Miguel Soto Martín — *miguel.soto@estudiante.uam.es*
Nicolás Victorino Ruiz — *nicolas.victorino@estudiante.uam.es*
Diego González Romero — *diego.gonzalezromero@estudiante.uam.es*

DOCUMENTO DE DIAGRAMAS

Entrega 3.2: Codificación de Interfaz Gráfica de Usuario (GUI)
Aplicación: NANO GYM

FECHA: 07/05/2023

1. Introducción

El programa main de la aplicación con interfaz gráfica se encuentra en src/mainGUI.java/MianGUI.java. Para ejecutar el gimnasio con algunos datos de usuarios, clientes, actividades y salas precargados se tiene que ejecutar previamente src/escenario/EscenarioGym.java.

Los principales usuarios precargados son:

- Cliente → usuario: pato, contraseña: 12345
- Monitor → usuario: moni, contraseña: 12345
- Administrador → usuario: admin, contraseña: 12345.

En nuestro caso, para poder ejecutar EscenarioGym.java, era necesario Eclipse (en Visual Studio no funcionaba bien).

Dentro de la aplicación, en algunas pestañas como actividades, donde se pueden llegar a listar todas las actividades que hay en el gimnasio en la primera semana, no se pueden llegar a ver todas por el tamaño de la ventana. Nuestra intención ha sido poner un scroll para poder ver todas las actividades, hasta el final, pero al intentarlo nos han surgido muchos problemas y, por falta de tiempo, hemos priorizado realizar una aplicación completa a añadirle el scroll.

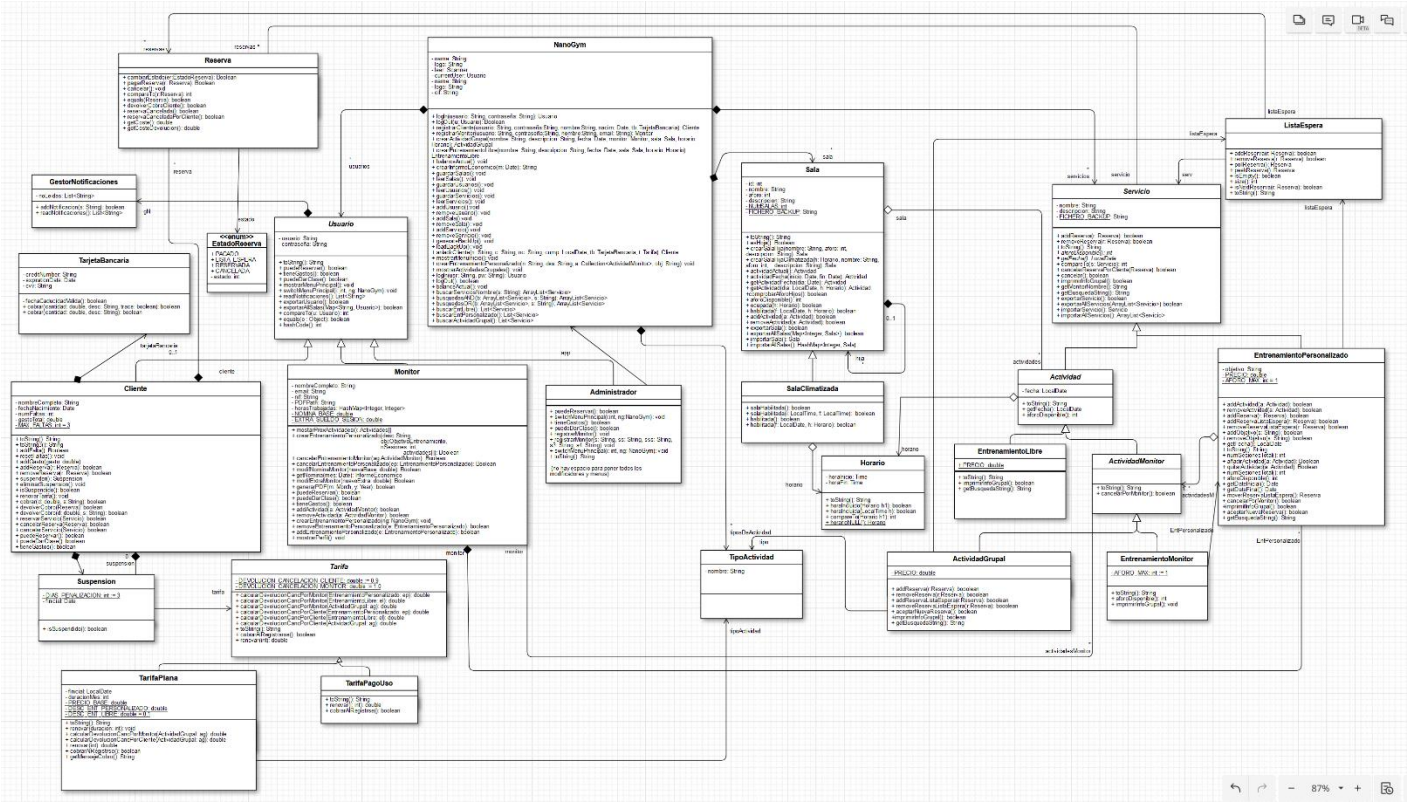
Para la entrega anterior realizamos un extenso y detallado programa main de prueba se encuentra en la carpeta testMain dentro de la carpeta test. Para utilizarlo puedes o bien registrarte como un Cliente propio, con tu nombre y contraseña y probar la experiencia de la aplicación, o bien registrarte como administrador (nick: admin, contraseña: 12345). También se puede probar la experiencia de monitor, utilizando el nick:hola, contraseña:12345, o como monitor con el usuario: adios, contraseña: 12345.

