Relación de ejercicios diagramas de clases UML

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

a. *La visibilidad de los atributos y funciones* → Atributos privados (-) y los métodos públicos (+)

b. *La multiplicidad de las asociaciones* →

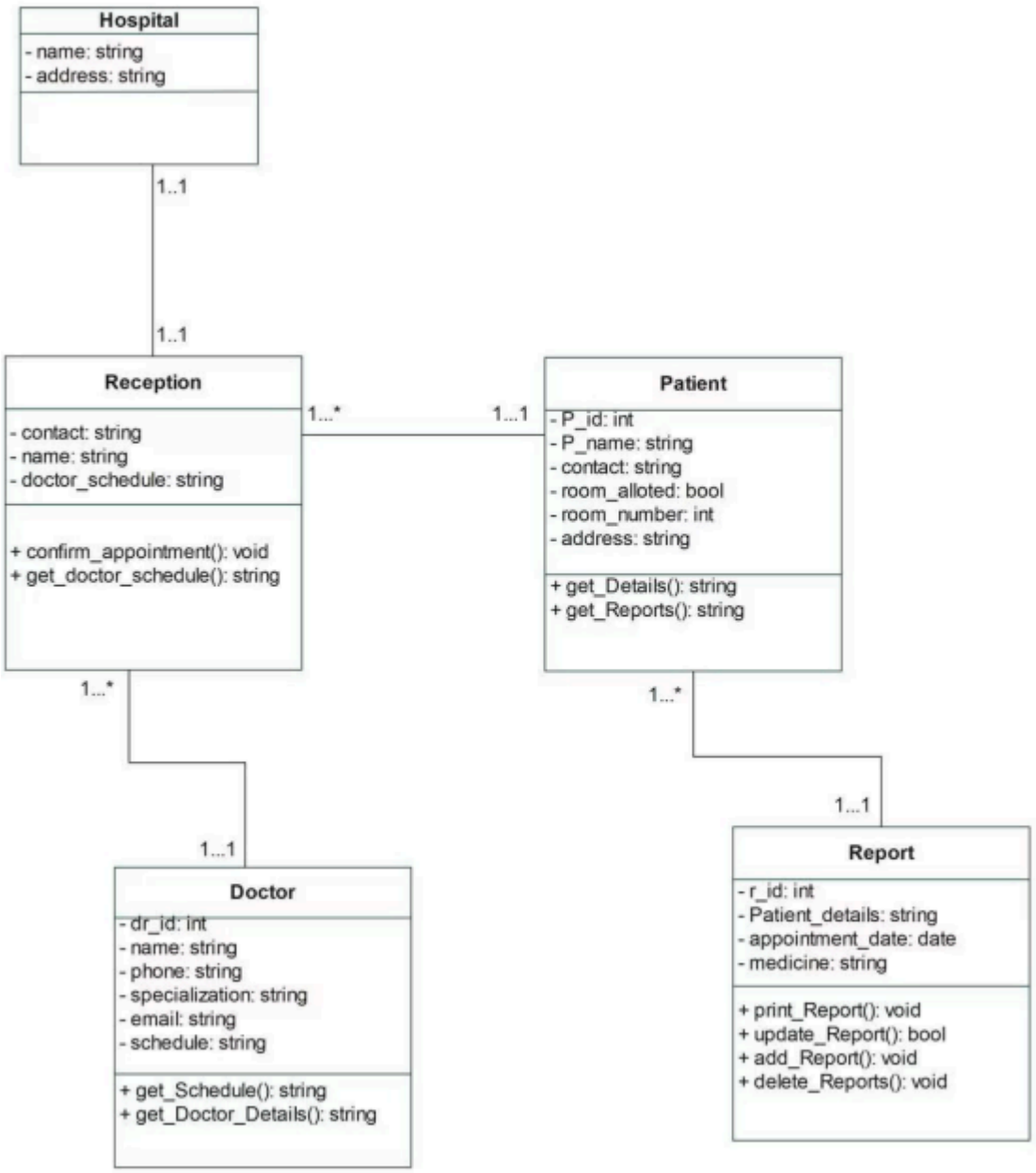
Hospital 1..1 – 1..1 Reception → Un hospital tiene una sola recepción
Hospital 1..* – 1..1 Patient → Un hospital tiene muchos pacientes
Patient 1..* — 1..1 Doctor → Cada paciente tiene un doctor, pero un doctor puede tener muchos pacientes.
Patient 1..* — 1..1 Report → Un paciente puede tener muchos informes, pero cada informe pertenece a un solo paciente.

c. *Qué tipo de relaciones hay entre las clases ¿hay alguna jerarquía?* → Todas son asociaciones, no hay herencia.

