ACTIVIDAD
INTRODUCCIÓN A PRINCIPIOS SOLID

Aprendizaje esperado

Utilizar principios básicos de diseño orientado a objetos para la implementación de una pieza de software acorde al lenguaje java para resolver un problema de baja complejidad.

Actividad

Contexto:

LogiTrack S.A., una empresa de logística y distribución de última milla, enfrenta un desafío técnico: su flota de vehículos de transporte es diversa (camiones, drones, motocicletas eléctricas, bicicletas repartidoras) y su sistema de gestión es obsoleto, rígido y difícil de mantener. Necesitan una solución modular que permita simular operaciones, gestionar consumo energético, planificar rutas y mantener el control del estado operativo de cada unidad.

Tu equipo ha sido contratado para desarrollar una aplicación base reutilizable y escalable, construida con principios de diseño orientado a objetos, para facilitar la gestión automatizada de su flota.

Objetivo de Aprendizaje

Utilizar principios básicos de diseño orientado a objetos para implementar una solución en Java que emplee polimorfismo (mediante herencia e interfaces), asegurando mantenibilidad y reutilización del código en un sistema realista de baja complejidad.

Requerimientos Específicos

1. Modelo de clases y herencia

- Crear una clase abstracta VehiculoLogistico con los siguientes atributos:
  - idVehiculo (String)
  - capacidadCargaKg (double)
  - ubicacionActual (String)
  - velocidadPromedioKmH (double)
- Métodos abstractos:
  - desplazar(String destino)
  - verificarEstado()

- Implementar al menos cuatro clases concretas:
    - CamionRefrigerado
    - DronRepartidor
    - MotoElectrica
    - BicicletaCarga

    Cada clase debe personalizar sus métodos según las particularidades de su funcionamiento.

2. Interfaces y polimorfismo

- Crear una interfaz Energizable con el método recargar(), para vehículos eléctricos o con baterías.
- Crear una interfaz CombustibleFosil con el método repostar(), para vehículos con motor a diésel o gasolina.
- Crear una interfaz Operativo con métodos calcularDesgaste(), registrarEntrega(), que deben implementar todos los vehículos.

3. Uso de clases auxiliares y composición

- Crear la clase RutaEntrega con atributos como origen, destino, distanciaKm, nivelDificultad.
- Implementar una clase SimuladorDistribucion que permita asignar rutas a vehículos, simular entregas y actualizar estados (batería, combustible, ubicación, entregas).

4. Aplicación de principios de diseño

- Aplicar los siguientes principios:
    - **Responsabilidad Única (SRP)**: cada clase debe tener una responsabilidad clara.
    - **Bajo acoplamiento y alta cohesión**: separar funcionalidades entre clases y usar interfaces para flexibilizar el sistema.
    - **Diseño reutilizable**: permitir que se agreguen nuevos tipos de vehículos o rutas sin necesidad de modificar el código existente.

5. Representación visual

- Elaborar un **diagrama de clases UML** que incluya:
  - Jerarquía de clases e interfaces.
  - Relaciones de composición.
  - Métodos sobrescritos y polimorfismo.

6. Simulación de uso

- En un programa principal (main), simular el siguiente flujo:
  - Crear una flota diversa de vehículos.
  - Crear rutas de entrega.
  - Asignar vehículos a rutas.
  - Simular entregas, desplazamientos, recargas y repostajes.
  - Generar una bitácora de operación por vehículo.

Entregables

- **Proyecto Java completo**, con:
  - Código fuente organizado y documentado.
  - Clases estructuradas con principios SOLID.
  - Simulaciones funcionales de entregas.
- **Documento técnico (PDF)** con:
  - Diagrama de clases UML.
  - Justificación de los principios de diseño aplicados.
  - Ejemplos de ejecución y resultados de la simulación.