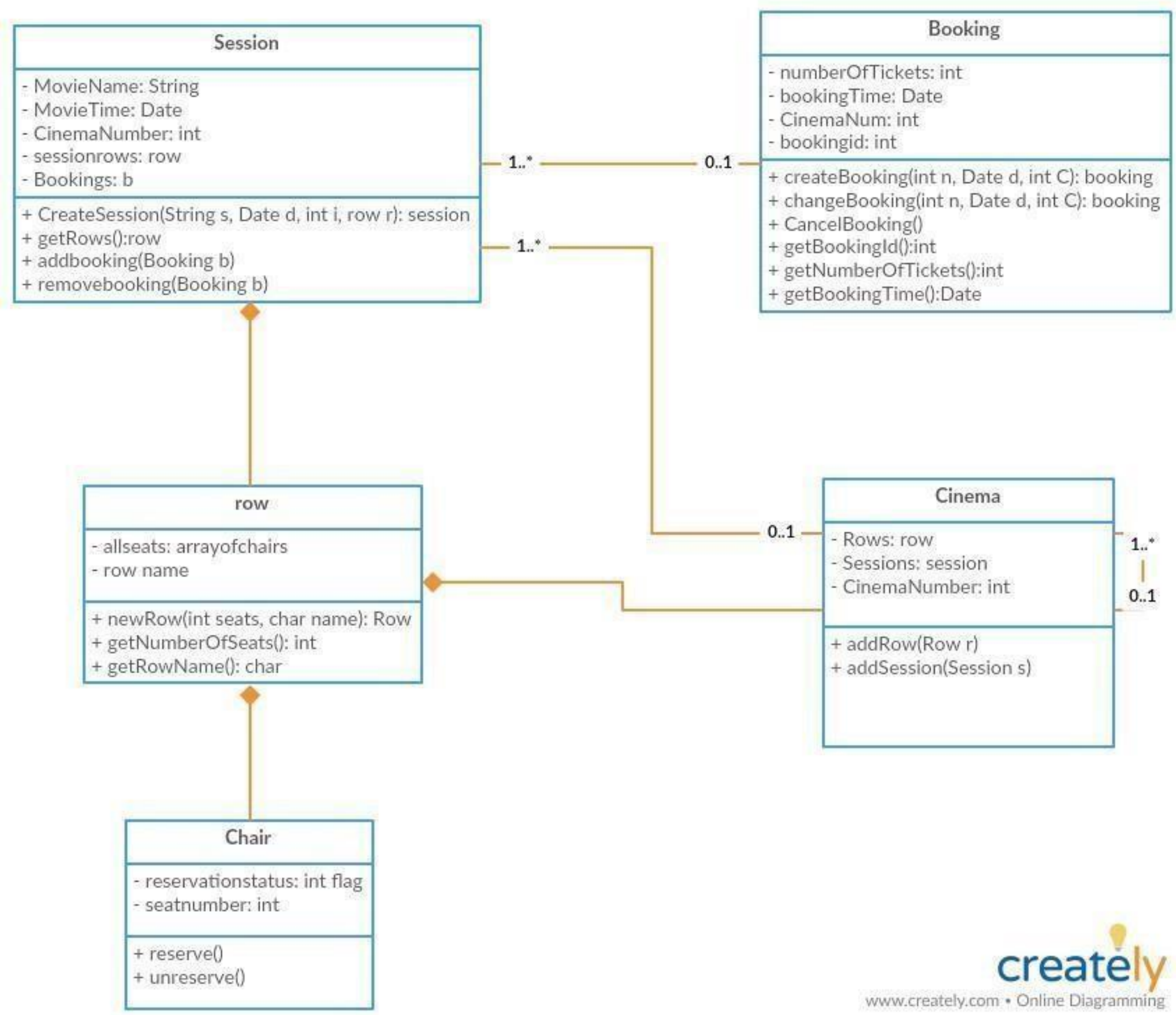
Table of Contents

UML Class Diagram

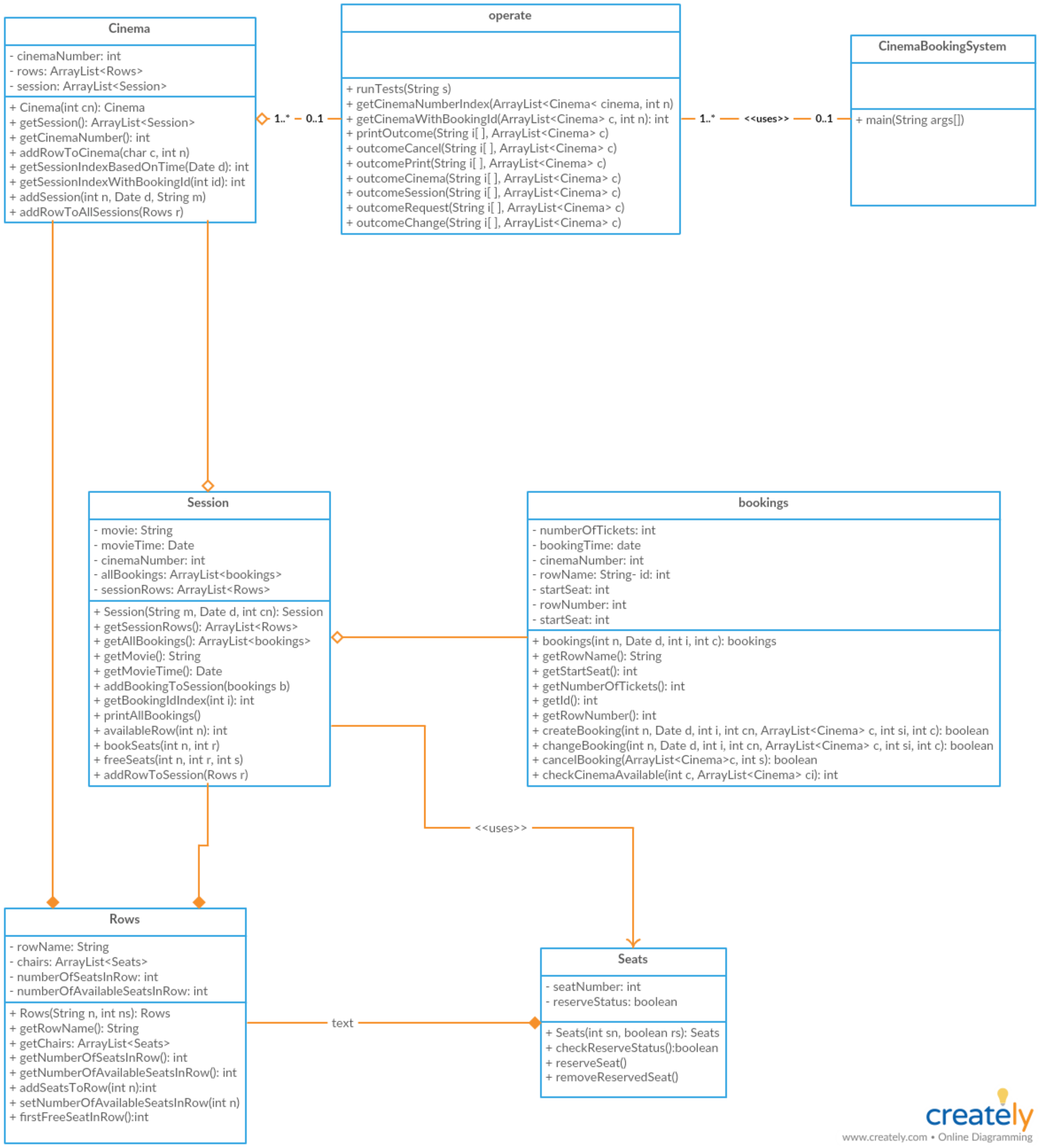
First Diagram Before Any coding done

My first UML Class Diagram I made before I started the project which was to try give me a basic idea of how I want to setup my design and relate all my classes, This was made before I started coding and it ended up being a similar design just needed a lot of more functions and also changed some attributes and added some as I went, however this shows how I first thought I wanted to setup the system