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

```
class Hospital {
   private Reception reception;
   private List<Patient> pacientes;
}

class Patient {
   private Doctor doctor;
   private List<Report> reports;
}
```



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

a. *La visibilidad de los atributos y funciones* → Todos los atributos son privados. No aparecen métodos.

b. *La multiplicidad de las asociaciones* →

Persona 1 — * ElementoHistorico
ElementoHistorico 1 — * Diagnóstico
Persona 1 — 1 Veterinario
Persona 1 — 1 Auxiliar
Factura 1 — * ElementoFactura

c. *Qué tipo de relaciones hay entre las clases, ¿hay alguna jerarquía?* → Hay herencia que serían:

Persona ← Física
Persona ← Jurídica
Personal ← Veterinario
Personal ← Auxiliar

Y asociaciones:

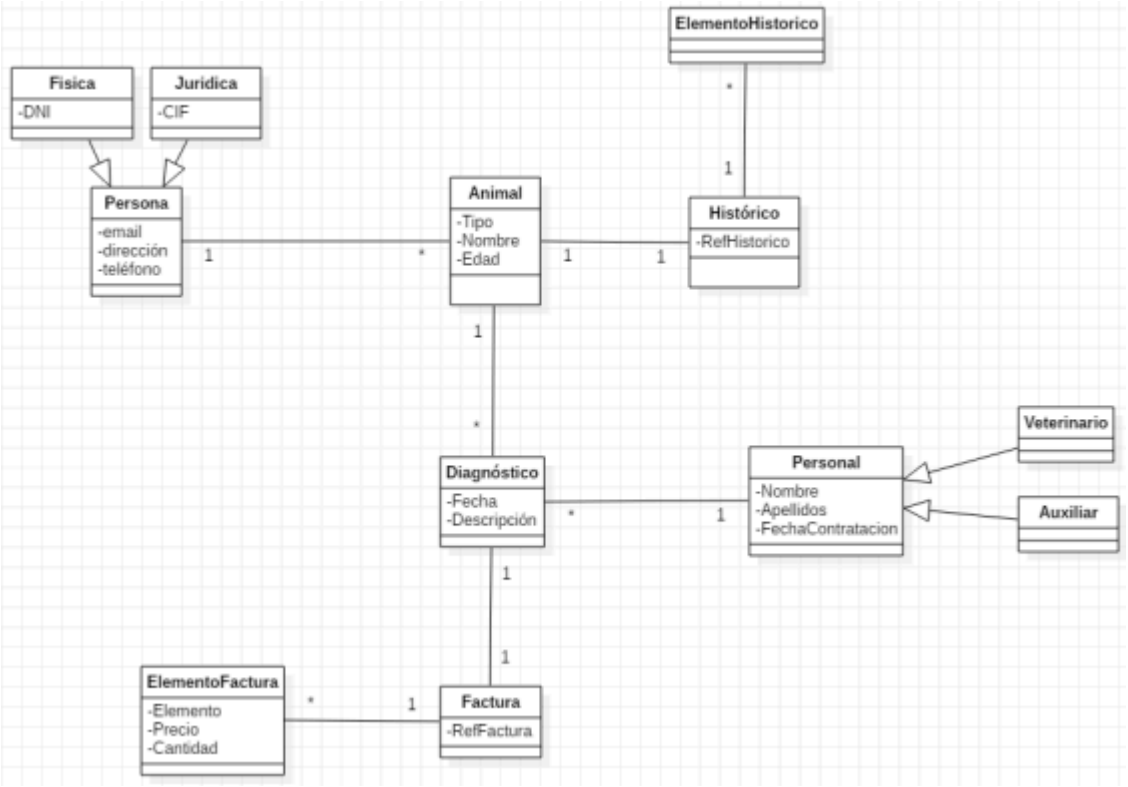
Persona – ElementoHistórico
ElementoHistórico – Diagnóstico
Factura – ElementoFactura

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

```
class Persona { ... }
class Fisica extends Persona { ... }
class Juridica extends Persona { ... }
```

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

Persona: getEdad(), getDirection()
ElementoHistorico: addDiagnostico()
Diagnóstico: imprimir()
Factura: calcularTotal()



3. **Dado el siguiente diagrama de clases:**

a. Retoca las relaciones entre clases ¿Debería ser alguna una composición o agregación? → Sí, por ejemplo:

Pedido – LineaProducto → Composición
Una línea no existe sin un pedido.
CarritoCompra – Producto → Agregación
El carrito contiene productos, pero estos existen fuera.

