1 Tabla de contenido

2 Colaboraciones principales

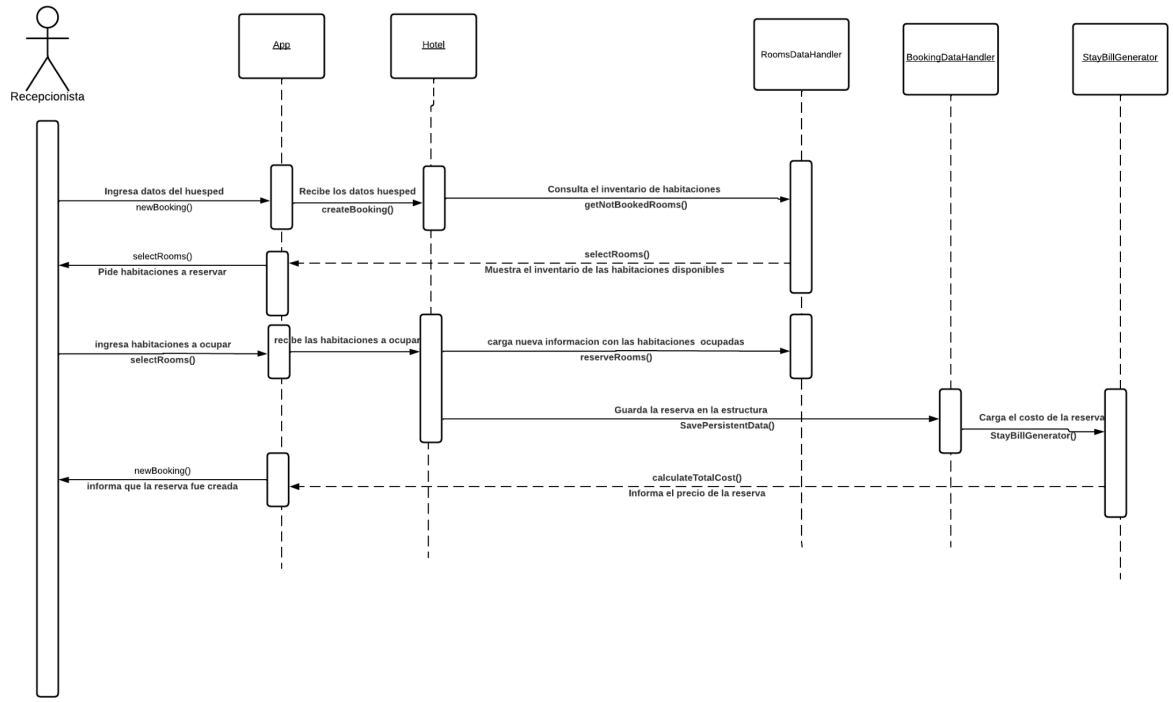
Teniendo en cuenta el diagrama de secuencia, se caracterizan las siguientes colaboraciones principales de la aplicación:

1. Reservar habitaciones
2. Registrar nuevos huéspedes
3. Consumir y pagar servicios
4. Hacer check-out

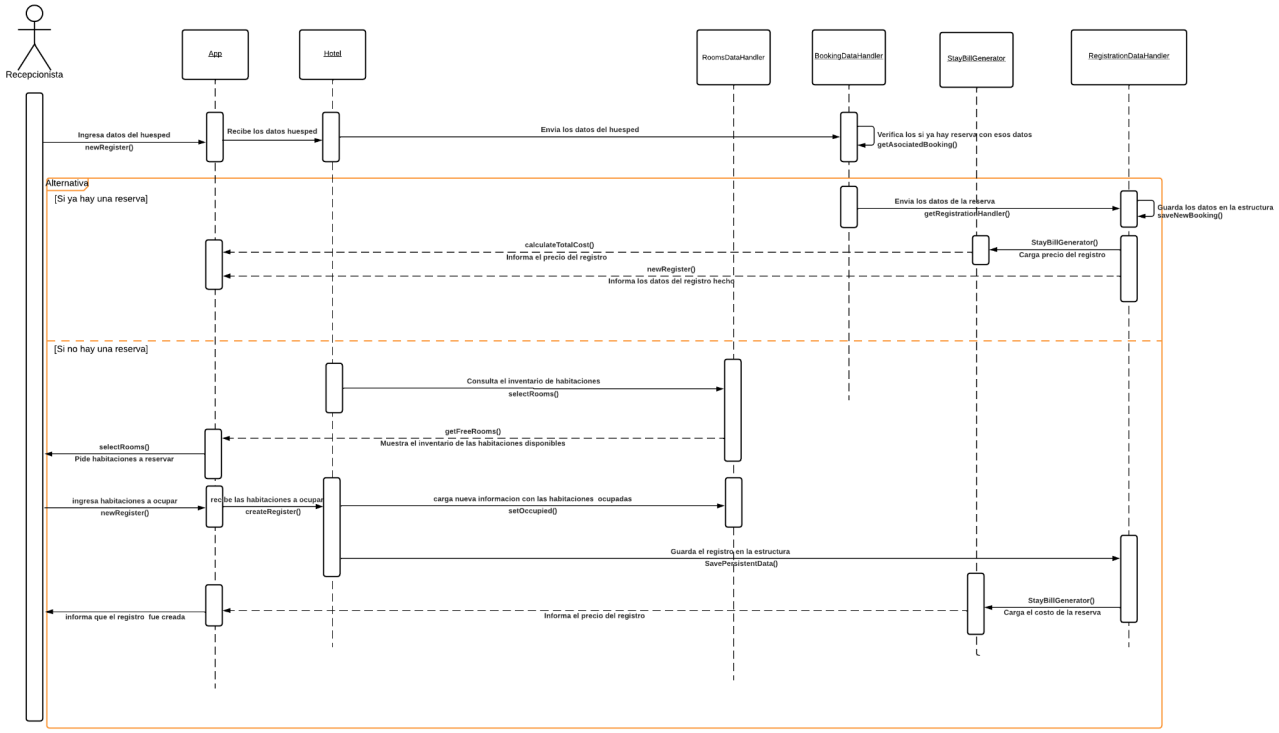
3 Diagramas de secuencia

Según las colaboraciones principales anteriores, se procede a construir cada diagrama de secuencia respectivamente.

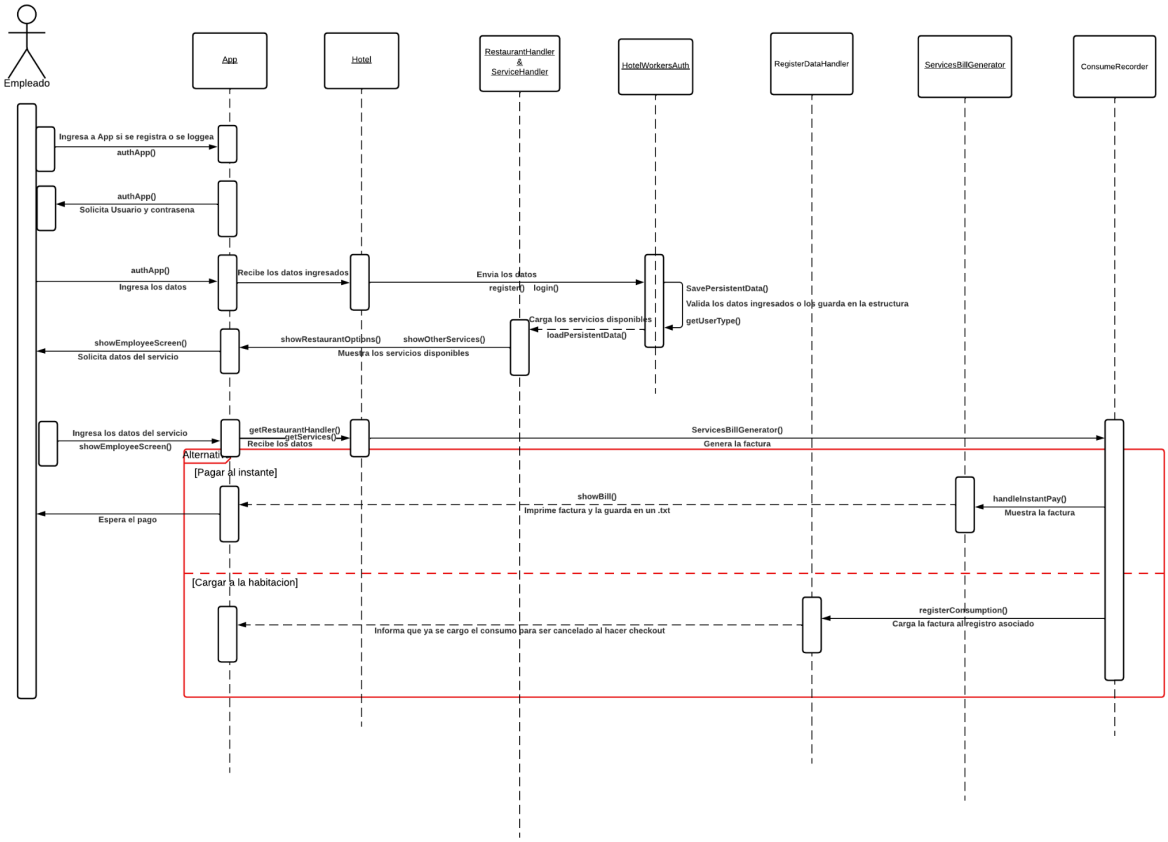
3.1 Reservar habitaciones:



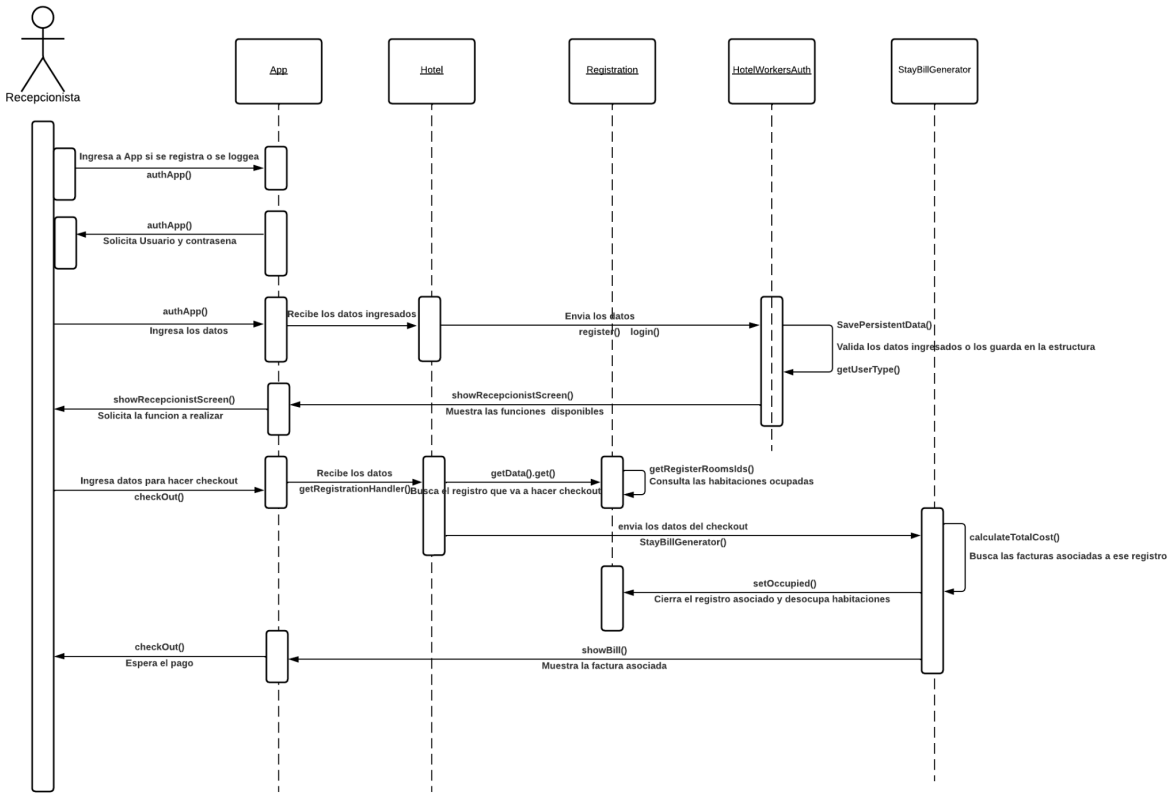
3.2 Registrar nuevos huéspedes



3.3 Consumir servicios y Pagar servicios



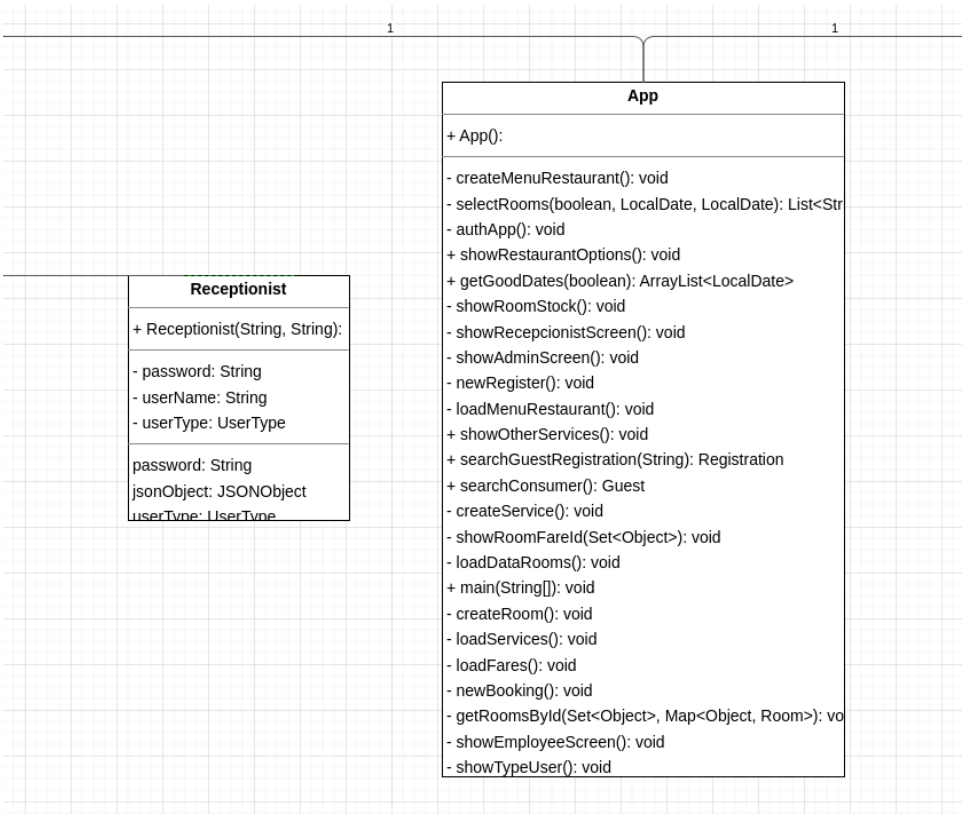
3.4 Hacer chek-out



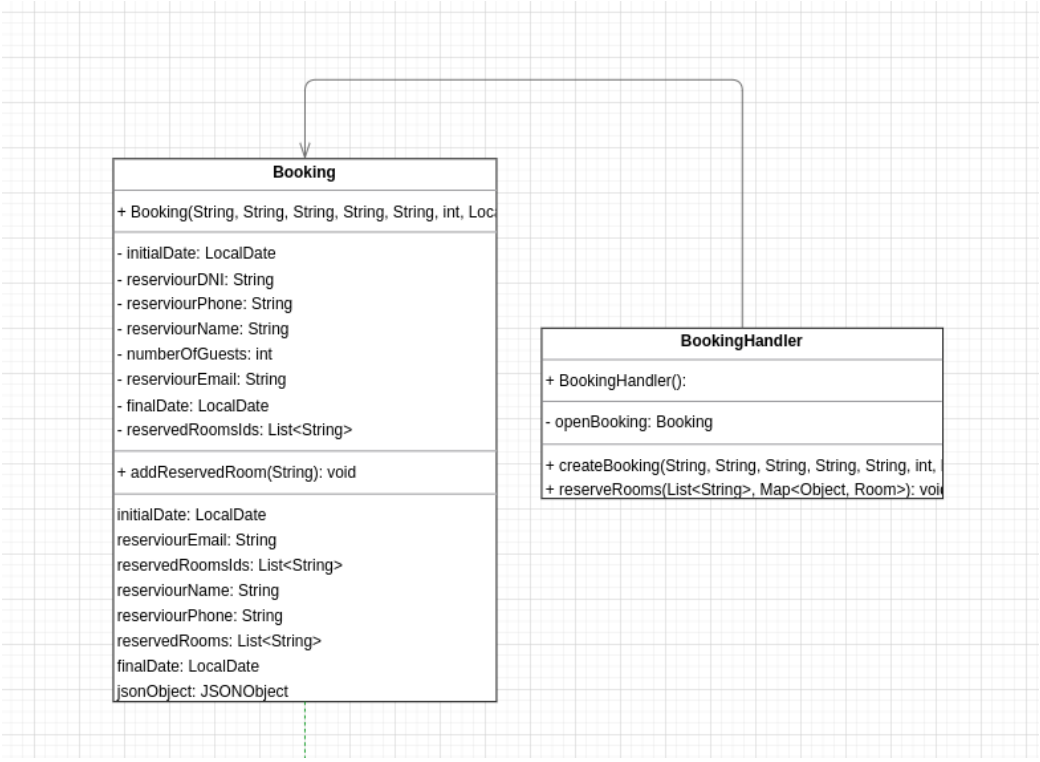
Asignación de responsabilidades

5. Reservar habitaciones

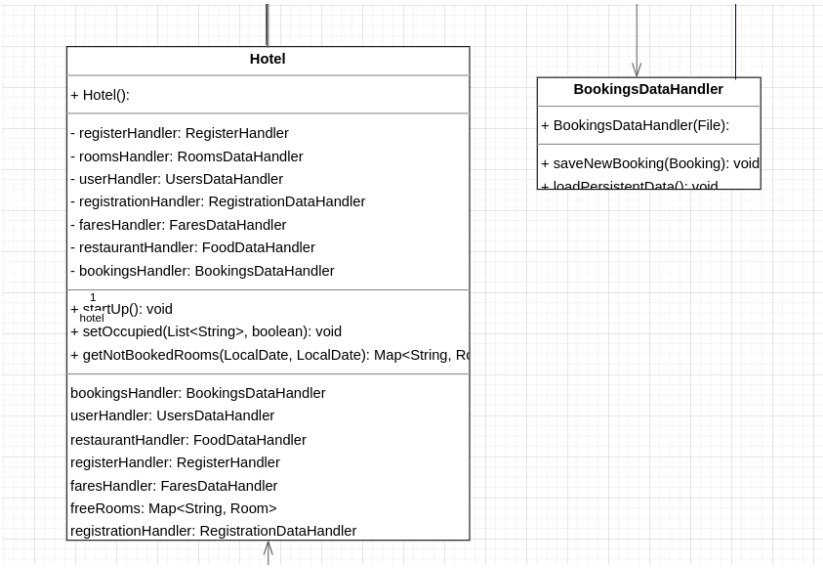
   Para reservar una habitación se debe seguir la siguiente secuencia de colaboraciones El usuario de tipo `Receptionist` hace uso de los métodos de la clase `App`, concretamente del método `newBooking`.



Desde el método `newBooking` se instancia a la clase `BookingHandler` quien es la encargada de realizar las operaciones sobre el nuevo objeto `Booking` (Reserva) que estamos creando. Desde `bookingHandler` podremos ejecutar las operaciones de creación de una nueva reserva y de asignar y reservar las habitaciones seleccionadas

