Ejercicios.

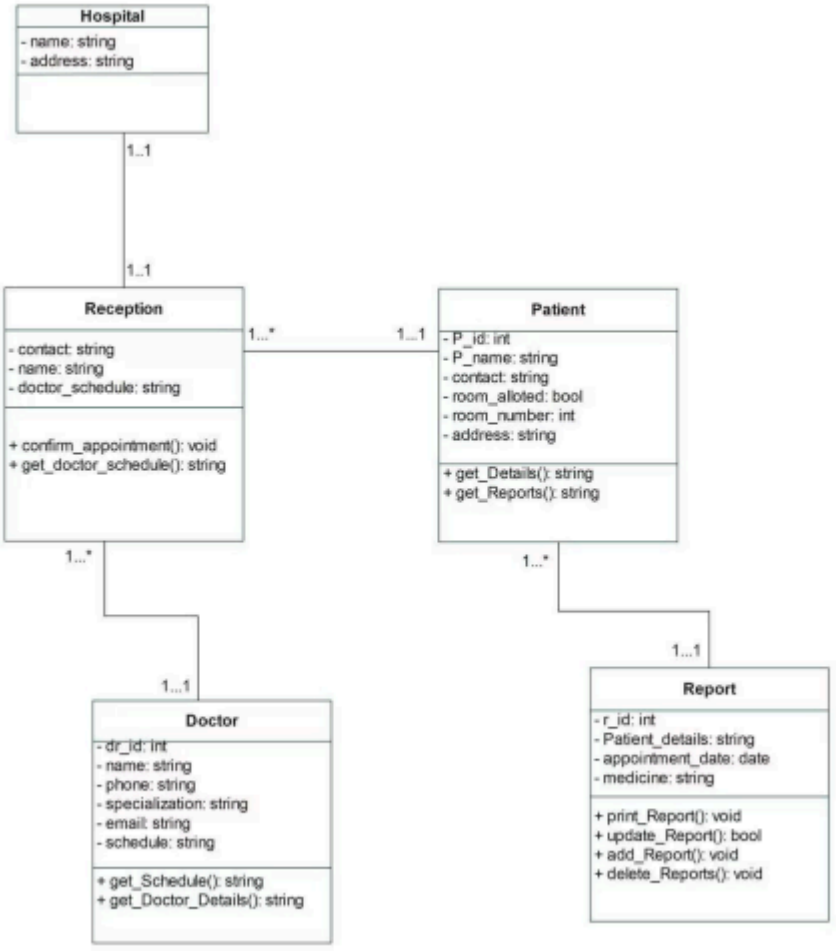
1. Estudia el siguiente diagrama. Fíjate en los siguientes aspectos:

a. La visibilidad de los atributos y funciones

b. La multiplicidad de las asociaciones

c. Qué tipo de relaciones hay entre las clases ¿hay alguna jerarquía?

d. ¿Cómo crees que se representarán esas relaciones al convertir el diagrama en código?



a) Visibilidad

Todos los atributos tienen el signo (-), así que son privados. No se pueden usar directamente desde fuera de la clase.

Todos los métodos tienen el signo (+), así que son públicos y sí se pueden usar desde otras clases.

b) Multiplicidad

Hospital – Reception: 1 a 1. Un hospital tiene una recepción y cada recepción pertenece a un hospital.

Reception – Patient: 1 a muchos (1..*). Una recepción gestiona varios pacientes, pero cada paciente tiene una sola recepción.

Reception – Doctor: 1 a muchos (1..*). Una recepción gestiona varios doctores, pero cada doctor pertenece a una sola recepción.

Patient – Report: 1 a muchos (1..*). Un paciente puede tener varios informes, pero cada informe es de un solo paciente.

c) Tipo de relaciones

Todas son asociaciones normales. No hay herencia, ni composición ni agregación. No hay jerarquía entre clases.

d) Cómo se representaría en código

Las relaciones 1 a 1 se harían con un atributo del tipo de la otra clase.

Las relaciones 1 a muchos se harían con una lista o colección (por ejemplo, un ArrayList) dentro de la clase que tiene muchos.