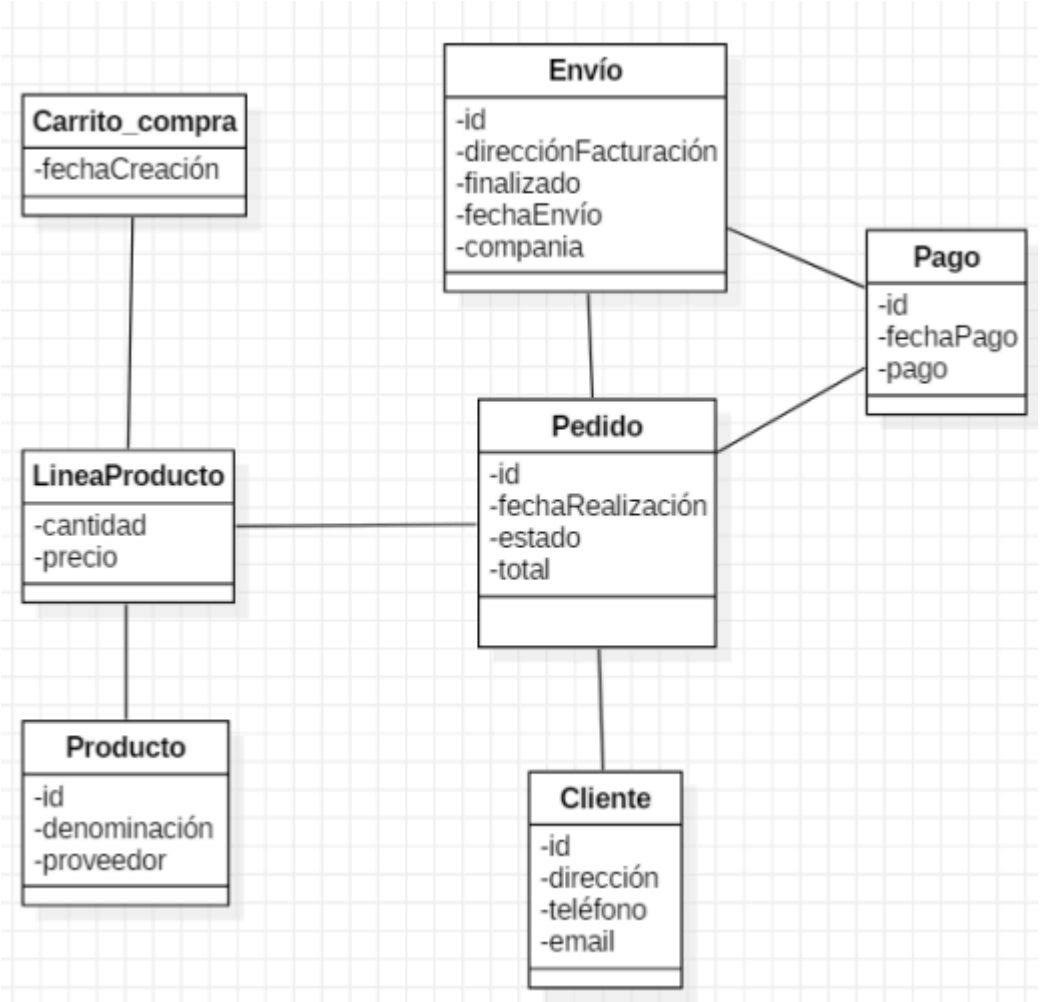
b. Reflexiona sobre cuáles serían las multiplicidades en las relaciones en las que tenga sentido ponerlas →

Cliente 1 — * Pedido
Pedido 1 — * LineaProducto
LineaProducto 1 — 1 Producto
Pedido 1 — 1 Pago
Pedido 1 — 1 Envío
CarritoCompra 1 — * Producto

c. Qué tipo de relaciones hay entre las clases ¿hay alguna jerarquía? → No hay herencia.

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

```
class Pedido {
   private List<LineaProducto> lineas;
   private Pago pago;
   private Envio envio;
}
```



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 →

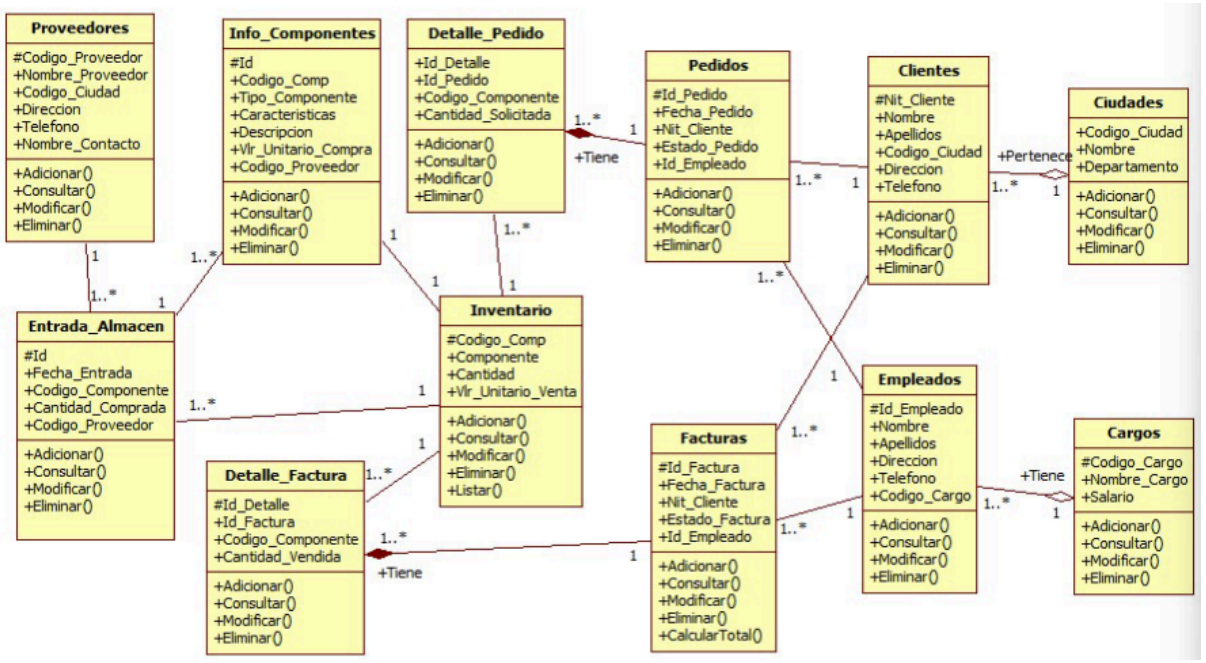
Agregación → se usa una lista normal
Composición → el objeto se crea y destruye dentro de la clase contenedora
Ejemplo:

```
class Pedido {
   private List<Detalle_Pedido> detalles; // composición
}
```