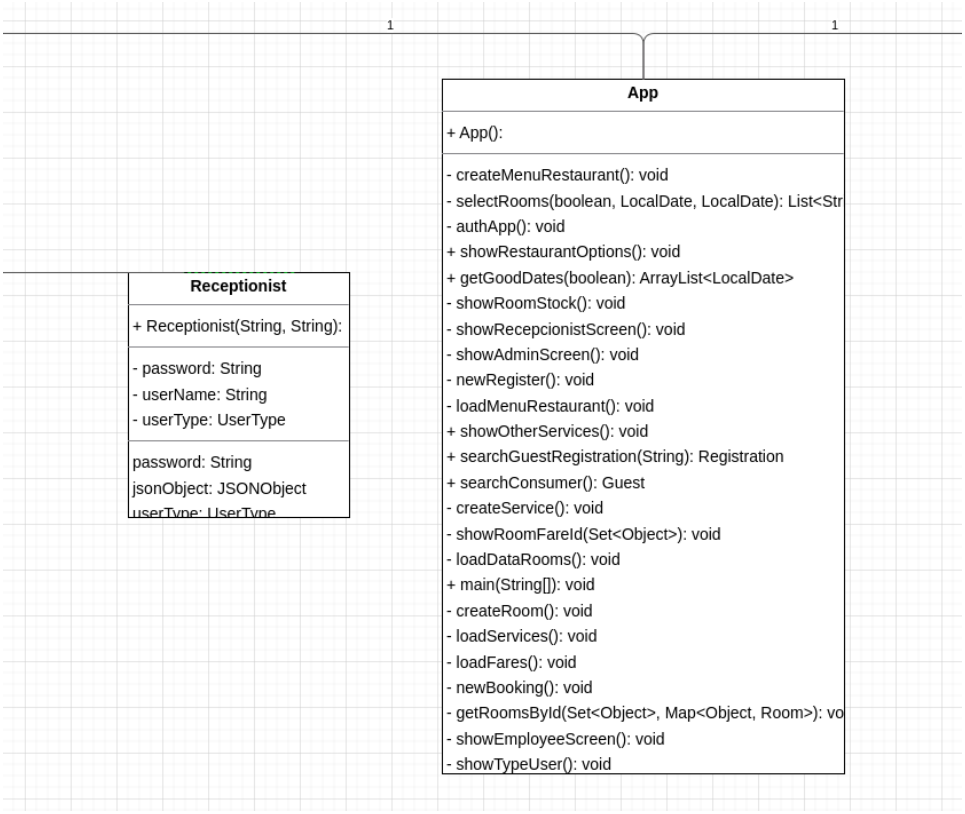


Finalmente, para guardar la información en nuestra persistencia hacemos uso de la clase `Hotel`, quien contiene la única instancia de `BookingDataHandler`, encargada de cargar y guardar sobre el archivo JSON en donde se encuentra alojada la información persistente.

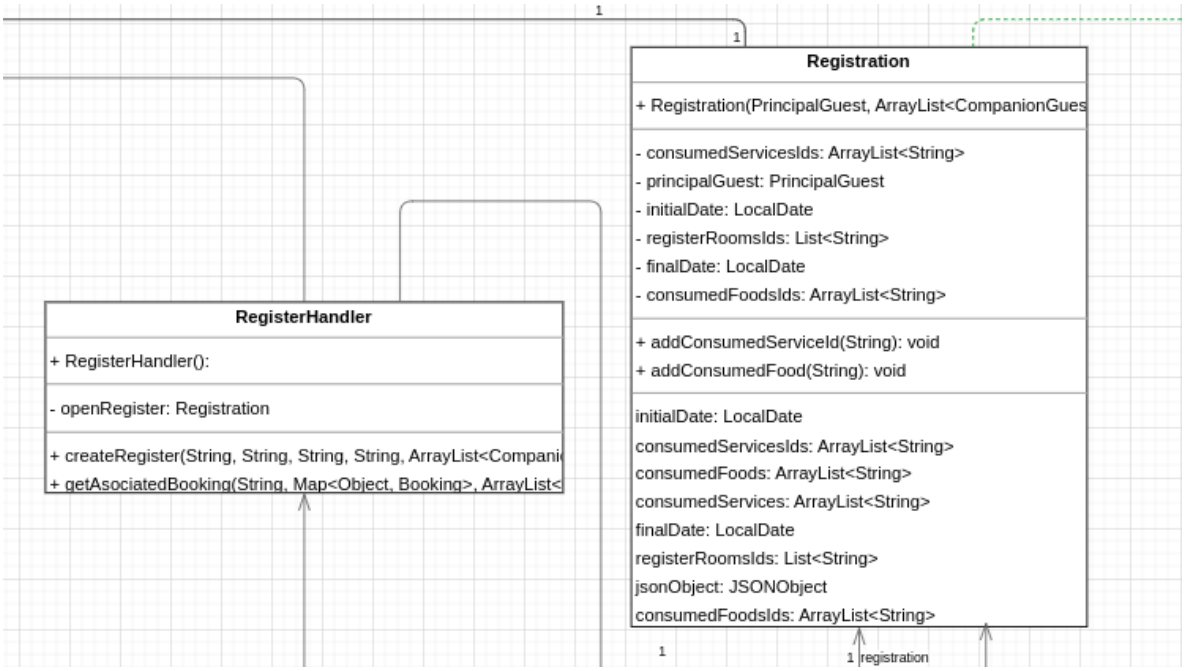


6. Registrar nuevos huéspedes

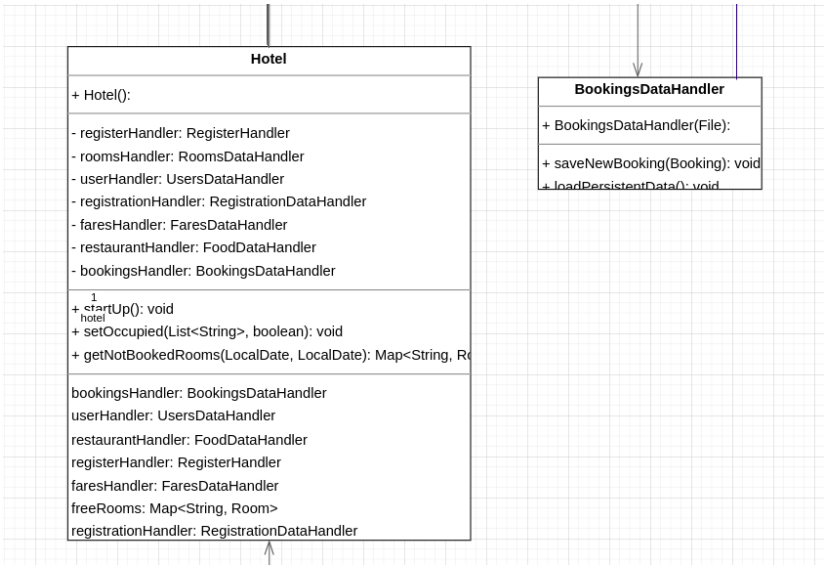
Igual que en el caso anterior, el usuario `Receptionist` hace uso de `App` para generar un nuevo registro, en este caso, a partir del método `newRegister`



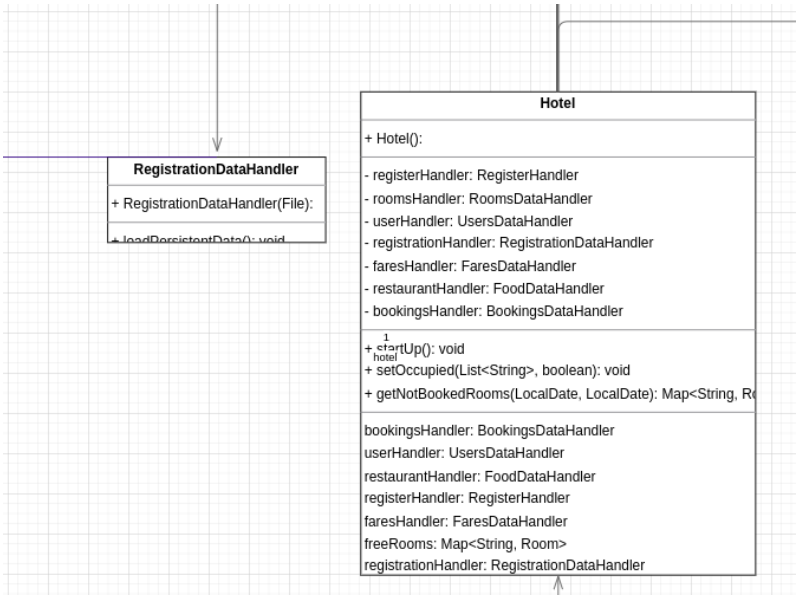
Para realizar la acción de registro, también se cuenta con una clase auxiliar (`RegisterHandler`) que realizará las operaciones sobre `Registration`.



