Práctica: Diseño de una solución aplicando SOLID y GRASP

Contexto

Una empresa de logística quiere desarrollar un **Sistema de Gestión de Entregas** para optimizar el seguimiento de los pedidos desde que son recogidos en el almacén hasta su entrega al cliente. Actualmente, la empresa maneja distintos tipos de envíos: **estándar, express y programada**, cada una con reglas y restricciones específicas.

Los transportistas pueden ser empleados de la empresa o servicios externos tercerizados, y la asignación de pedidos debe manejarse de manera eficiente. El sistema debe permitir la consulta de entregas, la asignación de pedidos a transportistas y la generación de reportes. Además, debe garantizar que los pedidos sean almacenados para su consulta posterior.

La empresa requiere una **aplicación web** con una interfaz que permita a los usuarios consultar y administrar las entregas. Se espera que el sistema sea **fácilmente extensible** y con **una buena organización del código** para permitir futuras modificaciones sin alterar su estructura principal.

Objetivo

Los estudiantes deben **diseñar la arquitectura de clases del sistema**, asegurándose de aplicar los principios de diseño **SOLID y GRASP**. El diseño debe presentarse en un **diagrama de clases UML** y acompañado de una justificación clara de las decisiones de diseño.

Para lograr un diseño bien estructurado, los estudiantes deben **pensar en las distintas partes que componen el sistema y sus interacciones**, asegurando una clara separación de responsabilidades.

Requerimientos del sistema

1. **Gestión de Pedidos**:
    - Cada pedido tiene un ID, una dirección de entrega, un estado (**pendiente, en tránsito, entregado**) y un tipo de envío.
    - Se debe poder calcular el **costo del envío** según su tipo.
    - Un pedido debe poder **asignarse a un transportista** disponible.

2. **Gestión de Transportistas**:
    - Hay dos tipos de transportistas: **empleados internos** y **servicios externos**.
    - Cada transportista tiene una capacidad máxima de pedidos asignables.
    - Un transportista puede **actualizar el estado** del pedido y marcarlo como entregado.
3. **Generación de Reportes**:
    - Se debe poder generar un **reporte de pedidos entregados** y **un ranking de transportistas más eficientes**.
4. **Interacción con el usuario**:
    - Debe existir una forma de que los usuarios **puedan consultar y modificar pedidos**.
    - La interacción con el sistema debe estar **bien separada de la lógica de negocio**.
5. **Persistencia de datos**:
    - Los pedidos y transportistas deben **almacenarse en una base de datos relacional o en archivos**.
    - Debe haber una **clase que gestione la persistencia**, desacoplada de la lógica de negocio.
6. **Extensibilidad y Flexibilidad**:
    - El sistema debe permitir la **adición de nuevos tipos de envíos** sin modificar el código existente.
    - Se deben **separar adecuadamente las responsabilidades** para garantizar bajo acoplamiento y alta cohesión.

Restricciones

1. **El diseño debe permitir la interacción con el usuario**, asegurando que las responsabilidades estén bien distribuidas.
2. **Los principios SOLID y GRASP deben estar claramente aplicados** en el diseño y explicados en la justificación.
3. **Las clases de persistencia deben estar desacopladas de la lógica de negocio**, permitiendo independencia entre ambas capas.
4. **Las reglas de negocio deben estar encapsuladas en clases especializadas**, evitando que otras partes del sistema implementen lógica directamente.
5. **Las clases encargadas de la interacción con el usuario no deben acceder directamente a la base de datos o a la lógica de negocio**.
6. **Se debe diseñar un mecanismo para la asignación de pedidos a transportistas**, asegurando que la lógica sea flexible y extensible.

Criterios de Evaluación

1. **Aplicación de principios SOLID**:
    - ¿Se usa el **Principio de Responsabilidad Única (SRP)** para evitar clases con múltiples responsabilidades?
    - ¿Se aplica el **Principio Abierto/Cerrado (OCP)** para facilitar la extensión del sistema?
    - ¿Se usa **Liskov Substitution Principle (LSP)** correctamente en las jerarquías de clases?
    - ¿Se implementa el **Principio de Inversión de Dependencias (DIP)** para evitar dependencias rígidas?
2. **Aplicación de principios GRASP**:
    - ¿Las responsabilidades están bien distribuidas según principios como **Creator, Expert, Controller, Polymorphism, Low Coupling, High Cohesion**?
    - ¿Se ha utilizado **Indirection** y **Pure Fabrication** cuando corresponde?
3. **Calidad del diagrama de clases**:
    - **Claridad** en las relaciones y asociaciones entre clases.
    - Uso correcto de **herencia, composición y agregación**.
    - Implementación de **interfaces y patrones de diseño** si es necesario.
4. **Justificación del diseño**:
    - Explicación del uso de cada principio en el diseño.
    - Argumentación sobre cómo las decisiones tomadas mejoran la mantenibilidad y extensibilidad del sistema.

Entregables

1. **Diagrama de clases UML detallado** con todas las entidades relevantes y sus relaciones.
2. Sustentación:
    a. **Justificación** explicando cómo se aplicaron SOLID y GRASP en el diseño.
    b. **Esbozo de la estructura del código** (opcional por bonificación).
    c. **Explicación de la interacción entre las distintas partes del sistema**, destacando la separación de responsabilidades.

Sustentación: 22 de marzo (grupo presencial) y 25 de marzo (grupo virtual)