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

Proveedores 1..* Info_Componentes
Pedidos 1..* Detalle_Pedido
Facturas 1..* Detalle_Factura
Ciudades 1..* Proveedores
Empleados 1..* Facturas

c. Observa la visibilidad de atributos y métodos ¿Cambiarías algo? → Muchos atributos aparecen públicos.
Deberían ser privados y gestionarse con getters/ setters.



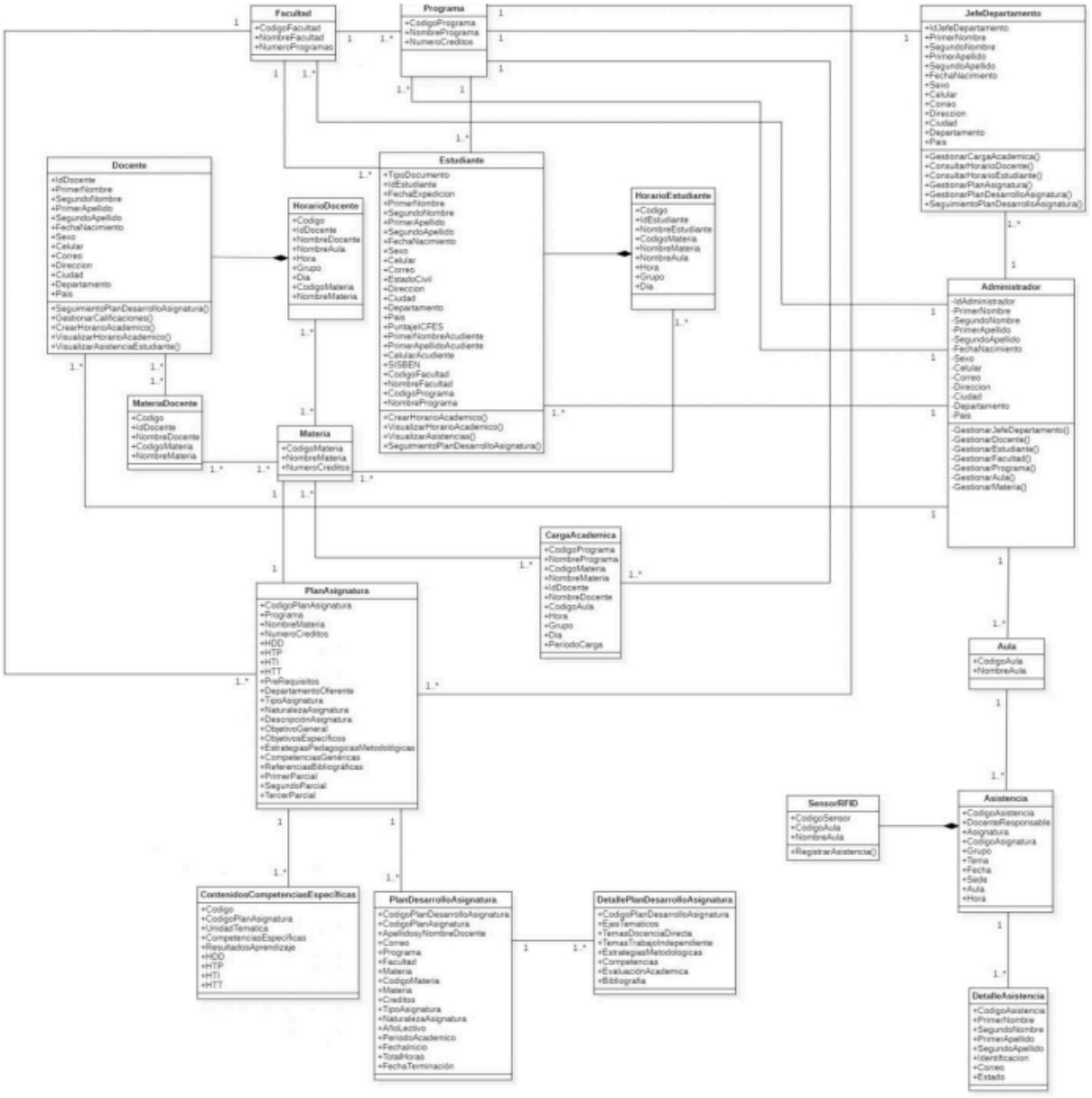
5. Dado el siguiente diagrama de clases, como se podría modificar para:

a. Que docentes y estudiantes optimicen su estructura → Ambos comparten datos personales, por lo que podríamos crear superclases.

b. Que un docente sea tutor de otro docente en prácticas → Como una asociación reflexiva

c. Que una misma asignatura pudiera estar en más de un programa → Cambiar relación a *..* (muchos a muchos) y crear una clase intermedia si se necesitan datos.

d. Que una materia docente la pueda impartir más de un profesor → Cambiar relación Docente – Materia a *..*



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.

