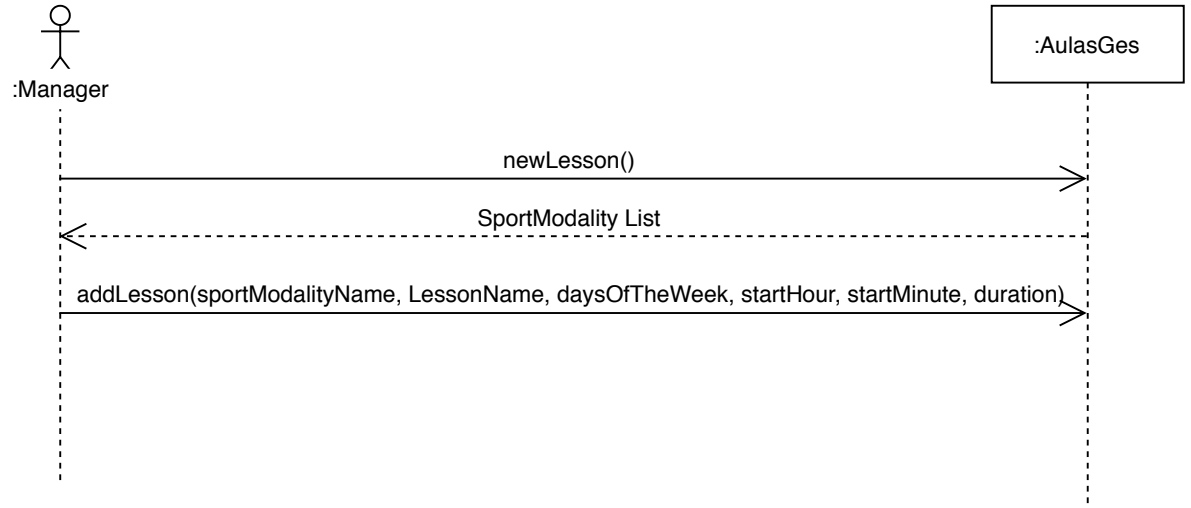
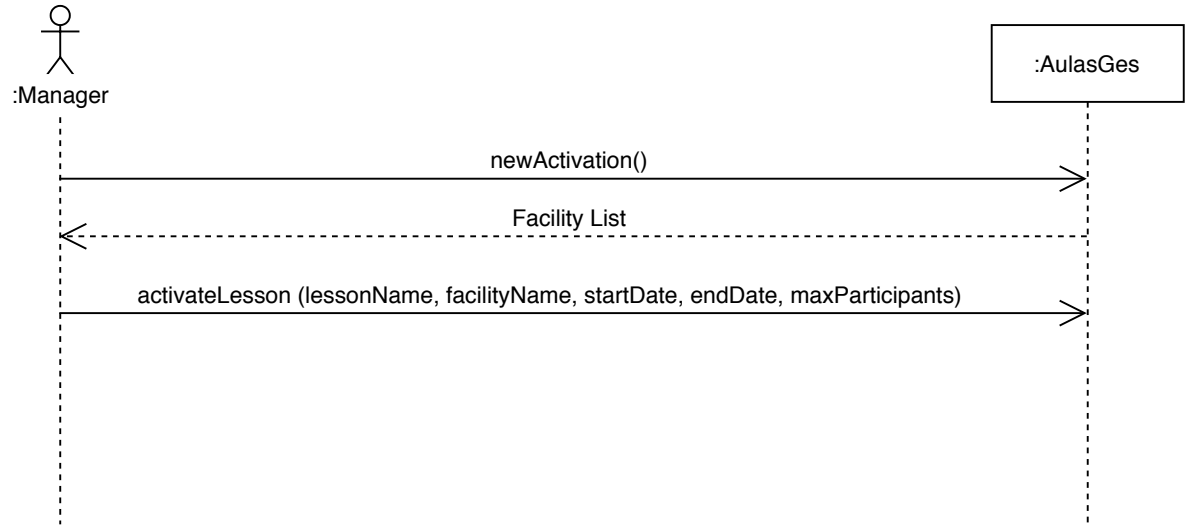
