

A través del desarrollo de este sistema, se busca no solo automatizar las operaciones de las estaciones de servicio, sino también optimizar la toma de decisiones a nivel gerencial, asegurando un mayor control sobre las ventas y la operación diaria de la red de estaciones de TerMax. El enfoque basado en la programación orientada a objetos facilitará la escalabilidad y el mantenimiento del sistema, permitiendo su adaptación futura a nuevos requerimientos y características.

PROPÓSITOS

General

Desarrollar un sistema de gestión integral basado en el paradigma de la Programación Orientada a Objetos para la red nacional de estaciones de servicio de TerMax, que automatice, supervise y controle eficientemente las operaciones diarias de venta de combustibles, optimizando el manejo de recursos, la administración de transacciones y la toma de decisiones estratégicas en tiempo real.

Específicos

Implementar la gestión de la red nacional de estaciones de servicio, que incluya la posibilidad de agregar o eliminar estaciones, así como fijar los precios del combustible según la región geográfica, garantizando la escalabilidad y flexibilidad del sistema.

Desarrollar un módulo para la administración de las estaciones de servicio, que permita agregar, activar o desactivar surtidores, consultar el historial de ventas y reportar la cantidad de litros vendidos por categoría de combustible en cada estación, asegurando un control preciso de las operaciones.

Crear un sistema de simulación de ventas de combustible, que asigne aleatoriamente un surtidor activo para realizar la transacción, registrando los datos de la venta, incluyendo la fecha, hora, cantidad y método de pago, con el fin de replicar las dinámicas comerciales reales.

Implementar un sistema de verificación de fugas de combustible, que compare la cantidad de combustible vendido y almacenado en los tanques para asegurar que corresponda con el inventario disponible, minimizando pérdidas y mejorando la seguridad en las estaciones de servicio.

Desarrollar un menú interactivo que permita acceder a las diferentes funcionalidades del sistema, facilitando la interacción del usuario con las opciones de gestión y simulación de manera amigable y eficiente.

Asegurar que el sistema sea modular y escalable, permitiendo futuras expansiones, como la incorporación de nuevas estaciones de servicio o surtidores, sin comprometer la funcionalidad propuesta.

DESCRIPCIÓN DEL PROBLEMA

La empresa TerMax, líder en la distribución de combustibles en Colombia, enfrenta un desafío importante en la gestión eficiente de su red nacional de estaciones de servicio. Actualmente, las operaciones relacionadas con la venta de combustibles, la administración de recursos y el control de transacciones se realizan de manera dispersa y sin un sistema integral que centralice la información. Este escenario genera problemas en la supervisión efectiva de las estaciones, dificultad en el control de inventarios, y una falta de visibilidad en tiempo real de las transacciones y el estado de los recursos.

Uno de los principales problemas identificados es la *falta de automatización* en las operaciones diarias de las estaciones de servicio, lo que obliga a los administradores locales a realizar procesos manuales de control, tales como la actualización del inventario de combustibles, el registro de ventas y la gestión de surtidores. Esto no solo incrementa la posibilidad de errores humanos, sino que también limita la capacidad de respuesta ante situaciones como la baja disponibilidad de combustible o el mal funcionamiento de surtidores.

Adicionalmente, *la carencia de un sistema de monitoreo centralizado* para las estaciones de servicio imposibilita una supervisión adecuada de los recursos y ventas a nivel nacional. Actualmente, no existe un mecanismo que permita a la gerencia central de TerMax tener un control detallado del inventario de cada estación, del rendimiento de los surtidores, ni de las ventas diarias, discriminadas por tipo de combustible y métodos de pago. Esta falta de visibilidad complica la toma de decisiones estratégicas, ya que se carece de información actualizada y precisa para ajustar precios, gestionar el suministro de combustible, o detectar posibles pérdidas, como fugas en los tanques de almacenamiento.