2. Ahora vamos con otro diagrama. Fíjate en los siguientes aspectos:

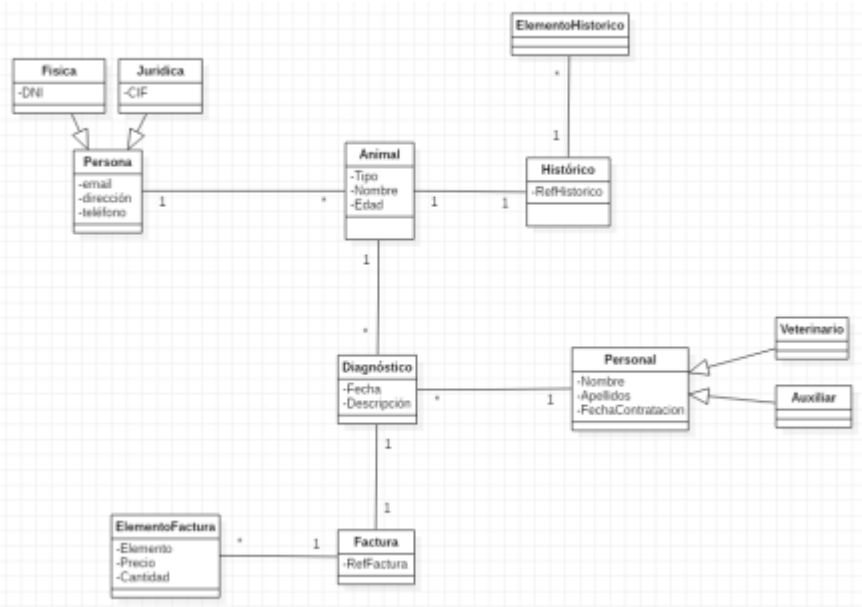
a. La visibilidad de los atributos y funciones

b. La multiplicidad de las asociaciones

c. Qué tipo de relaciones hay entre las clases ¿hay alguna jerarquía?

d. ¿Cómo crees que se representarán esas relaciones al convertir el diagrama en código?

e. No aparecen los métodos explicitados en las clases. Imagina cuales sería interesante incluir



a) Visibilidad

Todos los atributos aparecen con el signo (-), así que son privados.

No se muestran métodos en las clases, solo atributos.

b) Multiplicidad

Persona – Animal: 1 a muchos (1..*).

Animal – Histórico: 1 a 1.

Histórico – ElementoHistórico: 1 a muchos (1..*).

Animal – Diagnóstico: 1 a muchos (1..*).

Diagnóstico – Personal: muchos a 1.

Diagnóstico – Factura: 1 a 1.

Factura – ElementoFactura: 1 a muchos (1..*)

c) Tipo de relaciones y jerarquía

Hay asociaciones normales entre la mayoría de clases.

Sí hay jerarquía en dos casos:
Persona es la clase padre de Física y Jurídica (herencia).
Personal es la clase padre de Veterinario y Auxiliar (herencia).

No se ven composiciones ni agregaciones marcadas explícitamente.

d) Cómo se representaría en código

La herencia se representaría usando extends (en Java).

Las relaciones 1 a 1 se harían con un atributo del tipo de la otra clase.

Las relaciones 1 a muchos se harían con listas (por ejemplo, ArrayList).

Cada clase tendría sus atributos privados y métodos públicos.

e) Métodos que sería interesante incluir

En Persona: mostrar y actualizar datos.

En Animal: añadir diagnóstico y consultar histórico.

En Histórico: añadir elemento.

En Diagnóstico: crear y modificar diagnóstico.

En Factura: calcular total y añadir elemento.

En Personal: registrar diagnóstico y mostrar datos.

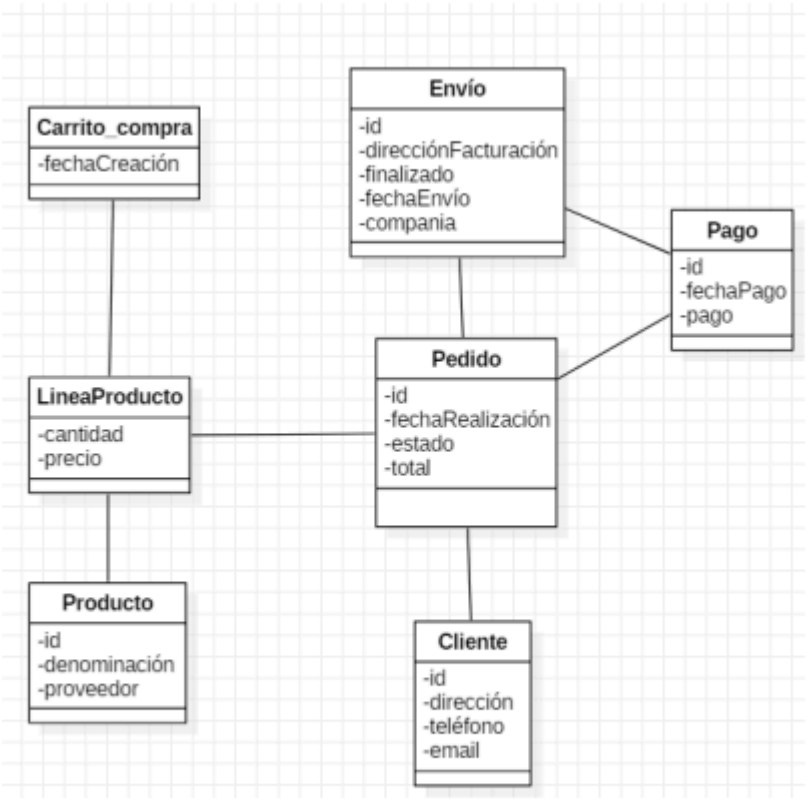
3. Dado el siguiente diagrama de clases:

a. Retoca las relaciones entre clases ¿Debería ser alguna una composición o agregación?

b. Reflexiona sobre cuales serían las multiplicidades en las relaciones en las que tenga sentido ponerlas

c. Qué tipo de relaciones hay entre las clases ¿hay alguna jerarquía?

d. ¿Cómo crees que se representarán esas relaciones al convertir el diagrama en código?



a) Retocar relaciones

Carrito_compra – LineaProducto debería ser composición

Pedido – LineaProducto también debería ser composición, porque un pedido está formado por sus líneas.

