Object Orientated Systems Development

CW1

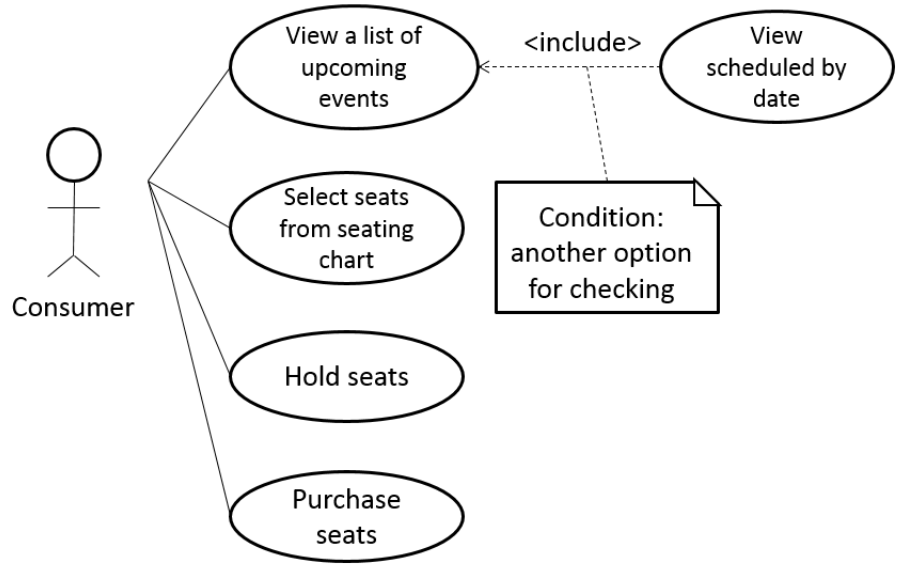
Report

Contents

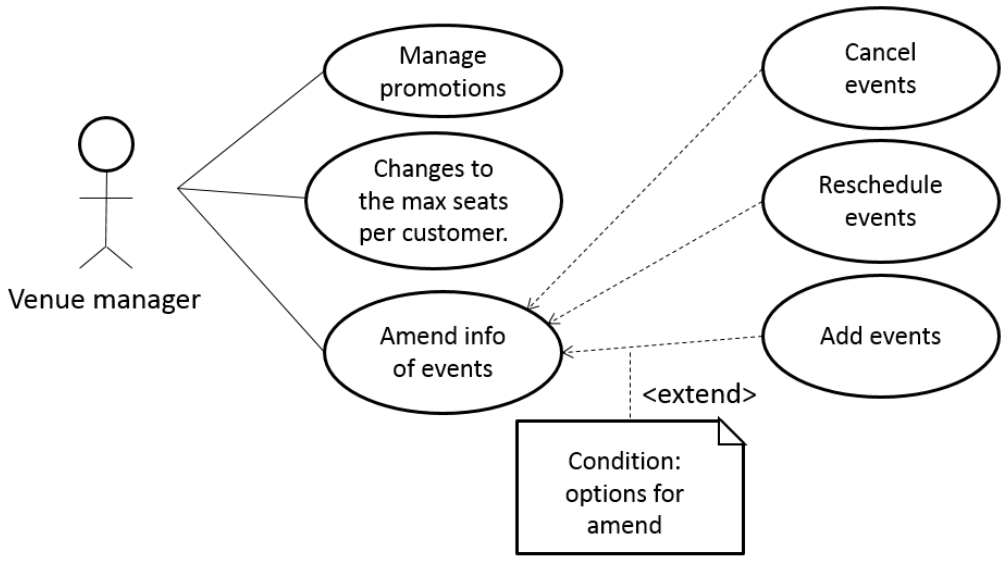
Section A: Design

Use Case Diagrams

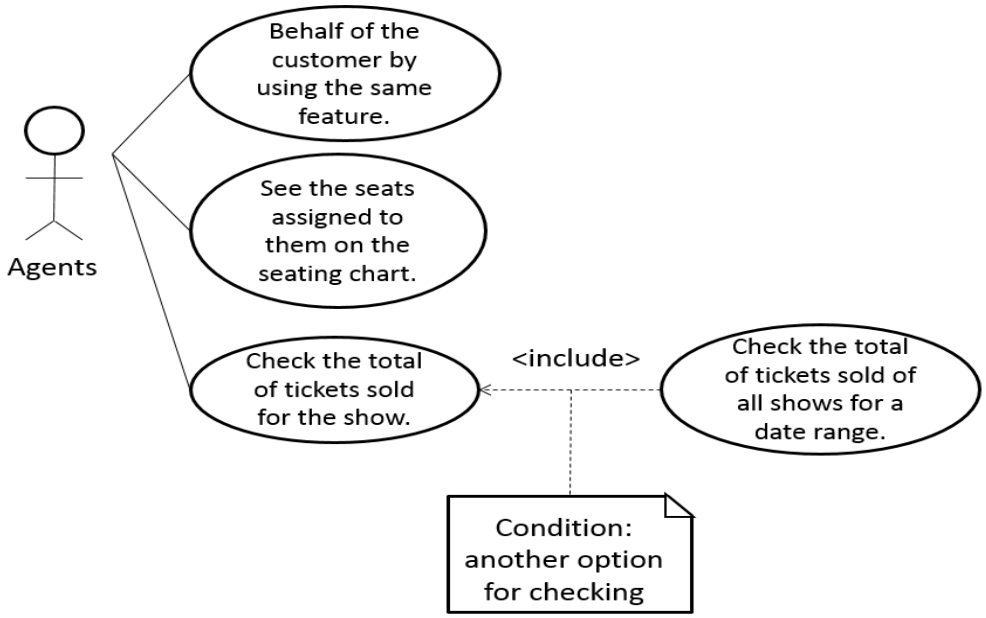
Consumer Use Case Diagram



Venue Manager Use Case Diagram



Agent Use Case Diagram



Use case diagrams are used to show the functionality of the system from the users' perspective. The diagram will include the feature of users could use for the system.

The use case diagrams for this case study will include the consumer, the venue manager, and the agent. The use case diagram for the consumer has a include relationship on the feature of viewing the upcoming events. On the diagram has stated it is an option for users to us. The reason of it is an include relationship and not a extend relationship is because the functionalities of it both is similar. Instead of making it become separate, using include relationship is easier to show the similarity. Include relationship not only used in the use case diagram for the consumer. It's also used in the agent use case diagram, for the method of checking the total number of ticket sold. The reason of using include relationship in here was because of the same reason from the consumer use case diagram. In the use case diagram for venue manager has used extend relationship. The extend relationship was used in the functionality of amend information of events. There are three options for venue manager to amend the information, which are cancel, add or reschedule events on the system. The reason of using extend relationship is because it is not always needed for the venue manager to do these changes on the system.

After the first stage of use case diagram has done, which is produce use case diagram for each user perspectives. And then we all review on it and giving suggestion on improvement. Other team members gave review about the relationship. At first, there is no relationship in each use case diagram. And then the other members think it should add relationship into it, so the functionalities doesn't look like duplicate. The first improvement have made, we all review the works again and found that the relationship in the diagram was in the wrong way round. After we done the second improvement, we all agreed on the work and start the second stage.

The second stage is to consolidate the diagrams into one and produce use case description. At the end, we came up of two use case diagrams instead of three. We have merge the use case diagrams

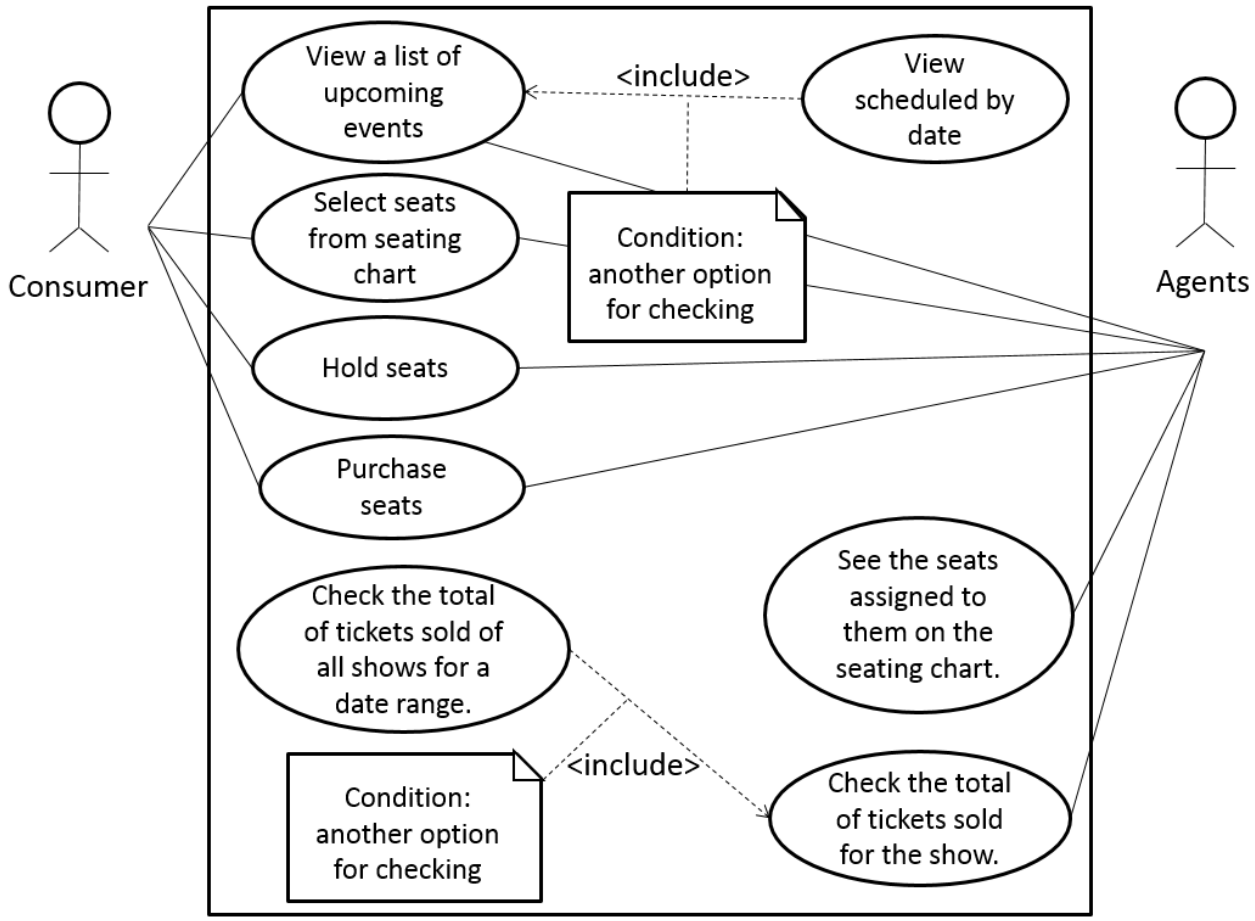
for agent and consumer together, because the agent have the same features as the customers and with one or two additional features. The new use case diagram for agent and consumer is named purchase ticket, because the functionalities in the diagram are about purchasing. We didn't make any changes on the venue manager use case diagram, because it is about managing the information, which is nothing related to purchasing. We named the diagram as manage information, and added use case description for it. We did do another review after the second stage has done, and we are all agreed and pleased for the work, so we didn't make any huge changes on the work.

Use Case Diagrams – Explanations

Consumer and Agent

The actors are given option by the system on choosing the event. After they choses, it will display a seating chart on screen for them to select the seat(s). Once they have select the seat(s) of want, it will hold the seat(s) for a short time, to allows the actors to finish the process of purchasing.