Otro problema clave *es la ausencia de un sistema para la detección automática de fugas de combustible*. En la situación actual, no se realiza un seguimiento adecuado de las discrepancias entre las cantidades de combustible despachadas y las disponibles en los tanques. Este vacío en el control puede resultar en pérdidas significativas para la empresa, además de presentar riesgos de seguridad.

Además, la *gestión manual de precios* por región y la incapacidad de realizar simulaciones de ventas son factores que incrementan la ineficiencia operativa. La variabilidad de los precios según la ubicación de las estaciones debería estar automatizada para evitar errores y facilitar el ajuste dinámico de los precios en función de la demanda y las condiciones de

mercado. De igual forma, la simulación de ventas permitiría prever escenarios futuros, evaluar el rendimiento de las estaciones y anticipar problemas operativos.

Por tanto, el problema principal se puede resumir en la necesidad de un *sistema de gestión integral* que permita a TerMax centralizar la operación de su red de estaciones de servicio, automatizar los procesos de venta, controlar el inventario de combustibles en tiempo real, detectar posibles fugas, y gestionar de manera eficiente los surtidores y las transacciones. Este sistema debe basarse en un enfoque robusto de programación orientada a objetos (POO), que permita un diseño modular, flexible y escalable, para que la solución sea adaptable a las futuras necesidades del negocio.

El desarrollo de este sistema abordará estos problemas mediante la creación de una plataforma que integre todas las operaciones relacionadas con la venta de combustibles, garantizando la integridad de los datos y proporcionando información clave para la toma de decisiones estratégicas, mejorando así la eficiencia operativa y reduciendo las pérdidas potenciales.

DESCRIPCIÓN DE TAREAS

Fecha	Nombre tarea	Actividades	Resultado
Octubre 9 a 10	**Análisis del problema:** Esta fase consiste en realizar una comprensión profunda del problema planteado, identificando los requerimientos clave del sistema y definiendo claramente las funcionalidades solicitadas.	Leer y analizar todo el documento del desafío. Identificar las entidades principales del sistema (estaciones de servicio, surtidores, transacciones, combustible). Definir los requisitos funcionales y no funcionales. Delimitar las relaciones entre los diferentes componentes del sistema (gestión de estaciones, surtidores, ventas y fugas de combustible).	Documento que describe claramente el análisis del problema y los requisitos del sistema.

Fecha	Nombre tarea	Actividades	Resultado
Octubre 10 a 11	**Análisis de Datos y Estructuras:** Determinar las estructuras de datos más adecuadas para representar los elementos del sistema (listas, mapas, arreglos dinámicos) y analizar la eficiencia de las mismas.	Identificar las estructuras de datos necesarias para representar estaciones, surtidores, transacciones y tanques de combustible. Elegir los tipos de datos correctos para atributos como cantidades de combustible, métodos de pago, precios por región, entre otros. Definir las funciones matemáticas o lógicas requeridas para cálculos como las ventas totales y la verificación de fugas.	Documento de especificación de los tipos de datos y estructuras de datos seleccionadas, con justificación para cada elección.

Fecha	Nombre tarea	Actividades	Resultado
Octubre 11 a 12	**Elaboración del Diagrama UML:** Crear un diagrama UML que represente visualmente las clases del sistema, sus atributos, métodos y relaciones entre ellas.	Diseñar las clases principales del sistema (RedNacional, EstacionServicio, Surtidor, Transaccion, Combustible). Definir las relaciones de agregación, asociación y dependencia entre las clases. Asegurarse de que todos los métodos públicos y privados estén correctamente representados en el diagrama. Revisar el diagrama para asegurar que todas las funcionalidades del sistema estén reflejadas en el diseño.	Diagrama UML completo y revisado, con una estructura clara de las clases y relaciones del sistema.