Pedido – Envío debería ser agregación o composición.

Pedido – Pago debería ser composición, porque el pago pertenece a un pedido concreto.

b) Multiplicidades

Carrito_compra – LineaProducto: 1 a muchos (1..*).

Pedido – LineaProducto: 1 a muchos (1..*).

LineaProducto – Producto: muchos a 1.

Pedido – Cliente: muchos a 1.

Pedido – Envío: 1 a 1.

Pedido – Pago: 1 a 1.

c) Tipo de relaciones y jerarquía

Hay asociaciones y composiciones (según lo que hemos retocado).

No hay herencia, así que no existe jerarquía entre clases.

d) Cómo se representaría en código

La composición se haría con atributos tipo lista dentro de la clase principal (por ejemplo, un Pedido tendría una lista de LineaProducto).

Las relaciones muchos a 1 se harían con un atributo simple (por ejemplo, LineaProducto tendría un Producto).

Pedido tendría atributos de tipo Cliente, Envío y Pago.

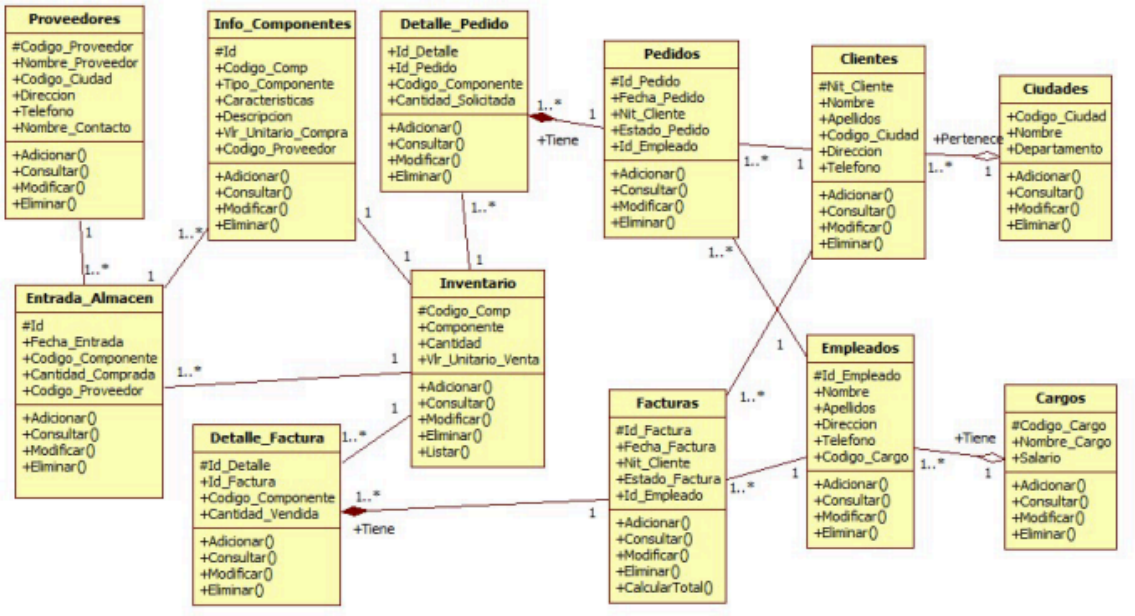
Todos los atributos serían privados y se accedería mediante métodos públicos.

4. Dado el siguiente diagrama de clases:

a. Observa las relaciones de agregación y composición e indica en qué influye en el código posterior que sean de un tipo o de otro

b. Revisa las multiplicidades y anota en qué clases aparecerán "colecciones" de objetos de otra clase al codificar

c. Observa la visibilidad de atributos y métodos ¿Cambiarías algo?



a) Agregación vs composición

Composición: el "todo" posee las partes → normalmente lista y al borrar el todo se borran las partes. (Ej.: Pedido–Detalle_Pedido, Factura–Detalle_Factura).

Agregación/asociación: objetos independientes → referencias simples, no se borran en cascada.

