Descripción general del proyecto

Este proyecto es una 'Java Application' que utiliza hibernate sobre base de datos MySQL para la persistencia de los datos. La configuración de hibernate se ha realizado en base a ficheros xml.

Génesis y desarrollo del proyecto

Tras leer el enunciado se ha tratado de realizar un enfoque 'ágil' al desarrollo del proyecto:

- producto incremental.
- añadir valor/funcionalidad en cada 'sprint'.
- aplicar conocimiento de otros módulos del curso en la medida de lo posible.

Las diferencias principales en el modelos de la base de datos respecto a ejercicio anterior (modelo base) son:

- se utiliza nomenclatura anglosajona.
- Los campos clave de las tablas son siempre campos numéricos.

El nombre de la base de datos utilizada es 'Aena' (ver `src/bbdd/Aena.sql`).

Airplane
El primer paso fue implementar la tabla 'airplane' avión (se ha decidido nombrar a todos los elementos [tablas, campos, métodos y atributos] en inglés), mapearla, implementar el pojo y comprobar que la conexión a la BBDD, el mapeo hibernate y los elementos persistentes están correctamente configurados: buscar un airplane en base a su id, así como recuperar todos los airplanes de la base de datos (el fichero '`src/bbdd/Aena.sql`' realiza una par de inserts de airplanes).

Airport / gates
Como segundo paso se abordó la problemática 'aeropuerto/puertas de embarque' (airport / gates).

Se implementa la tabla 'airport': la diferencia principal con el esquema presentado en la práctica anterior, es que el campo 'numPuertas' no se implementa (lo que aporta valor en el sprint no se implementa).

Se implementa la funcionalidad para recuperar objetos persistentes: obtener un aeropuerto por 'id', así como recuperar todos los aeropuertos (el fichero '`src/bbdd/Aena.sql`' realiza una par de inserts de aeropuertos).

Se añade la tabla 'gate'.

Se implementa la relación uno a muchos airport / gate uni-direcccional: desde un aeropuerto se permitirá navegar a sus puertas, pero desde una puerta no se podrá navegar a un aeropuerto. Este punto (relación one to many) ha sido uno de los puntos conflictivos: no es lo mismo definir la relación como set, list o bag. Otro aspecto es como se mapea la relación: inicialmente al eliminar una airport, no eliminaban sus gates: se dejaba en la tabla 'gate' con el idAirport a null. Finalmente encontré el parámetro adecuado para que no dejase elementos huérfanos en la tabla.

Airplane / boardingpasses
Como tercer paso se abordó la problemática 'avión/billetes' (airplane / boardingpass).

Se modifica la tabla la tabla 'airplane': incorporando los siguientes campos:

- totalNumSeats
- totalNumSeats
- numSeatsConfirmed
- numSeatsBooked

para poder llevar un control de los billetes vendidos (book) y los confirmados (se han montado en el avión).

Se implementa la funcionalidad listAirplaneByPlateNumber y setAirplaneTotalNumSeats (para poder 'configurar' una avión).

Se crea la tabla 'boardingpass' con un campo 'boarded' (el pasajero ha subido al avión) para controlar si un billete vendido (book) se ha confirmado (ha subido el pasajero al avión). *Nota importante: en e modelo original de la BBDD no existía éste campo*.

Se implementa la relación uno a muchos airplane / boardingpass uni-direcccional: desde un airplane se permitirá navegar a sus boardingpasses, pero desde un boardingpass no se podrá navegar a un airplane.

Se implementa las funcionalidad bookSeat y confirmSeat: no se pueden vender más billetes de los configurados para cada avión. Y sólo se pueden confirmar billetes si existen billetes vendidos sin confirmar.

Se implementa un dao: en vez abrir y cerrar sesiones en el programa principal se agrupa toda la funcionalidad de persistencia en un dao.

Aunque por 'programación' de los pojo's ya se responde a las consultas de dado un aeropuerto determinar las puertas de embarque disponibles, y dado un avión determinar el número de billetes vendidos y confirmados (recorriendo las listas de gates y boardingpasses). Se implementan las consultas específicamente con HQL y Criteria:

- `gatesAvailableAtAirport airportName metodo`
- `numBoardingpassesBookedFromAirplane plateNumber metodo`
- `NumBoardingpassesConfirmedFromAirplane plateNumber metodo`

Nota: método puede ser '`HQL`' o '`Criteria`'.

Los modelos ER y UML de la base han sido añadidos al proyecto.

Alcance del proyecto

Por problemas de tiempo no se ha implementado más que lo descrito.

Adicionalmente, en la última media hora de la última sesión de Spring roo se realizó un intento de ingeniería inversa que aunque si bien 'levantó' una web ésta no era capaz de hacer persistir datos. Tampoco el modelo obtenido era el esperado / implementado.

También he realizado otro intento utilizando Spring roo de implementar airport-gate, con un resultado parecido: puedo levantar una web que aunque si bien es capaz de persistir datos, las relaciones no están implementadas con en este proyecto: se necesitan más horas de 'vuelo' para poder 'replicar' el comportamiento del proyecto entregado.