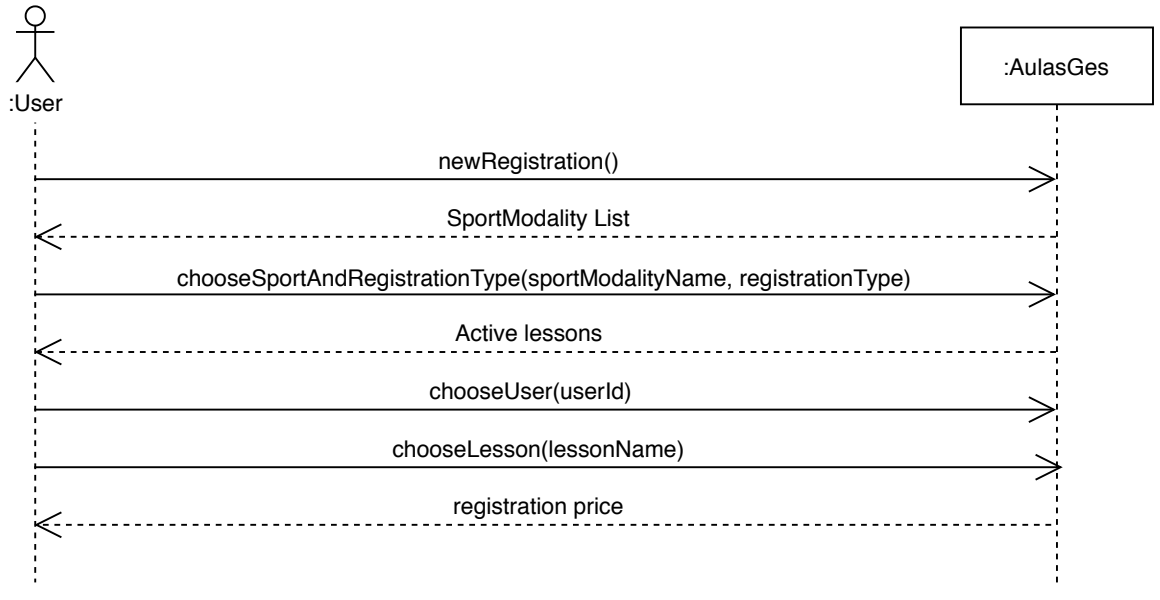
SSD 1) Criar Aula