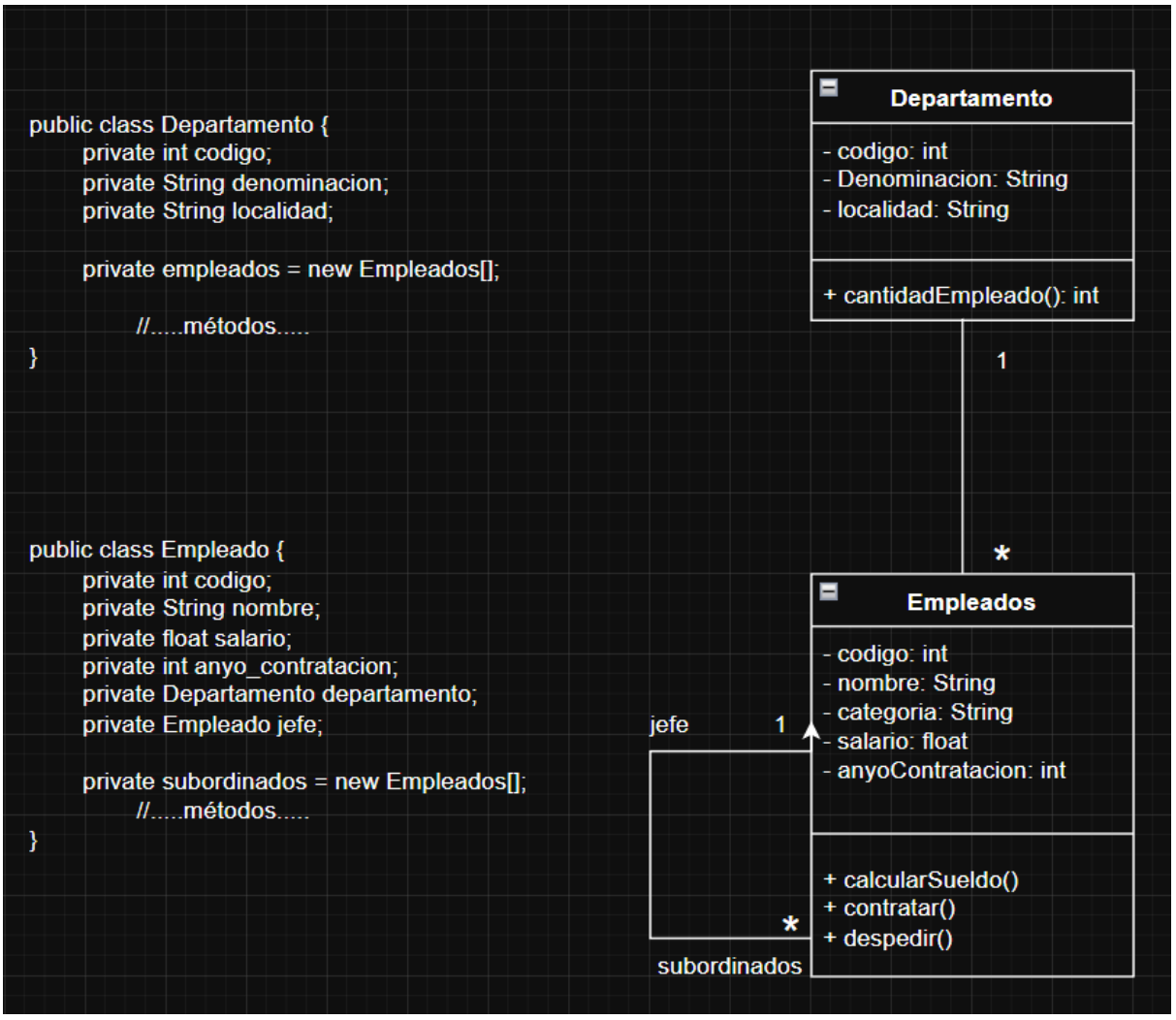
b. Los datos a considerar de un empleado son código, nombre, categoría, salario y año de contratación.