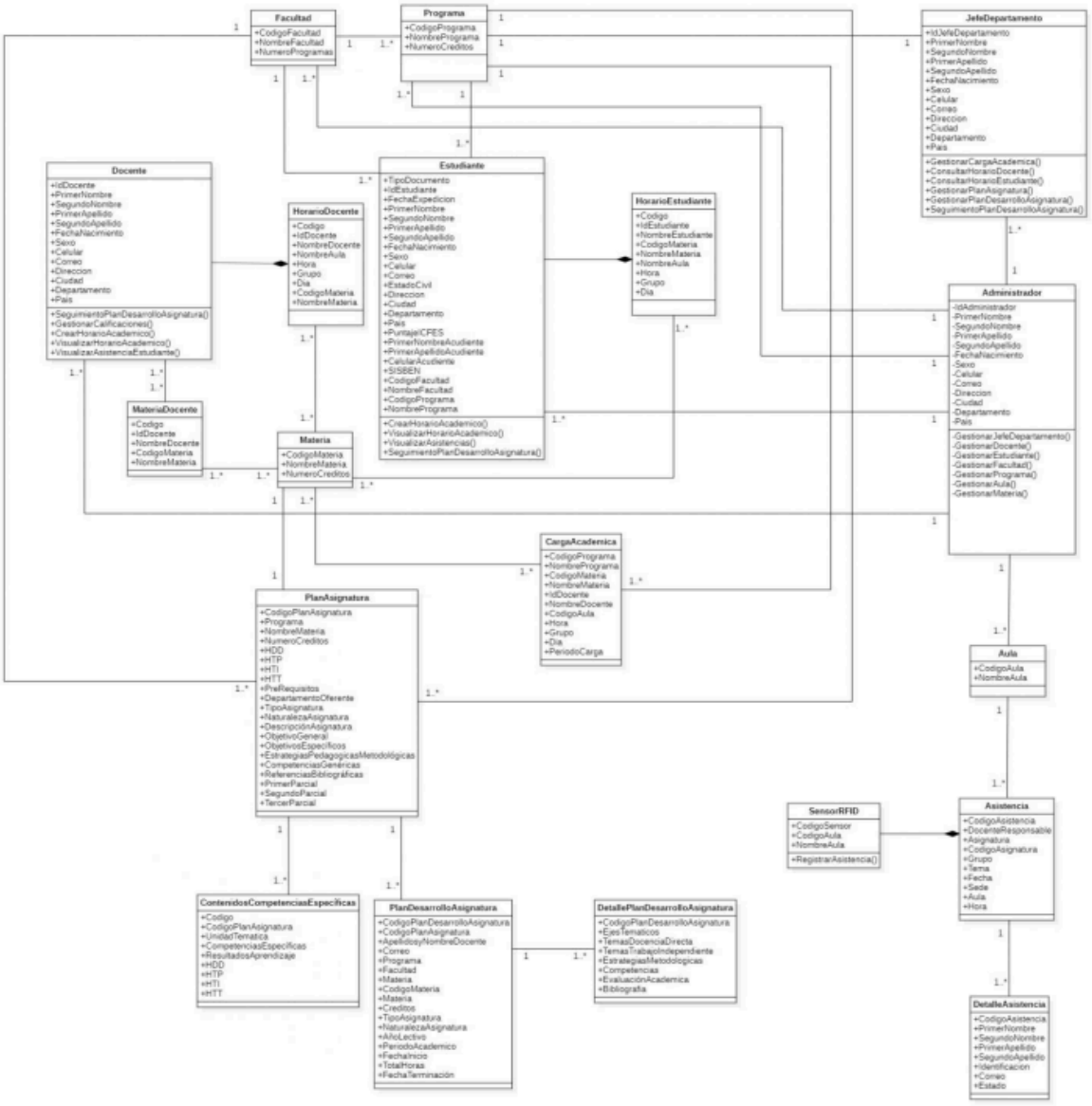
b) Dónde habrá "colecciones"

Pedidos → List<Detalle_Pedido>
Facturas → List<Detalle_Factura>
Clientes → List<Pedidos> y List<Facturas>
Proveedores → List<Entrada_Almacen>

c) Visibilidad

Los atributos los cambiaria a privados, -

5. Dado el siguiente diagrama de clases, como se podría modificar para:
• Que docentes y estudiantes optimicen su estructura
• Que un docente sea tutor de otro docente en prácticas
• Que una misma asignatura pudiera estar en más de un programa
• Que una materia docente la pueda impartir más de un profesor



1. Optimización de Docente y Estudiante
Crear una superclase Persona con los atributos comunes.
Docente y Estudiante heredan de Persona.

2. Docente tutor de otro docente en prácticas
Agregar una relación reflexiva en Docente:
Docente (Tutor) 1 — 0..* Docente (En prácticas).