A continuación, explicamos nuestro diagrama de clases actualizado, e incluimos explicaciones de las anteriores entregas.

1.1 DIAGRAMA DE CLASES (UML)

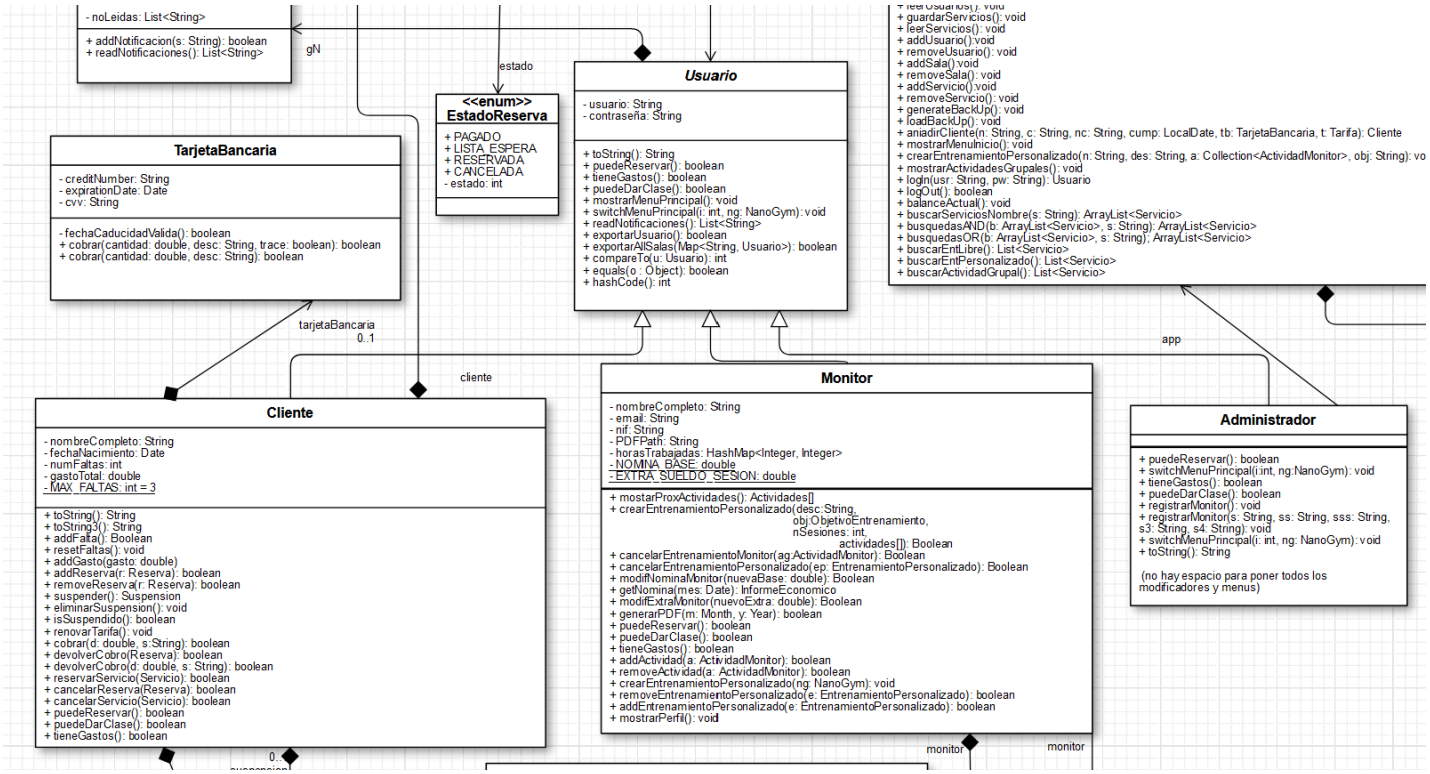
Para una mejor visualización del diagrama, vamos a comentarlo en pequeños fragmentos, ya que en caso de insertar una imagen de todo el diagrama sería imposible la lectura del mismo.
Para ver el diagrama al completo, dejamos este enlace: https://cacoo.com/diagrams/D3k0im8zZEBi6kwO/03AEC