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

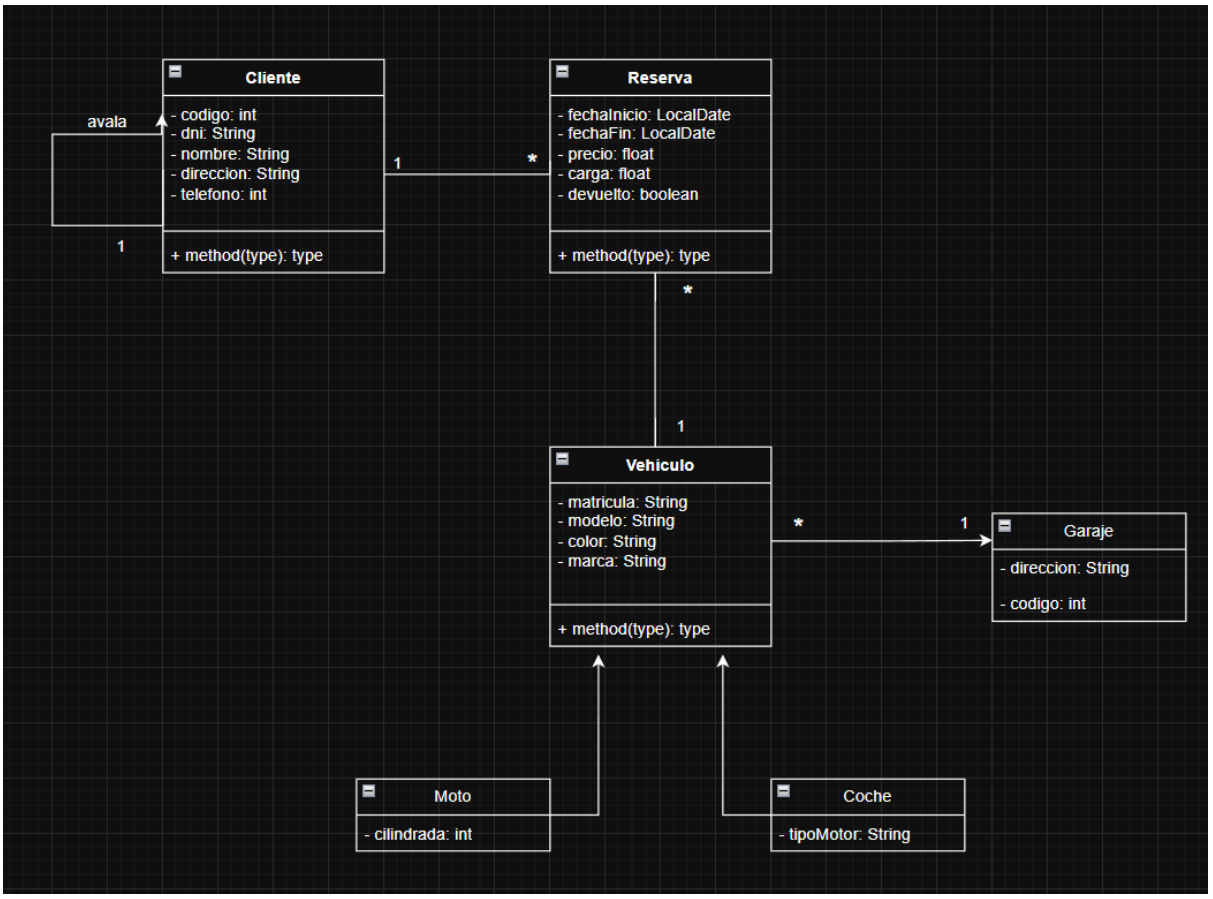
f. ¿Qué otros métodos deberían añadirse a cada clase, como mínimo?



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 desea 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,

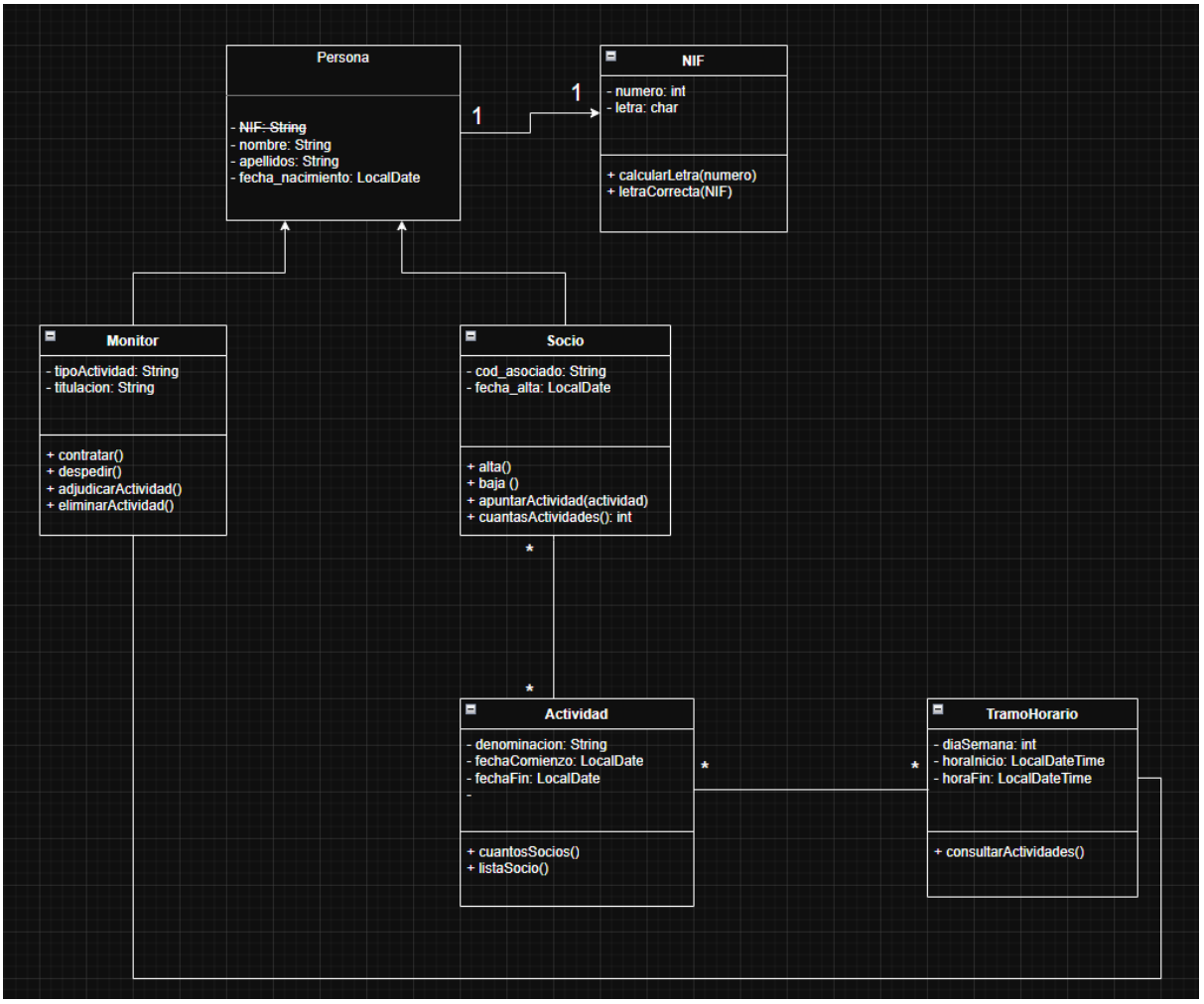
población y código postal). En un garaje puede estar estacionado más de un vehículo.
g. De cada vehículo, se requiere registrar su matrícula, modelo, color y marca. Si es una moto, además la cilindrada, y si es un coche, si es de gasolina, gasoil, eléctrico o híbrido
h. De momento, no detallaremos los 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:**

