Online Movie Ticket Booking System

Part 2

1. Introduction

The online movie ticket booking system is a web-based enterprise system that allows customers to book movie tickets and cinema manager to do order management. In this submission, we built the first feature in our proposal, which is building a movie management system. There are two roles in this feature, which are the cinema manager and customer.

- **Cinema Manager**

The cinema manager has the right to add, delete, edit and view movie information. Here, the movie information includes both the movie details and other movie related information, including cinemas, sessions and whether it's a 3D movie or not. The word 'session' represents the movie screening information, including when that session will start, when it will end and how many seats will be available.

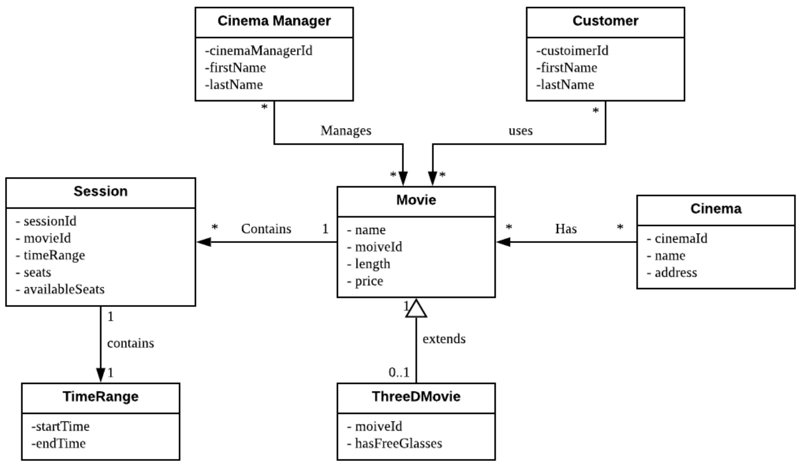
- **Customer**

Customer could view all movie information in the system. Customers could firstly get a list of all movies and they could view detailed information of one specific movie by clicking on the view button of that movie's row.

2. Use Cases

Here are the use case diagrams of both roles.

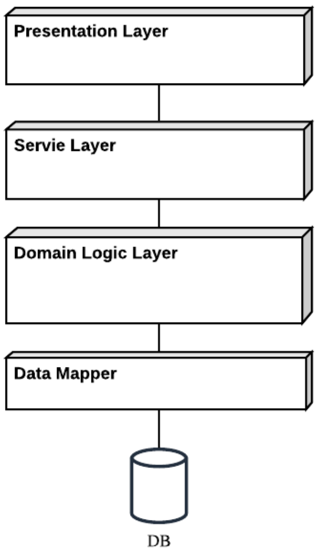
3. Domain Class Diagram & explanation



- **Cinema:** This class contains the information of a cinema.
- **Movie**: This class contains all information of a movie itself.
- **ThreeDMovie:** This class contains extra information of a 3D movie. It extends the Movie class. Some Movie object may have related ThreeDMovie object while some others may not. The inheritance relationship is class table inheritance, which means each class has one table and they share one sam primary key (movieId).
- **Customer:** This class contains all information of a customer.
- **Cinema Manager:** This class contains all information of a cinema manager.
- **Session:** This class contains all information of a session, and one session could only related with one movie object by the including movieId in attributes.
- **TimeRange:** This class contains all information of one time schedule, including the start time and end time. It is used by session object and one session object could only has one TimeRange object as one of its attributes.

4. High Level Architecture

The high level architecture diagram of the system could be shown as below:



Roles and Responsibilities of Layers'

Layer	Roles and Responsibilities
Presentation Layer	• Display Views and Information generated by controllers • Collect user input information • Transmit all business calls to the service layer
Service Layer	• Contains application-specific logic and handle with all application request • Involve multiple resources and actions and coordinate the application's response in each operation
Domain Logic Layer	• Contains the domain-specific logic which is independent of the system • Data access management
Data Mapper	• Provides the mapping between DataBase and Domain/Service Layer • Make the rest of application layers independent with the structure of DataBase

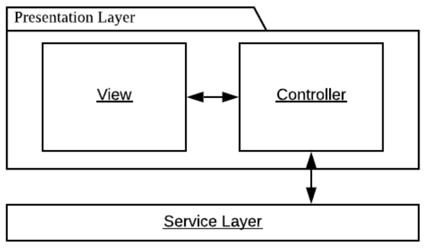
The component diagram of the system is shown as below:

Explanations

5. Presentation Layer

5.1. Pattern for Total Presentation Layer

There are two components included in the presentation layer, which are the Views and Controllers. The structure of presentation layer is shown as below.



