# Trabajo práctico N°9

## APIs REST

Para los siguientes ejercicios se pide implementar una API REST en Python utilizando flask, con la estructura de proyecto vista en clase y ofreciendo las rutas para las operaciones CRUD en cada recurso. Considera que cada clase almacena sus datos en un archivo JSON. Cuando inicia la aplicación debe cargar los datos, y luego mantenerlos actualizados a medida que se reciben peticiones de alta, baja o modificación.
Para ello puede iniciar cada ejercicio con la siguiente secuencia:

I. Crea una carpeta con el nombre del proyecto

II. Crea y activa el entorno virtual ".venv" para el proyecto.

III. Una vez activado el entorno virtual instalarle Flask.

IV. Crea el archivo "*app.py*" y las carpetas "*modelos*\" (para la gestión de datos), "*rutas*\" (para las rutas) y "*datos*\" (para los archivos json) . Dentro de la carpeta "*modelos*\" crea las carpetas "*entidades*\" y "*repositorios*".

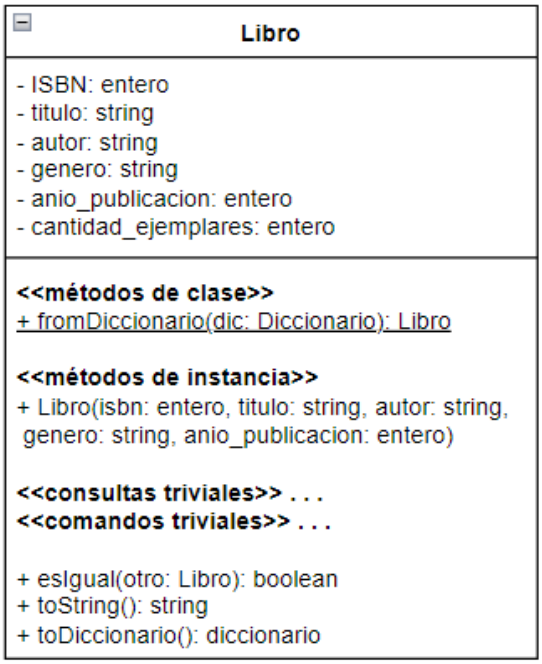
V. En la carpeta "*modelos**entidades*\" crear los archivos que contengan las clases del problema

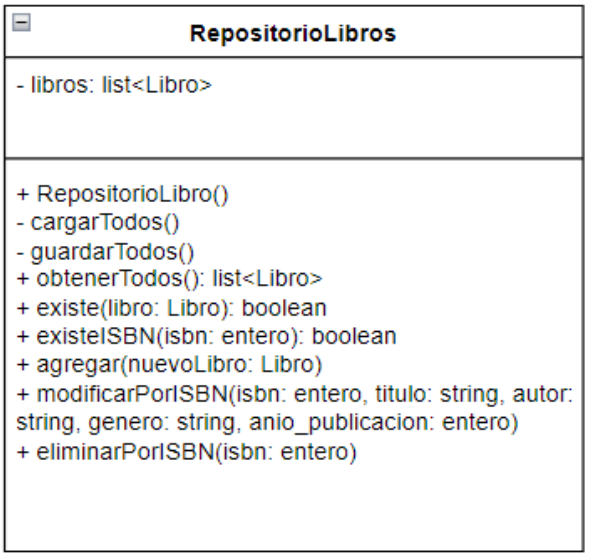
VI. En la carpeta "*modelos**repositorios*\" crear los archivos que gestionen las colecciones de objetos de las clases declaradas en el punto anterior. Son las clases que cargan los datos en memoria y gestionan los datos almacenados en los archivos, y el archivo "*repositorios.py*" que declara una variable para cada clase de repositorio y contiene una función que se encarga de inicializar los repositorios en esas variables cuando inicia la aplicación.

VII. En la carpeta "*rutas*\" crear los archivos que gestionarán las rutas, para ello debe definir los blueprints en cada archivo y luego en "*app.py*" importarlos y registrarlos.