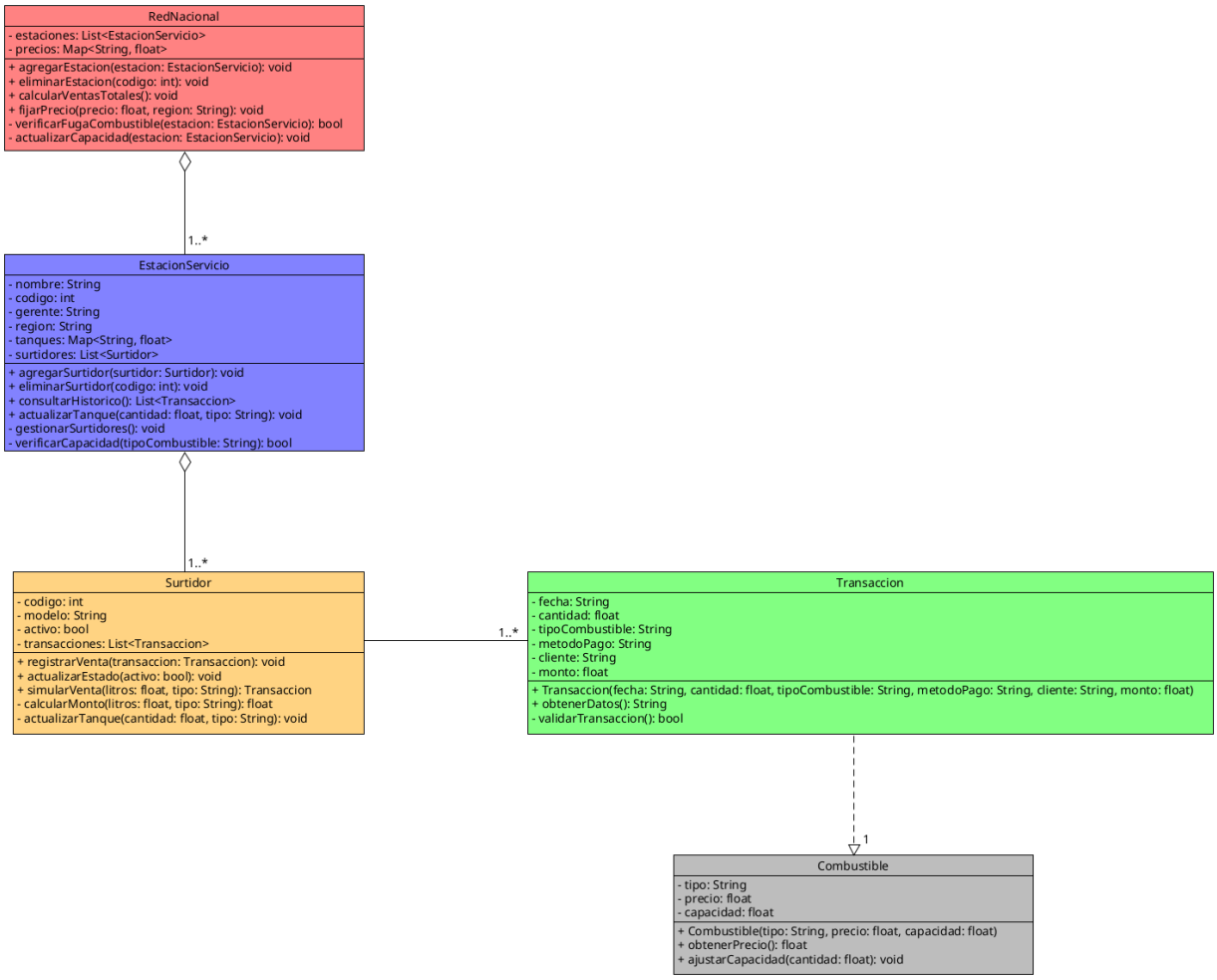
Fecha	Nombre tarea	Actividades	Resultado
Octubre 12	**Diseño de la implementación:** Basado en el análisis de datos y el diagrama UML, diseñar la implementación del sistema en C++, dividiendo el trabajo en módulos lógicos y definiendo los métodos y atributos necesarios.	Crear un esquema detallado de cómo se implementarán las funcionalidades clave del sistema (gestión de estaciones, ventas, verificación de fugas). Definir la interacción entre las clases a nivel de código. Especificar los constructores, getters, setters y funciones necesarias para que el sistema sea funcional.	Esquema de implementación detallado, listo para ser llevado a código.

Fecha	Nombre tarea	Actividades	Resultado
Octubre 12	**Creación del Repositorio en GitHub:** Configurar un espacio en GitHub para gestionar el código y el informe, haciendo commits regulares para documentar el progreso del desarrollo.	Crear el repositorio público en GitHub. Establecer un archivo README que incluya una breve descripción del proyecto y los requisitos del desafío. Hacer el primer commit con la estructura básica del proyecto (carpetas de código y documentos). Realizar commits regulares al menos una vez al día para documentar la evolución de la solución.	Repositorio de GitHub correctamente configurado y actualizado de manera regular.

Fecha	Nombre tarea	Actividades	Resultado
Octubre 13 a 15	**Implementación del sistema:** Convertir el diseño detallado en código funcional en C++, asegurando que todas las funcionalidades solicitadas estén implementadas de manera eficiente y que el sistema funcione de acuerdo a los requisitos.	Implementar la gestión de la red de estaciones (agregar/eliminar estaciones, fijar precios). Programar la administración de surtidores (agregar/eliminar surtidores, registrar ventas). Implementar la simulación de ventas con surtidores asignados aleatoriamente. Programar el sistema de verificación de fugas de combustible. Realizar pruebas unitarias y ajustes necesarios en el código.	Código C++ completo, funcional y documentado, implementando todas las funcionalidades requeridas.

Fecha	Nombre tarea	Actividades	Resultado
Octubre 15	**Elaboración del Video de Sustentaión:** Crear un video explicativo que resuma la solución planteada, con una presentación clara del análisis realizado, el diseño del sistema, la implementación, y ejemplos de su funcionamiento.	Presentar la estructura general del sistema y el diagrama UML. Explicar las principales funcionalidades implementadas en el sistema. Mostrar ejemplos en tiempo real de cómo funciona el sistema, simulando ventas y verificaciones. Justificar las elecciones técnicas realizadas durante el desarrollo (estructuras de datos, diseño POO, etc.).	Código C++ completo, funcional y documentado, implementando todas las funcionalidades requeridas.

DIAGRAMA DE CLASES



Explicación

Clase RedNacional

Atributos:

estaciones: List<EstacionServicio> (privado): Lista de estaciones de servicio que forman parte de la red nacional.

precios: Map<String, float> (privado): Mapa que asocia cada tipo de combustible con su precio en diferentes regiones.

Métodos Públicos:

+ agregarEstacion(estacion: EstacionServicio): void: Agrega una nueva estación a la red.

+ eliminarEstacion(codigo: int): void: Elimina una estación de la red basada en su código.

+ calcularVentasTotales(): void: Calcula las ventas totales de la red.

+ fijarPrecio(precio: float, region: String): void: Fija el precio de un tipo de combustible para una región específica.

Métodos Privados:

- verificarFugaCombustible(estacion: EstacionServicio): bool: Verifica si existe una fuga de combustible en una estación.

- actualizarCapacidad(estacion: EstacionServicio): void: Actualiza la capacidad de combustible disponible en una estación.

Esta clase gestiona la red nacional de estaciones de servicio, lo que incluye agregar o eliminar estaciones, calcular las ventas totales y fijar precios a nivel regional. También se encarga de la verificación de fugas de combustible, tal como se especifica en el desafío. El método verificarFugaCombustible es importante para garantizar la integridad del sistema, asegurando que no haya pérdidas de combustible en las estaciones.