Final UML diagram now that entire project is finished

This was edited and fixed up as I made changes to the project, this is the end result after I have finished my entire project, I realised I had to redesign some things and my issue was I visualised a cinema as an object rather than a real thing, when I imagined a cinema as a bunch of theatres with each theatre having a bunch of sessions and people making bookings for a session it clicked with me and I redesigned a lot of things. I also realised I needed a lot more functions than I thought I needed to help navigate


Cinema
- cinemaNumber: int
- rows: ArrayList<Rows>
- session: ArrayList<Session>
+ Cinema(int cn): Cinema
+ getSession(): ArrayList<Session>
+ getCinemaNumber(): int
+ addRowToCinema(char c, int n)
+ getSessionIndexBasedOnTime(Date d): int
+ getSessionIndexWithBookingId(int id): int
+ addSession(int n, Date d, String m)
+ addRowToAllSessions(Rows r)
1..*
0..1
operate
+ runTests(String s)
+ getCinemaNumberIndex(ArrayList<Cinema< cinema, int n)
+ getCinemaWithBookingId(ArrayList<Cinema> c, int n): int
+ printOutcome(String i[ ], ArrayList<Cinema> c)
+ outcomeCancel(String i[ ], ArrayList<Cinema> c)
+ outcomePrint(String i[ ], ArrayList<Cinema> c)
+ outcomeCinema(String i[ ], ArrayList<Cinema> c)
+ outcomeSession(String i[ ], ArrayList<Cinema> c)
+ outcomeRequest(String i[ ], ArrayList<Cinema> c)
+ outcomeChange(String i[ ], ArrayList<Cinema> c)
1..*
<<uses>>
0..1
CinemaBookingSystem
+ main(String args[])
Session
- movie: String
- movieTime: Date
- cinemaNumber: int
- allBookings: ArrayList<bookings>
- sessionRows: ArrayList<Rows>
+ Session(String m, Date d, int cn): Session
+ getSessionRows(): ArrayList<Rows>
+ getAllBookings(): ArrayList<bookings>
+ getMovie(): String
+ getMovieTime(): Date
+ addBookingToSession(bookings b)
+ getBookingIdIndex(int i): int
+ printAllBookings()
+ availableRow(int n): int
+ bookSeats(int n, int r)
+ freeSeats(int n, int r, int s)
+ addRowToSession(Rows r)
bookings
- numberOfTickets: int
- bookingTime: date
- cinemaNumber: int
- rowName: String- id: int
- startSeat: int
- rowNumber: int
- startSeat: int
+ bookings(int n, Date d, int i, int c): bookings
+ getRowName(): String
+ getStartSeat(): int
+ getNumberOfTickets(): int
+ getId(): int
+ getRowNumber(): int
+ createBooking(int n, Date d, int i, int cn, ArrayList<Cinema> c, int si, int c): boolean
+ changeBooking(int n, Date d, int i, int cn, ArrayList<Cinema> c, int si, int c): boolean
+ cancelBooking(ArrayList<Cinema>c, int s): boolean
+ checkCinemaAvailable(int c, ArrayList<Cinema> ci): int
<<uses>>
Rows
- rowName: String
- chairs: ArrayList<Seats>
- numberOfSeatsInRow: int
- numberOfAvailableSeatsInRow: int
+ Rows(String n, int ns): Rows
+ getRowName(): String
+ getChairs: ArrayList<Seats>
+ getNumberOfSeatsInRow(): int
+ getNumberOfAvailableSeatsInRow(): int
+ addSeatsToRow(int n):int
+ setNumberOfAvailableSeatsInRow(int n)
+ firstFreeSeatInRow():int
text
Seats
- seatNumber: int
- reserveStatus: boolean
+ Seats(int sn, boolean rs): Seats
+ checkReserveStatus():boolean
+ reserveSeat()
+ removeReservedSeat()
creately
www.creately.com • Online Diagramming

Project Design

My project intended to have a operate class which handles all requests from input files, this would operate as the backend of the cinema system. My cinema system itself works in a way such that each cinema is created by a Cinema request, it either adds a row to the cinema or creates the cinema with the row. My Cinema would contain a cinema number, a list of all sessions in that cinema and a list of rows.