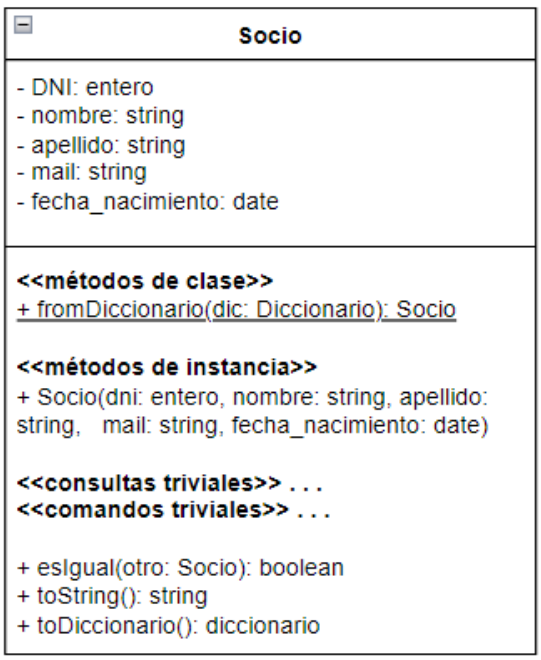
1) En una biblioteca almacenan la siguiente información de los libros: título, autor, género, año de publicación e ISBN. ISBN son las siglas de International Standard Book Number (Número Internacional Normalizado del Libro), y es un número que permite identificar de forma unívoca a un libro en particular.
   Desarrolle una API REST que gestione la información de los libros de la biblioteca.
   El servicio deberá proveer endpoints (rutas) para:
   a) consultar todos los libros registrados (GET)
   b) buscar un libro en particular por ISBN (GET)
   c) agregar un nuevo libro (POST)
   d) modificar los datos de un libro ingresando su ISBN (PUT)
   e) eliminar un libro del sistema dado su ISBN (DELETE)

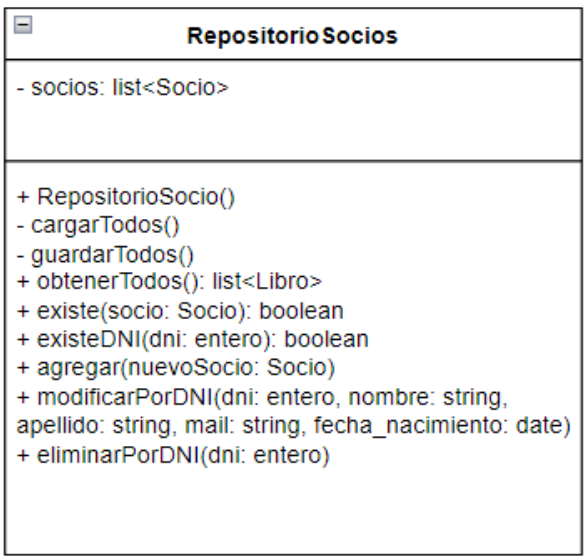
   Al agregar un nuevo libro no se debe duplicar la información existente, por lo tanto, hay que verificar que el libro no exista antes de agregarlo.
   Los datos del sistema están modelados por las siguientes clases:



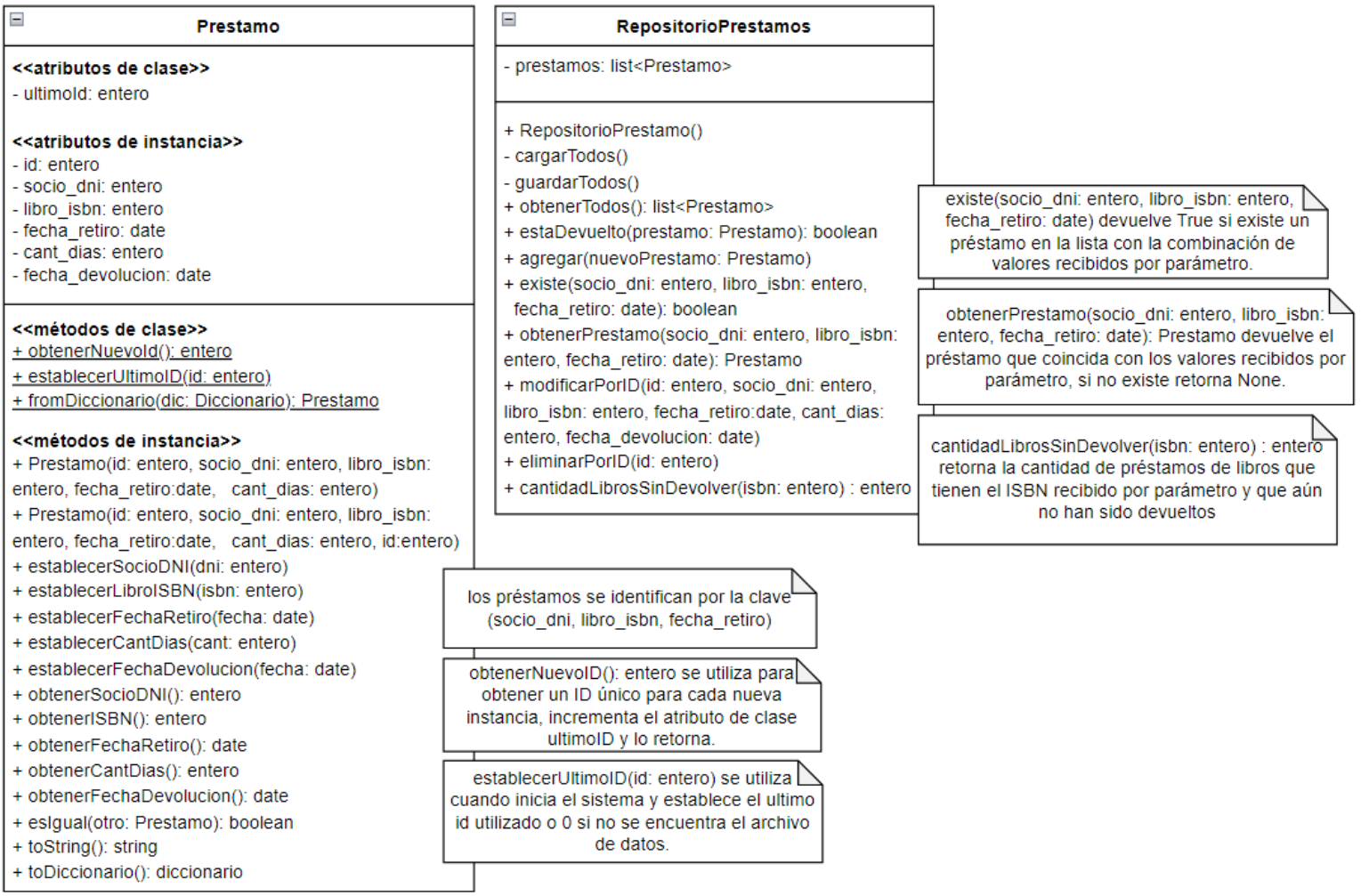




2) La biblioteca desea ahora implementar una API REST que permita gestionar la información de los socios. Para registrarse en la biblioteca se le pide a cada socio la siguiente información: número de DNI, nombre, apellido, un mail de contacto y su fecha de nacimiento.
   Cada vez que se registra un usuario el sistema debe controlar que no exista otro socio registrado con el mismo DNI.
   Desarrolle una API REST que gestione la información de los socios de la biblioteca.
   El servicio deberá proveer endpoints (rutas) para:
   a) consultar todos los socios registrados (GET)
   b) buscar un socio en particular por número de DNI (GET)
   c) agregar un nuevo socio (POST)
   d) modificar los datos de un socio por su DNI(PUT)
   e) eliminar un socio del sistema dado su DNI (DELETE).
      Los datos del sistema están modelados por las siguientes clases:

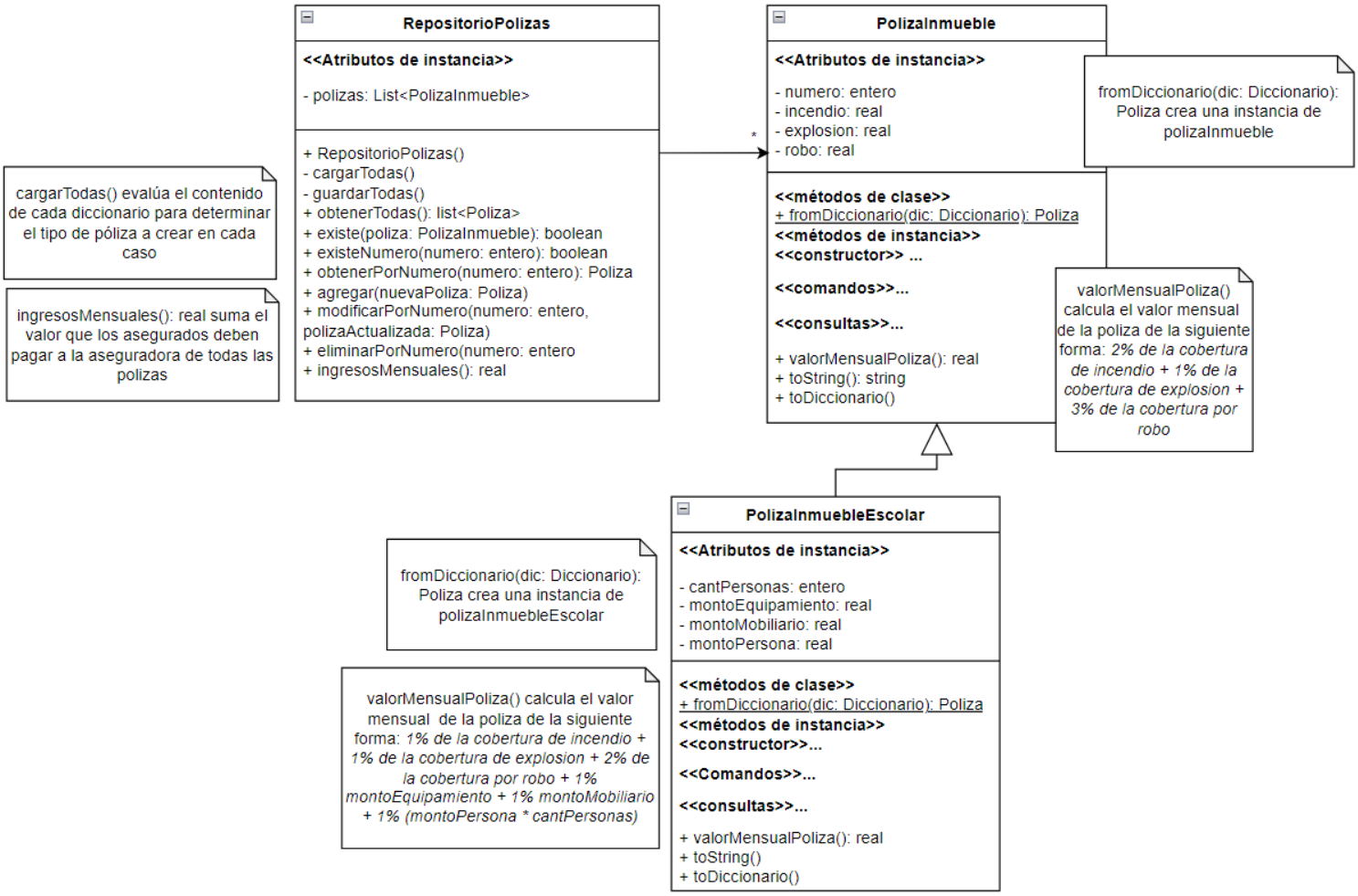


3) Ahora la biblioteca desea combinar la funcionalidad de ambas APIs en una nueva API REST, y agregarle al sistema la posibilidad de gestionar los préstamos de los libros. Para ello gestionará los recursos “Libros” y “Socios” de la misma forma que en los puntos anteriores (con validaciones adicionales en la eliminación), y añade el recurso “Préstamos”.
Cada préstamo tiene un número de identificación, que es único y se genera de forma incremental. El préstamo lo registra el personal de la biblioteca cuando un socio retira un libro de la biblioteca y en ese momento se registra el DNI del socio, el ISBN del libro que retira, y la fecha. Luego cuando el libro es devuelto se le establece la fecha de devolución (mediante la ruta de modificación de un préstamo).
Cada vez que se intenta registrar un préstamo se debe verificar que el libro solicitado posea ejemplares disponibles para ser prestados, y en caso que no hubiera ejemplares disponibles debe mostrar un mensaje indicando “El libro solicitado no tiene ejemplares disponibles para prestar”.
Observación: como los préstamos contienen datos que referencian a libros y socios se debe tener precaución con la eliminación de las instancias de libros y socios: antes de eliminar una instancia se debe verificar que no esté registrada en un préstamo. Es decir: no se debe poder eliminar un socio (o un libro) que está registrado en un préstamo.
El préstamo está modelado por las siguientes clases:



4) Una empresa de seguros mantiene información referida a las pólizas en una colección modelada en el siguiente diagrama de clases. Implemente una API REST que permita a la aseguradora realizar las operaciones CRUD sobre la colección de pólizas.