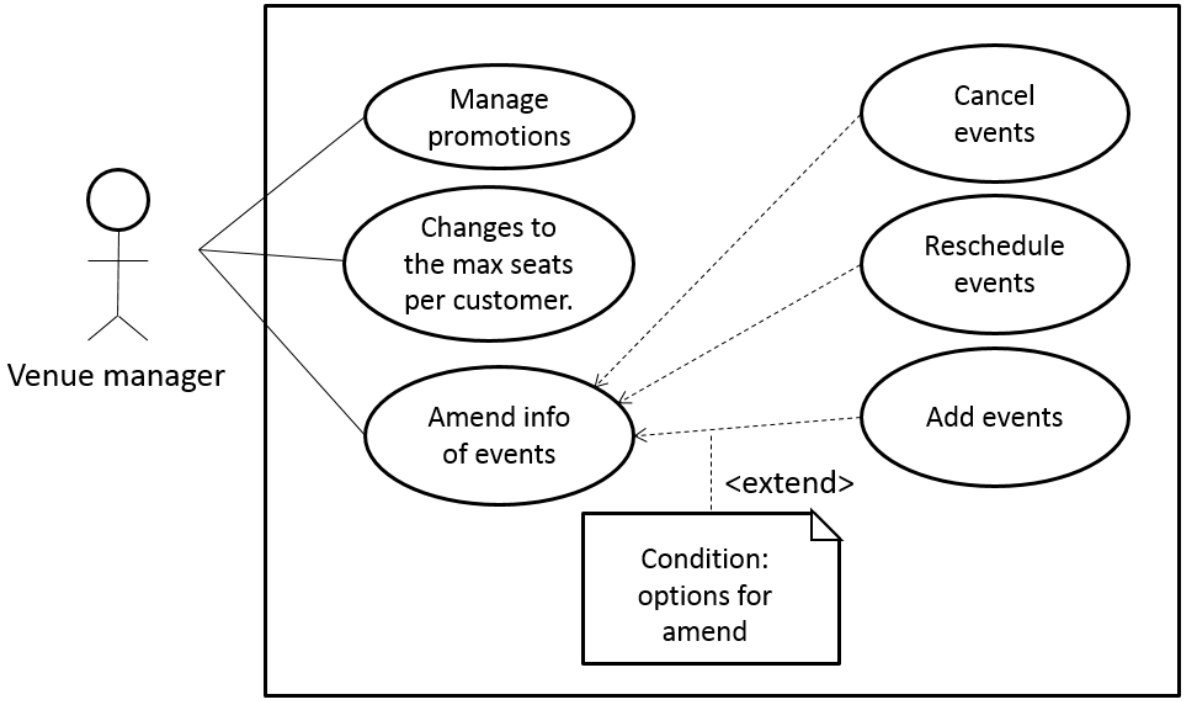
The agents will have additional features on the system, which is able to check the total number of tickets sold for the show or all the shows for a date range. They also able to see only the seats assigned to them on the seating chart. This information will help the agents on their business.



Use case diagram: purchase ticket

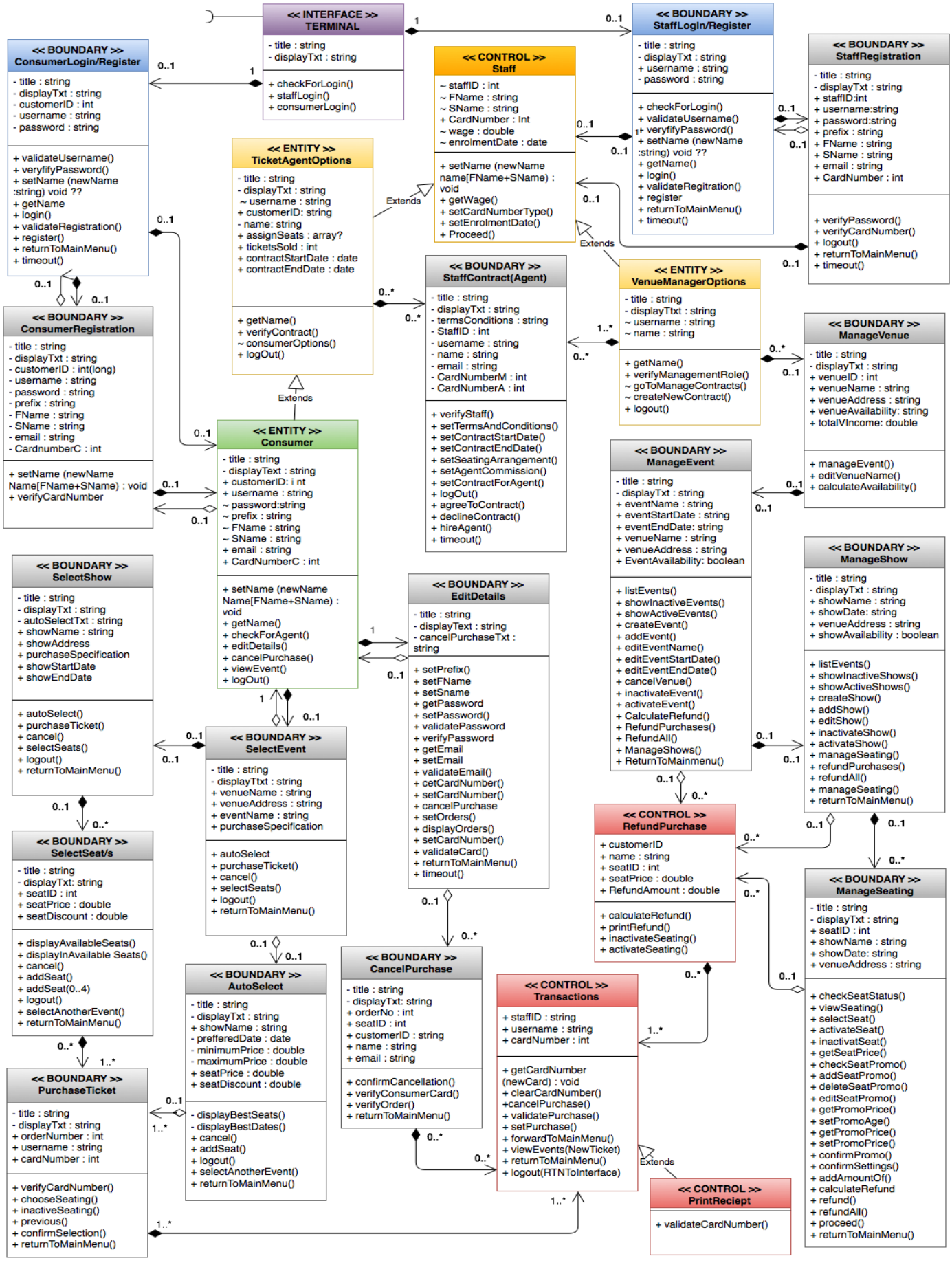
Venue Manager

The venue manager should be able to manage the information on the website. The things of venue manager can do on the website included manage promotions for shows, making changes to the max number of seats per customer for each show, and amend information of events. Amend information of events, which meant that it allows venue manager to add, cancel, or reschedule events.



Use case diagram: manage information

Class Diagram Outline



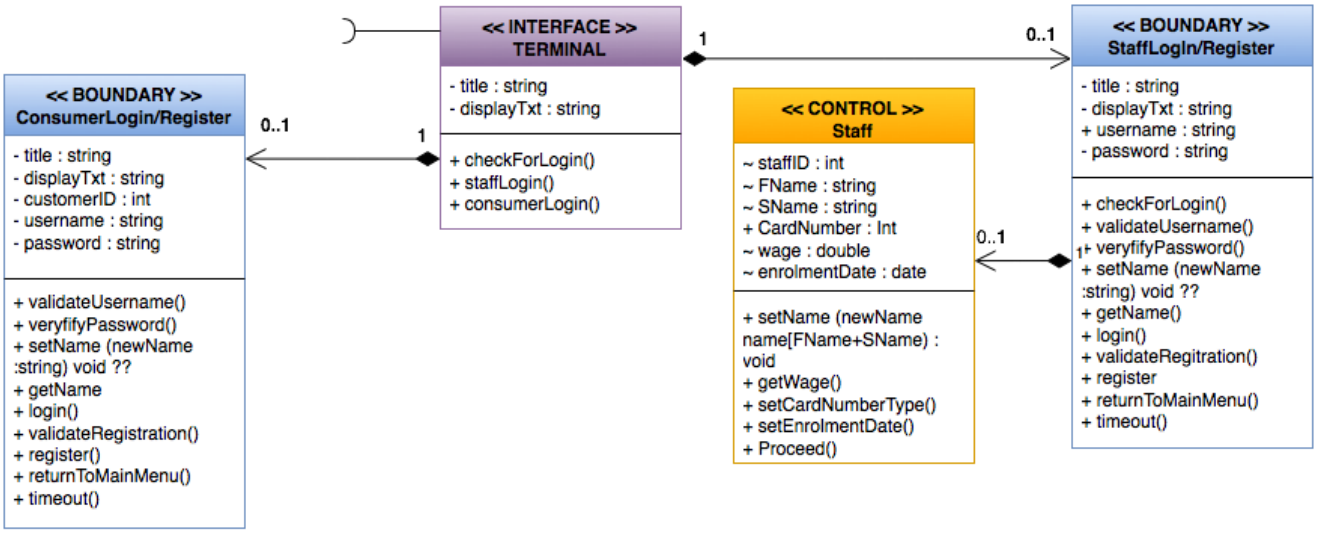
Class diagrams detailed explanation and justification

Class diagram is a diagram that contains class name compartment, attributes compartment and operations compartment. Attributes compartment that is the data that will be store or hold in the class, and operations compartment are about the method that will require those attributes to finish off. The system we need to create for the organisation will require quite a lot of data, so we also have create class diagrams to help to explain. When we doing the class diagrams, we did spend a lot of time on figuring the classes, this is because not all of us are fully understand how a class diagram should be like, and don't really got an idea in mind about the class diagrams for the project should be looks like. But after a few discussions, we did have a good start on the works. We do have a few versions for this task, since we have made a lot of changes and improvement compared with the first version. But the works also getting much more better along the process. Since it have lots of different classes and it is a big diagram, in the report will only pick some examples to talk about.

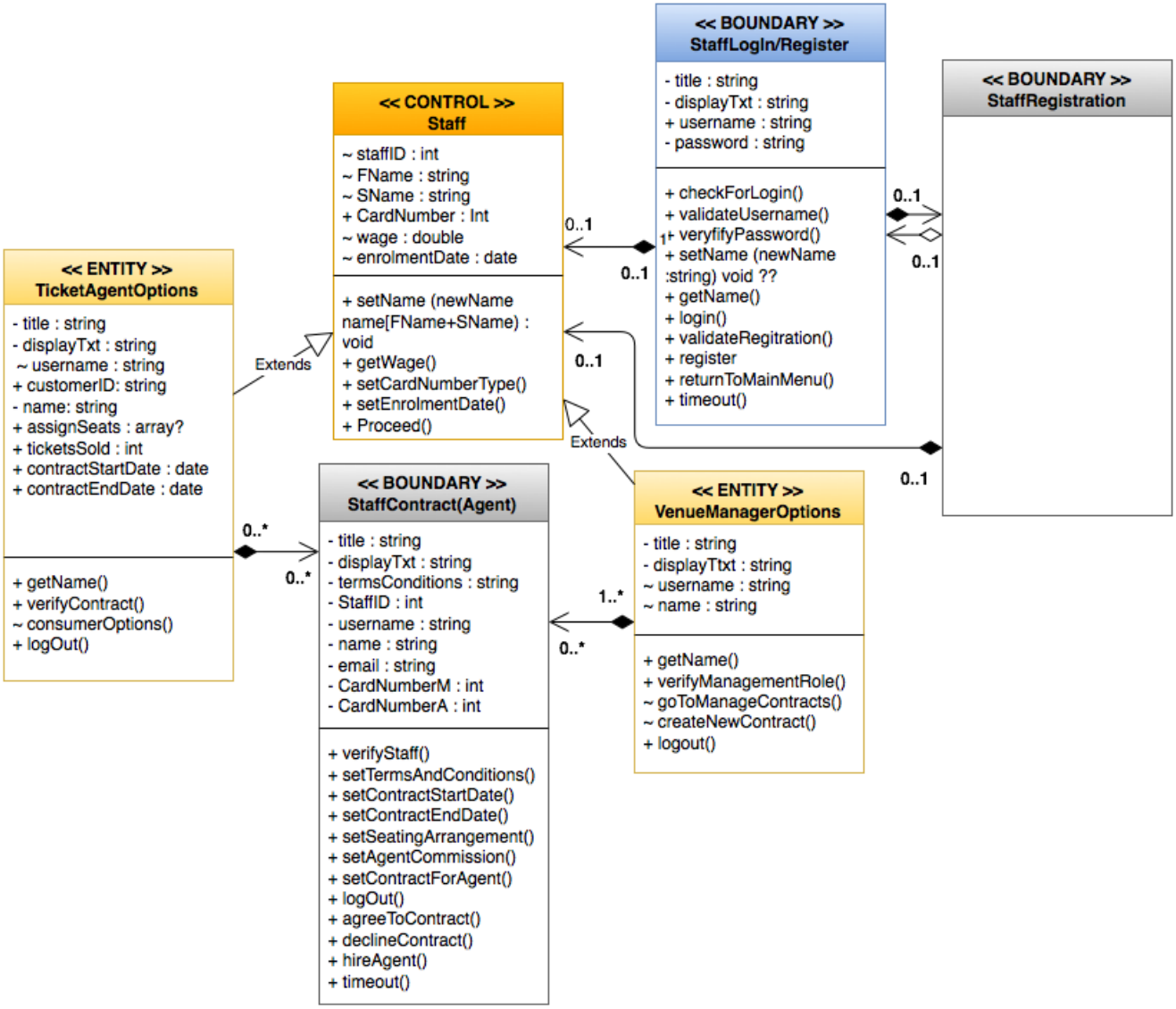
A Class Diagram is a constructed diagram that contains most if not all class names, attributes and operations compartments labelling the titles, fields, actions & functionality of each class in the diagram. The attributes compartment is literally the labelled information that will be stored or held in the relevant class whereas the operations compartment will be used to dictate the methodology that will require the relevant attributes to complete the class tasks. As well as a depiction of the individual class definitions, a class diagram will usually display the relationships between each class, inevitability designing the overall process of the system's functionality.