•MVC structure pattern is applied here.

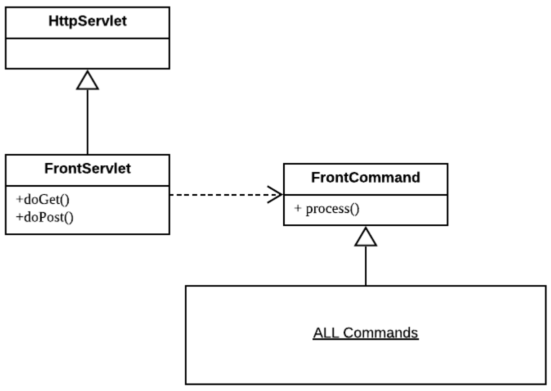
• Service Layer would provide domain information and handle with front-end requests.

• Controller will handles the interactions between the view and the model (domain objects).

•View will display information and collect user input through a user interface.

5.2 Pattern for Controller

Front Controller pattern is used on the controllers structure. The abstract structure of the controller is shown as below. Every time the FrontServlet gets a request from View, it will parse the url, and then it will create and use an appropriate command which will handle the request.



The reason why we applied Front Controller pattern is that:

- There is only one single entry controller, which will make both setting the configuration and implementation easier and more clear.
- As all commands inherent from FrontCommand class and different types of actions could be implemented separately, it's easier to extend this model by simply adding more command classes.

5.3 Actually Presentation Layer Structure Diagram

The actually presentation layer structure is like below

As the diagram's detail is hard to be seen clearly on this documentation, we provide the dropbox link so that the structure could be viewed clearly.

Dropbox Link:

https://www.dropbox.com/s/axxbxkzarc8l3v6/Presentation%20Layer%20Class%20Diagram.png?dl=0