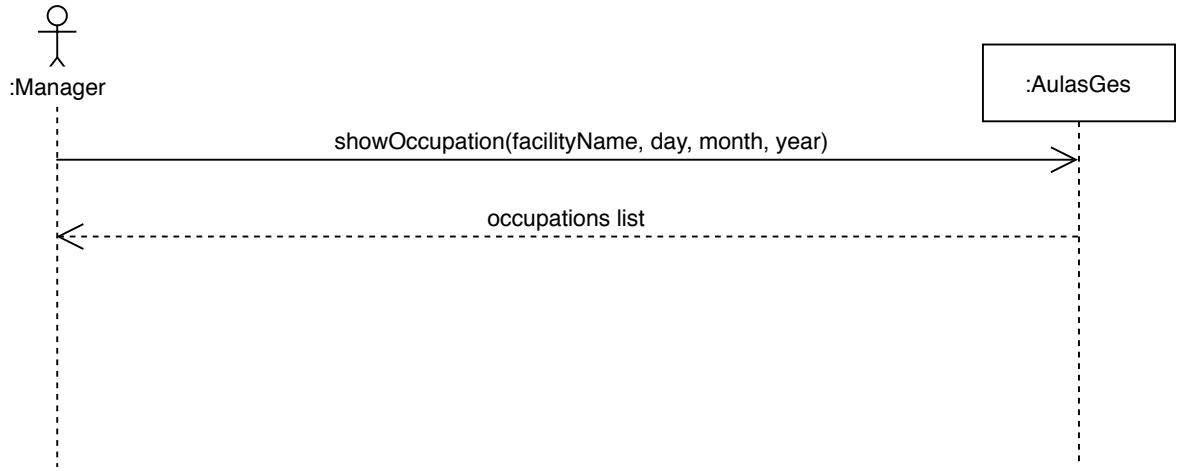
SSD 2) Ativar Aula



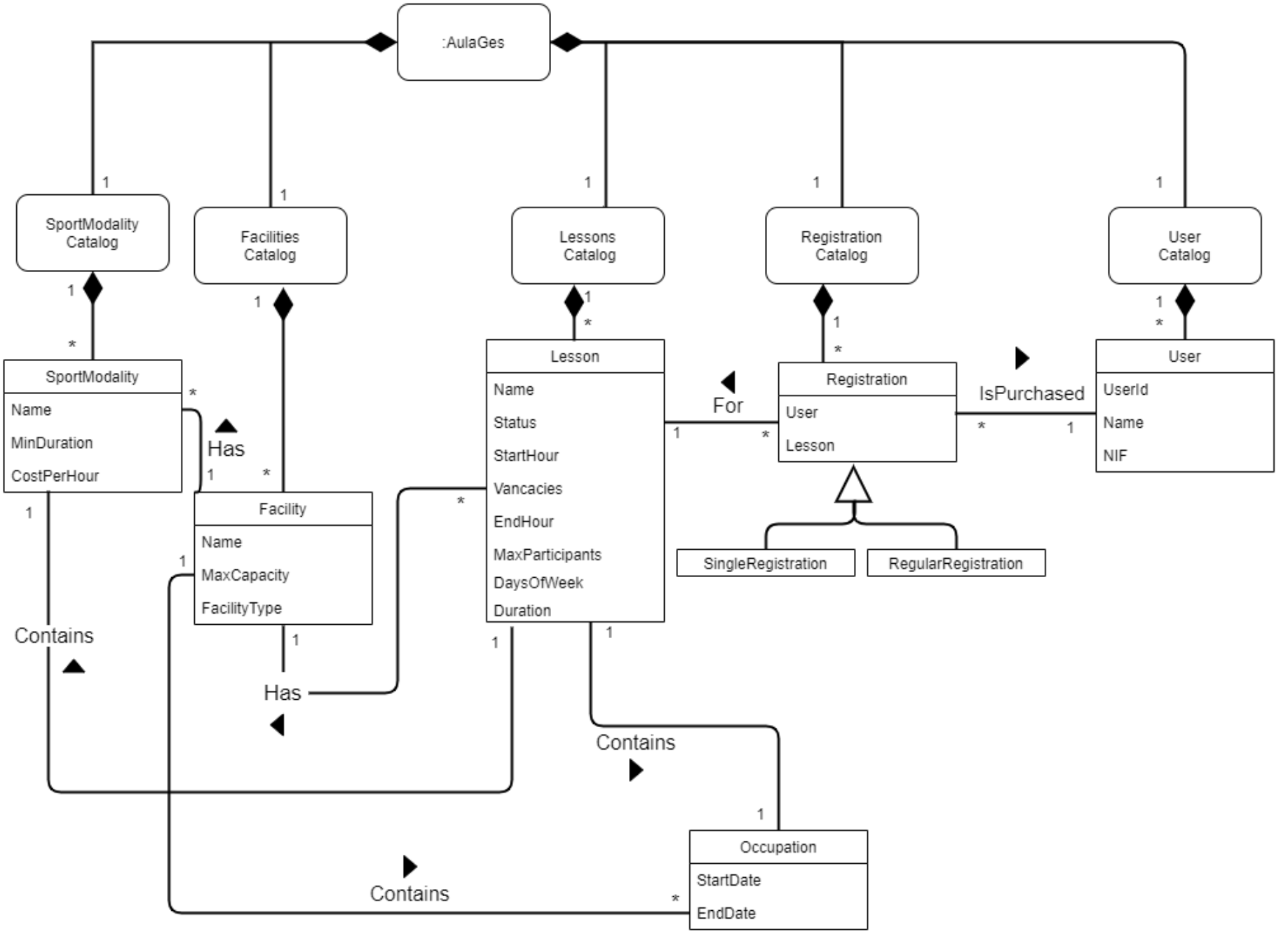