Diagrama de clases completo de la entrega 3.2:



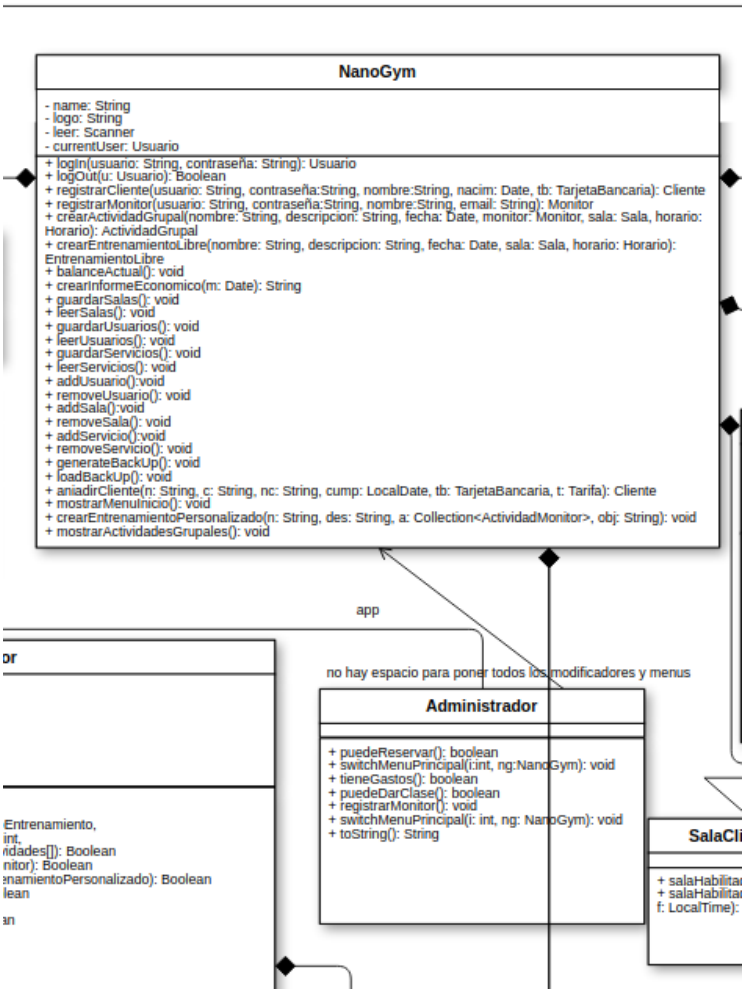
No hemos incluido los métodos setters y getters en el diagrama.