The specific system that we have been instructed to create for the BCPA organisation will require quite a lot of data. With that taken into consideration we have created a fully functional class diagrams displaying each class in the system and how they may generically interact with each other. When we designed the class diagrams (first partially then integrated into 1 diagram), we did spend a lot of time using deductive reasoning to depict how the system would be built regarding the functionality within classes but took the importance of class synchronicity into further consideration. At first we struggled blending our imaginations with the idea of implementing strong simplicity, but over a short period of time we figured out how we see the system to the work and how we can make it happen through the strong assistance of our class diagrams.

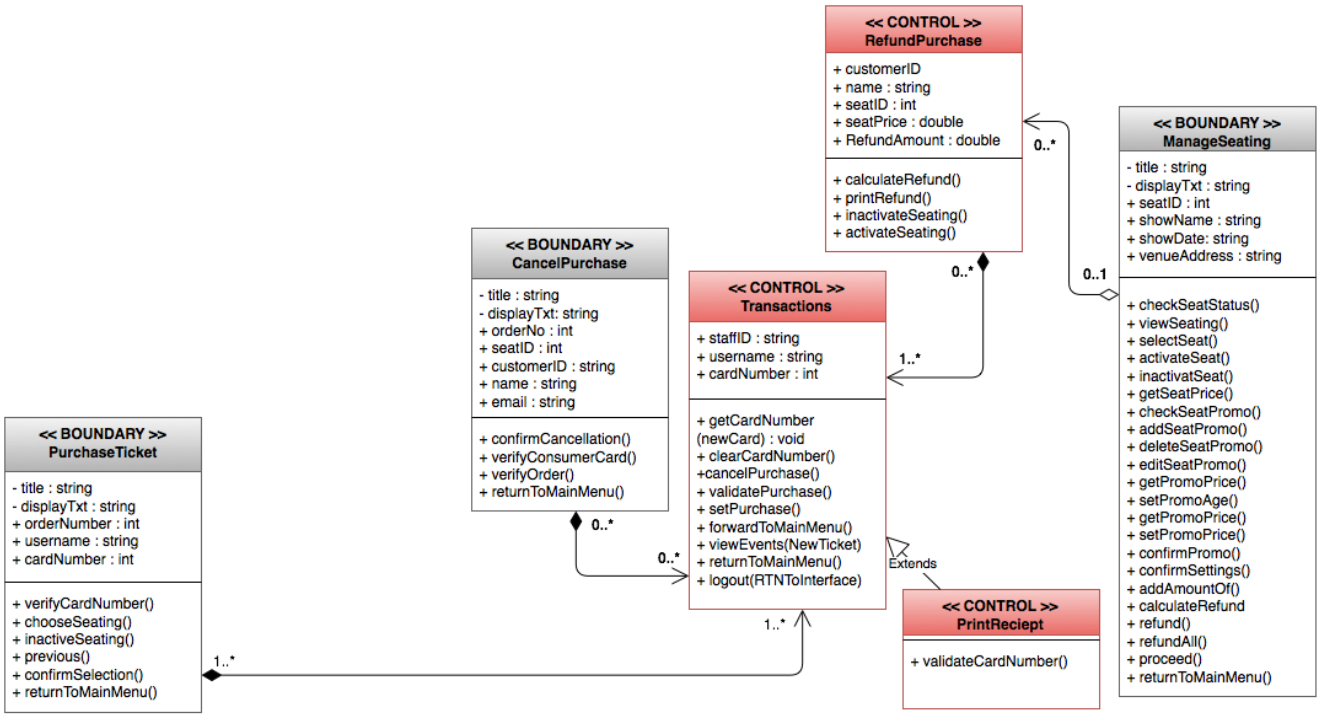
Since this particular system has many classes and is a large diagram, we have taken the liberty of not only displaying the entire system but also breaking it down in and for our report, mainly focusing on some of main class functionality examples bounding the elements that are used throughout the diagram/system.



The image above is the part of the class diagram displaying the online terminal interface class, as well as its link to the Staff & Consumer Login/Register option classes. The first row in the class box is the class name compartment, usually a filled with label that depicts the main functionality or representation of the class itself. The second defined row is the compartment for the attributes belonging to the class, which in most if not all cases is a list of labels describing the attribute fields (and their type) that are to be used in the class whether displayed, utilized by the user or in the system. The last row is the operation compartment, where a detailed list of the functions that can or will be used by the system through the relevant class upon relevance. If the row is blank, this usually is meant that the object doesn't have operation or attribute. For example, the class of printTicket (taken from the image following this report) does not particularly have attributes, this is because in this case it is only used to complete a set of commands from which the attributes have been set from in another class. Through analysing the various classes in this system diagram you can further depict the behaviour of each class and how this can and should affect the system functionality overall.



We did believe that there is a generalization structure in our system and diagram. Generalization is when the objects have similar in most details (can differ on some aspects), implicating a class with the ability to inherit and share attributes and/or operations. The image above is the generalization of Staff split into the Venue Managers & Ticket Agents that we believed should be included. As you can see the Staff class has attributes and operations that both the Venue Managers & ticket Agents can and will utilize. According to the image, we think that the Staff class is a major superclass, and the Agents and Manager Class boxes would be subclasses. We made these subclasses because they have special features on the system that are designed to cooperate with the organisation using real data and information. Although the Venue Manager and Ticket Agent classes are subclasses to the Staff class, they can be seen as a superclass to other class instances, meaning in is possible to have a super super superclass, which the average designer would label Object.



The image above displays the association between some on the classes in our system. Regarding class association, this methodology has been generically broken down into two main types of association, which are aggregation & composition. As consumers, managers and agents can all use the ticket machine to interact with the purchase of a ticket we deemed it important to state the associations our ManageSeating (and refundPurchase), CancelPurchase & PurchaseTicket classes will in most cases (depending on seat availability) interact with the Transactions class. If a seat that has been managed is now needed to refund a consumer's purchase, this procedure will inevitably activate a transaction linking the refund amount with the Consumers card number.

With that being said it can and may now be more understandable that a selection of seats may be managed with or without refunding a consumer's purchase and the information depicted from the RefundPurchase class can exist without the managing of a seat. This is known as aggregation and is represented with a non-filled diamond connection between classes representing the flow of initiation. Other forms of aggregation in our system and be found between the consumers ability to edit their details and the option of cancelling a purchase, or even the ability to auto-select a seat from the concurrent selection details proposed. This is because those are also actions or representations of direct functionality that can exist without the other class, it wouldn't be destroyed, causes a circuit break or loophole even if the events are being cancelled or rescheduled.

The image above also displays examples of composition in our class diagram. The operations in the transactions class are not strong enough to establish independency. All the operations in the transactions class are only operated through the specific Instance of purchasing a ticket or refunding a purchase and this is the reason of making it become composition. As each instance sets the values within the specific transaction made they are only needed to momentarily store and display the transaction details while holding no other purpose. Meaning that without the information processed through the purchasing of a ticket or refunding a purchase the ability to implicate a transaction would not exist, commonly known as a composition.

Multiplicity is a process mainly used in class diagrams to depict the cardinality between each classes as their collection of elements are designed to interact. Redefined, this particular type of expression describes the specific number of elements that each class can and will be interactive with at any given interval of use. For example, a class may have 0, 1, a predefined value (such as 4/5) or many (*) interactive instances of a class, as well as the likely chance of having the possibility of both 0 and many options of one associations cardinality.

With our specific class diagram and the vast amount of class functionality in this particular system it didn't take long for us to depict the fact that a lot of multiplicity can and should be implicated into it, typically as the system is designed to be used by multiple users for limited or a better yet designated outcome, as well as the system's ability to automate a function such as managing an event or show to refund many purchases. From the final class diagram below you will notice many states where multiplicity is used throughout the system.

Data Dictionary

Below on the next page is a data dictionary of basic terms and functions, the system is going to have once implemented as a group we all agreed what this should include.

The problem that we have met during the process, which is at the beginning we misunderstand what is a data dictionary. We thought that it should be contain the attributes that will need to use in the system, and also define the data type of it. But then we realise that it should be also contain the term and definition of it. So we have to spend time to add more things in it. Apart of this problem, this task is being fine. And the writing below is the explanation for the data dictionary.

Part 1 is the fundamental part of the system with all the necessary things that a system like the BCPA has to have as mandatory fields these are the following

- Customer ID
- Title
- Forename
- Surname
- Address 1 and 2
- Postcode
- Telephone Number
- Username
- Password

These are all major parts of the system as the consumers, agents will need to lookup details of shows. They have bought and consumers will need to buy tickets for a show they want to see etc.

The second bit of the data dictionary is some functions that may be in cooperated within in the system, in order for it to work effectively such as add amend and delete, as people that use the system need to be able to add new shows, amend details such as kick off times and delete events, that have happened to save users trying to book tickets for events that have already happened.

The third and final part we would say of the data dictionary is the general definitions for terms, generally used in the assignment. For example, Methods. It's the functions that users will be allowed to do on the system, such as purchase or amending information. And sequence model is a diagram to show the steps and methods of actor will need to follow for complete specific task.

Name	Data Type	Description
Customer ID	INT	Reference number of a customer
Title	String	Title Mr Mrs etc.
Forename	String	First name of a customer
Surname	String	Last name of a customer
Address 1	String	Line 1 of a customer's address
Address 2	String	Line 2 of a customer's address
Postcode	String	Post code of a customer string because of a mix of letters and numbers
Telephone Number	INT	INT as we are not using letters
Username	String	String as a username can be quite long
Password	String	String as a username can be quite long
Print Ticket	Function	Printing the ticket details for the customer
Card Number	INT	Only numbers are in card numbers
Order Number	String	Can contain numbers and text
Email	String	Can contain numbers and text
Staff ID	INT	Reference number of a member of staff
Username	String	Can contain numbers and text
Password	String	Can contain numbers and text
venueName	Function	Declare the name of the venue
ConfirmSeating	Function	Confirm the seats for a customer
ViewSeating	Function	Call this function to display the seats available to a customer
ViewSeatPrice	Function	Call this function to display the price brackets available to customers
AddAge	Function	Take information about the customer's age
AddAmount	Function	Add up the total cost of tickets
Refund	Function	Refund all the money back to the customer
RefundAll	Function	Refund all the customer's money
Proceed	Function	Move onto the next stage in the booking process

AddEvent	Function	Put a new event onto the system
AmendEvent	Function	Change details of an event on the system
DeleteEvent	Function	Delete the content of an event existing on the system
CancelEvent	Function	Take the event off the system
ResheduleEvent	Function	Make the event a different day time
Ticket Machine	N/A	The new online ticket system (OTS) that needed to be building for the client. The system should allow users to purchase from it.
Ticket	N/A	Consumer can buy it via the OTS. It only will be create when someone had purchased. Each is unique for different events.
Features	N/A	It is the functions in system that will be allow users to access for completing specific task, such as purchasing ticket.
Client	N/A	The client is the person or the organisation that require the system. The client of this project is Bucks Centre for the Performing Arts (BCPA).
Actor(s)	N/A	The person or people that need to take action for the process.
Consumer	N/A	This will be the people who will use the system to buy ticket for shows.
Venue manager	N/A	This is the staff from the client side that does the management for information that will display on the system for the public to view.
Agents	N/A	This is the people that cooperate with the client to run their business. These people will also need to use the system to book ticket or use to as a reference for their own business.
Methods	N/A	It's the functions that users will be allowed to do on the system, such as purchase or amending information.
Object	N/A	An object is the entity. It's help to find out the data that needed for create methods for the system.
Class	N/A	Objects with attributes and methods made up a class. A class can also divide into superclass and subclass.
Superclass	N/A	Another word for it is generalization, which is meant that defining the class for the object generally.
Subclass	N/A	Another word for it is specification. It is about define the object in a class in detail. For example, vehicle is superclass, but motorcycle and car are subclass.
Use case diagram	N/A	It is a diagram to show the things of an actor will need to do for complete specific task.
Class diagram	N/A	It is a diagram to show all the classes. This would help for implementation.