A la hora de crear un registro existe una particularidad y es que, si ya se realizó una reserva anterior, el registro se generará a partir de la información de la reserva, por lo que también se genera una interacción con `Hotel`, quien contiene a `BookingsDataHandler` y nos puede proveer de la información de la reserva

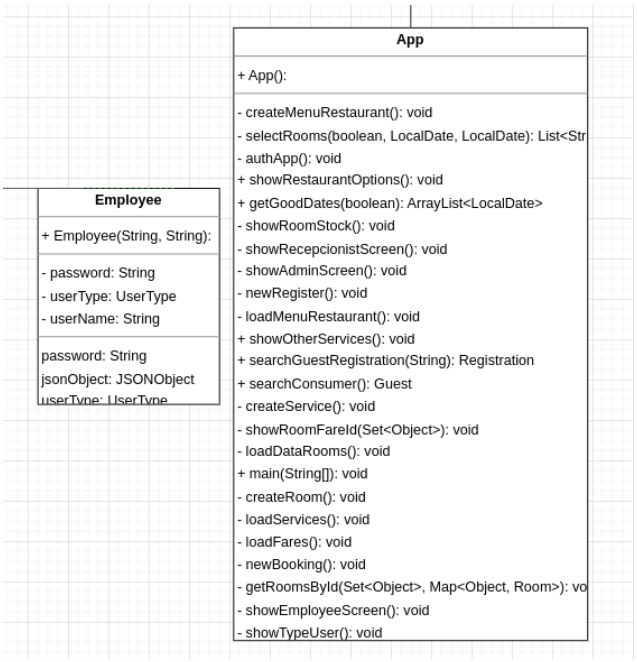


Finalmente, para hacer persistente la información hacemos uso de `Hotel`, quien tendrá a `RegistrationDataHandler`, clase encargada de guardar y cargar la información de los registros de huéspedes

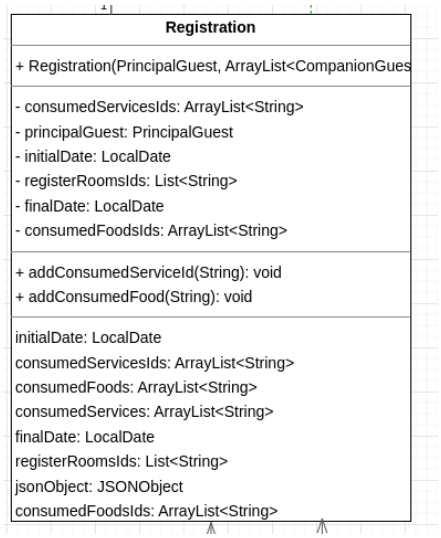


7. Consumir servicios

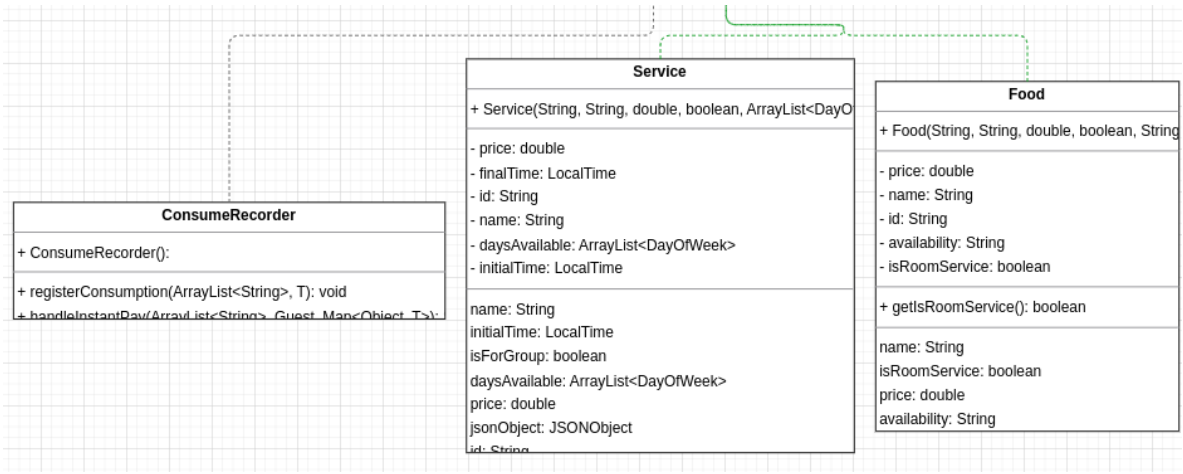
   EL encargado de registrar y cobrar el consumo instantáneo es el empleado del hotel que haya proveído el servicio, por lo que el usuario de tipo `Employee` hará uso del método `showRestaurantOptions` o `showOtherServices` sea el caso.



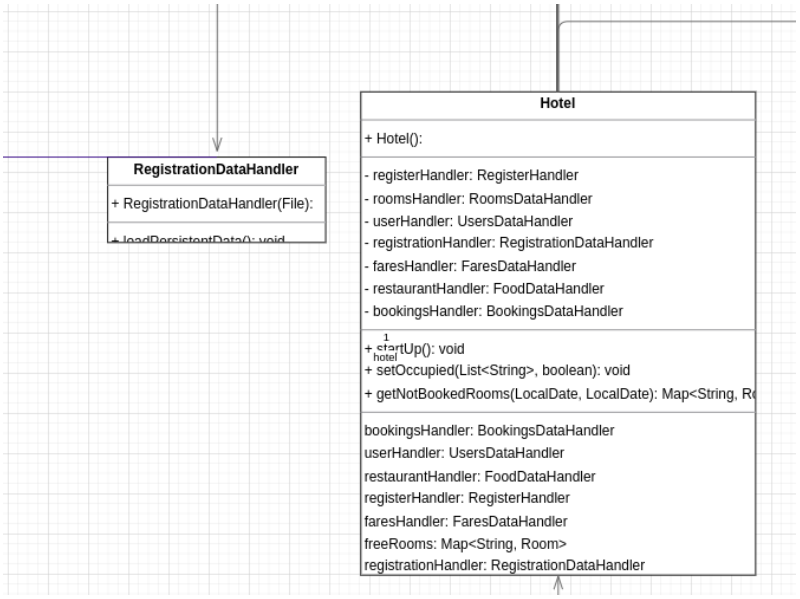
Para hacer el consumo de un servicio, deberá existir un huesped asociado, la información de dicho huesped se encontrará en el objeto `Registration` correspondiente, por lo que existe una colaboración con esta clase



