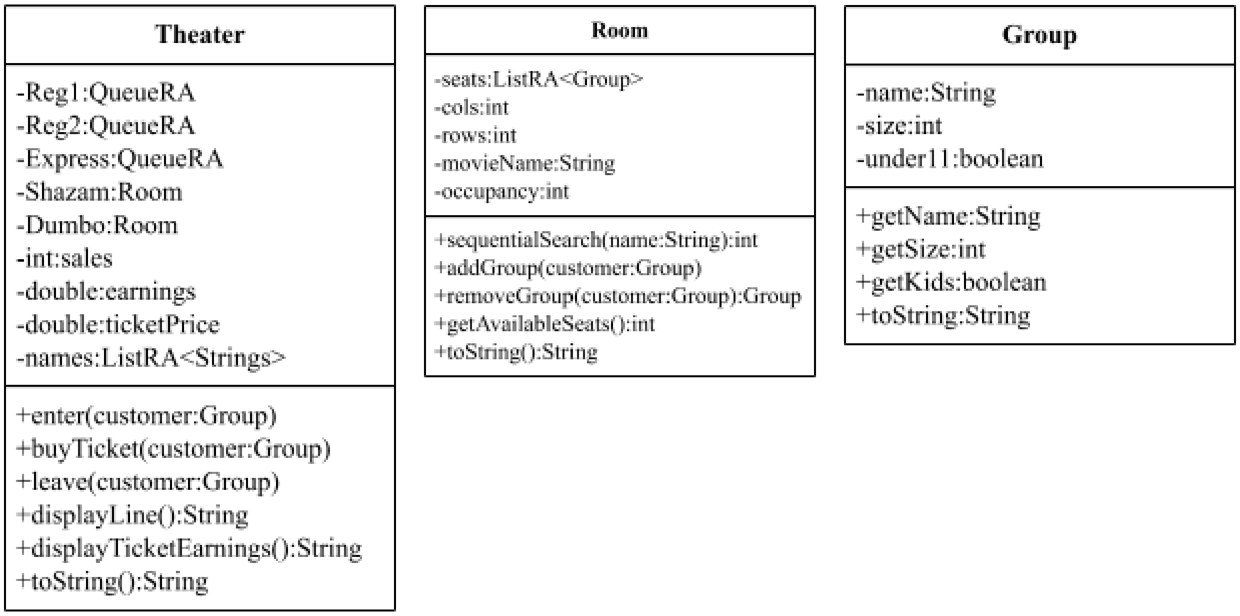
Project Write-up
4/29/2019
Matthew Tam & Chris Ancheta

The most frequent operation we expect to be performed during the entire execution of the program would probably be comparison operators when searching lists for existing customers. Because a lot of what the program is tasked with doing revolves a lot around inventory management and the handling of customers, we felt that the bulk of action items performed would include these operations. As such, we've decided to utilize a binary search as it is the most efficient solution that we know to implement. Since searching lists are our most frequent operation, it is important that we attempt to keep the searching algorithm as efficient as possible.

The ADTs that we decided to use are as follows:

- Queue using a Circular Resizable Array:
  - We plan on using these for processing each of the three lines of customers. Because they enter and leave in a queue and because we didn't want to waste cycles iterating a node structure to obtain information about the customers waiting in line, we decided that a queue using a circular resizable array would be most appropriate as well as the concerns mentioned previously regarding frequent operations.
- List using a Resizable Array:
  - We plan on using an array based list in our program to hold the list of groups currently present inside each movie as well as another list of strings that hold the unique names of each group of customers that are present inside the establishment. We chose to use a seperate master list because we felt it would be less time efficient if we were to search through each theater for a unique name. The possibility of running comparisons through each movie's list of groups seemed to expensive for what Chris and I determined was too simple a problem to waste the the energy on. Instead, we opted to sacrifice space in exchange for speed. Both lists we decided should be resizable because for movie occupancy instead of checking element in a list to determine occupancy, we will use math and calculate how much space is left. In that way, we will save a lot of space and time by not searching and running comparisons through a list as well as determining which elements within the list are empty. The master list belonging to the theater is resizable for the same reason that queues are resizable.