Activity diagram	N/A	It is a diagram to show the actions and decisions that need to done by actors.
Sequence model	N/A	It is a diagram to show the steps and methods of actor will need to follow for complete specific task.
Functional requirements	N/A	It is the requirements that expect to be done by the actors.
Non- functional requirements	N/A	It is the requirements that expect to be done by the system.
User interface	N/A	It is about the interaction between users and the computer system.
Pseudo code	N/A	It is an algorithm that using informal way, which meant it will use some programming language to do with.
Booking	N/A	Select event and seat, and then purchase ticket via the online ticket system.
Login	N/A	Users (venue manager, agents, and the consumer) have to sign in to the system with username and password for using the system.
Register	N/A	Consumers will need to register on the system by themselves to use the services.
Amend information	N/A	This is the feature only able to use by the venue manager, which allows them to reschedule, cancel, or add events. They also can use this feature to amend price or promotion for each event, or change information for the event such as description.
Promotion	N/A	Discount for specific events, people, or days. Venue manager are the only person that have the feature on the system to create promotion.
Held seat(s)	N/A	Seat(s) that have been selected by the consumer before completing transaction.
Seats	N/A	Consumers have to select the seat(s) of they want for the event. It can be multi selection for consumers that want to buy more than one ticket.
Seating chart	N/A	This will be display on screen when consumers decided to purchase ticket for the event. It will show the current available seats as well.
Auto seat selection	N/A	Consumers can use this feature if they don't mind the location. This feature will select the seat for the consumers after they have input the price range of they accept.
Internal clock	N/A	This will be display on screen when the seat(s) is placed in a hold state. It is to prompt consumer to continue the process.
Transaction	N/A	Consumer paying online to buy the ticket for event via the OTS.
Events	N/A	It is the shows that will be on soon, and able users to purchase ticket for the events to watch it. And the venue manager is the only person to create it on the OTS.

Design Phase Appendix 1 Meeting Minutes References

Meeting 1 -- Sunday 15th November 2015

This meeting was done virtually over a WhatsApp group chat, as we were all away in different places we talked about the following:

- Use Case Diagrams
- Class Diagram
- Data Dictionary

We discussed skills and what people wanted to do. It was decided that Elaine was going to do Use Case diagrams, Temi to do the Class Diagram and Robert to do the data dictionary, as he had previous experience with creating data dictionary and he found an old template he had used before.

Meeting 2 -- Monday 17th November 2015

This meeting was only 20 minutes as the team had to go somewhere.

Class Diagram was the only thing talked about, as this would take some time to produce, we decided to look at some examples to get an idea of what it should look like.

Meeting 3 -- Tuesday 18th November 2015

This meeting only 2 people was present due to sickness we discussed the quality of the data dictionary and started finalizing the report sections.

Meeting 4 -- Wednesday 25th November 2015

This meeting we went through whole of section A as it needed a check, as some things still were not correct.

Meeting 5 – Saturday 28th November 2015

This meeting was done virtually over a WhatsApp group chat.

We have finalizing the data dictionary explanations of each part of the 3 entities, as summaries had not been produced of what each field means what like customer ID could mean anything. So we have to summarize what customer ID meant, as any user who might come to use the system would not have known. We also explained the functionality of the system.

Meeting 6 – Sunday 29th November 2015

This meeting was done virtually over a WhatsApp group chat.

Class Diagram and Class Diagram explanation and also general catch up on assignment as a whole. As the deadline to hand in the work is fast approaching, we decided to have a group discussion on WhatsApp of where everything was at it turned out that Temi drafted out the class diagram. We stated the data dictionary had been finalized. However it was stated that the use case diagrams had to be formally re drawn on draw.io.

Robert and Temi agreed to help Elaine with this as this was quite a complex task to do and lastly some time was taken to think about using Blue J to implement the system as the deadline was not far away for the whole project to be handed in.

Meeting 7 – Tuesday 1st December 2015

We're structured the class diagrams to make the system more understandable for when we begin to code the system.

Meeting 8 – Wednesday 2nd December 2015

Robert and Elaine had a chat and decided the data dictionary needed more detail, and a new approach as the data dictionary was like in a database form not a valid one for programming the BCPA system in Java using the Blue J program.

Meeting 9 – Thursday 3rd December 2015

Robert and Elaine finished putting the data dictionary together, as some bits still need to be added to it. Also Elaine pointed out that they were some use case diagrams that had been missed out at the start of the report, and after a quick discussion they were inserted.

Meeting 10 – Wednesday 9th December 2015

The meeting was about class diagram, as we all think that there was some problem in the class diagram. Robert and Elaine have given feedback on the class diagram, to help Temi to do improvement.

Meeting 11 – Thursday 10th December 2015

After the meeting 10, Temi has done the improvement for class diagram. In this meeting, Temi has explained the changes of he has done, and also go over the whole thing again to make sure that we all understand and agree the work.

Design Phase Appendix 2 Record of Meetings

Meeting no.	Date	Attendees	Topics Discussed	Duration
1.	Sunday 15th November 2015	Robert, Elaine and Temi	Use Case Diagrams Class and data Dictionary	2 hours
2.	Monday 17th November 2015	Robert, Elaine and Temi	Class Diagrams	20 mins
3.	Tuesday 18th November 2015	Robert and Elaine	Starting Data Dictionary and Report	1 Hour
4.	Wednesday 25th November 2015	Robert, Elaine and Temi	Overall Section A	1 Hour
5.	Saturday 28th November 2015	Robert and Temi	Finalizing data dictionary explanations	1 hour 10 mins
6.	Sunday 29th November 2015	Robert, Elaine and Temi	Class Diagram and Class Diagram explanation and also general catch up on assignment as a whole	1 hour 20 mins
7.	Tuesday 1st December 2015	Robert Elaine and Temi	Whole Assignment meeting on Design particularly Class diagram in the design phase	1 Hour
8.	Wednesday 2nd December 2015	Robert and Elaine	Finalizing and re doing data dictionary	30 Mins
9.	Thursday 3rd December 2015	Robert and Elaine	Re doing data dictionary and tidying up the design phase report a	35 mins
10.	Wednesday 9th December 2015	Robert, Elaine and Temi	Class Diagram	1 hour 15 mins
11.	Thursday 10th December 2015	Elaine and Temi	Class Diagram	1 hour 15 mins

Section B: Implementation

Requirements Statement

Client Name

Bucks Centre For Performing Arts (BCPA).

Purpose of system

An entertainment venue wants to allow customers to order tickets for the Internet. The system must include the customer to view upcoming events or view scheduled shows by date, select seats from a seating chart, hold the seats while selection is being completed and purchase the selected seats.

Functional Requirements

Functional Requirement	Description
User booking a ticket	This is the user choosing his or her ticket for a certain show, e.g. Snow White
Check Event detail	Checking start time, end time date of show
Agent checks to see seat allocated	What seats have been allocated and what seats have not
Manage promotions	Promotions active and still being finalized
Manage show details	Changing information on a performance e.g. description, image, date and time
Check number of tickets sold for specific show	Agents can only view this to check to see how much has been made
Check number of tickets sold all together	Agents can only view this to check to see how much has been made

Non Functional Requirements

Non Functional Requirement	Description
Store customer details	Storing customer details e.g. contact details, payment details
Store Prices	Storing price brackets e.g. Junior, Student and Concession
Store Promotions	Storing start and end dates of the promotions
Edit promotions	Changing dates on promotions
Edit Prices	Changing prices of tickets
Print show details	List the show details and print them out
Print Customer's Ticket	Producing E ticket or paper ticket

Partial Class Diagrams Explanations

Here are our partial class diagrams that we have designed for the BCPA system here we are contributed with some ideas. However, we have had some difficulties here such as assigning the multiplicity for each phase and we were trying to get the relationship sorted out. But the problems have been solved since we have spent time to do research on understanding it, and keep on improve the work after we found something new that should be add on the diagram.

Diagram 1 – Venue Manager

Here is the venue manager section partial class diagram from early on. We knew the venue manager was going to be responsible for so many things in the BCPA system, so we have had to break it done in order for the system to work efficiently.

Venues Manager is entity and is responsible for manage venue, manage event, manage show, manage seating, print ticket, updating consumer cards and refund.

The venue manager has to be able to manage venues, events, and shows. In order to do that functionality has to be implemented, such as add, edit and cancel events under manage event.

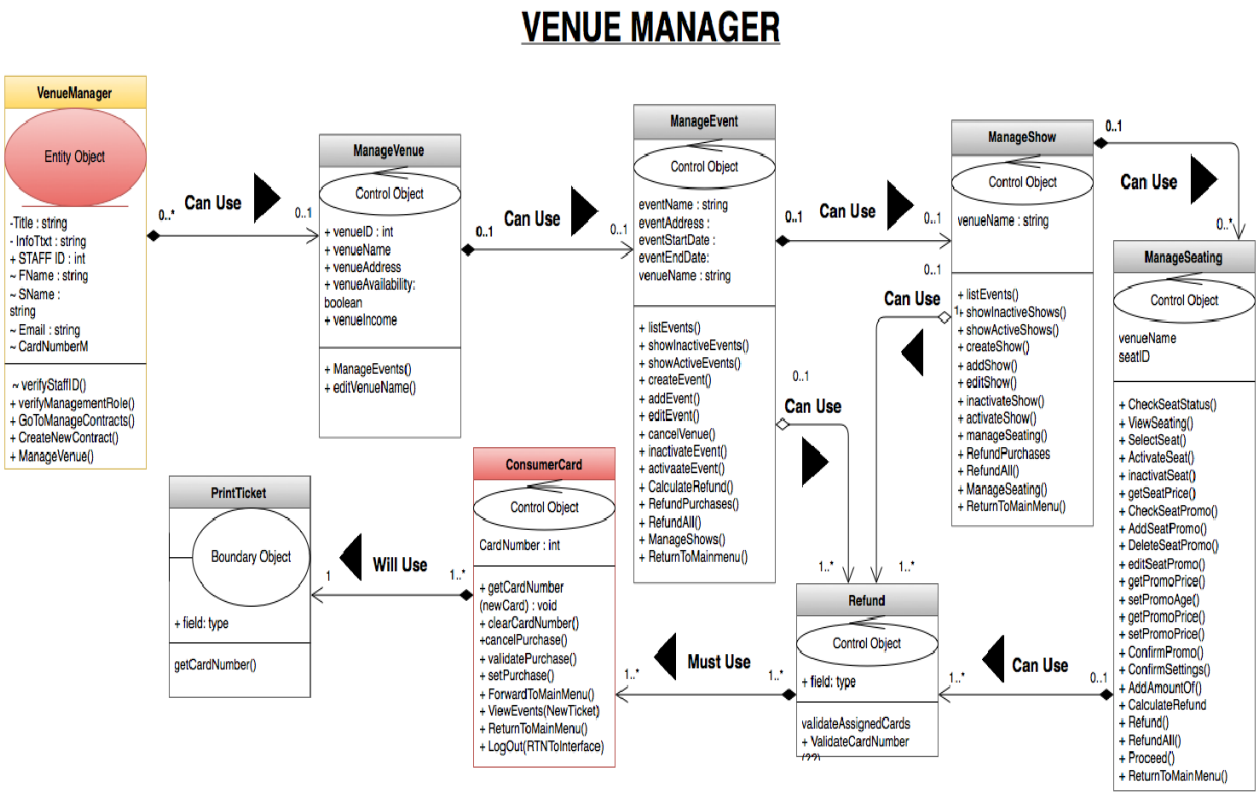


Diagram 2 – Consumer and Agent

Here is the second diagram of what a typical consumer would go through first. And they would login using forename, surname details etc.