1.1.1. Usuarios de la aplicación



Hemos optado por crear una clase abstracta de la que heredan los tres tipos de usuarios: Cliente, Monitor y Administrador. Hacemos esto porque todos comparten los atributos usuario y contraseña, y para poder tener un inicio de sesión único que sirva para los 3 tipos, ya que utilizan la misma información para hacerlo (usuario y contraseña).

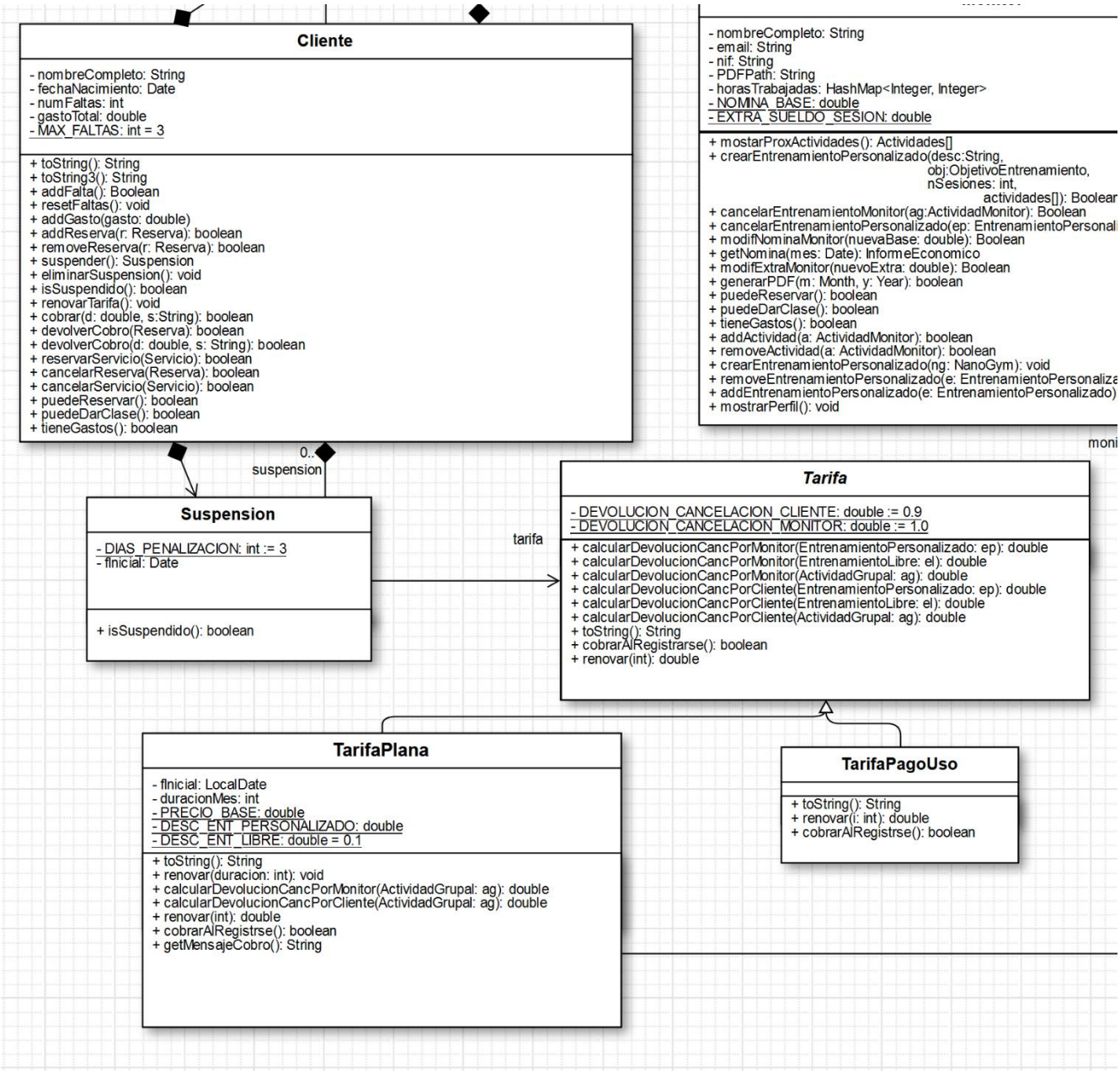
1.1.2. Gestión del gimnasio y administrador