3. Una asignatura en más de un programa

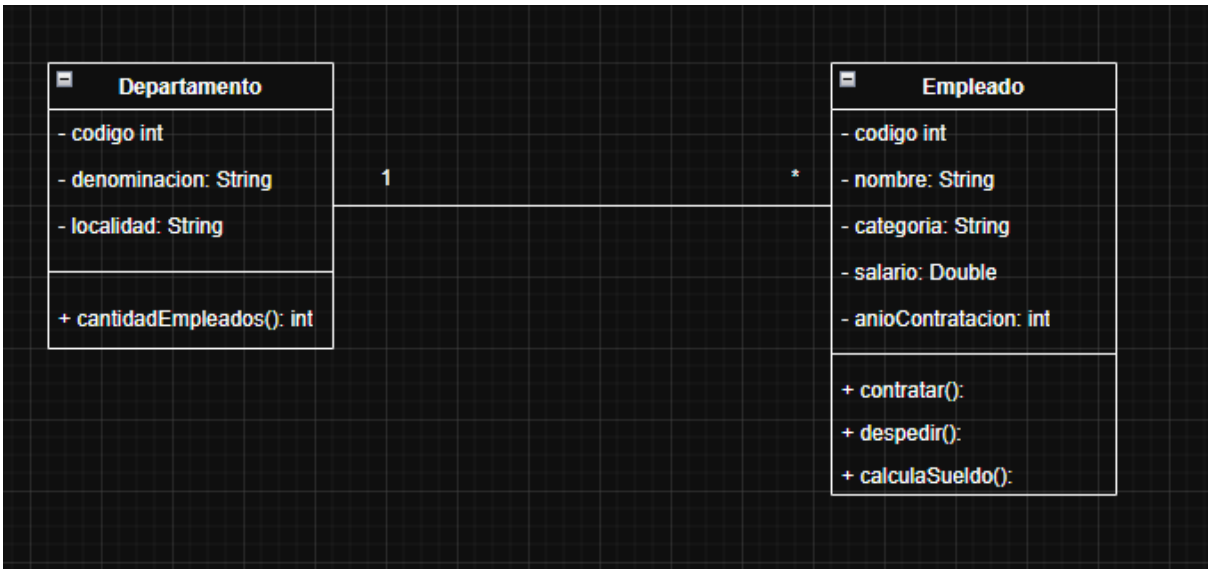
Modificar la relación Programa–Asignatura a muchos a muchos (N:M).
Opcional: crear clase intermedia ProgramaAsignatura.

4. Una materia impartida por varios profesores

Modificar la relación MateriaDocente–Docente a muchos a muchos (N:M).
Opcional: crear clase intermedia DocenteMateria.

6. Representa mediante un diagrama de clases realizado con la siguiente especificación de los datos persistentes de una aplicación:

a. Una empresa, se divide en departamentos, cada uno de los cuales está formado por un número de empleados.
b. Los datos a considerar de un empleado son código, nombre, categoría, salario y año de contratación.
c. De cada departamento, nos interesa manejar código, denominación y localidad. También se podrá realizar un recuento de los empleados que forman parte de un departamento
d. Un empleado puede ser jefe de varios empleados y empleado sólo tiene un jefe directo
e. Por otra parte, el empleado puede ser contratado, despedido y se podrá calcular su sueldo cada mes
f. ¿Qué otros métodos deberían añadirse a cada clase, como mínimo?



7. Se desea diseñar un diagrama de clases sobre los datos persistentes de las reservas de una empresa dedicada al alquiler de vehículos, teniendo en cuenta que:

a. Un determinado cliente puede tener en un momento dado hechas ninguna, una o varias reservas.

b. De cada cliente se desean almacenar su DNI, nombre, dirección y teléfono. Además, dos clientes se diferencian por un código único.

c. Cada cliente puede ser avalado por otro cliente de la empresa. De ser así, nos interesa saber quién avala a quien

