Tarea ED04_1 – Sistema de Gestión para un Concesionario de Vehículos

Introducción

Un concesionario desea implementar un sistema informático para gestionar los vehículos, los clientes y las ventas. El sistema debe ser flexible y permitir futuras extensiones para integrar nuevas funcionalidades. El objetivo de esta tarea es realizar el diseño de la aplicación haciendo uso de los diagramas de clase UML, siguiendo la notación específica del estándar. A continuación, se detallan los requisitos del sistema.

Contexto General

El concesionario trabaja con tres tipos de vehículos: **coches**, **motocicletas** y **camiones**. Cada tipo de vehículo tiene características comunes, como marca, modelo, precio base, y fecha de fabricación, pero también posee atributos específicos. Los **coches** tienen número de puertas y tipo de combustible. Las **motocicletas** tienen cilindrada y tipo de transmisión. Los **camiones** tienen capacidad de carga y número de ejes.

El concesionario también trabaja con diferentes **clientes**, que pueden ser **particulares** o **empresas**. Los clientes comparten algunos datos básicos como nombre, identificación (DNI o CIF), y datos de contacto. Las empresas tienen información adicional como nombre comercial y responsable de compras.

Funcionalidades del Sistema

1. **Gestión de Vehículos:**
   - Permitir agregar nuevos vehículos al inventario.
   - Consultar vehículos disponibles filtrando por tipo o rango de precio.
   - Calcular el precio final de un vehículo aplicando descuentos u ofertas específicas.
2. **Gestión de Clientes:**
   - Registrar nuevos clientes (particulares o empresas).
   - Consultar información de clientes registrados.
   - Identificar clientes frecuentes y asignarles un descuento fijo.
3. **Gestión de Ventas:**
   - Registrar una venta, asociándola a un cliente y a un vehículo.
   - Permitir que un cliente pueda comprar múltiples vehículos en una única transacción.
   - Registrar la fecha de la venta y calcular el monto total de la misma.
4. **Facturación y Reportes:**
   - Generar una factura detallada para cada venta, incluyendo los datos del cliente, los vehículos comprados, y el precio final.
   - Calcular las ganancias mensuales del concesionario basándose en las ventas registradas.

Requisitos Técnicos

- **Relaciones:**
  - Los vehículos son **propiedad** del concesionario.
  - Cada venta está asociada a un cliente y uno o más vehículos.
  - Los clientes particulares y empresas heredan de una clase abstracta **Cliente**.
  - Los vehículos tienen una relación jerárquica, ya que comparten características comunes.
- **Elementos adicionales:**
  - Implementar una interfaz **DescuentoAplicable** con un método `calcularDescuento` que sea implementado por los clientes frecuentes.
  - Usar atributos y métodos estáticos para llevar un conteo global del total de ventas realizadas.
  - Incorporar métodos en las clases, como `registrarVenta` en el concesionario o `calcularPrecioFinal` en los vehículos.
  - Siguiendo las directrices de los principios SOLID, especialmente el de **responsabilidad única**, deben crearse clases con responsabilidades claras, y por tanto, debemos evitar clases con múltiples responsabilidades. Ten esto en cuenta a la hora de realizar el diseño. Puedes hacer un primer diseño y luego, hacer un refinado intentando aplicar estas técnicas.

Instrucciones de la práctica

Realiza el diagrama de clases cumpliendo con las funcionalidades y requisitos definidos previamente. Realiza un análisis exhaustivo de los requisitos, siguiendo los pasos sugeridos en los apuntes. Añade las aclaraciones que consideres necesarias mediante el uso de notas dentro del propio diagrama.

Una vez analizado el problema, realiza el diagrama en **draw.io**. Puedes hacerlo tanto en la versión web como en la versión de escritorio, o incluso en Visual Studio Code, tal y como se define en los apuntes.

Preguntas de reflexión

Responde las siguientes preguntas basándote en tu solución al diagrama de clases:

- ¿Por qué es útil definir la interfaz `DescuentoAplicable` para los clientes?

Definir la interfaz DescuentoAplicable permite que diferentes tipos de clientes (por ejemplo, clientes VIP o empresas) implementen su propia lógica de descuento sin modificar la clase base Cliente

- ¿Qué ventajas aporta el uso de clases abstractas (`Vehiculo` y `Cliente`) para modelar jerarquías?