For this project we have designed four classes. The first class is Group, this stores a name, the number of people in the group, and if there are children under 11 years old present. This is all information needed in order to place customers in the appropriate line as well as determine if they can fit in one of the theaters. The second class is Room, this will function as a theater room for the showing movies. Each room has a number of seats defined at the start of the program. A Room can assign and unassign seats and provide information about how many seats are occupied. The last class is Theater. A Theater welcomes guests and places them in the appropriate line, and sells tickets. Theater is also what maintains that all group names are unique.

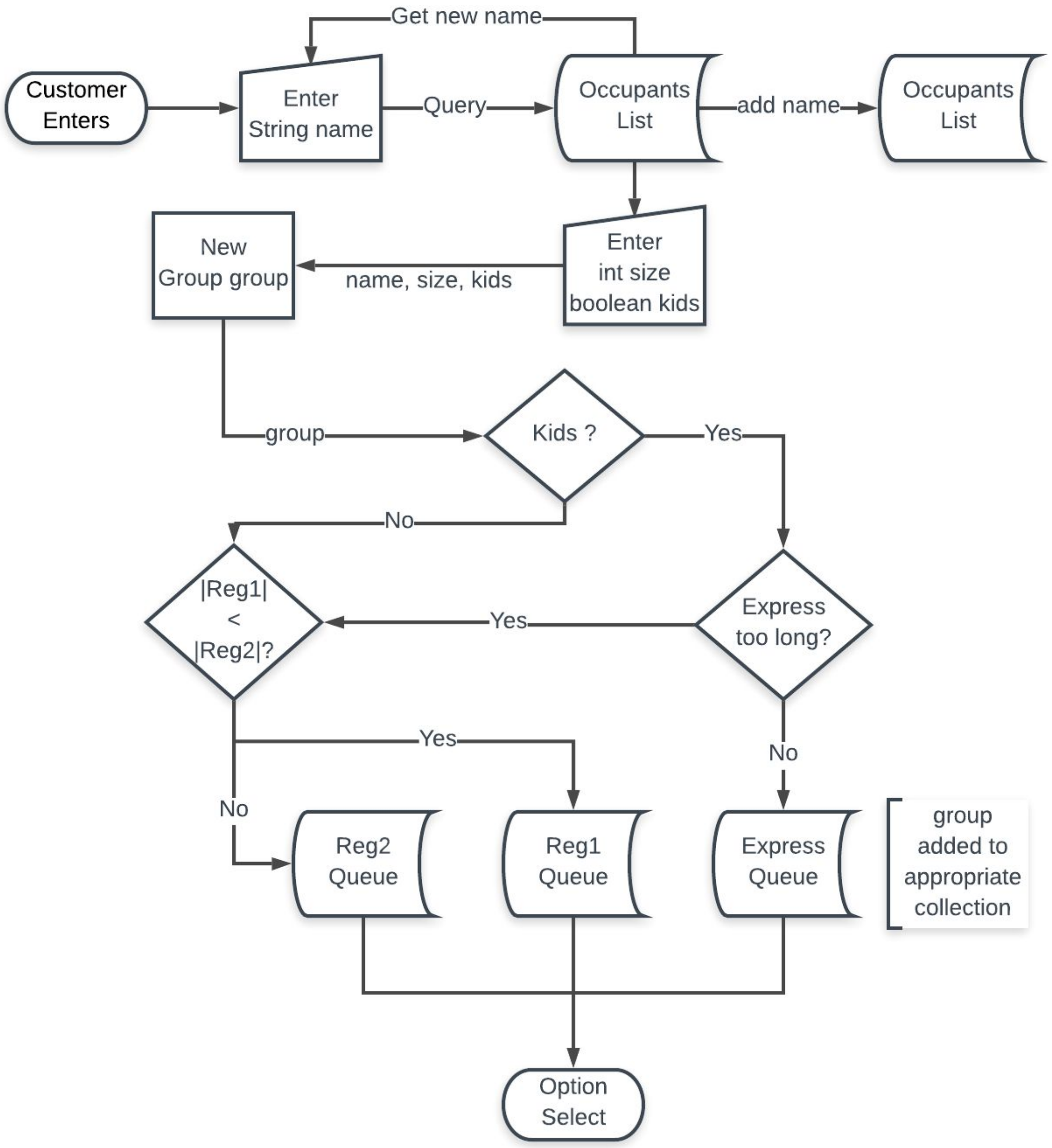


```
Theater
-Reg1:QueueRA
-Reg2:QueueRA
-Express:QueueRA
-Shazam:Room
-Dumbo:Room
-int:sales
-double:earnings
-double:ticketPrice
-names:ListRA<Strings>
+enter(customer:Group)
+buyTicket(customer:Group)
+leave(customer:Group)
+displayLine():String
+displayTicketEarnings():String
+toString():String
```