Descripción de los archivos principales del del proyecto

- `src/hibernate.cfg.xml`: fichero de configuración:
  - Session-factory.
  - Mappings.

- `src/hibernate.properties`: fichero de configuración hibernate con la base de datos.

- `src/log.txt`: log de progreso del proyecto.

- `src/bbdd/Aena.sql`: fichero 'sql' que contiene las queries para crear y configurar la base de datos MySQL utilizada en el proyecto. Contiene la inserción manual de algunos datos a fin de comprobar la aplicación.

- `src/resources/com/innova4b/aena/persistent`: mappings utilizados en la aplicación:
  - `Airplane.hbm.xml`
  - `Airport.hbm.xml`
  - `Boardingpass.hbm.xml`
  - `Gate.hbm.xml`

- `src/diagramas/er`: diagrama Entidad-Relación.

- `src/diagramas/uml`: diagrama UML.

- `src/com/innova4b/aena/dao`: interfaces de acceso a los datos.
  - `AirplaneDAO.java`
  - `AirportDAO.java`

- `src/com/innova4b/aena/daoImpl`: implementación de los interfaces de acceso a datos.
  - `HibernateUtil.java`: singleton de la session factory de acceso a la base de datos via hibernate.
  - `AirplaneDAOImpl.java`
  - `AirportDAOImpl.java`

- `src/com/innova4b/aena/persistent`: POJO's persistentes.
  - `Airplane.java`
  - `Airport.java`
  - `Boardingpass.java`
  - `Gate.java`

- `src/com/innova4b/aena/trans`: main de la aplicación java.

Opciones de ejecución del proyecto

El proyecto responde a los siguiente comando:

- `getAllAirplanes`: muestra la información referente a todos los aviones de la base de datos.
- `newAirplane plateNumber`: crea un nuevo avión en la base de datos. No permite repetir matrícula (plateNumber).
- `deleteAirplaneById plateNumber`: elimina de la base de datos el avión con la matricula indicada. Eliminará todos los billetes (boardingpasses) relacionados con el avión.
- `getAirplaneById plateNumber`: muestra la información referente al avión con la matrícula especificada.
- `configureAirplaneById plateNumber totalNumSeats totalNumSeatsToBook`: configura el avión con la matrícula especificada: número total de asientos y número total de billetes que se pueden vender / reservar (book).
- `bookSeatAtAirplaneById plateNumber`: crea una reserva (boardingpass) para el avión con la matrícula indicada si no ha sobrepasado el número máximo configurado para el mismo.
- `confirmSeatAtAirplaneById plateNumber`: confirma que una reserva (boardingpass) se ha utilizado (el pasajero ha subido al avión) si el avión con la matrícula indicada tiene alguna reserva disponible.
- `getAllAirports`: muestra la información referente a todos los aeropuertos de la base de datos: nombre, número de puertas (gate), número de puertas disponibles y cuáles son y número de puertas no disponibles y cuáles son.
- `GetAirportById name`: muestra la información del aeropuerto especificado: nombre, número de puertas (gate), número de puertas disponibles y cuáles son y número de puertas no disponibles y cuáles son.
- `newAirport name`: crea un nuevo aeropuerto con el nombre indicado. No permite repetir nombre.
- `deleteAirport name`: elimina el aeropuerto con el nombre indicado. Eliminará también todas las puertas (gates) asociadas al mismo.
- `addGateToAirport name`: añadir una puerta (gate) al aeropuerto indicado. La nueva puerta tendrá como número uno más que la anterior (se debería de realizar un control más completo: si se añaden y borran puertas, es fácil llegar a tener varias puertas con el mismo número (que no id)). Las nuevas puertas siempre están disponibles.
- `deleteGateFromAirport name gateNumber`: borra la puerta (gate) indicada del aeropuerto indicado.

- `changeGateStatusFromAirport name gateNumber`: cambia el estado de la puerta indicada del aeropuerto indicado. Si está disponible a no disponible, y viceversa.
- Dfgdfg
- `gatesAvailableAtAirport airportName metodo`: dado un aeropuerto, determina el número de puertas de embarque (gates) disponibles. Si metodo es 'HQL' realiza la consulta vía HQL, si metodo es 'Criteria' realiza la consulta vía Criteria.

- `numBoardingpassesBookedFromAirplane plateNumber metodo`: dado un avión, determina el número de asientos reservados (booked). Este número se corresponde con el número de billetes comprados para un avión determinado. Si metodo es 'HQL' realiza la consulta vía HQL, si metodo es 'Criteria' realiza la consulta vía Criteria.

- `NumBoardingpassesConfirmedFromAirplane plateNumber metodo`: dado un avión, determina el número de asientos ocupados. Este número se corresponde con el número de personas que están dentro de un avión determinado. Si metodo es 'HQL' realiza la consulta vía HQL, si metodo es 'Criteria' realiza la consulta vía Criteria.