Las clases abstractas permiten definir un comportamiento común para todas las subclases, evitando la duplicación de código. Además, facilitan la reutilización y aseguran que todas las subclases tengan ciertas características esenciales. En este caso, Vehiculo puede definir atributos como marca, modelo y precio, mientras que Cliente puede establecer datos generales como nombre y DNI.

- Justifica la relación empleada entre `Venta` y `Vehiculo`/`Cliente` (asociación, composición, agregación...).

  La relación entre Venta y Vehiculo es agregación, ya que un vehículo puede existir sin estar asociado a una venta, pero una venta necesita al menos un vehículo.
  La relación entre Venta y Cliente también es agregación, pues un cliente puede realizar múltiples compras, y una venta siempre está asociada a un cliente.

- ¿Cómo el uso de interfaces y herencia mejora la extensibilidad del sistema?

  La combinación de interfaces y herencia permite agregar nuevas funcionalidades sin afectar el código existente. Por ejemplo, una nueva clase ClienteCorporativo puede implementar DescuentoAplicable sin modificar Cliente, y una nueva clase Moto puede extender Vehiculo sin cambiar las clases existentes.

- ¿Qué ventajas ofrece dividir la clase `Concesionario` en varios componentes especializados?

  Dividir Concesionario en módulos como Ventas, Clientes y Vehiculos. permite seguir el principio de responsabilidad única (SRP), facilitando el mantenimiento y la escalabilidad del sistema.

- ¿Cómo garantizarías que el sistema sea extensible para incluir nuevos tipos de vehículos o clientes?

  Usando la herencia para permitir que nuevas clases (Camion, Motocicleta) extiendan Vehiculo, lo mismo pasa con cliente pudiendo expandir los tipos de clientes.
  Implementando interfaces como DescuentoAplicable para permitir nuevas estrategias de descuento.

- ¿Qué principios SOLID están mejor representados en este diseño y cómo?

  SRP (Responsabilidad Única): con la separación de Concesionario en diferentes gestores.
  OCP (Abierto/Cerrado): Uso de interfaces como DescuentoAplicable y herencia para extender sin modificar nuestro proyecto.
  LSP (Sustitución de Liskov): Las subclases de Vehiculo y Cliente pueden sustituir a sus clases base sin afectar la funcionalidad.
  DIP (Inversión de Dependencias): Se usan interfaces en lugar de depender directamente de clases concretas.

- ¿Es recomendable o deseable representar los constructores y getters/setters de las clases en el diagrama de clases? ¿Hay casos donde esa representación tenga más sentido? Justifica si, en tu caso, has decidido incluirlos en el diagrama o no.

  Depende del propósito del diagrama:
  - **Si el diagrama es de alto nivel, incluir constructores y getters/setters puede hacer que se vea sobrecargado.**
  - **Si el diagrama es detallado y se usa para la implementación, puede ser útil mostrar estos métodos.**

En este caso, como el diagrama está orientado a diseño conceptual, no incluí constructores ni getters/setters, ya que lo importante es la estructura y relaciones entre clases; ademas de que no los solicitaba el ejercicio.

Instrucciones de entrega

Una vez realizado el diagrama, **expórtalo en formato PNG**. Deberás entregar:

- Un **archivo pdf**, respondiendo a las preguntas de reflexión, y con la imagen del diagrama de clases (el PNG previo)
- El archivo .drawio generado por la aplicación

Deberás comprimir ambos archivos en uno único y con el nombre siguiendo el formato **apellidos_nombre_ED04_1.zip**

Puntuación Propuesta

El ejercicio será evaluado considerando los siguientes criterios:

1. **Identificación y uso correcto de relaciones (3 puntos):**
    - Composición (1 punto).
    - Asociación con multiplicidad (1 punto).
    - Herencia y uso de clases abstractas (1 punto).
2. **Uso de interfaces y elementos estáticos (2 puntos):**
    - Definición e implementación correcta de la interfaz (1 punto).
    - Uso de atributos/métodos estáticos en el contexto adecuado (1 punto).
3. **Implementación de métodos y funcionalidades (3 puntos):**
    - Métodos adecuados en cada clase (1.5 puntos).
    - Correcta representación de las funcionalidades del sistema (1.5 puntos).
4. **Claridad y completitud del modelo (1 punto):**
    - Representación clara de atributos y métodos.
    - Inclusión de multiplicidades y anotaciones relevantes.
5. **Respuesta a las preguntas de reflexión (1 punto)**