On the other side though we have the staff section, which has exactly the same process to login or register. And so staffs have separate logins to the customers, as they need to see how many agents are associated on a particular event, and also ticket agents are classed as staff as they need to check the customer details spec.

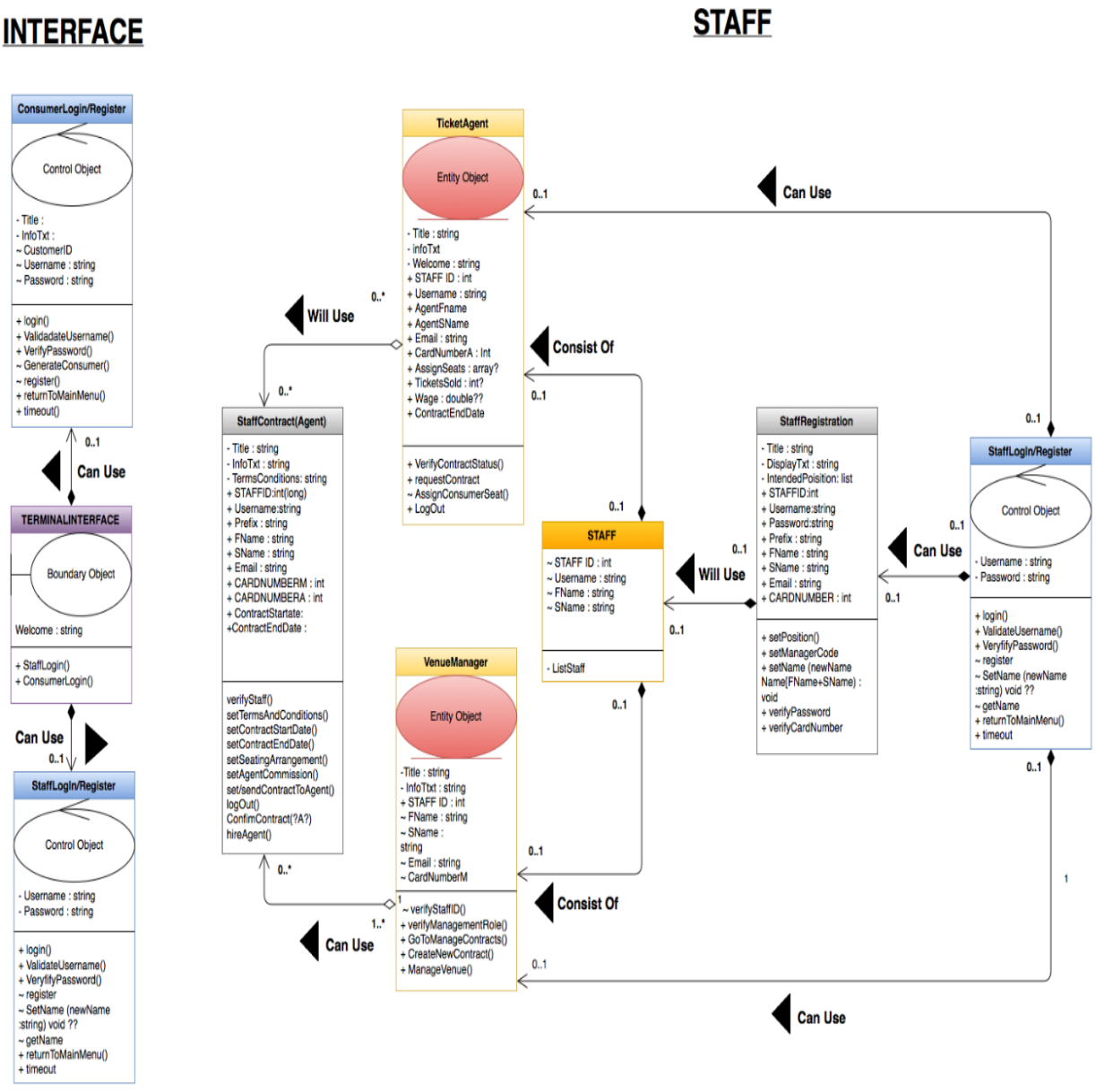


Diagram 3 – Ticket Agent

Here is the third diagram relating to ticket agent its quite a big one because he is responsible for so many things such as managing consumers details such as usernames passwords and verifying details etc. and also registering new consumers customers with all their details such as new username and basic things such as forename and surnames.

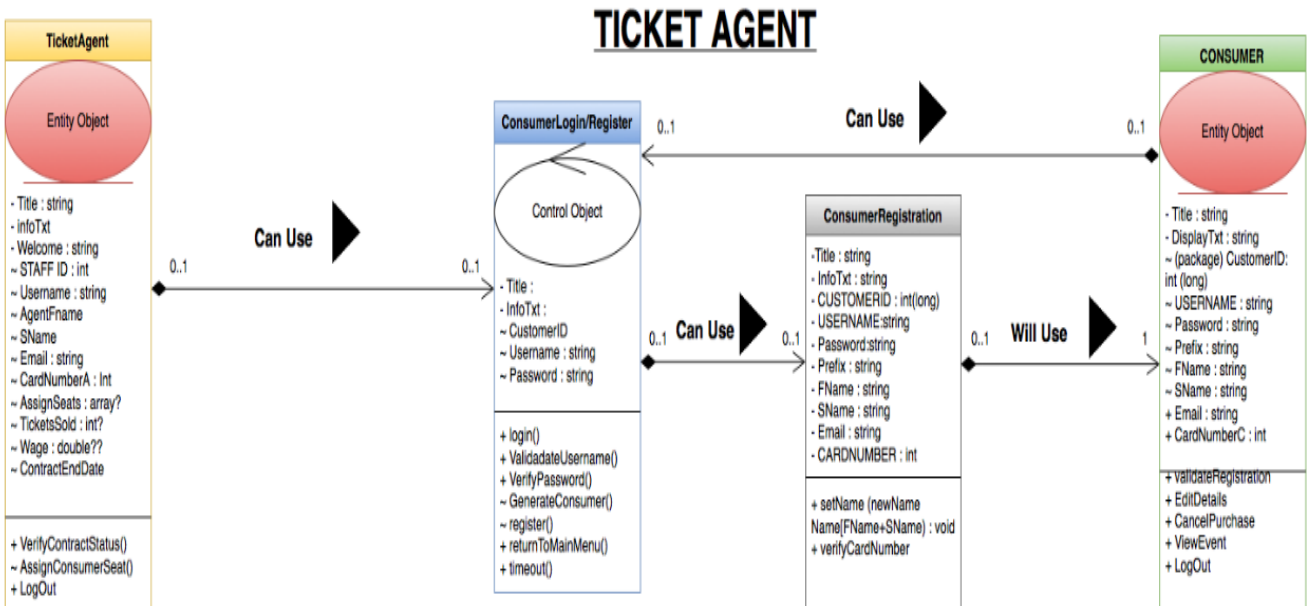


Diagram 4 – Consumer edit details

This diagram shows how the consumer would edit their details themselves, such as forename and surname, changed it password, or cancel an order for a show. There is also the inclusion of the consumer's card, as when they type it in the system they could check their purchase history.

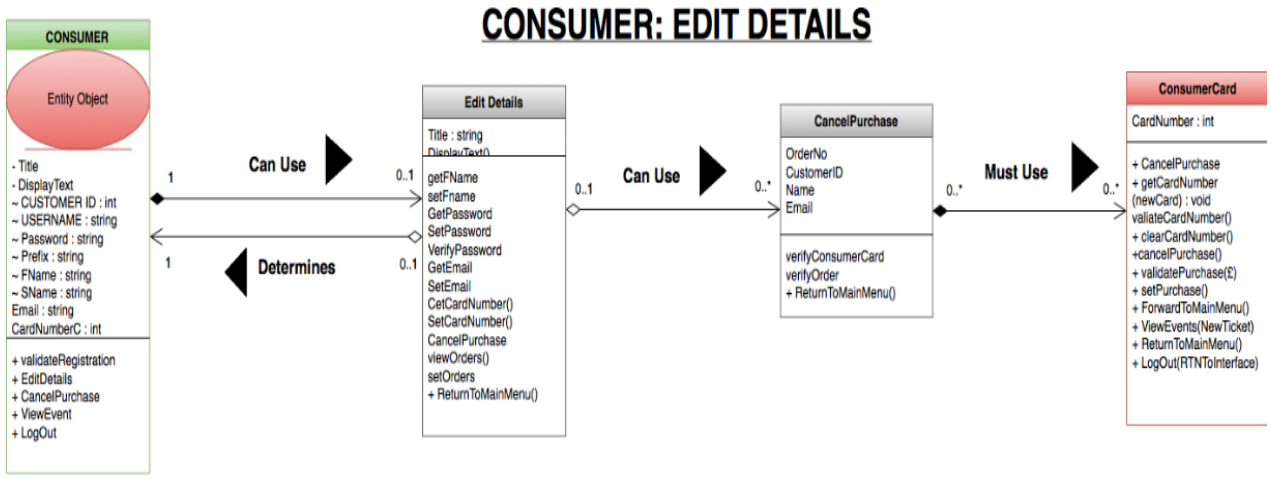


Diagram 5 – Consumer purchase seats

This last diagram shows a consumer process for purchasing seats for a show, such as Peter Pan, also it details the process involved.

Step 1: Would be the consumer logging in with the consumer details, they have set by the first time of their account creation with things such as forename, surname and email address.

Step 2: Will show the active events start date and end date, and then give the user an opportunity to select the show they want to go and see.