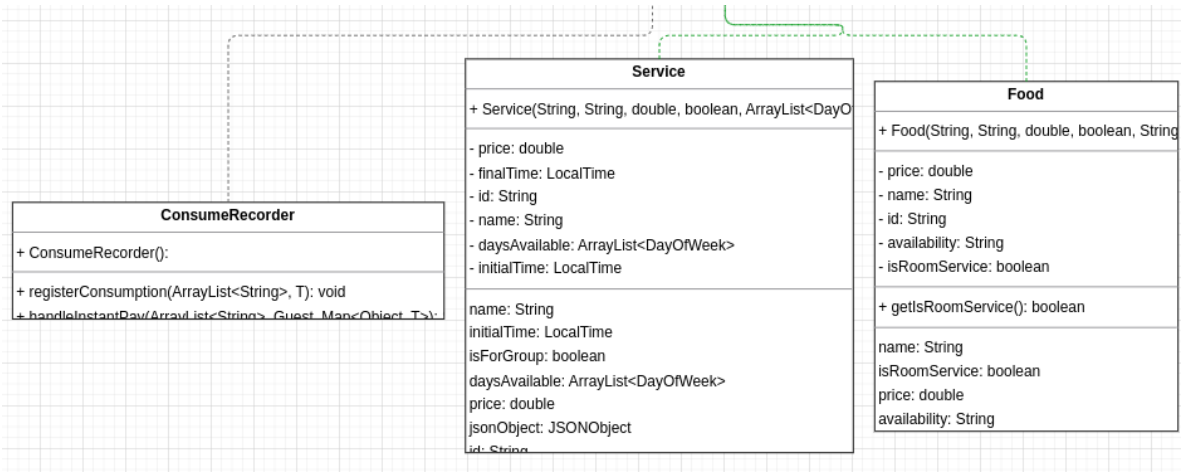
El grueso de la lógica del consumo de servicios se encuentra en `ConsumeRecorder`, en donde se van a registrar estos consumos, sean de servicios (`Service`) o de alimentos y bebidas (`Food`)



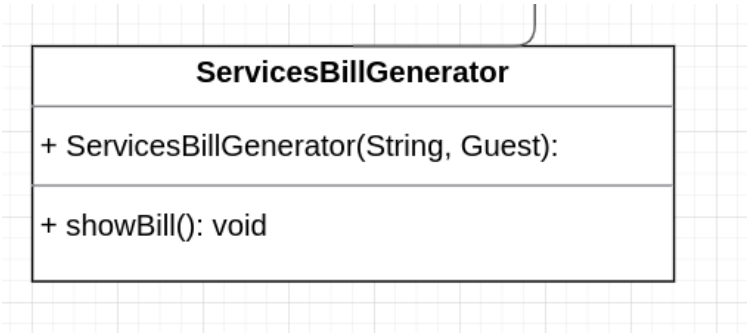
Finalmente, la información de lo que un huésped consumió le pertenece al registro de este huésped, por lo que para persistir la información de los consumos se harán entradas de estos consumos sobre `Registration` y por ende se guardará haciendo uso de `RegistrationDataHandler`.



8. Pagar servicios
   Los servicios ofrecidos por el Hotel se pueden pagar de dos formas, al momento de consumir el servicio/alimento, o al final en el checkout. Cuando el pago se realiza instantáneamente, se inicia desde consumeRecorder, en donde se encuentran los servicios que ya fueron registrados

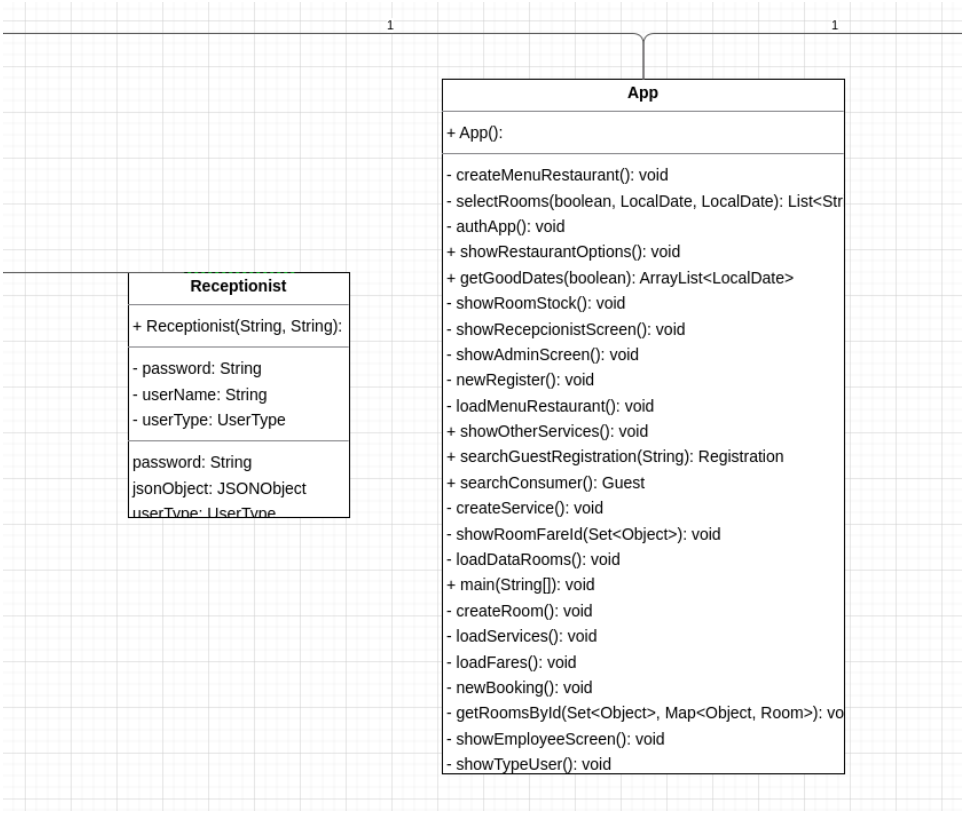


Finalmente para ejecutar el pago y mostrar la factura, se hace uso del servicio de `ServicesBillGenerator`, quien estará proveida de la información tanto del Huesped como del consumo y manejará el pago

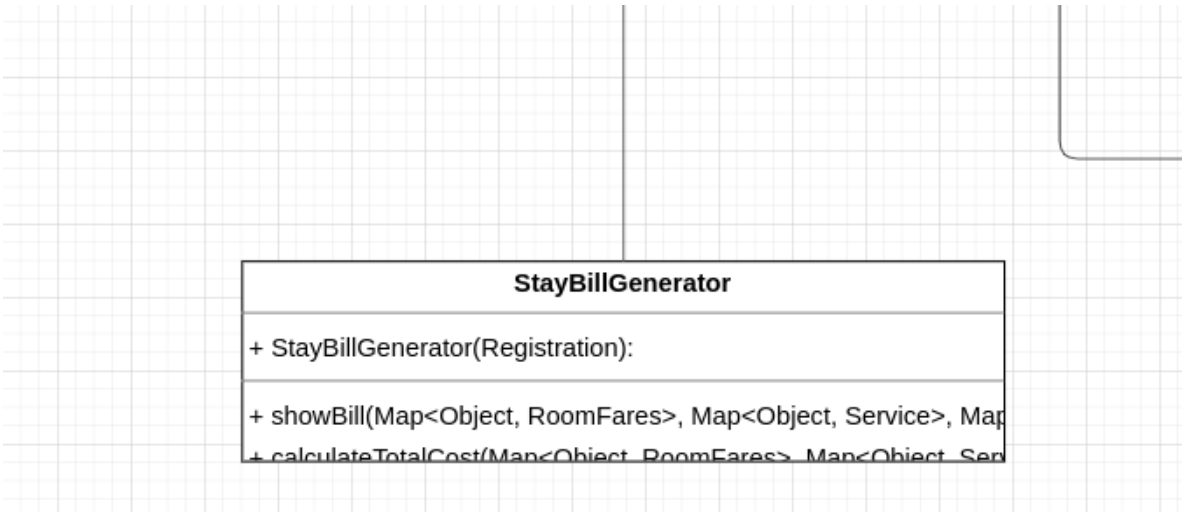


9. Hacer Check-out
   El encargado de hacer check-out es el recepcionista, quien ejecutará el método

`checkOut`, que realizará la acción mencionada en `App`.