5.4 Detail Responsibilities of all Commands

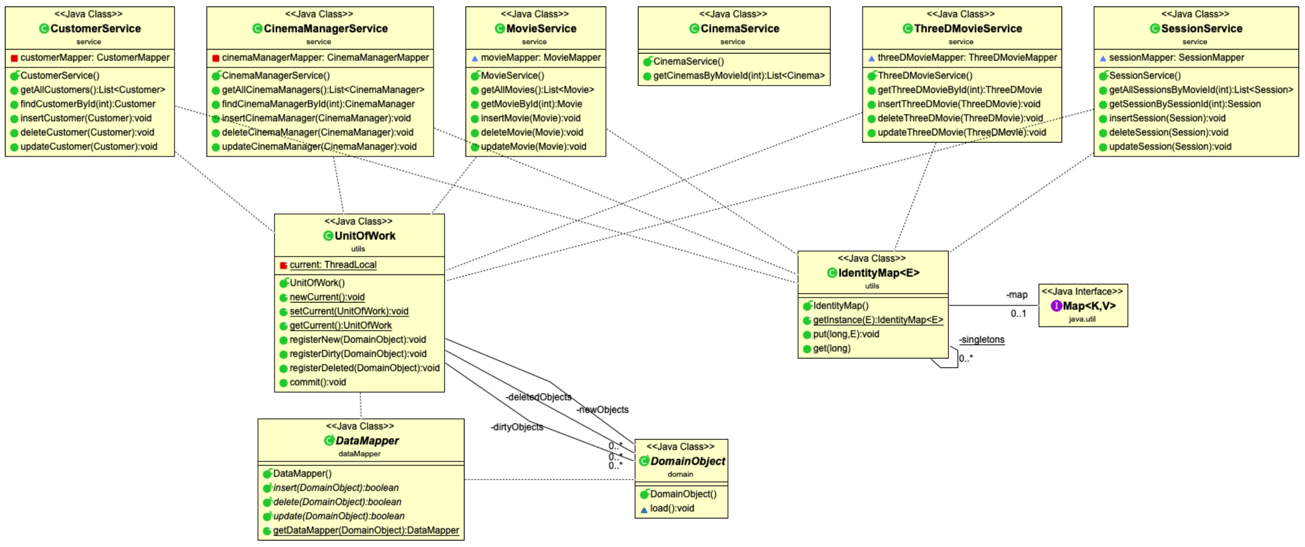
6. Service Layer

6.1 Service Layer General Design

The service layer is used to "Define an application's boundary" and "coordinates the application's response in each operation". Here **Operation Script** pattern is used to divide service layer with domain layer. According to this pattern's definition, domain layer will contain the domain-specific logic which is the domain itself and is independent of the system, like the type of Movie (3D or not). While the service layer contains the application-specific logic which will packages multiple domain classes, actions and resources and could be accessed by the presentation layer directly.

The reason why **Operation Script** pattern is applied is that it promotes re-use better than **Domain Facade** pattern. As the design and logic of the domain-specific logic could be reused by other systems and different actions could be implemented by simply replacing the application-specific logic into some others.

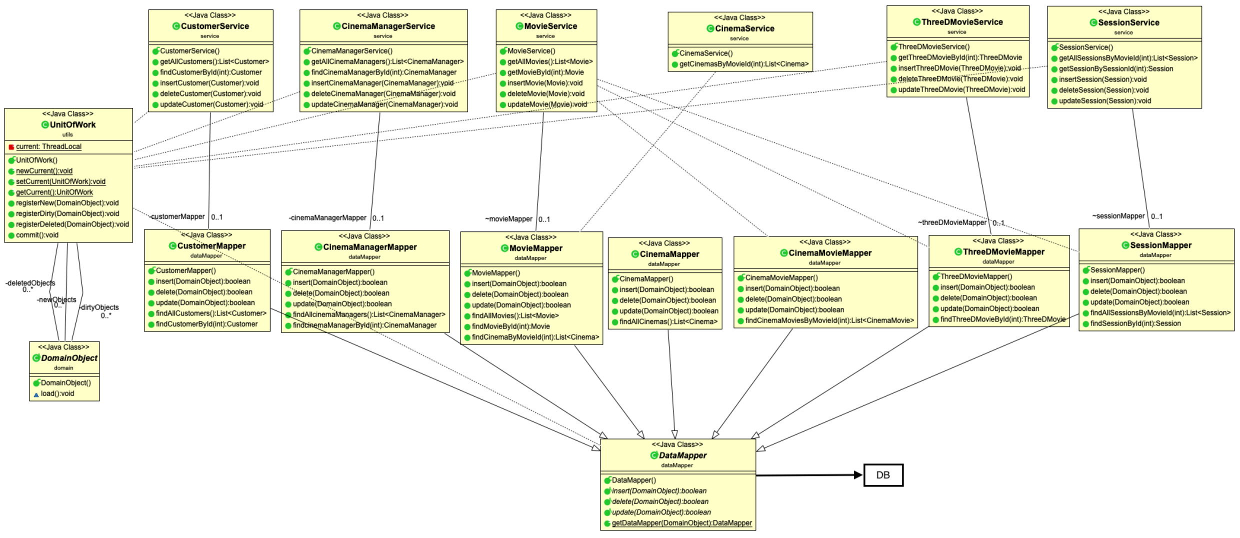
6.2 The Service Layer Class Diagram



6.3 Responsibilities of all Modules

6.4 Unit of Work

Unit of Work pattern is implemented and used by both Service Layer and Data Mapper Layer. The way it is used could be shown as below