SSD 3) Inscrever em Aula



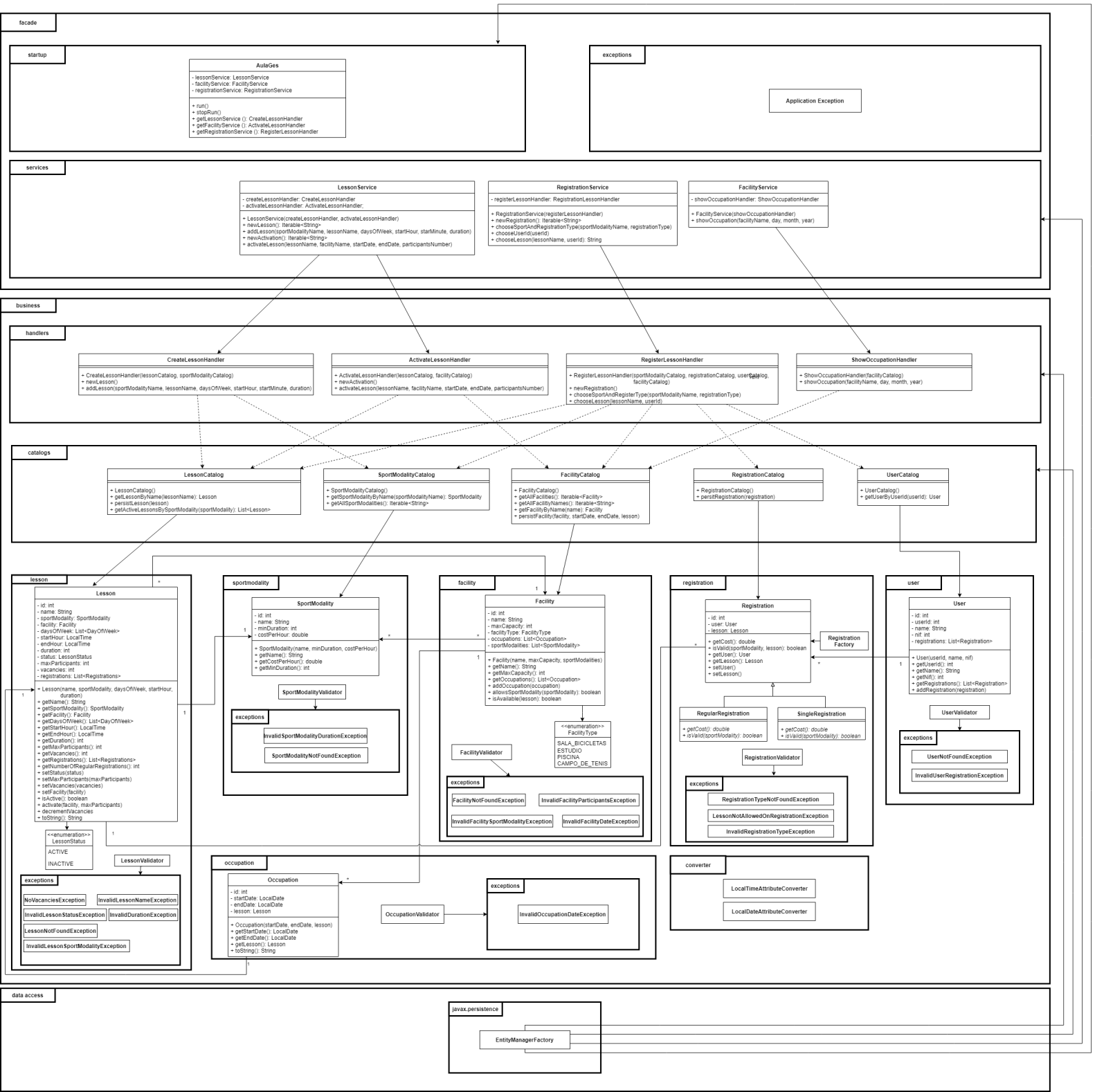
SSD 4) Visualizar Ocupação de Instalação