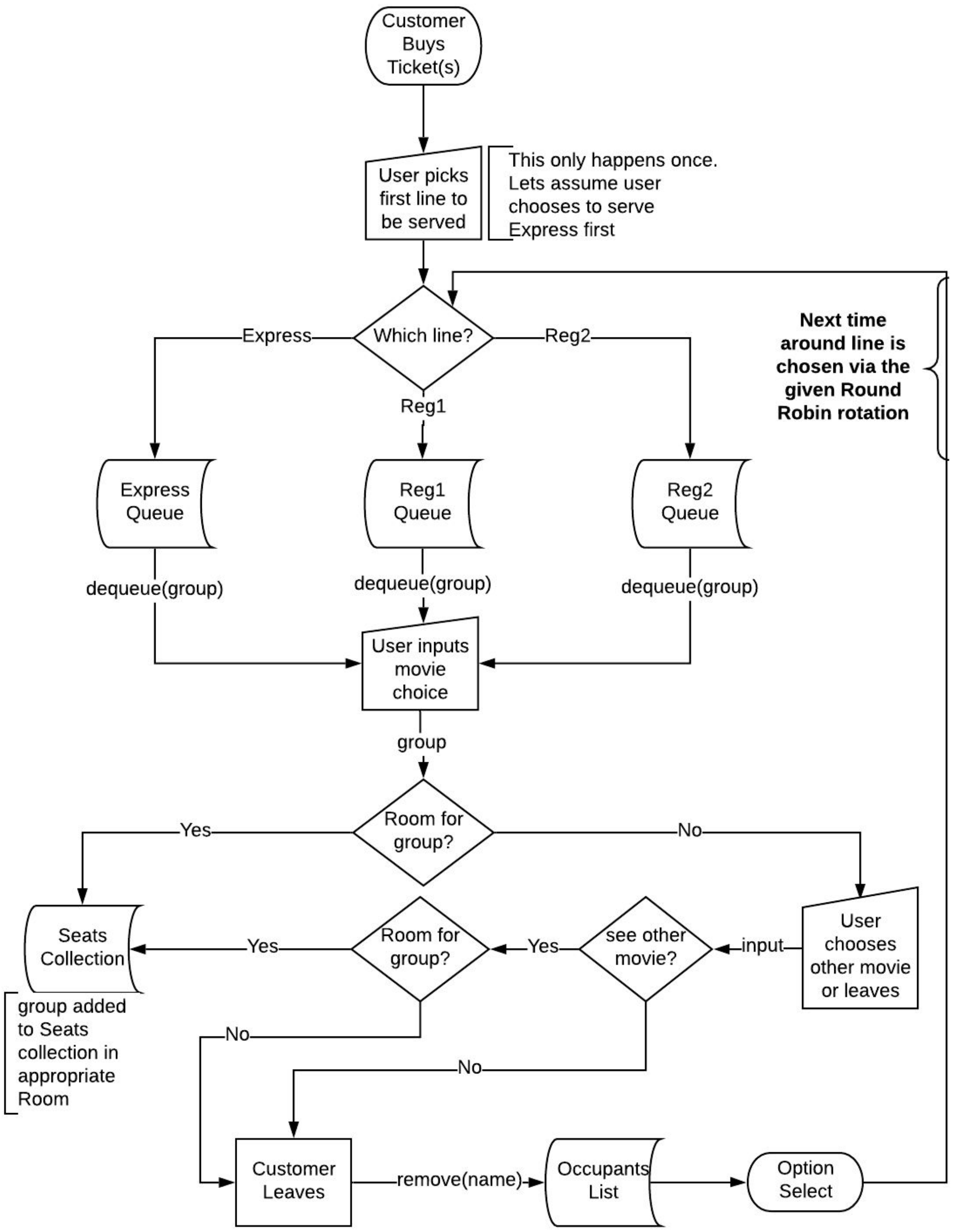
```
Room
-seats:ListRA<Group>
-cols:int
-rows:int
-movieName:String
-occupancy:int
+sequentialSearch(name:String):int
+addGroup(customer:Group)
+removeGroup(customer:Group):Group
+getAvailableSeats():int
+toString():String
```

```
Group
-name:String
-size:int
-under11:boolean
+getName:String
+getSize:int
+getKids:boolean
+toString:String
```

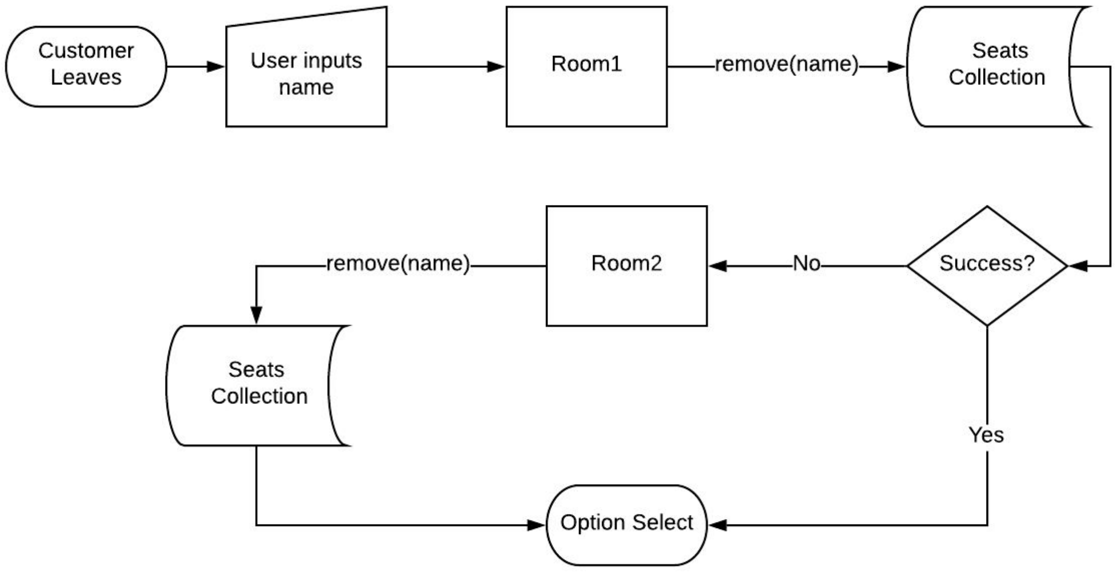
Option 1:


Customer Enters
Enter String name
Query
Get new name
Occupants List
add name
Occupants List
Enter int size boolean kids
name, size, kids
New Group group
group
Kids ?
Yes
No
Express too long?
Yes
|Reg1| < |Reg2|?
Yes
No
No
Reg2 Queue
Reg1 Queue
Express Queue
group added to appropriate collection
Option Select

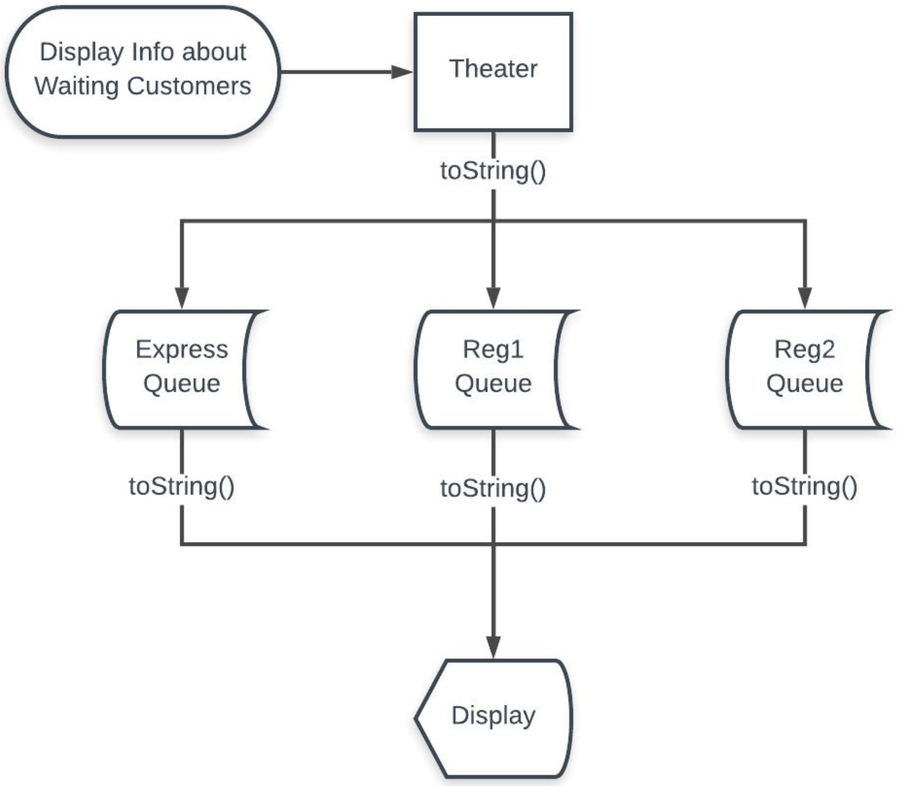
Option 2:


Customer Buys Ticket(s)
User picks first line to be served
This only happens once. Lets assume user chooses to serve Express first
Which line?
Express
Reg1
Reg2
Next time around line is chosen via the given Round Robin rotation
Express Queue
Reg1 Queue
Reg2 Queue
dequeue(group)
dequeue(group)
dequeue(group)
User inputs movie choice
group
Room for group?
Yes
No
Seats Collection
group added to Seats collection in appropriate Room
Yes
Room for group?
Yes
see other movie?
input
User chooses other movie or leaves
No
No
Customer Leaves
remove(name)
Occupants List
Option Select

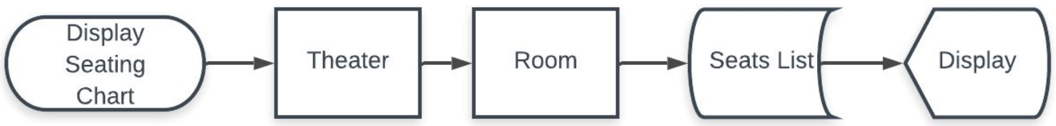
Option 3:



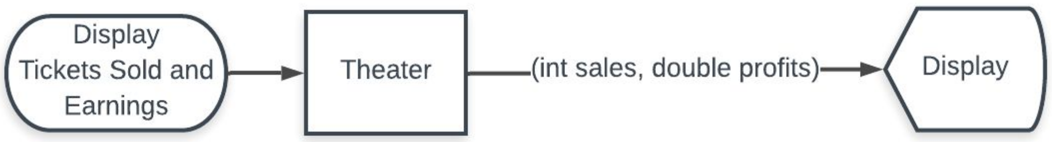
Option 4:



Option 5&6:



Option 7:


Display Tickets Sold and Earnings
Theater
(int sales, double profits)
Display

We're both collaboratively designing what needs to be implemented and will shift work dynamically according to the needs of the project.