Para poder gestionar la aplicación hemos creado la clase NanoGym, que contiene todas las funciones necesarias para ello. Esta clase también incluye los métodos para registrar e iniciar a los diferentes tipos de Usuario, además de métodos para filtrar los diferentes servicios y para ver el balance del gimnasio. Para guardar y gestionar la información del gimnasio también contiene a todos los usuarios, las salas, los servicios (o actividades) y los informes económicos del gimnasio.
Por otra parte, como ya hemos mencionado previamente, hemos creado la clase Administrador, cuyo único atributo es NanoGym, dándole acceso a toda la información contenida en esta, además de a todos sus métodos.

En administrador, no hemos sido capaces de poner todos los métodos, debido al poco espacio. Los métodos restantes solo son los encargados de imprimir la interfaz y de modificar parámetros del gimnasio.

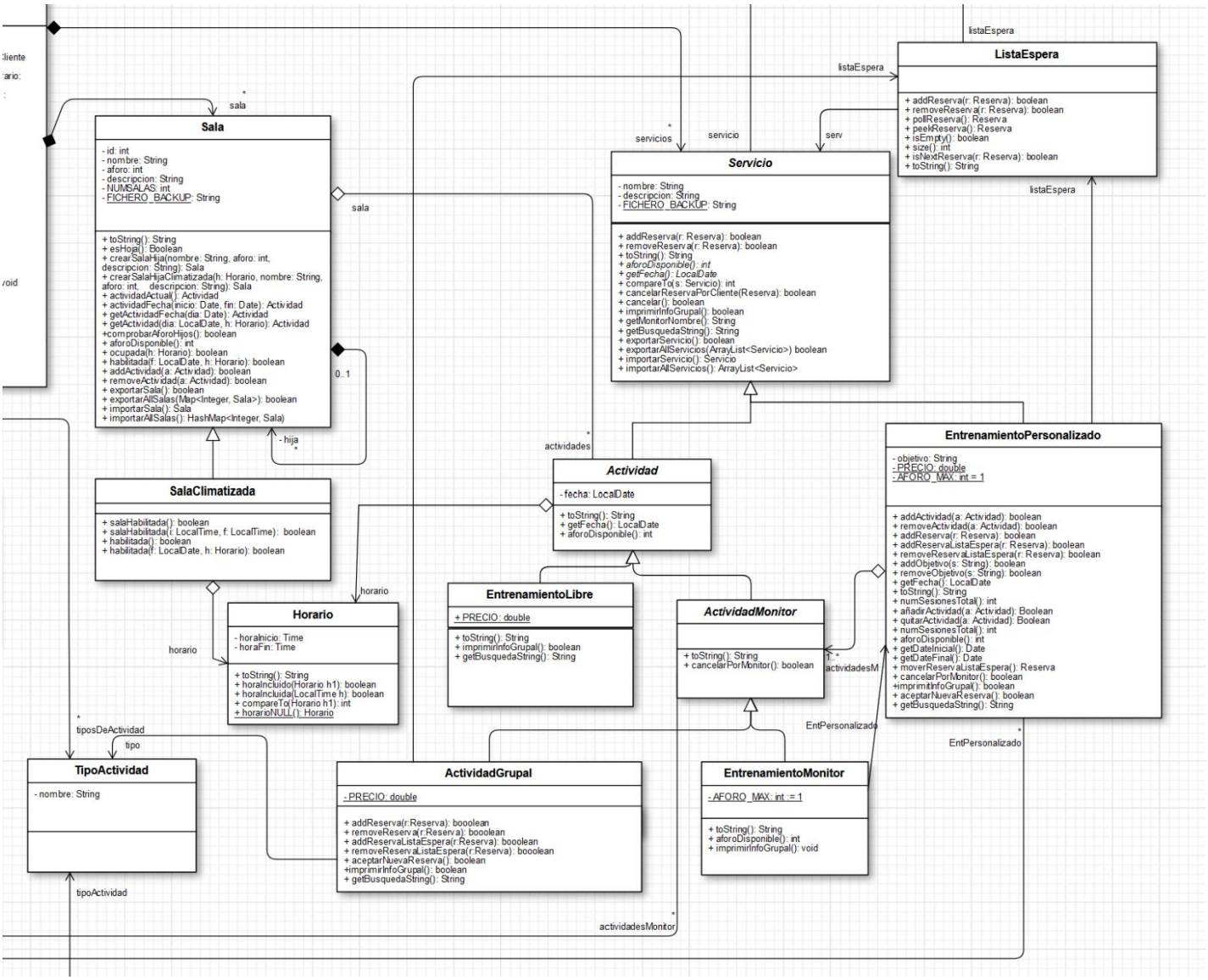
1.1.3. Cliente y tarifa



La clase Cliente contiene toda la información relevante del cliente, además del atributo estático MAX_FALTAS, que indica el número máximo de sesiones seguidas a las que puede faltar un cliente antes de ser suspendido. Además, contiene la clase abstracta Tarifa, de la cual hereden TarifaPlana y TarifaPagoPorUso. Está indica que tarifa tiene el cliente además de incluir información general para cada tarifa (precio base, devoluciones dependiendo de la tarifa, ...). En la tarifa plana se incluye además la duración de la tarifa que ha escogido el cliente (duracionMes). El cliente también contiene un atributo de clase tarjeta, que contiene la información de la tarjeta con la que paga el cliente.

Por último, Cliente puede incluir un atributo de tipo Suspensión. Esto se da cuando el número de faltas del cliente (numFaltas) es mayor o igual que MAX_FALTAS. Esta clase simplemente incluye la duración de la suspensión impuesta al cliente, y se elimina pasados DIAS_PENALIZACION (atributo de tipo estático).

