Vanessa Ciputra
804079209

SOFTENG 251 – Assignment 2
Domain Model Concepts + Design Justifications

The problem domain of this assignment revolves around a Theatre Booking System (TBS) development that stores information of artists, acts and performances that can be utilised by the user to access information and interact with the server. This is achieved by using an implementation of the interface TBSServer. From this, I obtained the following concepts that were required to accommodate the requirements of the problem:

- Theatres
- Seats
- Seats Availability
- Tickets Availability
- Tickets Issued
- Artists
- Acts
- Performance
- Prices/Fees
- Times/Schedule
- Sales

I linked all the concepts into blocks of relating information, producing:

- Tickets Availability/Tickets Issued → Performance → Seats/Seats Availability → Theatre
- Sales → Fees → Tickets → Performance → Theatre
- Artist → Acts → Prices → Theatre → Performance

As shown above, the recurring concepts are **Theatre** and **Performance**, which make up the following *main concepts* that link to all the other concepts:

- Performances – all details that are stored in the server, including Theatres, Artists, Acts, Seats, Schedules, Fees, lead to creating a specific performance which tickets are sold for. The user interaction will always lead up to accessing their chosen specific performance, and therefore it encompasses most of the other concepts in terms of information in one title.

- Theatre – the Theatre concept is important in the sense that it accommodates the practical technicalities of the implementation, including Seats, Seats Availability, Tickets and Tickets Availability to further represent Sales. It is required to maintain availability as well as accessing errors in terms of user interaction, such as issuing a previously bought ticket or an undefined seat. Information about tickets is also mainly linked to Theatres as the Seats information is also linked, and therefore the overall report of Sales, which encompasses Seats and Tickets, is stemmed from the Theatre concept.

In my design, the two main concepts above are represented in their own separate classes (Performance, Theatre), while I have also included three other classes which I define as sub-concepts (Artist, Act, Tickets), which in turn consists of their separate fields and methods. Each class consists of getters and setters – setters establish values of field that is stored in the particular instance of the object which are obtained from the TBSServerImpl class, and getters are called in the TBSServerImpl class as well as other classes to obtain particular fields and states when necessary.

Vanessa Ciputra
804079209

What separates the main classes from the sub-concept classes is that the main concepts also include methods that determine and calculate their own fields to provide *abstraction* without total reliance on the TBSServerImpl class, as exemplified by the getSeating() method in the Theatre class and the getPrices() method in the Performance class. The sub-concept classes mainly store the information on the server for the TBSServerImpl class to retrieve when necessary as I use them as pieces of information, albeit required for functionality, that make up the main concepts with abstraction. In the final methods in the TBSServerImpl class (such as schedulePerformances, issueTickets, and salesReport), the methods mainly focus on the Performance and Theatre classes as every other concept and information, even sub-concepts, have been stored in some way into the main classes. Therefore, the main classes of Performance and Theatre are standing foundations in my design to implement the interface of TBSServer, encapsulating the rest of the classes, concepts and methods.

In terms of my overall design, each class is adapted to ensure that they somehow link to at least one of the main concept classes, including the main classes themselves, with use of creating instances of objects of the classes in TBSServerImpl to link them all together. This allows versatility between classes and prevents the risk of detachment between the fields and responsibilities of the classes. Furthermore, the easy access from the instances of objects allows a quick retrieval of information for particular fields. Since each class has a unique ID associated with them, matching the ID with the specified input will allow retrieval of any piece of information in the class using specific getter methods.