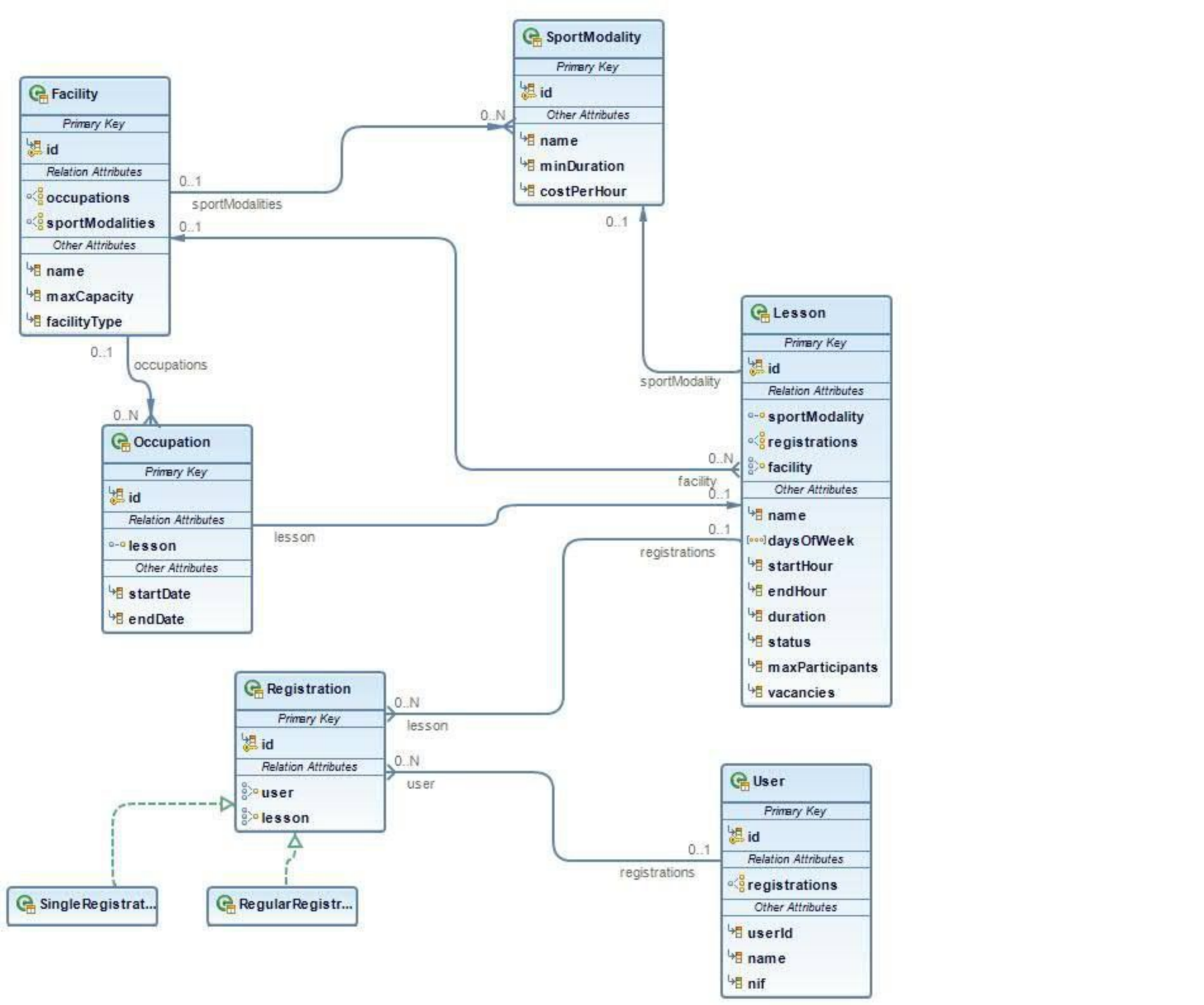


Modelo Dominio



Modelo de Classes




SportModality
Primary Key
id
Other Attributes
name
minDuration
costPerHour
Facility
Primary Key
id
Relation Attributes
occupations
sportModalities
Other Attributes
name
maxCapacity
facilityType
0..N
0..1
sportModalities
0..1
0..1
occupations
0..N
Lesson
Primary Key
id
Relation Attributes
sportModality
registrations
facility
Other Attributes
name
daysOfWeek
startHour
endHour
duration
status
maxParticipants
vacancies
sportModality
0..N
facility
0..1
0..1
registrations
Occupation
Primary Key
id
Relation Attributes
lesson
Other Attributes
startDate
endDate
lesson
Registration
Primary Key
id
Relation Attributes
user
lesson
0..N
lesson
0..N
user
User
Primary Key
id
Relation Attributes
registrations
Other Attributes
userId
name
nif
0..1
registrations
SingleRegistrat...
RegularRegistr...

Decisões importantes no desenho da aplicação

Para o desenvolvimento do desenho da aplicação do projeto AulaGes, decidimos recorrer a alguns *Design Patterns* (mais focados na implementação do caso de uso 3), a estratégias de mapeamento de dados com JPA e à criação de algumas classes de auxílio e exceções.

De maneira a implementar caso de uso em que era pedido que inscrevêssemos um utilizador, previamente registado no sistema, numa aula ativa, na meta 3 utilizamos o Padrão *Strategy* para definir as classes dos vários tipos de inscrição (avulso e regular). Adicionalmente, na meta 4 foi necessário criar uma classe factory de inscrições (seguindo o Padrão *Factory*), para que fosse possível criar as mesmas com os tipos indicados pelo utilizador, de forma a serem persistidos de seguida.