Since sessions can be thought of an an instance of that cinema, so in real life if you go to see a session of the movie, you visit the cinema at a specific time to view it, it would only make sense to make sure that in every situation the session rows would correspond to the cinema rows, i.e same number of rows and seats. A session contains a list of bookings for that session, the rows for that session. Since a session is an instance of the cinema the rows in the session can't be the same for each session and have to be treated differently for each session, each session initially has the same row structure as the cinema it's in, however when bookings are made, seats get reserved in rows for that session, hence why I made the session rows deep copies of the cinema rows so when I change my session rows it wouldn't effect every other session and the cinema's rows. A session also contains a movie name, and a time which the movie is showing.

A booking is a reservation of seats for a certain Session. Bookings contain the first seat in the booking, the number of tickets in the booking, the cinema number its showing in and a booking id, and which row the booking is located. When creating/deleting bookings, we want to keep track of which row the booking is in to know where our booking is, we also want to keep track of the starting seat to know where we are going to book or free seats, from start seat -> start seat + number of tickets.

Rows are a list of Seats simply, they also have an identifier which is their row name.

Seats are the core of the entire project, they will have a reservation status and seat number, all bookings will be made based on the number of unreserved seats in a row.

The operate class will use all previous classes in combination in order to handle user requests from input files, it will read in all the input from a text file line by line. The comments '#' will be separated from the request, which can be any request. Only requests with "Cinema" "Session" "Request" "Change" "Cancel" "Print" will be processed, the others will be ignored. These requests will perform different things which are listed below in the walkthroughs. The operate class will read every line and be able to handle any input and will cease at the end of file input.

Walkthrough

For Cinema Request

1. Operate class reads input line by line
2. Operate class separates the line from comment
3. Operate class reads the first word in the separated string, and checks if it is "Cinema"
4. If it isn't go to other checks or skip
5. Operate class checks if the input is valid by parsing the input, checks if row is string
6. Operate class checks if cinema number is an integer
7. Operate class checks if number of seats is >0 and is integer
8. Operate class checks if the cinema number exists in the list of cinemas
9. If the cinema number doesn't exist it will go to the Cinema class with the input
10. Cinema class would create the Cinema with the Cinema Number
11. Cinema class will then add the row to that cinema (and all existing sessions)
12. If the cinema number exists already
13. Cinema class will add that row to the cinema and all sessions in that cinema with the cinema number

For Print Request

1. Operate class reads input line by line
2. Operate class separates the line from comment
3. Operate class reads the first word in the separated string, and checks if it is "Print"
4. If it isn't go to other checks or skip
5. Operate class will check if the input is valid by parsing the input, checks if the cinema number is an int
6. Operate class will also check if the session time is date formatted
7. If it isn't skip
8. Operate class will then find the correct session in that cinema with the cinema number, to display information for
9. The session class will print the name of the movie in that session with a colon and a new line
10. The session class will then print all bookings in the session in the correct format

For session Request

1. Operate class reads input line by line
2. Operate class separates the line from comment
3. Operate class reads the first word in the separated string, and checks if it is "Session"
4. If it isn't go to other checks or skip
5. Operate class will check if the input is valid by parsing the input, checks if the Time of the session is a validly formatted date in HH:MM
6. Operate class will check if cinema number is an integer
7. Operate class will then take the rest of the separated part of the line as movie name
8. Cinema class will create a new session and add it to the cinema with the cinema number

For “Request” requests (Booking create)

1. Operate class reads input line by line
2. Operate class separates the line from comment
3. Operate class reads the first word in the separated string, and checks if it is “Request”
4. If it isn’t go to other checks or skip
5. Operate class will check if the input is valid by parsing the input, checks if the Time of the session is a validly formatted date in HH:MM
6. Operate class will check if the cinema number is an integer
7. Operate class will check if the booking id is an integer
8. Operate class will check if if the number of tickets is an integer > 0
9. Operate class will try to find the cinema with that cinema number (if it doesn’t exist booking rejected)
10. Operate class will then try to find the session with that time (it it doesn’t exist booking rejected)
11. Operate class will then attempt to make a booking for that session with the number of tickets requested
12. The Session class will try to find the first free row in that session which can hold the number of tickets requested
13. if no row-> return false for create booking booking rejected
14. if the row exists we will now try to find the first free seat in the row (helps us keep track of bookings)
15. The Session class will then book the seats for that booking from the start seat -> start seat + number of tickets.
16. The booking class will then update all its information to keep track of the booking
17. If any point any part of creating booking failed, returns false and booking rejecgted
18. Otherwise print out the booking : booking id startsteat-finishseat