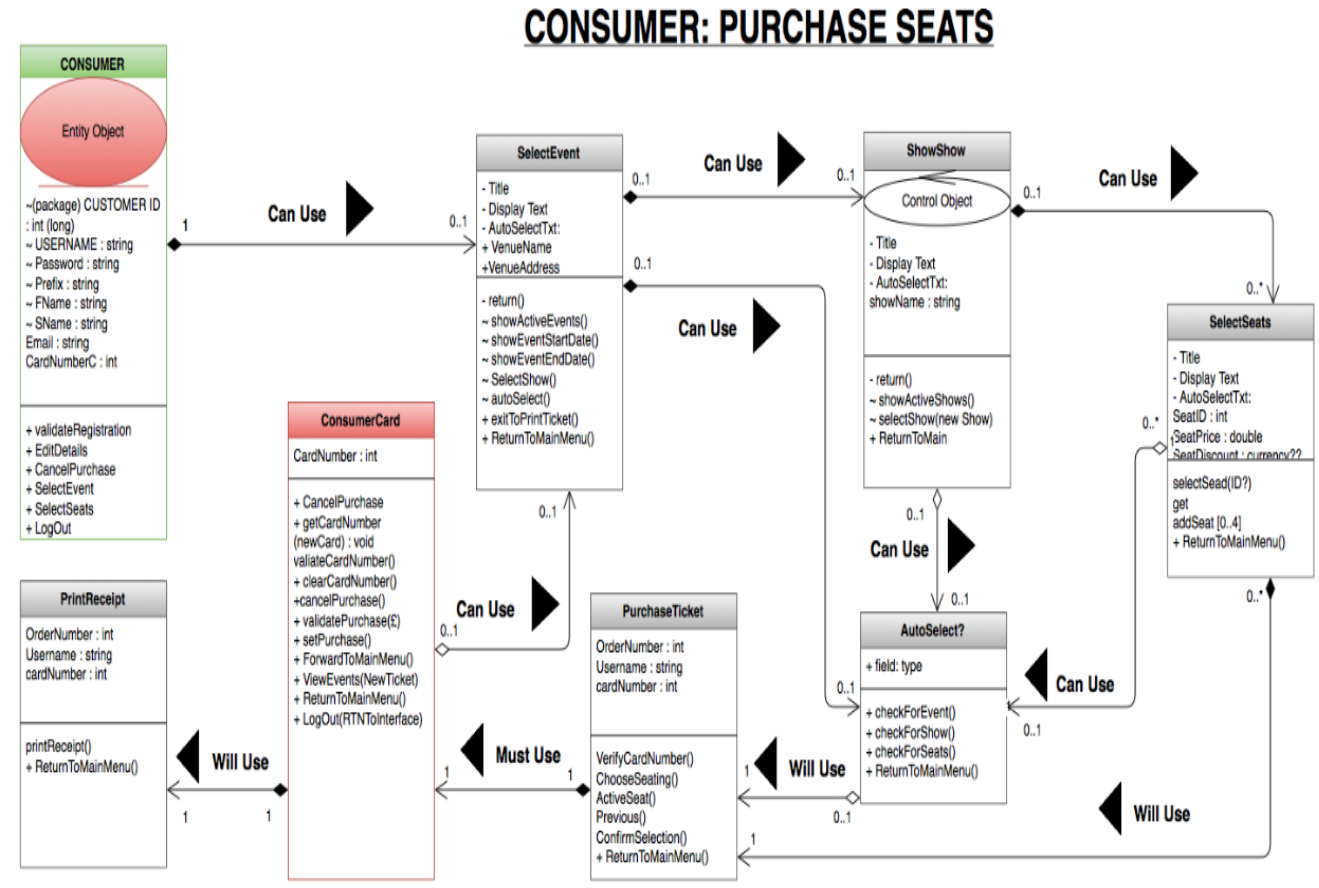
Step 3: Show the show, which would display things such as the name title of the show.

Step 4: The user would select the seats for the performance they want, and choosing the price brackets of junior student and adult.

Step 5: Does the user want the best available seats or choose a precise area row and block.

Step 6: The consumer will input their card details, such as card number. These will be validated with the system of records, to make sure the transaction matches the consumer's record actually stored on the system.

Step 7: Final stage is print out the consumer's receipt for the show they have bought tickets for.

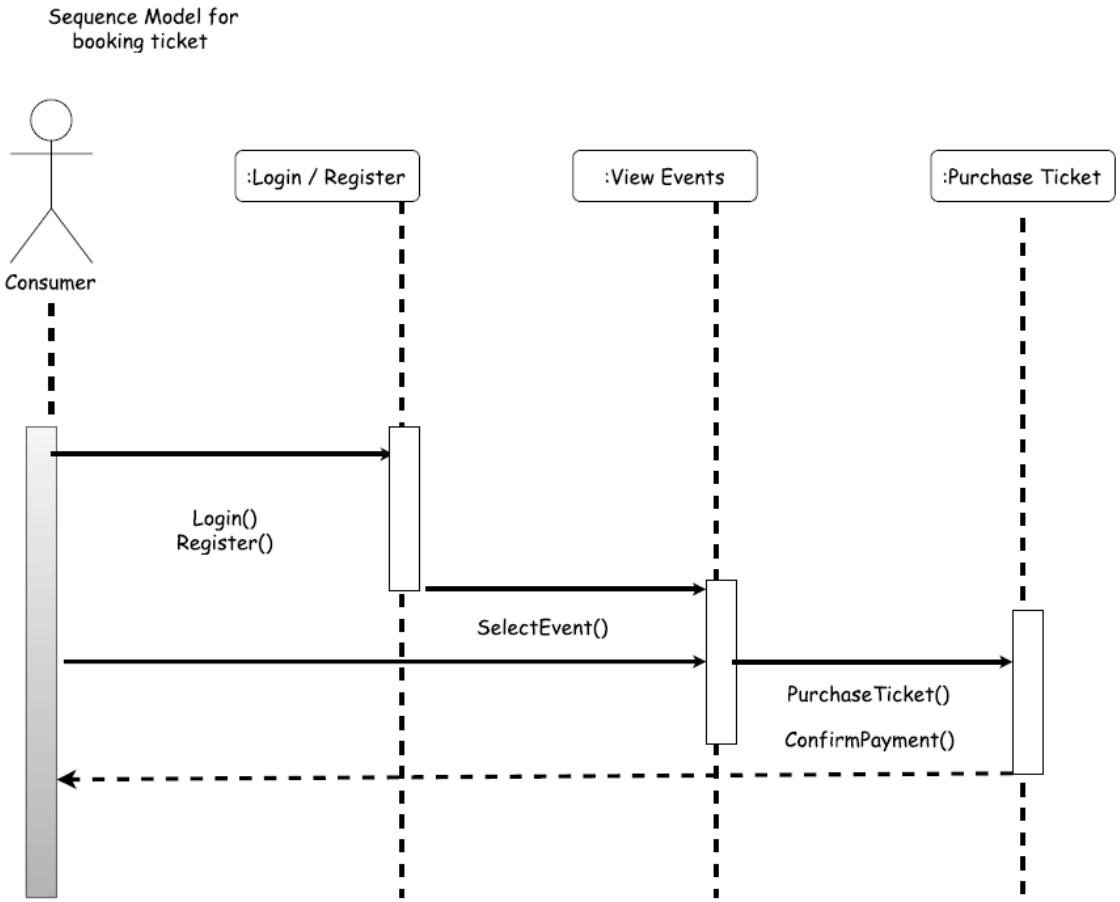


Sequence Model Overview

Sequence model is to show the interaction between objects. This will help developers to figure out the methods that needed for specific process. There is one huge problem that we have met during working on this, which we are having misunderstanding it. At first we done the diagram in wrong way, we made it into activity diagram. When we done the activity diagrams, luckily we have checked it again and found this big mistake. Since activity diagram and sequence model are not the same thing, one is more about the system and the other is more about the users. So to get this done correctly, we all sat together to do the works again. After our hard works and time spend, we finally get the right diagrams done. The paragraphs below are the explanation of the sequence diagrams that done by us.

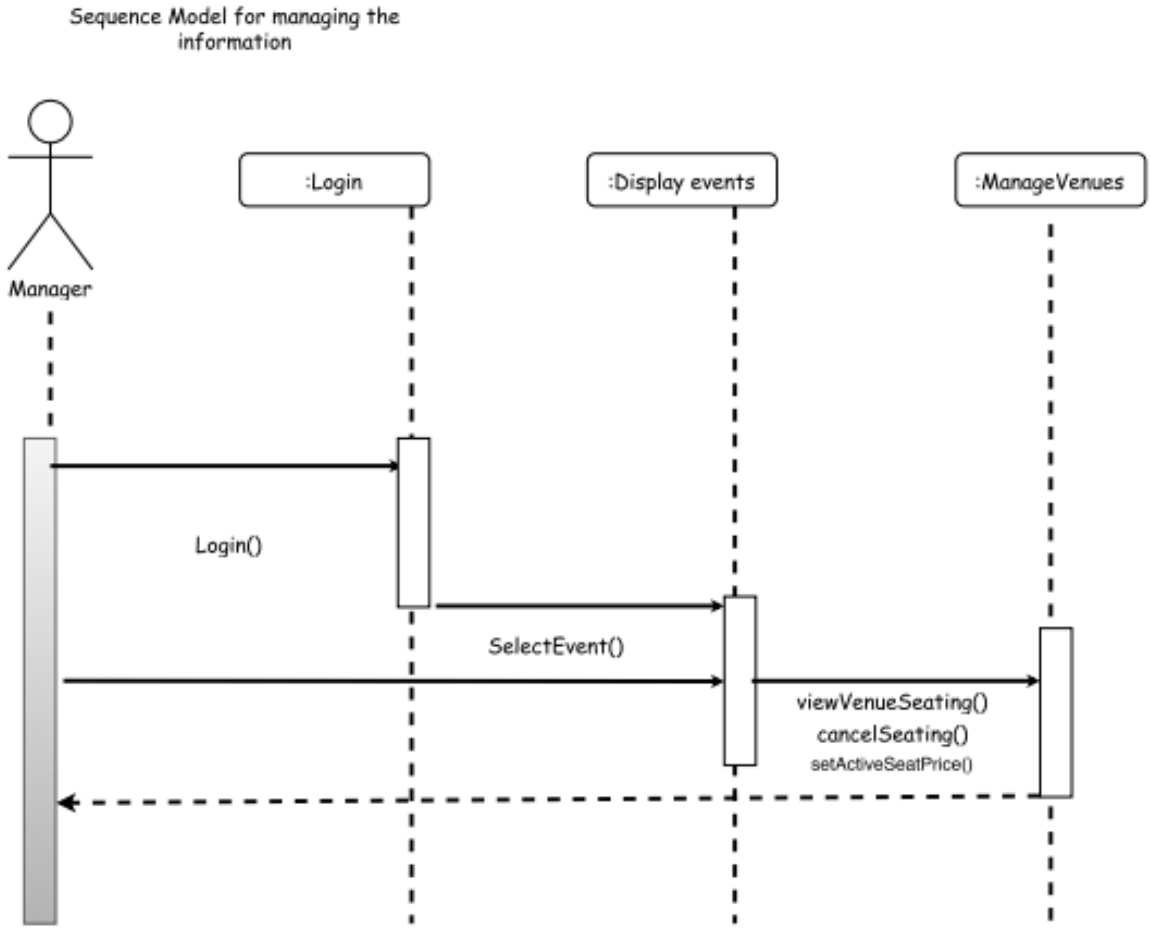
Sequence Model – Booking a ticket

One of the sequence models we have done is the process of booking ticket. The objects involved in it, included the actor consumer, login, view events, and purchase. The first arrow in the diagram shows the first thing require consumer to do is login or register on the system, to allow them to access the system. The second thing is select event that want to view, and then purchase ticket for the show. At the end there is a dashes arrow, which is to show the system will return back the info to actor. There is an arrow between actor and view event, this is to show that actor can view event's details without doing the login or register step. For this step, we're assuming that actor has already logged on.



Sequence Model – Managing the information

Another sequence model we have done is about managing information. The actor involved is the venue manger, the other objects that contained in the diagram are login, display events, and manage venue. This diagram is similar to the previous one. It's also included four objects in total. The first two steps are same as the previous diagram, which also require actor to login and select the event to do the final step. And again, we also made an assumption in this diagram, which is assume that the actor had logged on to the system before doing this process. The final step in the diagram is managing the info. To complete this step, there are three methods that allow venue manager to do. The manager doesn't have to operate all three methods. They can only choose the function of needed to do the task. After these steps went through, the system will return back the information to actor.



Pseudo code

Pseudo code is the section that done before the implementation. This is the algorithm for the classes and methods that need to be create later. This will help us to do the implementation, because these are the steps when creating the methods, and also this will have the method name, variables name and parameter that should be include. This will save us time when implementation. And it is lucky that when we doing this task didn't meet any huge problem.

Manager class

1. Declare the code for using array list.
2. Declare the name for constructors that will need to be use later.
3. Declare the name for array list.
4. Declare variables that will need to use later.
5. In default class:
    a. Set new constructor for Purchase class.
    b. Set new constructor for Cancel Purchase class.
    c. Set new constructor for Manage Event class.
    d. Set new constructor for Promotion class.
    e. Set array list to store data in Manage Event class.
6. Set new method called setDiscount (int discount)
    a. Call set discount method from Promotion class.
    b. Assign value to the parameter.
7. Set new method called buy (int buy)
    a. Call buy method from Purchase class.
    b. Assign value to the parameter.
8. Set new method called pay (int pay)
    a. Call pay method from Purchase class.
    b. Assign value to the parameter.
9. Set new method called checkBalance ()
    a. Call get balance method from Purchase class.
    b. Assign value to the parameter.
10. Set new method called refund (int refund)
    a. Call refund method from Cancel Purchase.
    b. Assign value to the parameter.
11. Set new method called editName (String name)
    a. Call set name function from Manage Event class.
    b. Assign value to the parameter.
    c. Add the value to array list.
12. Set new method called editMax (int max)
    a. Call set max function from Manage Event class.
    b. Assign value to the parameter.
    c. Add the value to array list.
13. Set new method named editPrice (int price)
    a. Call set price function from Manage Event class.
    b. Assign value to the parameter.
    c. Add the value to array list.