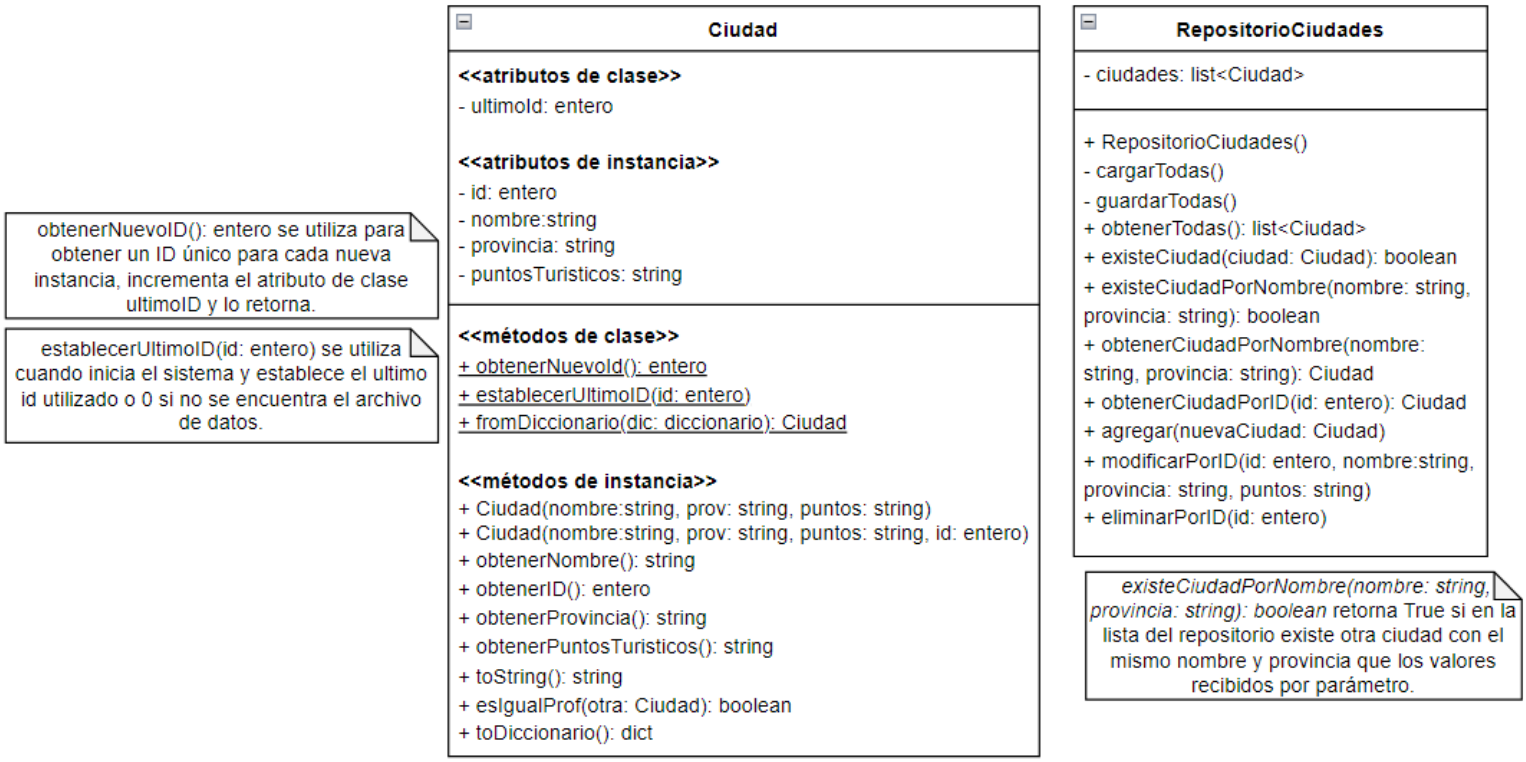

5) Una agencia de viajes especializada en destinos turísticos de Argentina necesita un sistema de gestión de viajes para organizar y ofrecer paquetes turísticos grupales a sus clientes. La agencia requiere almacenar, actualizar y visualizar la información de cada destino, el tipo de viaje, el alojamiento y el medio de transporte para cada paquete turístico.
Cada paquete de viaje grupal tiene una ciudad, fechas de ida y vuelta, una descripción del viaje, un tipo de viaje (entre las opciones: turismo, educativo, egresados o aventura), un medio de transporte (entre las opciones avión, colectivo, tren, crucero), el hotel donde se alojarán y el precio del paquete. También se debe mantener información sobre el cupo máximo y el cupo actual de personas en cada paquete grupal. Cada paquete tiene un número que lo identifica, pero además cada paquete es único considerando la clave (Ciudad, Hotel, FechaSalida, FechaVuelta, TipoViaje, Transporte).
La empresa almacena información de las ciudades que ofrece como destino. Cada ciudad tiene un nombre, la provincia donde se ubica, y una descripción de los puntos turísticos a donde se puede ir a pasear. Cada ciudad tiene un id, y además se puede identificar mediante el nombre y la provincia, ya que puede haber ciudades con el mismo nombre en distintas provincias.