For Change Request

1. Operate class reads input line by line
2. Operate class separates the line from comment
3. Operate class reads the first word in the separated string, and checks if it is “Request”
4. If it isn’t go to other checks or skip
5. Operate class will check if the input is valid by parsing the input, checks if the Time of the session is a validly formatted date in HH:MM
6. Operate class will check if the cinema number is an integer
7. Operate class will check if the booking id is an integer
8. Operate class will check if if the number of tickets is an integer > 0
9. Operate class will then check if the previous booking exists
10. Operate class will then attempt to change the Booking
11. Session class will keep track of the old booking information
12. Session class will then free the old seats of the booking (incase booking change is in same cinema)
13. Session class will then attempt to create a new booking in that cinema (keeping in mind the booking we are trying to change had it’s seats freed)
14. If there is no row we will recreate the booking we freed seats for (change rejected)
15. If at any point we can’t create new booking re-assign old seats -> change rejected

16. Session class will create the new booking with same id in the new session
17. Booking class will update the booking information from the old booking to the new booking
18. Session class will remove the old booking from the old sessions booking list
19. Session class will add the new booking to the new sessions booking list
20. Return true if successful
21. Operation class will output, change + booking id + startseat-startseat+numberoftickets
22. (will output Change rejected if any conflictions during change requested)

For Cancel Request

1. Operate class reads input line by line
2. Operate class separates the line from comment
3. Operate class reads the first word in the separated string, and checks if it is “Request”
4. If it isn’t go to other checks or skip
5. Operate class will check if the input is valid by parsing the input, checks if the booking id is a valid integer
6. Operate class will then check if the booking exists in any cinema with the booking id
7. If it doesn’t exist -> cancel rejected
8. If it does it will find the cinema with the cinema number
9. Operation class will find the session with the booking id
10. Operation class will then attempt to cancel i.e remove that booking from booking list in that session
11. The session class will free the seats in the booking from start->number of tickets for that booking in that row
12. The session class will then remove that booking from the list of bookings
13. The session class will return true if successful or false if unsuccessful
14. Will output Cancel + booking id cancelled if successful
15. If at any point failure or return false will output Cancel rejected

Explaining how bookings work

Boolean return -> true – successful/ false - unsuccessful
Bookings are created in the booking class, it will use the session class since all bookings are added to session and effect session rows directly

Bookings class will check if a cinema with the cinema number exists in the list of cinema first
If there is no cinema with that cinema number return ->false, otherwise continue
Bookings class will call session class to check if there is an available row to create booking for
If there is no available row return -> false, otherwise continue
Bookings class will then call the row class to find the first free seat in the first available row
Bookings class will then call the session class to book the seats from the start seat to start seat + number of tickets

Bookings class will then keep track of all the booking info which includes

- The row number (which rows appears from input file)
- The row name
- The first seat for the bookings
- The bookings id
- The number of tickets for the bookings
- The cinema for the booking

Explaining how Change works

Boolean return -> true – successful/ false - unsuccessful
Changes effect bookings so they are performed in the Bookings class
First it will gather old booking information index for the cinema and session in their arraylist
It will then check if the old booking even existed in first place by checking row name or start seat since they are -1 before being created so if -1 return false no booking exists
Then it will check if there is a cinema for the change request, if the cinema requested for change doesn't exist return false
Then it will free the old bookings seats
It will then check if there is a row in the cinema for the changed booking
If there is no row, rebook the seats we freed and return false
Otherwise it will create the new booking in the same way as described above
After new booking is created, it will delete the old booking from the bookings list in the session
In a change request we add this new booking to the new session with new info
Update the booking information same as above

Explaining how Cancel works

Boolean return -> true – successful/ false - unsuccessful
Cancel effect bookings so they are performed in the bookings class
First it will find the cinema with the session that has the booking id
If the booking id doesn't exist return false
If the row number/start seat == -1 return false
Then we want to simply free the seats
Then we want to remove the booking from the bookings list in the session