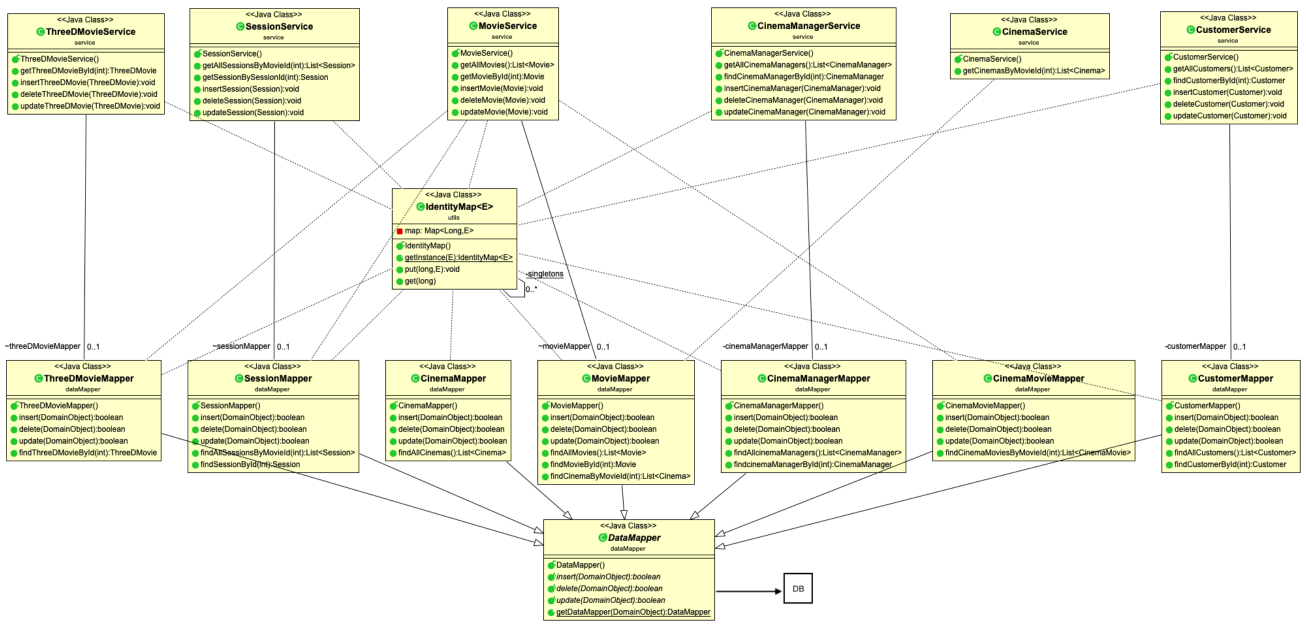


Explanations:
The unit of work pattern is used to maintain a list of objects that are affected by a business transaction. The reason why we applied it is that this pattern could make the DB connected operation more efficient, as it will tag all the objects changes in one business logic operation and commit all of them to the database together. For example, in the implementation of deleteMovie(Movie) function in MovieService Class, if the user want to delete one movie object, all related Session objects, Cinema_Movie objects and ThreeDMovie objects will be deleted as well. By using the Unit of Work pattern, the system will tag all related objects as "deleted" and commit all to the DB at the end of this method. It's an efficient and simple way to implement the business logic of deleting movie.

The way how UnitofWork achieves the commit operation is related with Data Mapper Layer and will be explained there. (The explanation of commit&DataMapper: UnitofWorkinDataMapperLayer)

6.5 Identity Map

Identity Map pattern is implemented and used by both Service Layer and Data Mapper Layer, the way it is used could be shown as below:


<<Java Class>>
ThreeDMovieService
service
ThreeDMovieService()
getThreeDMovieById(int):ThreeDMovie
insertThreeDMovie(ThreeDMovie):void
deleteThreeDMovie(ThreeDMovie):void
updateThreeDMovie(ThreeDMovie):void
<<Java Class>>
SessionService
service
SessionService()
getAllSessionsByMovieId(int):List<Session>
getSessionBySessionId(int):Session
insertSession(Session):void
deleteSession(Session):void
updateSession(Session):void
<<Java Class>>
MovieService
service
MovieService()
getAllMovies():List<Movie>
getMovieById(int):Movie
insertMovie(Movie):void
deleteMovie(Movie):void
updateMovie(Movie):void
<<Java Class>>
CinemaManagerService
service
CinemaManagerService()
getAllCinemaManagers():List<CinemaManager>
findCinemaManagerById(int):CinemaManager
insertCinemaManager(CinemaManager):void
deleteCinemaManager(CinemaManager):void
updateCinemaManager(CinemaManager):void
<<Java Class>>
CinemaService
service
CinemaService()
getCinemasByMovieId(int):List<Cinema>
<<Java Class>>
CustomerService
service
CustomerService()
getAllCustomers():List<Customer>
findCustomerById(int):Customer
insertCustomer(Customer):void
deleteCustomer(Customer):void
updateCustomer(Customer):void
<<Java Class>>
IdentityMap<E>
utils
map: Map<Long,E>
IdentityMap()
getInstance(E):IdentityMap<E>
put(long,E):void
get(long)
-singletons
0..*
~threeDMovieMapper
0..1
~sessionMapper
0..1
~movieMapper
0..1
-cinemaManagerMapper
0..1
-customerMapper
0..1
<<Java Class>>
ThreeDMovieMapper
dataMapper
ThreeDMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findThreeDMovieById(int):ThreeDMovie
<<Java Class>>
SessionMapper
dataMapper
SessionMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllSessionsByMovieId(int):List<Session>
findSessionById(int):Session
<<Java Class>>
CinemaMapper
dataMapper
CinemaMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCinemas():List<Cinema>
<<Java Class>>
MovieMapper
dataMapper
MovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllMovies():List<Movie>
findMovieById(int):Movie
findCinemaByMovieId(int):List<Cinema>
<<Java Class>>
CinemaManagerMapper
dataMapper
CinemaManagerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllcinemaManagers():List<CinemaManager>
findcinemaManagerById(int):CinemaManager
<<Java Class>>
CinemaMovieMapper
dataMapper
CinemaMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findCinemaMoviesByMovieId(int):List<CinemaMovie>
<<Java Class>>
CustomerMapper
dataMapper
CustomerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCustomers():List<Customer>
findCustomerById(int):Customer
<<Java Class>>
DataMapper
dataMapper
DataMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
getDataMapper(DomainObject):DataMapper
DB