14. Set new method named printRefund ()
    a. Call print refund method from Cancel Purchase class.
    b. Print details for refund.
15. Set new method named printEvent()
    a. Call print event's details method from Manage Event class.
    b. Print details for event's details.
16. Set new method named printTicket()
    a. Call print ticket details method from Purchase class.
    b. Print details for ticket's details.

ManageEvent class

1. Declare variables that will need to use later.
2. In default class, set parameters' name(String event name, int price, int max) :
    a. Assign default value for parameters.
3. Set a method named setName (String newName), which set new name for the event.
    a. Assign value for event name.
4. Set a method named setMax (int newMax), which set the max of tickets that allow to purchase each time.
    a. Assign value for the max number.
5. Set a method named setPrice (int newPrice), which set price for ticket.
    a. Assign value for the price.
6. Set a method named getPrice(), which get the price of ticket.
    a. Return value of price.
7. Set a method named getMax() , which get the max number.
    a. Return value of max.
8. Set a method named printEventDetails()
    a. Print the details that been stored on screen.

Purchase class

1. Declare name for constructors that will create later.
2. Declare variables that will use later.
3. In default class:
    a. Create new constructor for ManageEvent class.
    b. Create new constructor for Promotion class.
4. Set new method for buyTicket(int number of ticket)
    a. If number of ticket is less than or equal to the max number
        i. Calculate the total cost of it.
        ii. Apply discount to it, depending on is there a discount or not.
        iii. Print out the details.
    b. Else
        i. Print a warning message.
5. Set new method to getBalance()
    a. Return the value of balance.
6. Set a pay method called insertMoney (int amount)
    a. If the amount is greater than zero.
        i. Assign the amount value to balance.

b. Else
    i. Print out a warning message.

7. Set a printTicket() method.
    a. If balance is greater than or equal to the price.
        i. Print out the details of purchase.
        ii. Assign new value to the total.
        iii. Assign new value to the balance.
    b. Else
        i. Print a warning message.

CancelPurchase class

1. Declare the name for constructor that will create later.
2. Declare variables.
3. In default class:
    a. Create constructor for Purchase class.
4. Set a cancel(int amountToRefund), which is a method for refund.
    a. Calculate how much left in balance after refund.
5. Set a method named print()
    a. Print refund details.

Promotion class

1. Declare variables.
2. In default class:
    a. Set default value for discount.
3. Set method named setPromo(int discount)
    a. Assign new value to discount.

Consumer class

1. Declare name for constructor that will create later.
2. Declare variables and set default value to it.
3. In default class:
    a. Create new constructor for Purchase class.
4. Set new method called buy (int Buy)
    a. Call the buy method from Purchase class.
    b. Assign new value to the variable of number of ticket.
5. Set new method called pay (int Pay)
    a. Call the pay method from Purchase class.
    b. Assign new value to the variable of money have pay.
6. Set new method named checkBalance()
    a. Call check balance method from Purchase class.
    b. Return value.

Agent class

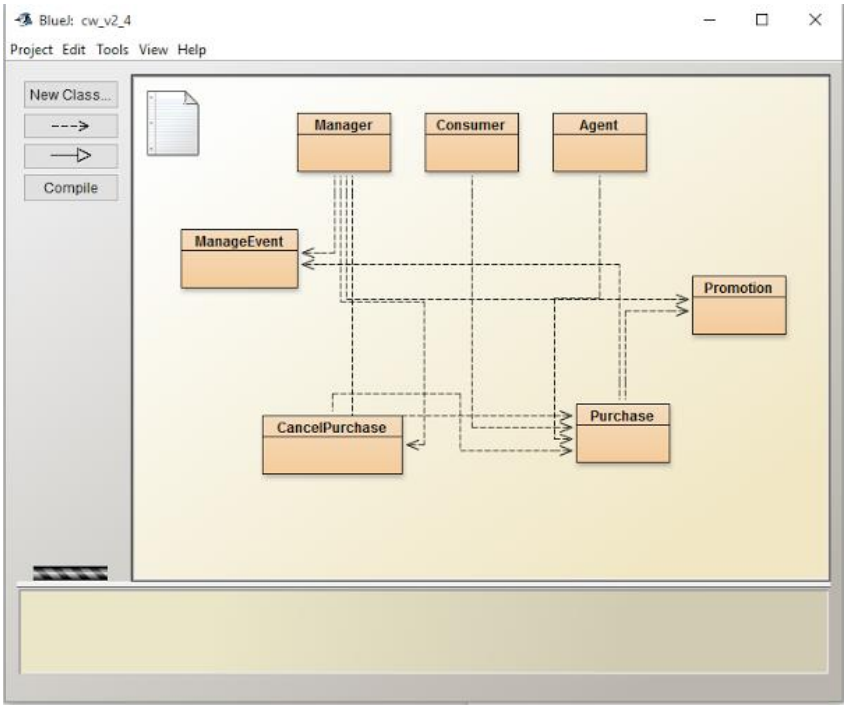
1. Declare name for constructor that will create later.
2. Declare variables and set default value to it.
3. In default class:
    a. Create new constructor for Purchase class.

4. Set new method called buy (int Buy)
    a. Call the buy method from Purchase class.
    b. Assign new value to the variable of number of ticket.
5. Set new method called pay (int Pay)
    a. Call the pay method from Purchase class.
    b. Assign new value to the variable of money have pay.
6. Set new method named checkBalance()
    a. Call check balance method from Purchase class.
    b. Return value.

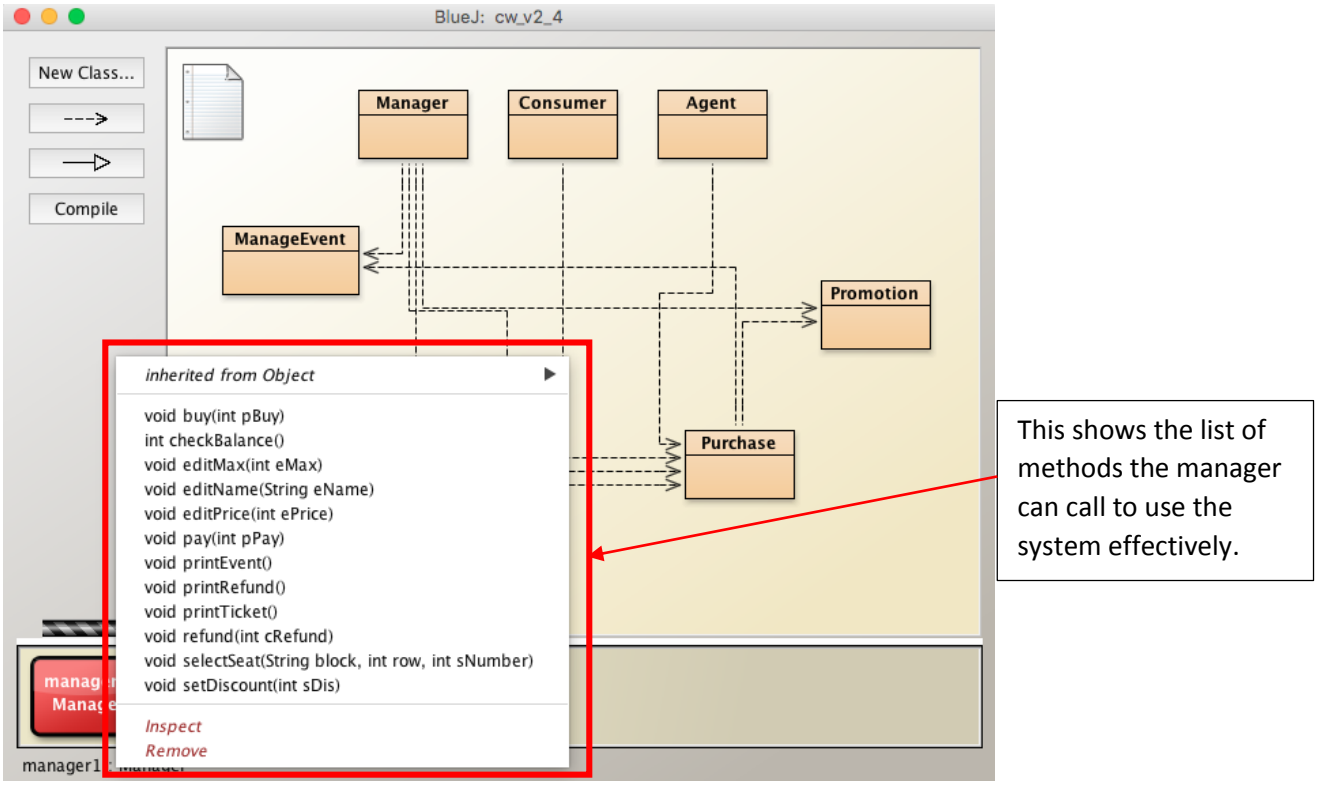
Implementation

When we start the implement the system in Blue J, the first thing we do is write all the code in class, just to make sure it will be working. After we have tried the code are working, we then start to make it into different classes. To make a class able to use methods or variables from other classes, will need to set new constructor for it. There was a big problem occur during the process, which when compile the class it pop up a error message about over flow stack. Since there is no other message that can help to understand what's going on, we then copy the code from class to a new project one at a time, to find out which part is going wrong. After found out the part that causes error, we done some research on Internet, to find out solution for over flow stack error. After looking at the explanation on Internet and compare it with the exercises about java that we have done before, we finally understand that the error is because of the we are trying to make some of the classes get data for some sort of variable from the Manager class, and this is not allow since the Manager class is using the methods from the classes. The things of we try to do will make the class going to a situation like loop. To solve this problem, we make the data that need to access by other classes being store in the Manager Event class as well. Apart of this big problem, we did also met some other problems, such as using wrong parameters in method for getting variables or calling methods from other classes. We do also done some silly mistakes such as forgot to put in the parameters or forgot to declare the variables at the beginning of the coding.

When we think this is all done, we start to do some testing for the functions. We then found out the Purchase class was not working correctly. No matter what is the max number of tickets per purchase, it is also saying "it should be no more than 0. " This is the message created by us, which is should be only to be show when the number of tickets that going to be sell are more than the max number that been set. The message print on screen and saying no more than 0, that is meant the Purchase class was not getting data from Manage Event class. That's why the number of showing is the default number, 0. We then urgently inspect the classes to try to fix the problem. This is because if the Purchase class is not able to access data from Manage Event class, it will make the Cancel Purchase class have wrong data. Unfortunately we can't find out why this is happening and we don't have enough time, so we do have to give up on fixing it. Otherwise, all the methods are working and didn't cause any error.



This image shows the user interface, which meant when users open up the system, this is what they will see on screen. It may seem a bit complex, but users will mainly and should be only use the classes at top. There is a ReadMe file as well; inside the file has the description and user instruction for this system, so it should be not a problem for users to understand how to use the system.



This image shows how it will be look like after a new object has been created. When users right click on the object, there will be a list of methods that could be do by the object. All the methods are named a name that is easy understanding, this will make it easy for the users to know which method do what. Once users have selected the method of they would like to use, it will pop up a box for users

to input data. All methods are with description about what is expected to be input, and this will be show in the box for input data. We believe everything is easy to understand since it is all with descriptions.