Desde este punto se encarga de, en primer lugar manejar el costo y la factura del alojamiento y de los servicios consumidos de los cuales el pago se postergó hasta el final. `StayBillGenerator` se encargará de calcular el costo y mostrar las facturas



Finalmente para cambiar el estado de las habitaciones, de ocupada a no ocupada se hará uso del método `setOccupied` en `Hotel`.

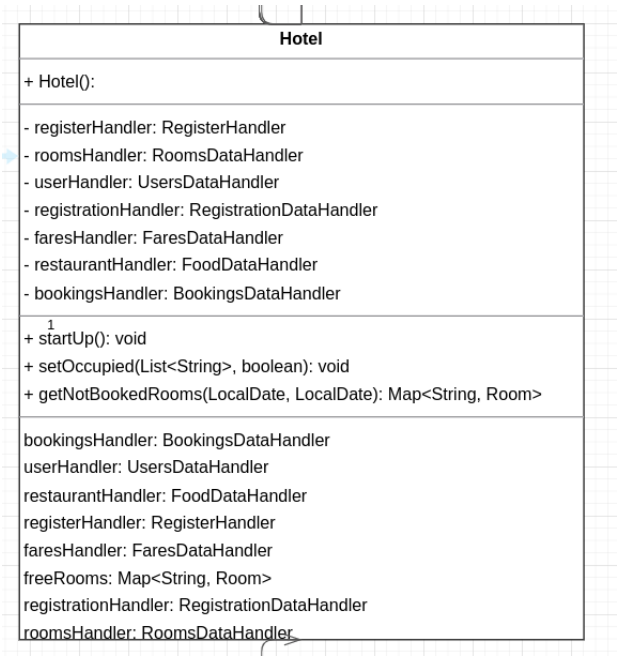
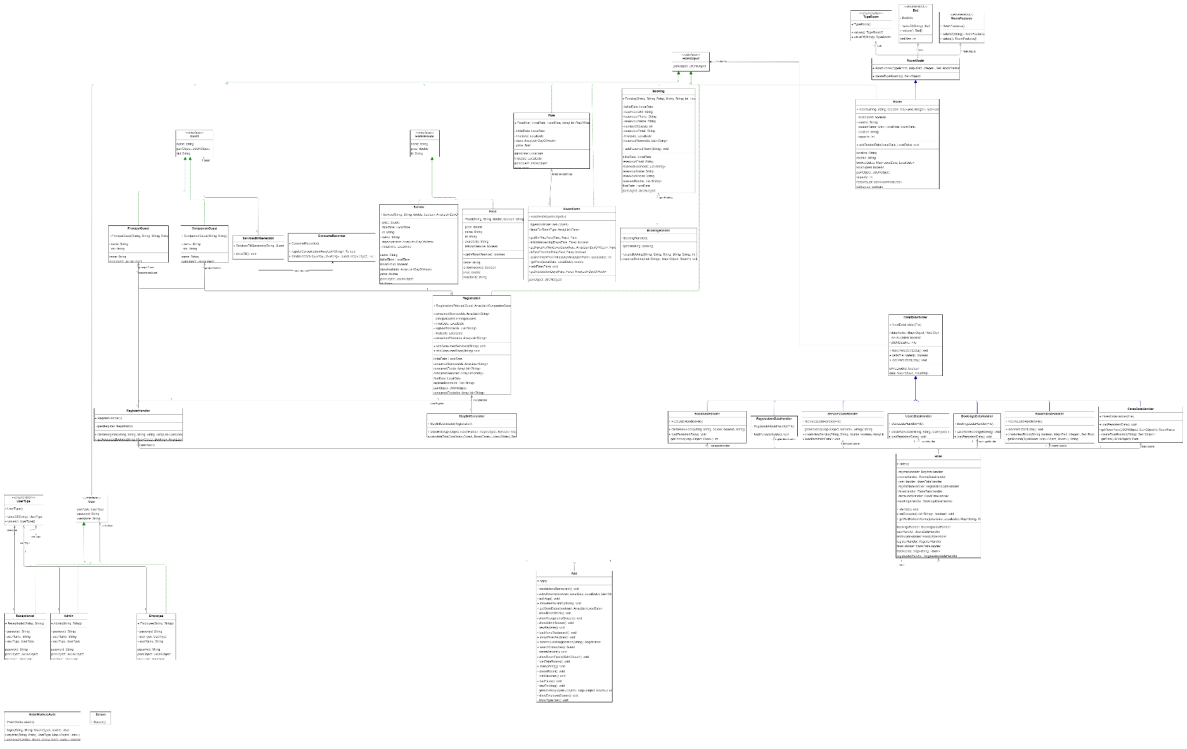


Diagrama de clases



Glosario

1. **HotelObject**: Interfaz usada para modelar diversas clases. Tiene un único método llamado getJsonObject. Este método retorna el JSONObject de cada objeto instanciado con el fin de poder almacenar los datos y que puedan persistir.
2. **Guest:** Interfaz que extiende de HotelObject. Esta interfaz tiene como objetivo modelar los posibles tipos de huésped. Tiene 3 métodos declarados: getName, getDni y getJsonObject.
3. **PrincipalGuest:** Clase que implementa Guest. Con esta clase se modelan los huéspedes principales, es decir, aquellos que son responsables del grupo. Sus datos son tomados para realizar el registro a nombre de dicho huésped. Cuenta con un nombre, dni y número de teléfono.
4. **CompanionGuest:** Clase que implementa Guest. Esta clase modela los huéspedes que no son principales. Cuentan con un nombre y dni.
5. **StayBillGenerator**: Clase con la que se genera la factura al momento de realizar check out, cuando se genere esta factura se deben añadir las comidas y los servicios que no fueron pagados instantáneamente, es decir, que se cargaron a la habitación. Tiene un objeto de registro, una lista de los ids de lo servicios consumidos y una lista de los ids de las comidas consumidas
6. **ServicesBillGenerator:** Clase con la que se genera la factura del pago de un servicio al momento de pagarlo instantáneamente. Tiene como atributos el texto de una factura y el huésped que está realizando el pago.
7. **ConsumeRecorder:** Por medio de esta clase se modelan los registros de los consumos de un huésped. Tiene dos métodos, uno para añadir el servicio a la lista de servicios que tiene un registro para que al hacer check out se paguen, el otro método es usado para realizar el pago instantáneo de un servicio.
8. **HotelService:** Interfaz que extiende de HotelObject. Tiene 3 métodos, getName, getPrice y getId. Esta clase es usada para generar el pago de facturas de forma efectiva.
9. **Service**: Clase que implementa HotelService. Por medio de esta se modelan los posibles servicios que puede tener el restaurante. Cuenta con un id, un nombre, un precio, si es para grupo o no, los días de la semana en los que está disponible, su hora de inicio y su hora final
10. **Food**: Clase que implementa HotelService. Por medio de esta se modelan todas las comidas que puede tener el restaurante. Cuenta con un id, un nombre, un precio, si es disponible para llevar a la habitación y la disponibilidad de la comida.
11. **RoomFares**: Clase que implementa HotelObject. Esta clase tiene como objetivo realizar todas las verificaciones de las tarifas, específicamente para que se realice un registro, que una habitación seleccionada tenga una tarifa asignada, si hay dos tarifas para un mismo rango de fechas se seleccione la más barata, etc. Cuenta con una lista de tarifas (Fare) y un conjunto con la información de la habitación (camas, características, tipo).

