Jack Burns

Requirements

User Cases

Domains

Relations

Constraints

Module Dependency Diagram

UML Diagram

Problem

Your goal is to design a Java application that will manage the sales of movie tickets in a cinema complex. The complex consists of several theaters that show movies at different times. The same movie may be showing in more than one theater.

The ticket prices may be different for children, adults, and seniors. There may also be a different price for the matinees.

The cinema complex wants to be able to analyze its sales by various criteria - tickets sold to seniors, tickets for a specific movie, matinee ticket sales, etc.

Some cinema complexes may also have luxury theaters with higher fixed price for all patrons.

Requirements

The system shall allow the user to update the sales data for any valid movie/time/theater combination.

Sales data will be categorized and priced in accordance to the status of the ticket purchaser (e.g. child, senior, adult) and can accommodate any number of different pricing amounts.

The means by which the user may update sales information is the Console.

The system may be comprised of any number of theaters and movies shown at various times.

A movie may be shown in multiple theaters at the same time.

The user may analyze its sales by querying the system for data

Use Cases

Use Case 1: Update Sales (Console)

• User enters movie, theater, time, number of child tickets, number of adult tickets, number of senior tickets

• The system processes request and updates sales log and system data

Use Case 2: Querying for Information (Console)

• User queries for a given data set (e.g. how many tickets for X movie)

• The system returns report based off query

Use Case 3: Update Sales Error (Console)

• User enters an incorrect set of data (e.g. more tickets sold then seats exist in the theater)

• The system updates log and ignores data

Use Case 4: Querying for Information Error (Console)

• User queries for a nonexistent data set

• The system updates log and ignores data

Use Case 5: Update Price data overlap (Console)

• User enters Price data for category that already exists

• The system will update the already established category with new amount

Domains:

User Interface: Input module

Reports results of queries

Data Manager: Module that keeps track of the data, verifies its integrity, processes data updates

Relations:

User Interface: Interacts with the user, provides selection of specific theater/movie/time combinations to update sales data for the set combination. Also allows the user to update any of the theater/movie/time combinations that will then be reflected in the User Interface sales update selection process. Interacts with data manager when user wants to query for specific data.

Data Manager: Handles all information about the cinema complex used for sales calculations theaters, schedules, movies, and other characteristics. Is responsible for integrity of data (e.g. theater can’t be showing two movies at the same time). Interacts with User Interface to report back based off user query.

Constraints:

The cinema complex must be designed in such a way that all movies screenings have a time and a theater. No theater can air two movies at the same time. Multiple theaters can play the same movie.

Module dependency diagram:

+--------------------+ +--------------------+

| User Interface |<------>| Data manager |

+--------------------+ +--------------------+

Data Model:

**Movie**: Represents a movie

String Movie.name: Name of the Movie

Int Movie.length: Length of the Movie

**Price**:

String Price.cat: Price category title

String Price.amount: Amount of $

**Theater**: Represents a theater

String Theater.name: Name of Theater

Int Theater.capacity: Number of seats in this Theater

ArrayList<Show> Theater.shows: The Shows playing in this theater

**Show**: Represents a Show

Int Show.movie: Movie index

Int Show.theater: Theater index

Int Show.time: Tme show begins

Int Show.length: Length of movie being aired

ArrayList<Integer> Show.sales: Tracks sales

**TicketSales**: Represents the Cinema complex/TicketSeller

ArrayList<Theater> TicketSales.theaters: The theaters in the cinema

ArrayList<Movie> TicketSales.movies: The movies airing in the cinema

ArrayList<Price> TicketSales.prices: The prices used in the cinema

ArrayList<String> TicketSales.sales: Accumulate the sales log data

ArrayList<String> TicketSales.log: Accumulate the sales log data

String TicketSales.manager: Accumulate manager log

Int reportRequests: Tracks number of report requests

UML Diagram:

