Resumen Ejecutivo

Este informe documenta el diseño, implementación y validación de un sistema de gestión para una empresa de transporte, desarrollado en Python con enfoque de Programación Orientada a Objetos (POO). El sistema cubre la administración de la flota, el personal (conductores y mecánicos), la operación de viajes y el ciclo de mantenimiento. Se evidencia de forma explícita la aplicación de abstracción, herencia, polimorfismo y encapsulamiento. El resultado es una solución funcional.

1. Objetivos y Alcance

Objetivo general: diseñar y desarrollar una aplicación en Python que modele un sistema de gestión para una empresa de transporte, aplicando de forma demostrable los pilares de la POO.

Alcance funcional:

- Gestión de flota con operaciones CRUD (crear, listar con filtros por estado, eliminar con validaciones).
- Gestión de personal: conductores (asignables a vehículos) y mecánicos (asignan mantenimientos).
- Operación de viajes con actualización de odómetro y cálculo de costos por km (polimorfismo).
- Ciclo de mantenimiento: envío a taller, registro de mantenimientos y liberación del vehículo.
- Reportes de estado de flota, personal y bitácoras de viajes/mantenimientos.

2. Requisitos del Parcial y Trazabilidad

El parcial exige evidenciar POO (abstracción, herencia, polimorfismo y encapsulamiento) y entregar código ejecutable, con guía de uso y demostración. La siguiente tabla mapea los requisitos clave con elementos concretos del sistema:

Requisito	Elemento implementado	Cómo se evidencia
Abstracción	Clases Employee, Vehicle	Métodos abstractos rol(), costo_por_km(), arrancar() obligan implementación en subclases.
Herencia	Employee→Driver/Mechanic; Vehicle→Car/Motorcycle/Truck	Jerarquías funcionales y especializaciones (e.g., Truck añade ejes).
Encapsulamiento	Atributos privados __nombre, __odometro_km, __status	Acceso mediante @property y métodos regulados.
Polimorfismo	costo_por_km(), arrancar()	Comportamiento según tipo concreto sin condicionales externos.
Gestión de flota	CRUD de vehículos en Company	Crear/listar/eliminar con validaciones de estado y desasignación segura.
Operación	Viajes y mantenimientos	Bitácoras (TripLog, MaintenanceRecord) y reportes consolidados.
Demostración	Menú interactivo	Ejecución fluida en Google Colab con entradas por consola.

3. Diseño y Arquitectura (POO)

3.1 Decisiones de diseño

- Separación conceptual entre dominio (vehículos, personal) y casos de uso (acciones coordinadas en Company).
- Uso de clases abstractas para garantizar contratos estables y facilitar extensibilidad (p. ej., agregar *Bus* o *Supervisor*).

- Enums para estados del vehículo (seguridad y legibilidad).
- Propiedades con validaciones para preservar invariantes (datos consistentes).

3.2 Patrón de fachada liviana

La clase Company actúa como fachada, es decir, centraliza operaciones de negocio, orquesta entidades y simplifica el uso desde interfaz.

3.3 Extensibilidad

- Nuevos vehículos: basta con heredar de Vehicle e implementar costo_por_km() y arrancar().
- Nuevos roles: herencia desde Employee con su rol().
- Reglas: agregar métodos en Company sin afectar entidades (bajo acoplamiento).

4. Modelo de Dominio

El sistema considera las siguientes entidades y relaciones:

- Vehicle (abstracta) = Car, Motorcycle, Truck (Truck añade ejes).
- Employee (abstracta) = Driver, Mechanic.
- Company administra colecciones de vehículos y empleados, y mantiene bitácoras de viajes/mantenimientos.

5. Casos de Uso y Reglas de Negocio

5.1 CRUD de Vehículos

- Crear: tipos permitidos car, motorcycle, truck (camión requiere *ejes*).
- Listar: admite filtro opcional por VehicleStatus.
- Eliminar: prohibido si está *EN_VIAJE*; desasigna conductores enlazados para mantener consistencia.

5.2 Asignaciones

Solo se puede asignar un conductor a un vehículo *DISPONIBLE*; se permite desasignar para cambios operativos.

5.3 Viajes

Un viaje incrementa el odómetro y calcula el costo por km según el tipo concreto (polimorfismo). Si se supera el umbral desde el último mantenimiento (10.000 km), se emite alerta.

5.4 Taller y Mantenimiento

Un mantenimiento solo se registra si el vehículo está en estado *EN_TALLER*, con detalle y costo. Tras finalizar, vuelve a *DISPONIBLE* y se marca el odómetro como último mantenimiento.

6. Implementación y Principios de Calidad

- Seguridad de estados: restricciones en operaciones (no eliminar en viaje, no viajar sin asignación, etc.).
- Mensajes claros y manejo básico de errores en la interfaz de consola.
- Legibilidad: nombres expresivos, tipado gradual (typing), y separación de responsabilidades.
- Portabilidad: 100% librerías estándar, sin dependencias externas.
- Demostrabilidad: flujo “seed” y menú interactivo listos para uso en Colab.

7. Plan de Pruebas y Resultados

#	Escenario	Acción	Resultado esperado	Estado
1	Crear camión	tipo=truck, ejes=3	Vehículo creado con ID y ejes en descripción	OK
2	Eliminar vehículo DISPONIBLE	eliminar_vehiculo(ID)	Eliminación exitosa	OK
3	Eliminar vehículo EN_VIAJE	forzar viaje y eliminar	Error controlado (no permitido)	OK
4	Asignación	asignar_conductor(driver, veh)	Solo si vehiculo DISPONIBLE	OK
5	Viaje	iniciar_viaje	Odómetro aumenta y costo se calcula	OK
6	Alerta mantenimiento	acumular >= 10.000 km	Mensaje de alerta	OK
7	Registrar	taller→registrar	Cambia a DISPONIBLE	OK

	mantenimiento		y marca último km	

8. Conclusiones y Trabajo Futuro

El sistema cumple y demuestra los pilares de la POO con una arquitectura clara, mantenible y extensible. La interfaz por consola permite validar la lógica de negocio sin dependencias externas.

9. Guía de Ejecución (Colab/local) PARA TENER EN CUENTA

1. Abrir Google Colab y pegar el código completo en una celda.
2. Ejecutar la celda y seguir el menú por consola.
3. Opcional (local): guardar como main.py y ejecutar python main.py.

Hecho Por: Juan Diego Trujillo Narvaez