12. **Fare**: Clase que implementa HotelObject. Por medio de esta se modelan las tarifas de cada habitación. Cuenta con un precio, una fecha de inicio, una fecha final, y una lista con los días en los que se aplica la tarifa (teniendo en cuenta que la tarifa puede variar para días entre semana y fines de semana.
13. **BookingHandler**: Clase que permite crear nuevas reservas . Adicionalmente, con esta clase se añaden las habitaciones reservadas a la reserva. Cuenta con openBooking, atributo que es usado para crear una nueva reserva.
14. **Booking**: Clase que implementa HotelObject. Por medio de esta clase se modelan todas las ***reservas*** que realicen los recepcionistas. Tiene el nombre de la persona con la cual queda guardada la reserva, su DNI, número de teléfono, email, número de tarjeta de crédito, el número de huéspedes que se hospedarán (contando al principal), la fecha final de la reserva, la fecha final de la reserva y la lista que contiene los ids de las habitaciones reservadas.
15. **Room**: Clase que extiende RoomModel e implementa HotelObject. Se usa para modelar todas las habitaciones que son creadas en la aplicación. Contiene los atributos de roomId, location, capacity, isOccupied, bookedDates, type, beds, featuresList.
16. **RoomModel**: Es una clase que se usa para modelar habitaciones que tienen atributos iguales. Estos son el tipo de habitación, el número de camas y las características que tiene la habitación (cocina, balcón, etc).
17. **TypeRoom**: Es una enumeración usada para modelar los posibles tipos de habitación, esto con el fin de que al momento de crear una nueva habitación solo se pueda aceptar uno de los tipos que se encuentran definidos en esta enumeración. La enumeración contiene los valores: STANDARD, SUITE, DOUBLE_SUITE.
18. **Bed**: Es una enumeración que sirve para modelar los tipos de camas disponibles para una habitación. Cada tipo de cama posee la cantidad de personas que puede almacenar. Es fundamental para la creación de habitaciones en el hotel.
19. **RoomFeature**: Clase tipo enumeración que contiene las características principales que puede llegar a tener una habitación. Por ejemplo, si tiene balcón, vista del paisaje, o cocina.
20. **HotelDataHolder**: Clase abstracta que se encarga de modelar y gestionar la persistencia de datos del Hotel. Por ejemplo, funciona para guardar los datos en la persistencia o cargar los datos.
21. Registration: Clase que permite la modelación de los registros del hotel. Cuenta con un huésped principal, una lista de acompañantes, una lista con los ids de las habitaciones reservadas, una lista de comidas consumidas (es decir todas aquellas que no son pagadas instantáneamente), una lista de servicios consumidos (todos aquellos que no son pagados instantáneamente), una fecha inicial y una fecha final.
22. **FoodDataHandler**: Clase que funciona para crear, cargar o obtener los datos de las comidas/ bebidas asociadas al restaurante del hotel.
23. **RegistrationDataHandler**: Clase que funciona para crear, cargar o obtener los datos asociados a los registros en el hotel. Hay que tener en cuenta que los registros asociados al hotel, es la información que contiene los datos de los huéspedes y del grupo asociado, junto con las habitaciones que van a ocupar en un rango de fechas. Asimismo, la informacion de los consumos de servicios o del restaurante qué van a hacer el huésped o

sus invitados. En pocas palabras, toma los datos de las personas que van a ingresar al hotel. Todos los datos creados se guardan como un JSON.

24. **ServicesDataHandler**: Clase que funciona para crear, cargar o obtener los datos asociados a las reservas de un hotel. Todos los datos creados se guardan como un JSON.
25. **UsersDataHandler**: Clase que funciona para crear, cargar o obtener los datos asociados a los usuarios de un hotel. En otras palabras, esta clase es utilizada para llevar un registro de los usuarios (administrador, recepcionista y empleado) que pertenecen a la aplicación.
26. **BookingsDataHandler**: Clase que funciona para crear, cargar o obtener los datos asociados a las reservas en el hotel. En este caso, las reservas es la información de las habitaciones que van a ser utilizadas en un rango de fechas definido por la persona que va a reservar, junto con la información de quien va a reservar. Crear una reserva en el hotel, no significa que ya sea un huésped del hotel, o que se pidan los datos de sus acompañantes (esto lo hace el registro)
27. **RoomsDataHandler**: Clase que funciona para crear, cargar o obtener los datos asociados a las habitaciones en el hotel. Todos los datos creados se guardan como un JSON.
28. **FaresDataHandler**: Clase que funciona para crear, cargar o obtener los datos asociados a las tarifas en el hotel, las tarifas cargadas son por tipo de habitación, es decir por habitaciones que contengan exactamente las mismas características. Todos los datos creados se guardan como un JSON.
29. **RegisterHandler**: Clase que se encarga de crear un nuevo registro, tomando los datos del nuevo huésped como su nombre, dni,email, número de teléfono, entre otros. Para ello, instancia a las clases Principal Guest y Registration.
30. **UserType**: Enumeración que contiene los tipos de usuarios de la aplicación (ADMIN,RECEPCIONIST, EMPLOYEE). Se utiliza en la autentificación de usuarios para saber que tipos de usuario se registran o inicia sesión dentro de la aplicación
31. **User**: Clase abstracta que obliga a tener la obtención de los datos para los usuarios de la aplicación.
32. **Recepcionist**: Clase que contiene los datos del usuario recepcionista, como su nombre, contrasena o tipo de usuario
33. **Admin**: Clase que contiene los datos del usuario Admin, como su nombre, contrasena o tipo de usuario
34. **Employee**: Clase que contiene los datos del usuario Employee, como su nombre, contrasena o tipo de usuario
35. **HotelWorkersAuth:** Clase con la que se genera toda la autentificación de usuarios. Por medio de esta clase se hace el inicio de sesión de usuarios existentes y el registro de usuarios nuevos.
36. **Hotel**: Clase por las cual se controlan todos los DataHandler. Esta es la única clase que debe instanciarlos con el fin de evitar errores al momento de guardar y cargar datos. Sus atributos son todos los DataHandlers.
37. **App**: Se encarga de mostrar la consola con las interacciones a cada usuario de la aplicación. .En otras palabras, funciona como un View.