Explanations:
By keeping loaded objects in a map, identity map could make the loading of one object only happens once. The advantages are:

- By only one loading and maintaining only one same object, clash of different operations could be prevent as there is only one object across all the system. In this way, concurrent accessing's safety could be guaranteed.
- Apart from that, it could also decrease the cost as duplicate objects loading will be prevented.

The way in which service classes use identity map is as follows:

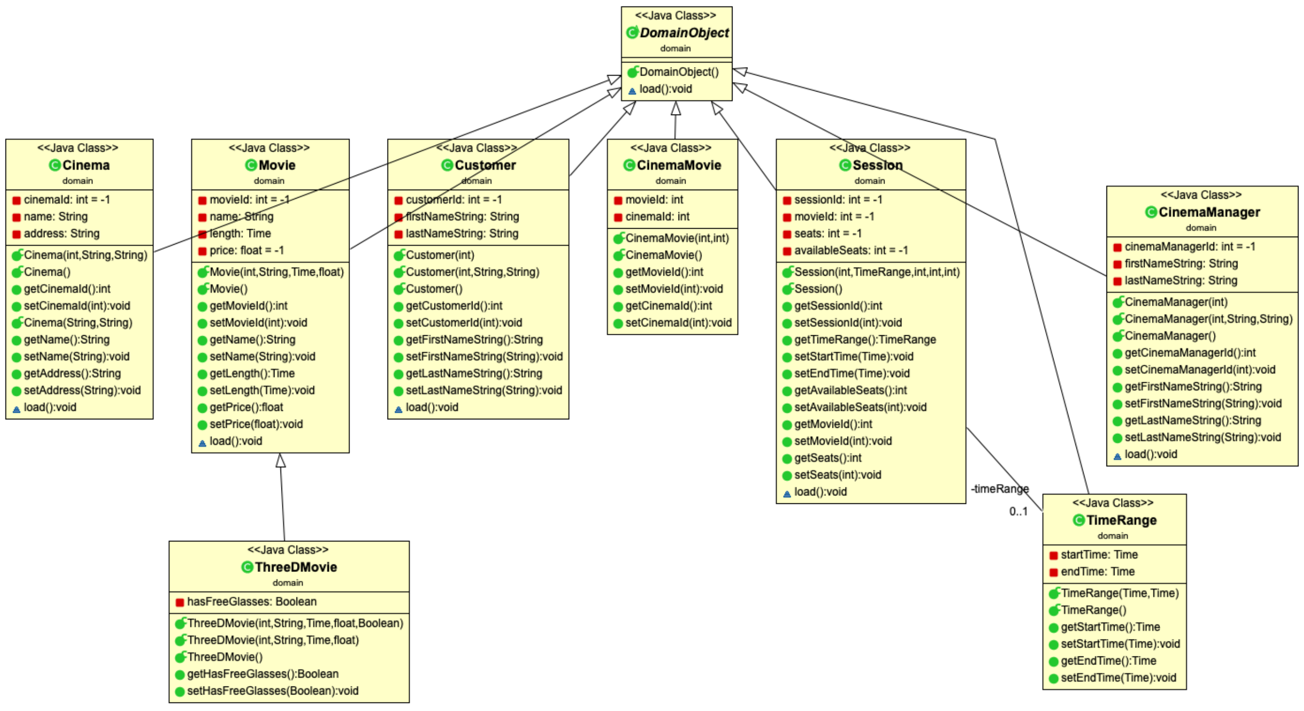
- Every time a view type of service is called, the service class will firstly check if the required object is already existed in identity.
- If so, it will load from the object from identity map and return it directly.
- If not, it will call related Data Mapper class and function, the function will load the object's information from database and assign the information to one object. That object will be put into identity map and then return. Thus, singleton is still maintained across all layers.