Clase EstacionServicio

Atributos:

nombre: String (privado): Nombre de la estación de servicio.

codigo: int (privado): Código único que identifica la estación.

gerente: String (privado): Nombre del gerente de la estación.

region: String (privado): Región donde está ubicada la estación.

tanques: Map<String, float> (privado): Mapa que asocia tipos de combustible con la cantidad almacenada en los tanques.

surtidores: List<Surtidor> (privado): Lista de surtidores disponibles en la estación.

Métodos Públicos:

+ agregarSurtidor(surtidor: Surtidor): void: Agrega un surtidor a la estación.

+ eliminarSurtidor(codigo: int): void: Elimina un surtidor de la estación.

+ consultarHistorico(): List<Transaccion>: Consulta el historial de transacciones de la estación.

+ actualizarTanque(cantidad: float, tipo: String): void: Actualiza la cantidad de combustible en los tanques de la estación.

Métodos Privados:

- gestionarSurtidores(): void: Gestión interna de los surtidores en la estación (activación, desactivación o mantenimiento).
- verificarCapacidad(tipoCombustible: String): bool: Verifica si hay suficiente capacidad de un tipo de combustible en los tanques.

Esta clase modela cada estación de servicio en la red, incluyendo sus tanques de combustible y surtidores. Es clave para las funcionalidades de agregar o eliminar surtidores, gestionar los niveles de combustible en los tanques y consultar el historial de transacciones, lo cual es parte de los requisitos del sistema. El método gestionarSurtidores se encarga de la administración interna de los surtidores, y verificarCapacidad asegura que no se excedan los límites de almacenamiento de combustible.

Clase Surtidor

Atributos:

codigo: int (privado): Código identificador del surtidor.

modelo: String (privado): Modelo del surtidor.

activo: bool (privado): Indica si el surtidor está activo.

transacciones: List<Transaccion> (privado): Lista de transacciones realizadas por este surtidor.

Métodos Públicos:

+ registrarVenta(transaccion: Transaccion): void: Registra una venta en el surtidor.

+ actualizarEstado(activo: bool): void: Actualiza el estado del surtidor (activo o inactivo).

+ simularVenta(litros: float, tipo: String): Transaccion: Simula la venta de una cantidad de litros de un tipo específico de combustible.

Métodos Privados:

- calcularMonto(litros: float, tipo: String): float: Calcula el monto de la venta según la cantidad de litros y el tipo de combustible.

- actualizarTanque(cantidad: float, tipo: String): void: Actualiza la cantidad de combustible en el tanque después de una venta.

Los surtidores son los encargados de realizar las ventas y registrar las transacciones. Esta clase se encarga de gestionar las operaciones diarias de cada surtidor, como simular ventas, registrar transacciones y ajustar la cantidad de combustible, tal como se pide en los requerimientos del desafío. Además, el método simularVenta permite cumplir con la funcionalidad de simular transacciones.

Clase Transaccion

Atributos:

fecha: String (privado): Fecha en la que se realizó la transacción.

cantidad: float (privado): Cantidad de combustible vendido.

tipoCombustible: String (privado): Tipo de combustible que se vendió.

metodoPago: String (privado): Método de pago usado (efectivo, tarjeta de débito, tarjeta de crédito).

cliente: String (privado): Identificador del cliente.

monto: float (privado): Monto total de la transacción.

Métodos Públicos:

+ Transaccion(fecha: String, cantidad: float, tipoCombustible: String, metodoPago: String, cliente: String, monto: float): Constructor que inicializa una transacción.

+ obtenerDatos(): String: Devuelve los datos de la transacción en un formato de texto.

Métodos Privados:

- validarTransaccion(): bool: Valida si una transacción es correcta o si hubo algún error.

Esta clase es crucial para registrar las ventas en las estaciones de servicio. El sistema debe registrar las transacciones que incluyen la fecha, cantidad de combustible, el método de pago y el cliente, lo cual está alineado con los requerimientos de la simulación de ventas y la gestión del historial de transacciones.