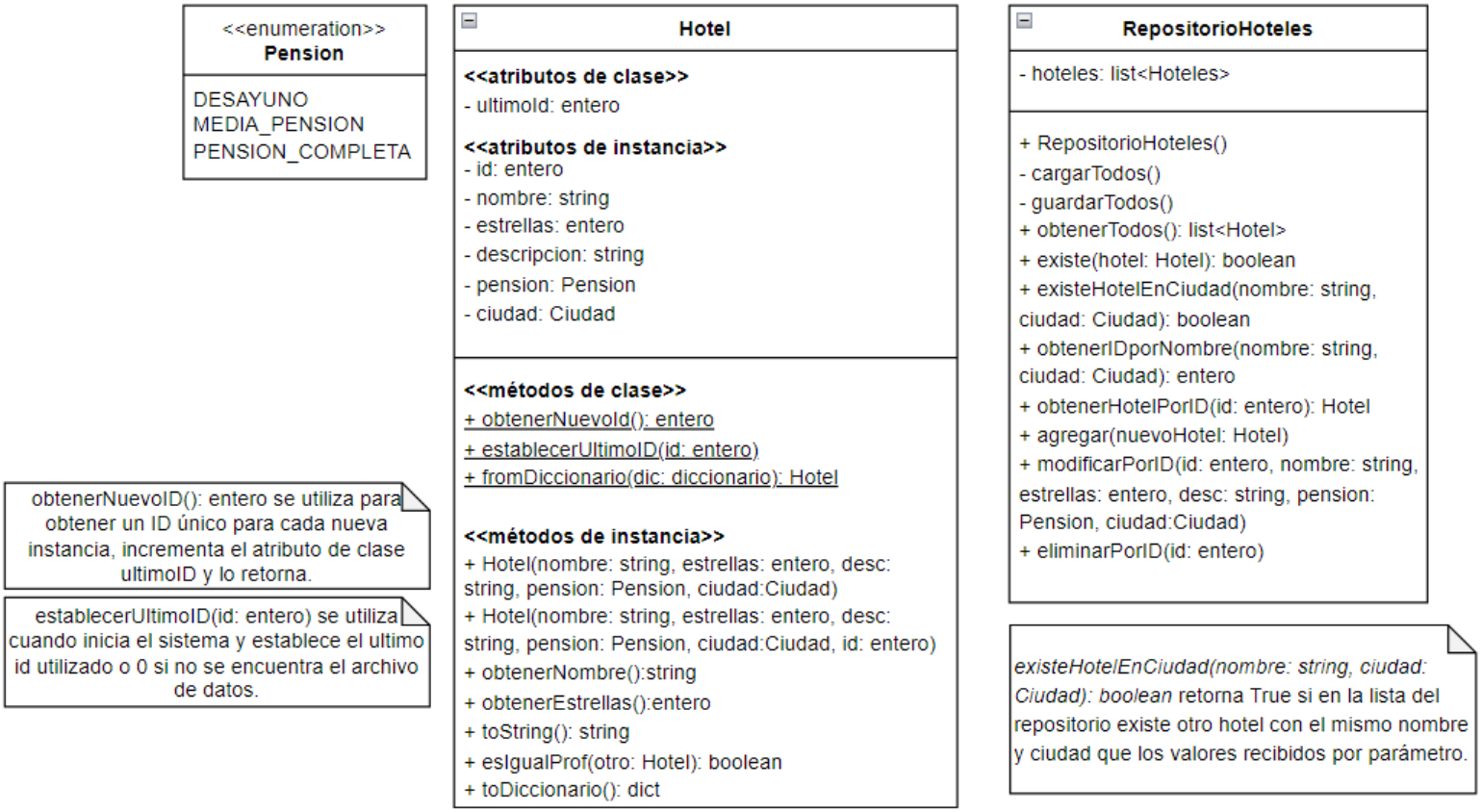
Cada paquete turístico incluye un alojamiento en la ciudad de destino. De cada alojamiento se tiene el nombre del hotel, la ciudad donde se ubica, una breve descripción, la categoría de estrellas del alojamiento y el tipo de pensión (entre las opciones: desayuno, media pensión, pensión completa). Los hoteles tienen un número de identificación en el sistema, y también se identifican mediante el nombre y la ciudad donde se encuentran ubicados (no deben existir dos hoteles con el mismo nombre en la misma ciudad).
Los paquetes turísticos se ofrecen con un tipo de transporte que se utilizará para ir y volver del destino turístico (por ejemplo, avión, colectivo, tren, etc.).
Desarrolle una API REST que gestione la información de las ciudades, hoteles y paquetes de viaje que ofrece la empresa. Implemente los endpoints (rutas) para ofrecer las operaciones CRUD en cada recurso.