(As most identity map related description is listed here, there won't be any more duplicate explanation in the Data Mapper Layer section even it's also used by Data Mapper Layer).

7. Domain Layer

7.1 General Design

Multiple patterns are applied in the design of Domain Layer, including Lazy Load, Identity field, Foreign Key Mapping, Association Table Mapping, Embedded Value and Class Table Inheritance. The Domain Layer Class Diagram is as follows:



Explanations:

- All classes in Domain Layer extend DomainObject class, which could make the design more clear and clean, and it's also necessary for UnitofWork implementation.
- As Operation Script devision principle is used, the domain later only contains domain-specific logic which is independent with the system. All application-specific operations (like deleting, inserting and updating) will only be implemented in the Service Layer.

7.2 Responsibilities of All Classes:

7.3 Lazy Load:

Ghost pattern is used in the many domain layer classes, which is easy to implement and could decrease the times needed to access the database. The way in which it achieve the outcome is that every time instead of query only the required field at one time, the pattern will force the object to query all fields it has, and the next some other attributes of that object is required, there is no needs to query again as it has already been loaded. The implementation of Lazy Load increases the efficiency of accessing data and all related operations.

7.4 Object-to-relational structural design

Multiple patterns of object-to-relational structural design are implemented in the system, including Identity field, Foreign Key Mapping, Association Table Mapping, Embedded Value and Class Table Inheritance. Here are some details.

7.4.1 Identity Field

This pattern is used between many classes and database tables, for example, the movidId field in Movie Class in Domain Layer represent the primary key of Movie table in database. The advantages of this pattern is that it is simple to implement and every time we could easily identify the corresponding row of domain objects using the identity field.

7.4.2 Foreign Key Mapping

This pattern is used to Maps an association between objects to a foreign key reference between tables. It is implemented in our system as well. For example, the movieId field in Session class represents the foreign key relationship between Session table and Movie Table in database. The reason why we implement this pattern is that it's efficient for representing one-to-many relationship in database, and the mapping between the domain relationships and database tables could be done simply and mechanically.

7.4.3 Association Table Mapping

This pattern is used to save the many-to-many relationship between database tables. It is implemented in out system as well. For example, the relationship between Cinema table and Movie table in database is many-to-many, as a result, an association table called Cinema_Movie is created in the database and the table is used in the system for searching all cinemas that could display one specific movie. The advantage of this pattern is that it's necessary and simple to represent many-to-many relationship in database and the mapping between the domain relationships and database tables could also be done simply and mechanically.

7.4.4 Embedded Value
This pattern is used to map one table's information into several objects in the Domain Layer when some object is more suitable to be represented as several classes in Domain Layer but not necessary to be stored separately in the database. Our system implementing this pattern by the Session class and TimeRange class in Domain Layer, which is stored as one table (Session) in the DB. The reason why we do this is that the TimeRange is quite different with other attributes of session object and it's reasonable to define it separately as a domain object. But in the database table, we usually collect time range information and other session information together, especially after implementing lazy load pattern on Session class. The advantage of this pattern is that it could make the relational database and related domain objects neater.