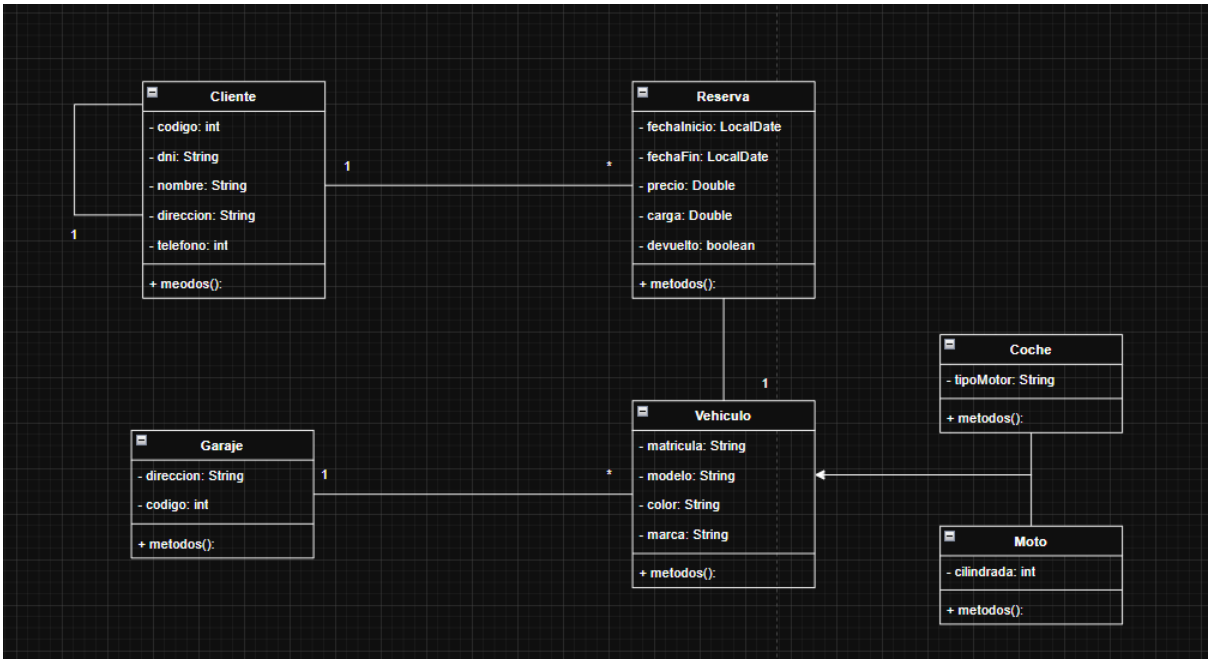
d. Una reserva la realiza un único cliente y siempre involucra a un sólo vehículo.

e. Es importante registrar la fecha de inicio y final de la reserva, el precio del alquiler del vehículo (que puede ser distinto según fechas y otros criterios: fidelidad, ofertas…), los litros de gasolina en el depósito en el momento de realizar la reserva, y un indicador de si ha sido devuelto.

f. Todo vehículo tiene siempre asignado un determinado garaje del que se saca y al que se devuelve. Del garaje necesitamos conocer dirección (calle, número, población y código postal). En un garaje puede estar estacionado más de un vehículo.

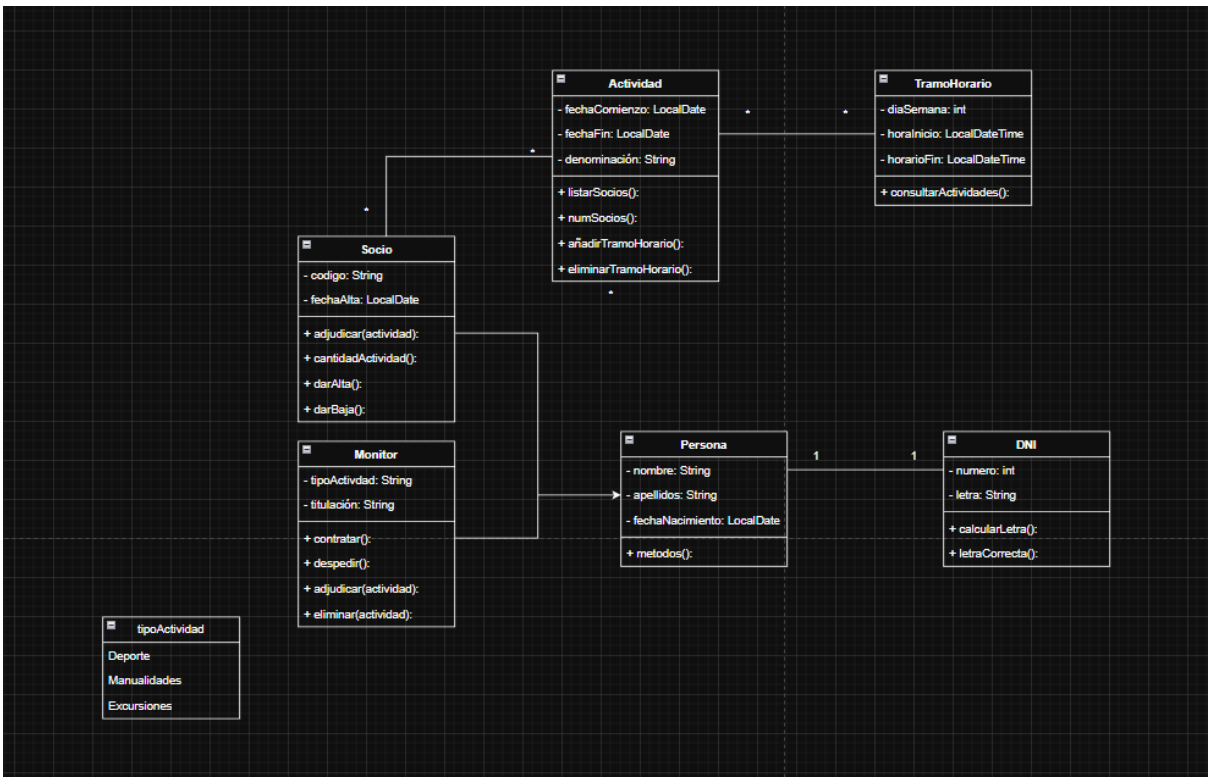
g. De cada vehículo, se requiere registrar su matrícula, modelo, color y marca. Si es una moto, además la cilindrada, y si es un coche, si es de gasolina, gasoil, eléctrico o híbrido

h. De momento, no detallaremos los métodos relevantes en cada clase, y sólo tendremos en cuenta sus atributos.