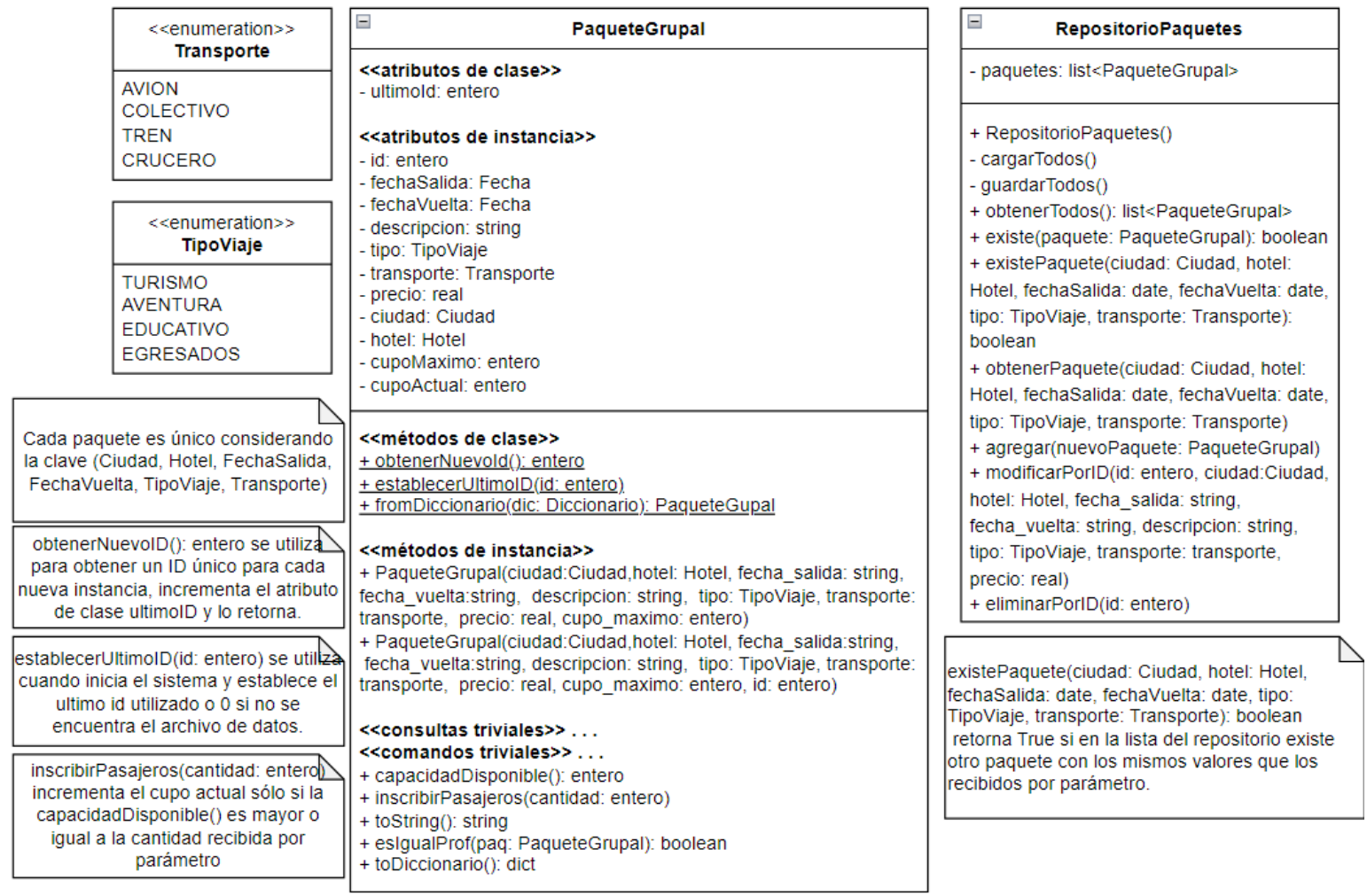
*Observaciones*:

a) Tanto en las creaciones como en las modificaciones de objetos existentes se deberá verificar que no se esté duplicando información mediante las acciones solicitadas. Por ejemplo, si se recibe una solicitud para crear un recurso que ya existe, o si recibo una solicitud de actualización con valores que corresponden a otro objeto que ya existe, el sistema debe retornar un mensaje indicando la situación de error.
b) En el caso de las ciudades y hoteles, como son clases que forman parte de los atributos de otras clases se debe tener precaución con la eliminación de las instancias: antes de eliminar una instancia se debe verificar que no esté siendo utilizada por otro objeto. Por ejemplo: no se debe poder eliminar una ciudad que está siendo utilizada en un paquete turístico o en un hotel, y tampoco se puede eliminar un hotel que esté asignado en algún paquete turístico.

Puede adaptar las clases del punto 3 del trabajo práctico 8 según el siguiente diagrama:




<<enumeration>>
Pension
DESAYUNO
MEDIA_PENSION
PENSION_COMPLETA
Hotel
<<atributos de clase>>
- ultimoId: entero
<<atributos de instancia>>
- id: entero
- nombre: string
- estrellas: entero
- descripcion: string
- pension: Pension
- ciudad: Ciudad
<<métodos de clase>>
+ obtenerNuevoId(): entero
+ establecerUltimoID(id: entero)
+ fromDiccionario(dic: diccionario): Hotel
<<métodos de instancia>>
+ Hotel(nombre: string, estrellas: entero, desc: string, pension: Pension, ciudad:Ciudad)
+ Hotel(nombre: string, estrellas: entero, desc: string, pension: Pension, ciudad:Ciudad, id: entero)
+ obtenerNombre():string
+ obtenerEstrellas():entero
+ toString(): string
+ esIgualProf(otro: Hotel): boolean
+ toDiccionario(): dict
RepositorioHoteles
- hoteles: list<Hoteles>
+ RepositorioHoteles()
- cargarTodos()
- guardarTodos()
+ obtenerTodos(): list<Hotel>
+ existe(hotel: Hotel): boolean
+ existeHotelEnCiudad(nombre: string, ciudad: Ciudad): boolean
+ obtenerIDporNombre(nombre: string, ciudad: Ciudad): entero
+ obtenerHotelPorID(id: entero): Hotel
+ agregar(nuevoHotel: Hotel)
+ modificarPorID(id: entero, nombre: string, estrellas: entero, desc: string, pension: Pension, ciudad:Ciudad)
+ eliminarPorID(id: entero)
obtenerNuevoID(): entero se utiliza para obtener un ID único para cada nueva instancia, incrementa el atributo de clase ultimoID y lo retorna.
establecerUltimoID(id: entero) se utiliza cuando inicia el sistema y establece el ultimo id utilizado o 0 si no se encuentra el archivo de datos.
existeHotelEnCiudad(nombre: string, ciudad: Ciudad): boolean retorna True si en la lista del repositorio existe otro hotel con el mismo nombre y ciudad que los valores recibidos por parámetro.


<<enumeration>>
Transporte
AVION
COLECTIVO
TREN
CRUCERO
<<enumeration>>
TipoViaje
TURISMO
AVENTURA
EDUCATIVO
EGRESADOS
PaqueteGrupal
<<atributos de clase>>
- ultimoId: entero
<<atributos de instancia>>
- id: entero
- fechaSalida: Fecha
- fechaVuelta: Fecha
- descripcion: string
- tipo: TipoViaje
- transporte: Transporte
- precio: real
- ciudad: Ciudad
- hotel: Hotel
- cupoMaximo: entero
- cupoActual: entero
<<métodos de clase>>
+ obtenerNuevoId(): entero
+ establecerUltimoID(id: entero)
+ fromDiccionario(dic: Diccionario): PaqueteGupal
<<métodos de instancia>>
+ PaqueteGrupal(ciudad:Ciudad,hotel: Hotel, fecha_salida: string, fecha_vuelta:string, descripcion: string, tipo: TipoViaje, transporte: transporte, precio: real, cupo_maximo: entero)
+ PaqueteGrupal(ciudad:Ciudad,hotel: Hotel, fecha_salida:string, fecha_vuelta:string, descripcion: string, tipo: TipoViaje, transporte: transporte, precio: real, cupo_maximo: entero, id: entero)
<<consultas triviales>> . . .
<<comandos triviales>> . . .
+ capacidadDisponible(): entero
+ inscribirPasajeros(cantidad: entero)
+ toString(): string
+ esIgualProf(paq: PaqueteGrupal): boolean
+ toDiccionario(): dict
RepositorioPaquetes
- paquetes: list<PaqueteGrupal>
+ RepositorioPaquetes()
- cargarTodos()
- guardarTodos()
+ obtenerTodos(): list<PaqueteGrupal>
+ existe(paquete: PaqueteGrupal): boolean
+ existePaquete(ciudad: Ciudad, hotel: Hotel, fechaSalida: date, fechaVuelta: date, tipo: TipoViaje, transporte: Transporte): boolean
+ obtenerPaquete(ciudad: Ciudad, hotel: Hotel, fechaSalida: date, fechaVuelta: date, tipo: TipoViaje, transporte: Transporte)
+ agregar(nuevoPaquete: PaqueteGrupal)
+ modificarPorID(id: entero, ciudad:Ciudad, hotel: Hotel, fecha_salida: string, fecha_vuelta: string, descripcion: string, tipo: TipoViaje, transporte: transporte, precio: real)
+ eliminarPorID(id: entero)
Cada paquete es único considerando la clave (Ciudad, Hotel, FechaSalida, FechaVuelta, TipoViaje, Transporte)
obtenerNuevoID(): entero se utiliza para obtener un ID único para cada nueva instancia, incrementa el atributo de clase ultimoID y lo retorna.
establecerUltimoID(id: entero) se utiliza cuando inicia el sistema y establece el ultimo id utilizado o 0 si no se encuentra el archivo de datos.
inscribirPasajeros(cantidad: entero) incrementa el cupo actual sólo si la capacidadDisponible() es mayor o igual a la cantidad recibida por parámetro
existePaquete(ciudad: Ciudad, hotel: Hotel, fechaSalida: date, fechaVuelta: date, tipo: TipoViaje, transporte: Transporte): boolean retorna True si en la lista del repositorio existe otro paquete con los mismos valores que los recibidos por parámetro.