Clase Combustible

Atributos:

tipo: String (privado): Representa el tipo de combustible (Regular, Premium, EcoExtra).

precio: float (privado): Almacena el precio del tipo de combustible.

capacidad: float (privado): La cantidad de combustible disponible en los tanques.

Métodos Públicos:

+ Combustible(tipo: String, precio: float, capacidad: float): Constructor que inicializa los atributos de un tipo de combustible.

+ obtenerPrecio(): float: Devuelve el precio del combustible.

+ ajustarCapacidad(cantidad: float): void: Ajusta la capacidad de combustible al incrementar o decrementar la cantidad disponible.

Esta clase es esencial para la gestión de los diferentes tipos de combustible dentro de cada estación. El sistema debe manejar distintas categorías de combustibles con diferentes precios y capacidades, como se menciona en el documento. Este tipo de modelado facilita la asignación de combustibles en las estaciones y los surtidores.

Relaciones y multiplicidades

RedNacional ↔ EstacionServicio: Composición 1..* ↔ 1. Una red tiene varias estaciones de servicio, pero cada estación pertenece a una única red.

EstacionServicio ↔ Surtidor: Composición 1..* ↔ 1. Una estación de servicio tiene varios surtidores, y cada surtidor pertenece exclusivamente a una estación.

Surtidor ↔ Transaccion: Asociación 1..* ↔ 1. Un surtidor puede realizar múltiples transacciones, pero cada transacción está asociada a un surtidor específico.

Transaccion ↔ Combustible: Asociación 1 ↔ 1..*. Cada transacción registra un tipo de combustible específico, pero ese combustible puede estar asociado a múltiples transacciones.

Este modelado de relaciones con sus multiplicidades responde directamente a los requisitos del desafío:

Gestión de estaciones y surtidores: La relación de composición entre RedNacional, EstacionServicio y Surtidor refleja cómo se gestiona jerárquicamente la red nacional. Cada estación tiene sus surtidores, y la red tiene control sobre todas las estaciones del país.

Registro de ventas y transacciones: La asociación entre Surtidor y Transaccion permite que cada surtidor registre todas las transacciones realizadas, lo cual es clave para mantener un historial de ventas preciso.

Categorías de combustible: La asociación entre Transaccion y Combustible permite que el sistema gestione las diferentes categorías de combustible vendidas, lo que es fundamental para calcular las ventas por categoría en cada estación.

Este diseño cumple con los objetivos del sistema, como la simulación de ventas, el cálculo de ventas totales y por tipo de combustible, la gestión de los precios y la verificación de fugas, asegurando un control preciso y eficiente de la red de estaciones de combustible.

PROBLEMAS DE DESARROLLO

Verificación de capacidad de combustible: Riesgo de ventas superiores a la cantidad disponible.

Verificación de fugas de combustible: Posibles pérdidas no detectadas si no se monitorea adecuadamente.

Simulación de ventas: Problemas en la asignación aleatoria de surtidores o inconsistencias en la cantidad vendida.

Actualización de precios: Posible inconsistencia en los precios por región si no se gestiona correctamente.

Gestión de códigos de surtidores: Posible duplicación de códigos o errores en la activación/desactivación de surtidores.

Historial de transacciones: Riesgo de sobrecarga del sistema o pérdida de información valiosa si no se maneja adecuadamente.

Asignación de capacidad de tanques: Riesgo de asignaciones ineficientes que afecten la operación de las estaciones.

Cálculo del monto de ventas: Posibilidad de errores financieros si el cálculo no es preciso.

Manejo de memoria dinámica: Problemas de rendimiento o errores de memoria si no se gestiona correctamente.

Estos problemas potenciales requieren especial atención en el diseño y desarrollo del sistema, para asegurar que el sistema de comercialización de combustible funcione de manera eficiente y cumpla con los requerimientos del desafío.