Para realizar as validações da camada de negócio, verificámos que muitas delas iriam ser novamente utilizadas em diferentes casos de uso. Por isso, decidimos criar classes de validação para os diferentes conceitos do programa e evitar assim repetição de código tornando-o mais legível. Criámos também várias exceções com essa finalidade, e também para dar ao utilizador uma melhor leitura dos erros que possam vir a acontecer quando este correr a aplicação.

Os tipos de dados utilizados para descrever as datas no projeto foram o LocalTime, LocalDate e LocalDateTime, devido ao problema de versões do JPA referido nas aulas teóricas e práticas, houve necessidade de implementar as classes de conversão destes tipos para java.sql.Date e java.sql.Time.

Com o intuito de mapear as classes criadas na meta 3, para criar a base de dados da meta 4 utilizámos várias anotações JPA. Nos atributos de listas com objetos de relações decidimos aplicar o padrão *Lazy Load,* de forma a apenas carregar esses objetos quando fossem chamados, tornando a aplicação mais eficiente. Para mapear as relações de herança optámos por definir as anotações *@Inheritance(strategy = InheritanceType.SINGLE_TABLE) @DiscriminatorColumn(name = "REGISTRATIONTYPE", discriminatorType = DiscriminatorType.STRING),* com a finalidade de mapear todas as classes da hierarquia em apenas uma tabela na coluna denominada por "*REGISTRATIONTYPE*".

Melhorias que poderiam ser impostas

Como não foi possível modificar alguns aspetos da fase 3 para a fase 4, pois seria necessário modificar bastante código, referimos aqui alguns aspetos que poderiam ser melhorados, são estes:

- Criar uma classe *ActiveLesson*, de forma a distribuir a complexidade da classe *Lesson* para o caso de uso *Ativar Aula*;

- Criar uma classe *Timetable*, para gerir melhor as datas e horas das ocupações e aulas, respetivamente;

- Melhorar o código em termos de legibilidade, para que seja mais percetível a quem não conheça o projeto e caso haja uma nova iteração do mesmo.