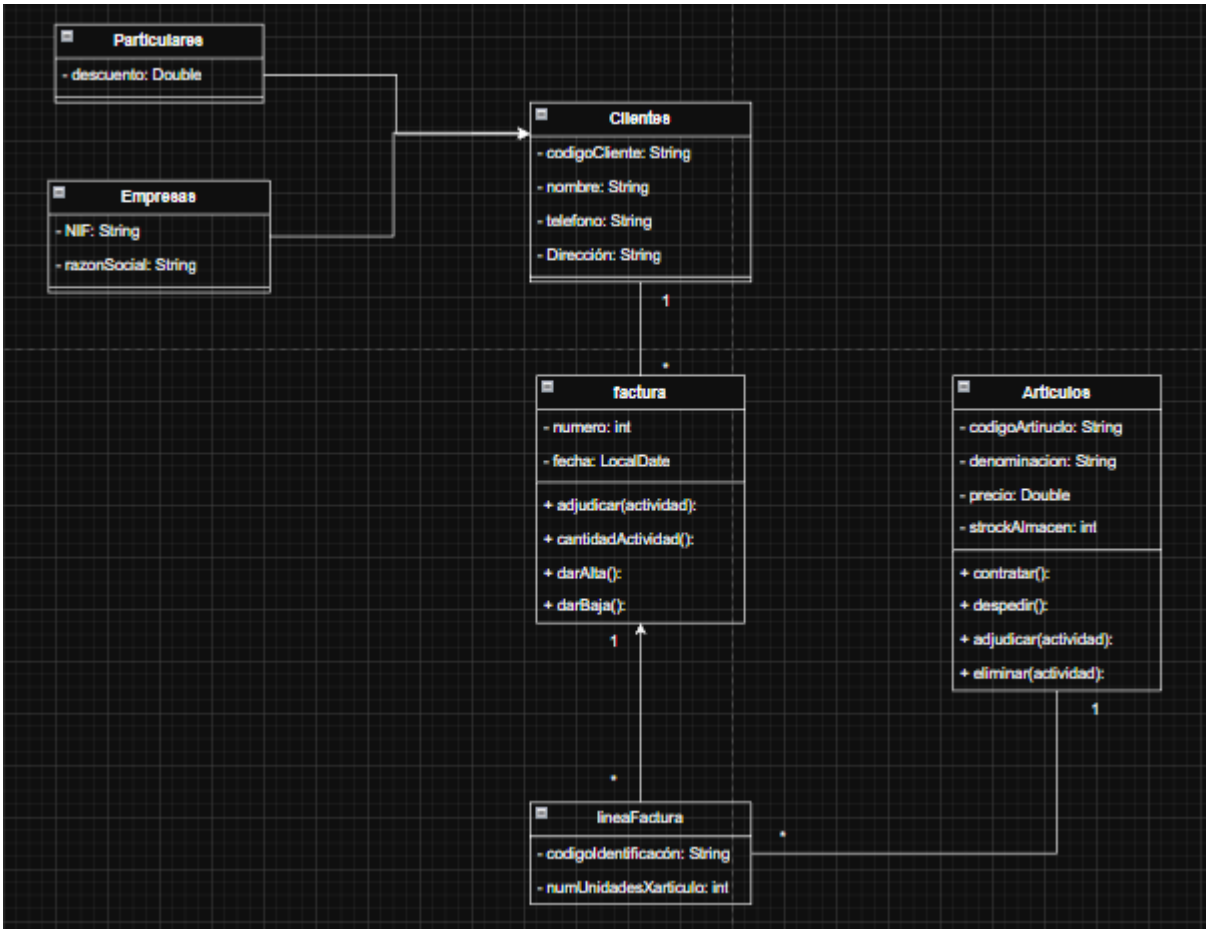
8. Diseña un diagrama de clases que modele solo las clases persistentes relativas al proceso de gestión de las personas que se apuntan a una asociación, en la cual podrán realizar distintos tipos de actividades lúdicas y deportivas. Utiliza el entorno que desees:

- **El sistema deberá tener en cuenta a todas las personas relacionadas con la asociación, ya sean socios o monitores**
- **De cada persona interesa conocer sus datos básicos: NIF, nombre completo y fecha de nacimiento**
- **Un socio puede darse de alta o de baja y apuntarse a una o más actividades. Cuando cada nuevo socio se da de alta, se le asigna un código de asociado, alfanumérico y se anota la fecha de alta**
- **Queremos también poder computar a cuantas actividades está apuntado un socio, y listarlas, y así mismo, dada una actividad, saber cuántos socios están apuntados y listarlos**
- **De un monitor, interesa conocer el tipo genérico de actividad que imparte (deporte, manualidades, excursiones,) y la titulación que tiene para hacerlo. Un monitor puede ser contratado, despedido, adjudicado a una actividad, o eliminado de ésta**
- **La fecha tiene tres campos (día, mes y año) de tipo entero. Queremos poder calcular los meses transcurridos entre dos fechas, y en que cae una fecha a partir del momento actual y un número de meses**
- **El NIF se modela como una clase con un campo de tipo entero llamado DNI y un campo de tipo carácter llamado letra. Queremos poder calcular la letra a partir del DNI y comprobar si una combinación letra-DNI es correcta**
- **Una actividad tiene una denominación, una fecha de comienzo, una de fin, y uno o varios tramos horarios. Queremos poder añadir y eliminar tramos a una actividad**
- **El tramo horario incluye el día de la semana (de 1 a 7) y la hora de comienzo y de fin. Es interesante saber qué actividades se ofertan en un tramo dado**
- **Una actividad además es impartida por uno o varios monitores y también un monitor podrá impartir uno o más talleres o actividades diferentes**


Actividad
- fechaComienzo: LocalDate
- fechaFin: LocalDate
- denominación: String
+ listarSocios():
+ numSocios():
+ añadirTramoHorario():
+ eliminarTramoHorario():
TramoHorario
- diaSemana: int
- horaInicio: LocalDateTime
- horarioFin: LocalDateTime
+ consultarActividades():
Socio
- codigo: String
- fechaAlta: LocalDate
+ adjudicar(actividad):
+ cantidadActividad():
+ darAlta():
+ darBaja():
Monitor
- tipoActivdad: String
- titulación: String
+ contratar():
+ despedir():
+ adjudicar(actividad):
+ eliminar(actividad):
Persona
- nombre: String
- apellidos: String
- fechaNacimiento: LocalDate
+ metodos():
1
1
DNI
- numero: int
- letra: String
+ calcularLetra():
+ letraCorrecta():
tipoActividad
Deporte
Manualidades
Excursiones

9. Se desea realizar el análisis de un sistema de gestión informática de un pequeño almacén dedicado a vender artículos a clientes. Para ello se necesita manejar los datos de clientes, artículos, facturas y detalle de las facturas:

- **Los datos de los clientes son: código de cliente, nombre, teléfono y dirección**
- **Los datos de los artículos son: código de artículo, denominación, pvp y stock de almacén • Los datos de las facturas son: número de factura y fecha de la factura**
- **Hay que tener en cuenta que un cliente puede tener muchas facturas o ninguna, y que cada factura está compuesta de una o varias líneas de factura**
- **Cada línea de factura tiene un código que las identifica, así como un código del artículo que se factura y el número de unidades de este**
- **Existen dos tipos de clientes: empresas y particulares**
- **De los clientes empresa se necesita saber el CIF y la razón social**
- **De los particulares, el porcentaje de descuento**



11. Se desea realizar el análisis de un sistema de gestión informática de una pequeña agencia que oferta viajes a sus clientes

- El sistema debe proporcionar una ventana inicial de login que dará paso a otra ventana con una serie de menús que conectará con el resto de ventanas de la aplicación, permitiendo realizar las siguientes acciones:

- La gestión de reservas de viajes para realizar reservas, modificar reservas, consultar reservas, borrar reservas y generar e imprimir facturas

- El mantenimiento de datos de clientes, para mantener actualizados los datos se realizarán operaciones de consultas, altas, bajas y modificaciones de datos de clientes, y además debe permitir generar listados de clientes

- Mantenimiento de datos de viajes, para mantener actualizados los datos se realizarán operaciones de consultas, altas, bajas, modificaciones e informes de viajes. Disponemos de una base de datos donde están almacenados los datos de los clientes, los viajes, las reservas, las fechas de viaje. Los datos son los siguientes:

Datos de clientes: código de cliente, nombre, teléfono y dirección

Datos de viajes: código, nombre, plazas y precio

Datos de las reservas: número de reserva y estado de la reserva

Un cliente puede realizar muchas reservas y una reserva es de un solo cliente

Igualmente, de un viaje se pueden realizar muchas reservas, y una reserva pertenecerá a un viaje

Los viajes se ofertan en varias fechas de viaje. De estas se necesita saber la fecha de comienzo y de fin. Varios viajes pueden compartir las mismas fechas.

También se cuenta con la información de un catálogo de viajes, los datos del catálogo son; código, destino, procedencia, temporada y precio.

Los viajes se crean a partir del catálogo

En relación con la lógica de la aplicación, necesitaremos diseñar clases de control que realicen las operaciones CRUD directas de cada una de las clases correspondientes a cada entidad