7.4.5 Class table inheritance
This pattern is used to represent an inheritance hierarchy of classes with one table for each class. In our system, it is used between Movie and ThreeDMovie classes and Movie and ThreeDMovie table. The detailed information is as follows:

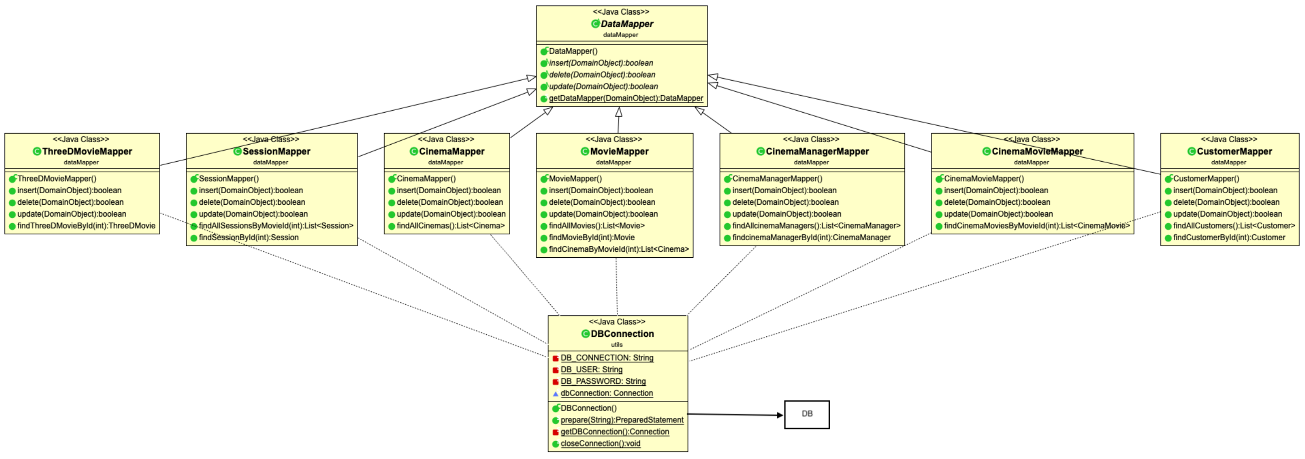
#diagram here

The reason why we use this inheritance is that it's easier to implement than concrete table inheritance and the table design is clear and without any wasted space (which is one advantaged compared with single table inheritance).

8. Data Mapper Layer (Data Source Layer)

8.1 General design
Data mapper pattern is used in the data source layer. The significant advantage of applying data mapper pattern is that it could keep other layers and objects independent with the database. All these layer could behavior as they don't know the existence of database. Apart from that, Unit of Work (will be explained later) and Identity Map (Already explained in Service Layer Section) are also used in the Data Mapper Layer to optimize the system's performance.

The structure of Data Mapper Layer is as follows:


<<Java Class>>
DataMapper
dataMapper
DataMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
getDataMapper(DomainObject):DataMapper
<<Java Class>>
ThreeDMovieMapper
dataMapper
ThreeDMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findThreeDMovieById(int):ThreeDMovie
<<Java Class>>
SessionMapper
dataMapper
SessionMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllSessionsByMovieId(int):List<Session>
findSessionById(int):Session
<<Java Class>>
CinemaMapper
dataMapper
CinemaMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCinemas():List<Cinema>
<<Java Class>>
MovieMapper
dataMapper
MovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllMovies():List<Movie>
findMovieById(int):Movie
findCinemaByMovieId(int):List<Cinema>
<<Java Class>>
CinemaManagerMapper
dataMapper
CinemaManagerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllcinemaManagers():List<CinemaManager>
findcinemaManagerById(int):CinemaManager
<<Java Class>>
CinemaMovieMapper
dataMapper
CinemaMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findCinemaMoviesByMovieId(int):List<CinemaMovie>
<<Java Class>>
CustomerMapper
dataMapper
CustomerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCustomers():List<Customer>
findCustomerById(int):Customer
<<Java Class>>
DBConnection
utils
DB_CONNECTION: String
DB_USER: String
DB_PASSWORD: String
dbConnection: Connection
DBConnection()
prepare(String):PreparedStatement
getDBConnection():Connection
closeConnection():void
DB

All mapper classes extends DataMapper Class, which could make the design more clear and also essential for implementing UnitofWork Pattern.

8.2 Responsibilities of mapper classes:

8.3 Unit of Work

The UnitofWork class use the Data Mapper Layer in its commit function.

// TODO