1.1.4. Servicios

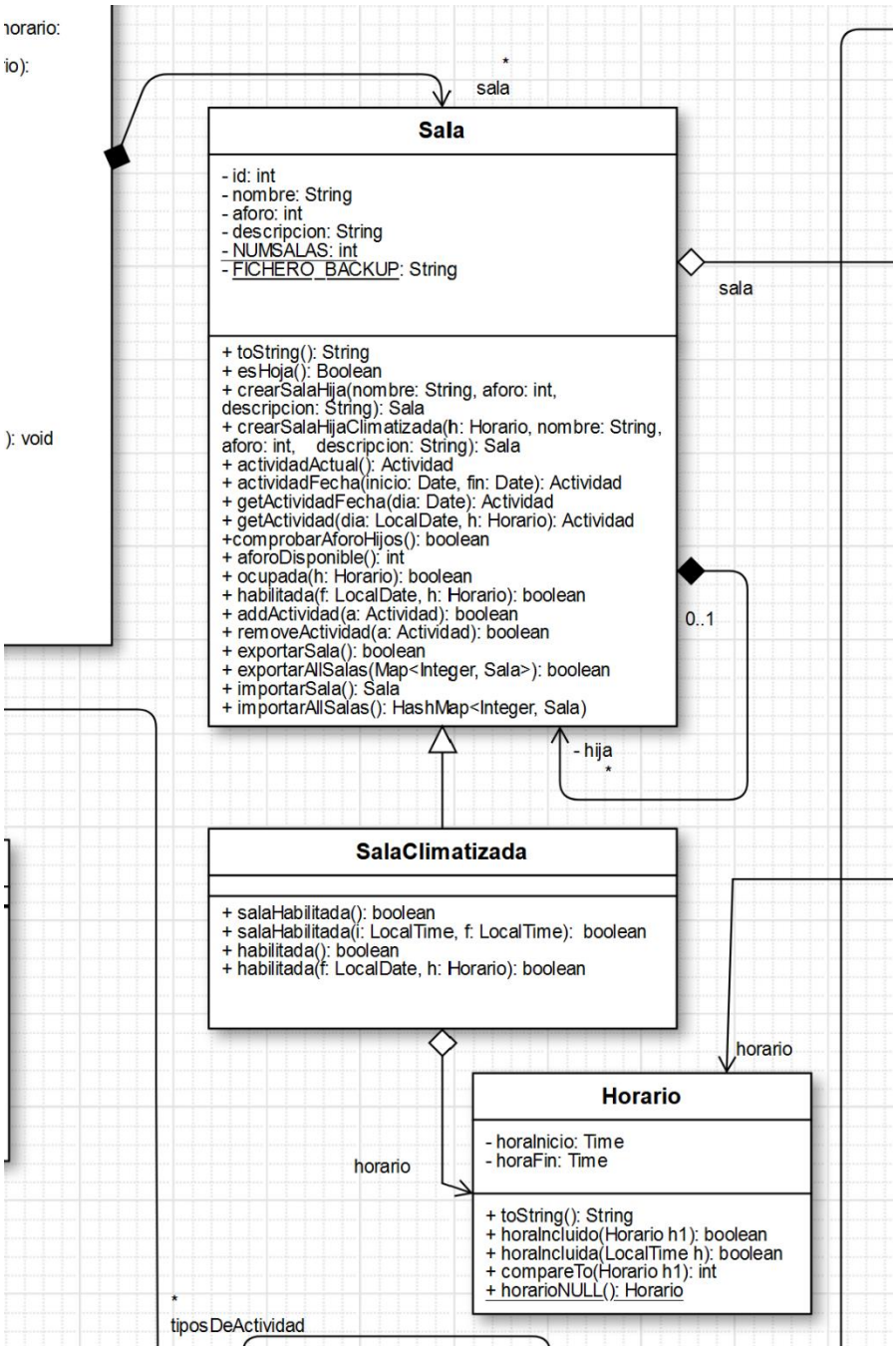


Para organizar las diferentes actividades hemos optado por separarlas en 3 grupos principales, haciendo uso de clases abstractas. La primera división la realizamos con la clase Servicio, que nos divide la clase EntrenamientoPersonalizado (conjunto de actividades que incluye un entrenamiento personalizado, incluyendo su precio, además del objetivo de este (para esto empleamos una enumeración)) del resto de actividades.
La segunda división la hacemos en la clase Actividad, que nos divide las actividades con monitor (ActividadMonitor, que es abstracta) de las actividades sin monitor (EntrenamientoLibre). Esta clase incluye la fecha en la que se va a realizar la actividad, además de un atributo de tipo Horario, que marca la hora de inicio y fin de la sesión. Por último, también se incluye un atributo de tipo Sala, que indica en que sala se va a realizar la actividad.

La tercera y última división se hace con la clase ActividadMonitor, que divide las actividades grupales (ActividadGrupal, que incluye la actividad a realizar en la sesión como una enumeración) de las sesiones de entrenamiento personalizado (EntrenamientoMonitor). Esta clase está contenida por EntrenamientoPersonalizado, ya que ambas pueden formar parte de una sesión de entrenamiento personalizado.