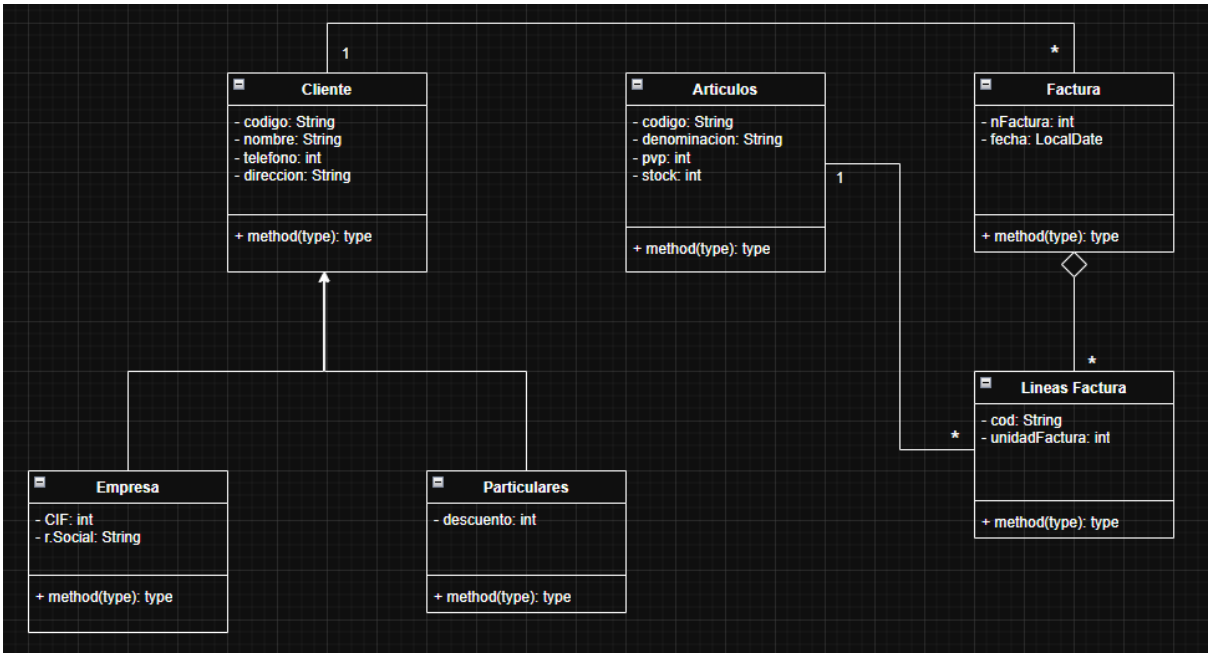
• El sistema deberá tener en cuenta a todas las personas relacionadas con la asociación, ya sean socios o monitores
• De cada persona interesa conocer sus datos básicos: NIF, nombre completo y fecha de nacimiento
• Un socio puede darse de alta o de baja y apuntarse a una o más actividades. Cuando cada 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



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



10. **Realiza utilizando el diagrama de clases de la siguiente especificación: Se desea realizar el análisis de un sistema de gestión informática de una pequeña empresa formada por empleados y departamentos. Para ello se dispone de una base de datos donde están almacenados los datos de unos y otros. Consideraremos las clases y relaciones del ejercicio anterior, pero en vez de modelizar sólo las clases persistentes, lo extenderemos a la aplicación completa.**

• Requisitos funcionales: el sistema a analizar debe permitir el acceso al administrador para mantener la información de la base de datos y para generar informes. o El administrador es un empleado y es el único usuario autorizado a entrar en el sistema y operar. Será el encargado del mantenimiento de los datos de la base de datos. En posteriores refinamientos añadiremos funciones adicionales del administrador. o Las operaciones a realizar por él son las siguientes: o El mantenimiento de datos de empleados incluye altas, bajas, modificaciones y consultas (CRUD) El mantenimiento de datos de departamentos incluye altas, bajas, modificaciones y consultas (CRUD) o Generación de informes

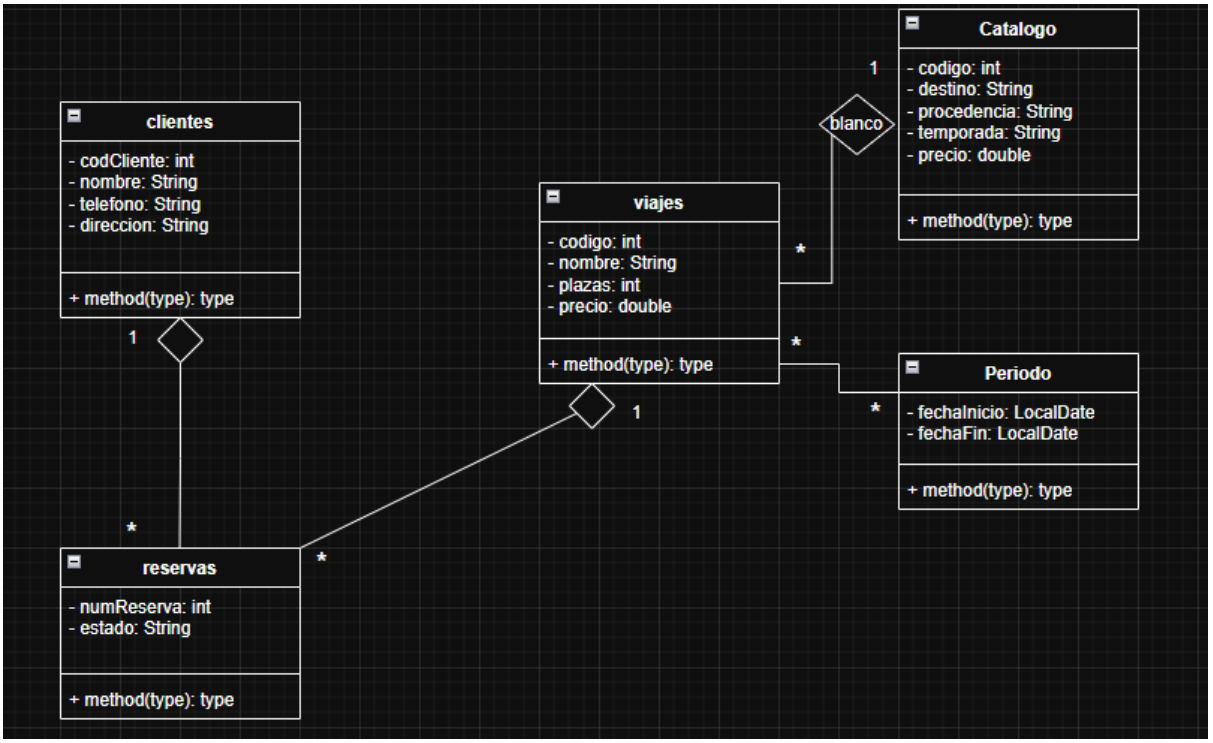
• Identificación de clases de diseño: consideramos los tres tipos distintos de clases de análisis para la definición de las clases del ejercicio: Entity, Control y Boundary. Así pues, se crearán los siguientes tipos de clases: o Clases de tipo Entidad: serán las clases persistentes utilizadas para almacenar la información, asociadas a la BD, consideramos Departamento y Empleado o Clases de Control: su objetivo es controlar las operaciones que se hacen con los datos de la BD (altas, bajas, modificaciones y consultas). Consideramos una clase para las operaciones con datos de empleados y otra para las operaciones con departamentos o Clases Interfaz: clases que se diseñarán para la interactuación del operador con el sistema, es decir, las interfaces gráficas. Se podría contar con una ventana de conexión o inicial, que, de paso tras el logeo a un panel de control de administración, y a su vez al resto de paneles de la aplicación. Estos paneles serán:

- uno para el mantenimiento de empleados
- otro para el mantenimiento de departamentos
- otro para la gestión de informes. Estos tres paneles heredarán ciertos aspectos del panel principal
- Identificación de paquetes: en este último paso agruparemos las clases en paquetes. Se creará un paquete para las clases Entidad, otro para las de Control, y otro para las clases Interfaz. La idea es separar los datos, la lógica y la interfaz

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


clientes
- codCliente: int
- nombre: String
- telefono: String
- direccion: String
+ method(type): type
1
*
reservas
- numReserva: int
- estado: String
+ method(type): type
*
1
viajes
- codigo: int
- nombre: String
- plazas: int
- precio: double
+ method(type): type
*
1
blanco
*
Catalogo
- codigo: int
- destino: String
- procedencia: String
- temporada: String
- precio: double
+ method(type): type
*
Periodo
- fechaInicio: LocalDate
- fechaFin: LocalDate
+ method(type): type