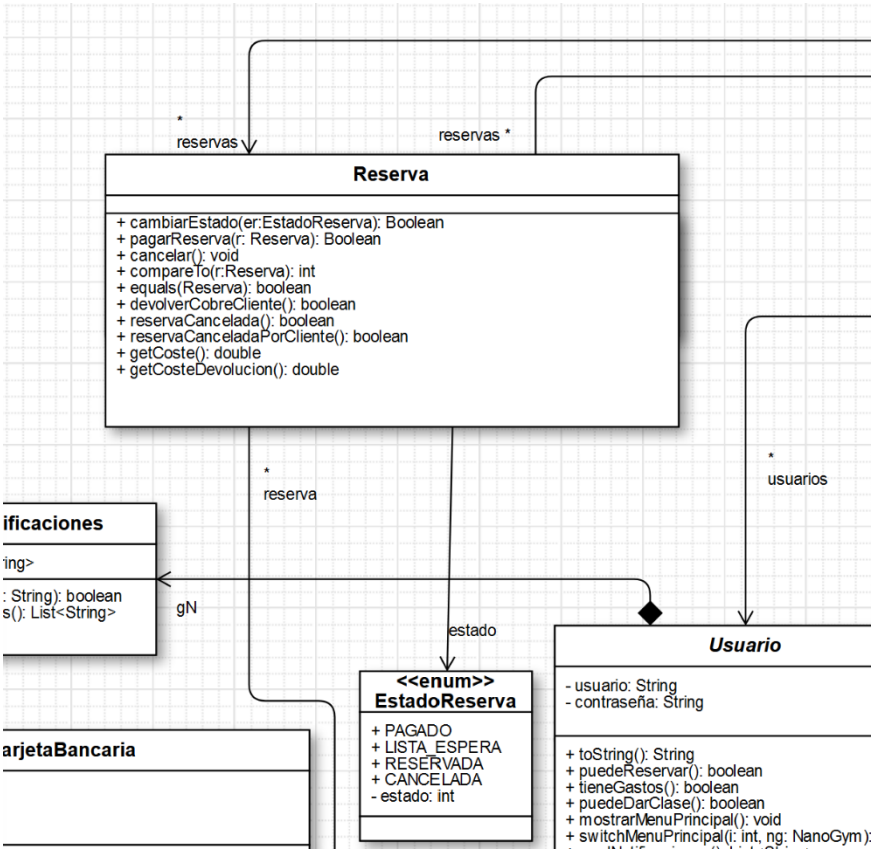
Por último, los servicios tienen una lista de espera, la cual guarda en orden las reservas que se han realizado después de que se llenara el aforo, y en caso de haber alguna cancelación, va incluyendo en orden estas reservas.

1.1.6. Salas



Las salas se dividen en salas sin climatizar (Sala), y en salas climatizadas (SalaClimatizada). La sala no climatizada incluye un id, que para asignarlo a cada sala y que sea único se hace uso del atributo estático NUMSALAS, que guarda el número de salas que hay. Al añadir una nueva sala se aumenta este número en 1 y se utiliza como id de la sala.
Las salas climatizadas heredan de la clase Sala, e incluyen un atributo de clase Horario, que indica el rango de tiempo en el que está climatizada la sala.
Por último, las salas se pueden contener a sí mismas. Esto crea una relación de salas padre y de salas hijo. El aforo de la sala padre debe ser igual a la suma del aforo de sus salas hija.

1.1.6. Reservas



La clase Reserva sirve para gestionar la lista de espera de los diferentes servicios, además de guardar la información de los Clientes que han reservado y los Servicios que han sido reservados. Esto es útil a la hora de cobrarle a un Cliente cuando sale de una lista de espera, además de facilitar la búsqueda de los Servicios a los que está apuntado un cliente, pudiendo además mostrárselos con la interfaz más fácilmente.
Para llevar el estado de una reserva (ver si está pagada, en lista de espera, cancelada o reservada), hemos empleado una enumeración (EstadoReserva).

1.1.7. Serialización

En esta entrega hemos implementado la serialización dentro de las clases principales: Sala, Servicio y Usuario.