Implemented code

Manager class

```
import java.util.ArrayList;

/**
 * Description: A class for the venue manager.
 * This class allows venue manager to manage the event's details.
 * There are methods in this class are calling from the other class.
 *
 * @author: Robert Collcott, Elaine Foon and Temi Dare
 * @version 2.4
 *
 */

public class Manager
{
   // declare the costructor that will need to be use later.
   private Purchase buyIt;
   private CancelPurchase cancelP;
   private ManageEvent manage;
   private Promotion promo;
   //array list
   private ArrayList<ManageEvent> data;

   // Store variable.
   private int checkTheBalance;
   private int BuyTic=0;
   private int Payment=0;
   private int RefundIt=0;
   private int Max=0;
   private int Price=0;
   private String Name;
   private int totalDiscount;
   private int price;
   private int max;
   private String eventName;
   private int disc;
   public int rowNo;
   public int seatNumber;
```

```
public String blockS;


// method from Manager class
/**
 * This is the own (default) class of this.
 * In here will create constructors, to make it able to call methods from other classes.
 */
public Manager()
{
   buyIt= new Purchase();
   cancelP= new CancelPurchase();
   manage= new ManageEvent(eventName, price, max);
   promo= new Promotion();

   data= new ArrayList<ManageEvent>();
}

 /**
 * select seat that wish to buy.
 */
public void selectSeat(String block, int row, int sNumber)
{
   blockS= block;
   rowNo= row;
   seatNumber= sNumber;

   System.out.println("Block: "+ blockS);
   System.out.println("Row: "+ rowNo);
   System.out.println("Seat number: "+ seatNumber);
}


//method from promotion class
/**
 * method to set the discount.
 */
public void setDiscount(int sDis)
{
 promo.setPromo(sDis);
 disc= disc+ sDis;
}


// methods from Purchase class
```

```
/**
 * this is the method of purchase ticket.
 */
public void buy(int pBuy)
{
   buyIt.buyTicket(pBuy);
   BuyTic= pBuy;
}

/**
 * method of pay for the ticket(s).
 */
public void pay(int pPay)
{
   buyIt.insertMoney(pPay);
   Payment=pPay;
}

/**
 * check the balance of how much have been pay.
 */
public int checkBalance()
{
   checkTheBalance= buyIt.balance;
   return checkTheBalance;
}


// from CancelPurchase class
/**
 * method for process the refund.
 */
public void refund(int cRefund)
{
   cancelP.cancel(cRefund);
   RefundIt=cRefund;
}

// from ManageEvent class
/**
 * set event's name
 */
public void editName(String eName)
{
   manage.setName(eName);
```

```
    Name=eName;
    data.add(new ManageEvent(Name, Price, Max));
  }

  /**
   * set max number of ticket that allow to buy each time.
   */
  public void editMax(int eMax)
  {
    manage.setMax(eMax);
    Max= eMax;
    data.add(new ManageEvent(Name, Price, Max));
  }

  /**
   * set price for the event.
   */
  public void editPrice(int ePrice)
  {
    manage.setPrice(ePrice);
    Price= ePrice;
    data.add(new ManageEvent(Name, Price, Max));
  }


  // Print methods from all classes
  /**
   * print details about refund
   */
  public void printRefund()
  {
    cancelP.print();
  }

  /**
   * print ticket/purchase details
   */
  public void printTicket()
  {
    buyIt.printTicket();
  }

  /**
   * print details about the event
   */
```

```
    public void printEvent()
    {
        manage.printEventDetails();
    }
}
```

Consumer class

```
/**
 * Write a description of class Consumer here.
 *
 * @author: Robert Collcott, Elaine Foon and Temi Dare
 * @version 2.4
 */
public class Consumer
{
    private Purchase buyIt;
    public int BuyTic=0;
    public int Payment=0;
    public int checkTheBalance;

    /**
     * Constructor for objects of class Consumer
     */
    public Consumer()
    {
        buyIt= new Purchase();

    }

    // from Purchase class
    /**
     * this is the method of purchase ticket.
     */
    public void buy(int pBuy)
```

```
    {
        buyIt.buyTicket(pBuy);
        BuyTic= pBuy;
    }

    /**
     * method of pay for the ticket(s).
     */
    public void pay(int pPay)
    {
        buyIt.insertMoney(pPay);
        Payment=pPay;

    }

    /**
     * check the balance of how much have been pay.
     */
    public int checkBalance()
    {
        checkTheBalance= buyIt.balance;
        return checkTheBalance;

    }
}
```

Agent class

```
/**
 * Write a description of class Consumer here.
 *
 * @author: Robert Collcott, Elaine Foon and Temi Dare
 * @version 2.4
```

```
 */
public class Agent
{
  private Purchase buyIt;
  public int BuyTic=0;
  public int Payment=0;
  public int checkTheBalance;

   /**
    * Constructor for objects of class Agent
    */
   public Agent()
   {
      buyIt= new Purchase();

   }

   // from Purchase class
   /**
    * this is the method of purchase ticket.
    */
   public void buy(int pBuy)
   {
      buyIt.buyTicket(pBuy);
      BuyTic= pBuy;
   }

   /**
    * method of pay for the ticket(s).
    */
   public void pay(int pPay)
   {
```

```
        buyIt.insertMoney(pPay);
        Payment=pPay;

    }

    /**
     * check the balance of how much have been pay.
     */
    public int checkBalance()
    {
        checkTheBalance= buyIt.balance;
        return checkTheBalance;

    }
}
```

Manage Event class

```
/**
 * Write a description of class ManageEvent here.
 *
 * @author: Robert Collcott Elaine Foon and Temi Dare
 * @version 2.4
 */
public class ManageEvent
{
    // variables
    public String newName;
    public int newMax;
    public int newPrice;
    public int price;
    public int max;
    public String eventName;
```

```
/**
 * Constructor for objects of class ManageEvent
 */
public ManageEvent(String eventName, int price, int max)
{
  this.eventName= eventName;
  this. price= price;
  this. max= max;
}

/**
 * set event's name.
 */
public void setName(String newName)
{
   eventName= newName;
}

/**
 * Input the max number of tickets that allow to buy for current event.
 */
public void setMax(int newMax)
{
   max= newMax;
}

/**
 * Input the price for ticket.
 */
public void setPrice(int newPrice)
{
```

```
    price = newPrice;
}

 /**
 * Return The price of a ticket.
 */
public int getPrice()
{
    return price;
}

/**
 * Display the max number of tickets that allow to buy for current event.
 */
public int getMax()
{
    return max;
}



/**
 * print event's details
 */
public void printEventDetails()
{
    System.out.println();
    System.out.println();
    System.out.println("--------------------");
    System.out.println("Event's name: " + eventName);
    System.out.println("Price: £" + price);
    System.out.println("Max per purchase: " + max);
```

```
        System.out.println("--------------------");
    }
}
```

Purchase class

```
/**
 * Description: A class to store all the methods for buy ticket.
 *
 * @author: Robert Collcott, Elaine Foon and Temi Dare
 * @version 2.4
 */
public class Purchase
{
    // Call variable from ManageEvent class.
    private ManageEvent manage;
    // call method/ variable from promotion class.
    private Promotion promo;
    // The total of ticket of buying.
    public int noTicket=0;
    // The amount of money entered by a customer so far.
    public int balance=0;
    // The total amount of money collected by this machine.
    public int total=0;
    // amount of money
    public int amount=0;

    //declare variable
    public String eventName;
    public int price;
    public int max;
    public int rowNo;
    public int seatNumber;
```

```
public String blockS;
public int numberTicket;

/**
 * Constructor for objects of class Purchase
 */
public Purchase()
{
   manage= new ManageEvent(eventName, price, max);
   promo= new Promotion();
}

/**
 * Return the number of ticket that want to buy.
 */
public void buyTicket(int noTicket)
{
   numberTicket= noTicket;

   if(noTicket<=manage.max)
   {
   total= noTicket * manage.price;
   total= total- promo.discount;

   System.out.println("");
   System.out.println("-----------------------------");
   System.out.println("You have buy "+ noTicket +" ticket(s).");
   System.out.println("Discount: £"+ promo.discount);
   System.out.println("Total cost: "+ total);
   System.out.println("-----------------------------");
   System.out.println("");
   }
```

```
    else
    {
        System.out.println("");
        System.out.println("----------------------");
        System.out.println(" Should be no more than "+ manage.max);
        System.out.println("----------------------");
        System.out.println("");
    }
}

 /**
 * Return The amount of money already inserted for the next ticket.
 */
public int getBalance()
{
    return balance;
}

/**
 * Receive an amount of money from a customer.
 * Check that the amount is sensible.
 */
public void insertMoney(int amount)
{
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount rather than: " +
                           amount);
    }
}
```

```
 /**
  * Print a ticket if enough money has been inserted, and
  * reduce the current balance by the ticket price. Print
  * an error message if more money is required.
  */
 public void printTicket()
 {
    if(balance >= manage.price) {
        // Simulate the printing of a ticket.
        System.out.println("");
        System.out.println("");
        System.out.println("You have buy "+ noTicket +" ticket(s).");
        System.out.println("####################");
        System.out.println("Discount: £"+ promo.discount);
        System.out.println("####################");
        System.out.println("Cost in total: " + manage.price + " cents.");
        System.out.println("####################");
        System.out.println("");

        // Update the total collected with the price.
        total = total + manage.price;
        total= total- promo.discount;
        // Reduce the balance by the prince.
        balance = balance - manage.price;
    }
    else {
        System.out.println("You must insert at least: " +
                           (total - balance) + " more cents.");

    }
 }
```

```
}
```

Promotion class

```
/**
 * Write a description of class Promotion here.
 *
 * @author: Robert Collcott, Elaine Foon and Temi Dare
 * @version 2.4
 */
public class Promotion
{
   // declare the  variable
   public int discount;

   /**
    * Constructor for objects of class Promotion
    */
   public Promotion()
   {
      discount= 0;
   }

   /**
    * method of promotion for saturday (evening) shows
    */
   public void setPromo(int disOne)
   {
    discount= discount+ disOne;
   }

}
```

Cancel purchase class

```
/**
 * Write a description of class CancelPurchase here.
 *
 * @author: Robert Collcott, Elaine Foon and Temi Dare
 * @version 2.4
 */
public class CancelPurchase
{
   // call variable from purchase clss.
   private Purchase cBalance;
   // the amount left in balance.
   public int left;
   //amount to refund
   public int amountToRefund;

   /**
    * Constructor for objects of class CancelPurchase
    */
   public CancelPurchase()
   {
      cBalance= new Purchase();
   }

   /**
    * Refund method.
    */
   public void cancel(int amountToRefund)
   {
      left= cBalance.balance - amountToRefund;
```

```
    }


    /**
     * Print details
     */
    public void print()
    {
        System.out.println("");
        System.out.println("------------------------");
        System.out.println(" Amount to refund: " + amountToRefund);
        System.out.println(" Balance: " +left);
        System.out.println("------------------------");
        System.out.println("");
    }
}
```

Testing

Test no.	Description	Input value	Expected result	Actual result
1.	Buy ticket method	3	3	Should be no more than 0
2.		0	0	0
3.	Check balance method	60	Return balance, 60	Return balance, 60
4.		0	Return balance, 0	Return balance, 0
5.	Edit max method	5	5	5
6.		0	0	0
7.	Edit name method	"XXX Live"	XXX Live	XXX Live
8.		0816	Error message	Error message
9.	Edit price method	30	30	30
10.		0	0	0
11.	Pay method	60	60	60
12.		0	0	0
13.	Set discount method	10	10	10
14.		0	0	0
15.	Select seat method	"G", 3, 11	Block: G Row: 3 Seat number: 11	Block: G Row: 3 Seat number: 11
16.		1, "r", 12	Error message	Error message
17.	Refund method	30	30	30
18.		0	0	0
19.	Print refund details method	N/A	Print refund details like the total.	Print refund details like the total.
20.	Print event details method	N/A	Print event's details like price, event name and max number.	Print event's details like price, event name and max number.
21.	Print ticket details method	N/A	Print ticket details like total cost and number of ticket bought.	Print ticket details like total cost and number of ticket bought.

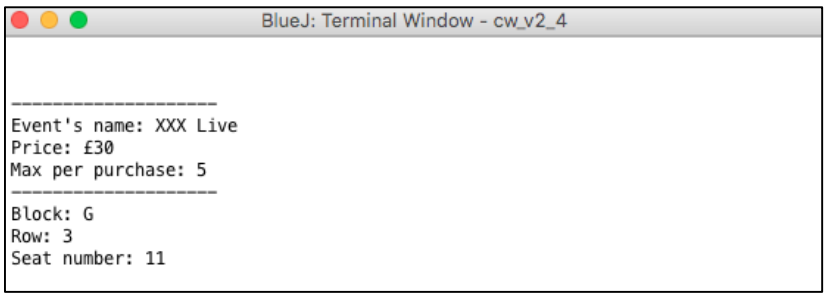


Figure 1: result for test no. 4, 5, 6, 15, 20

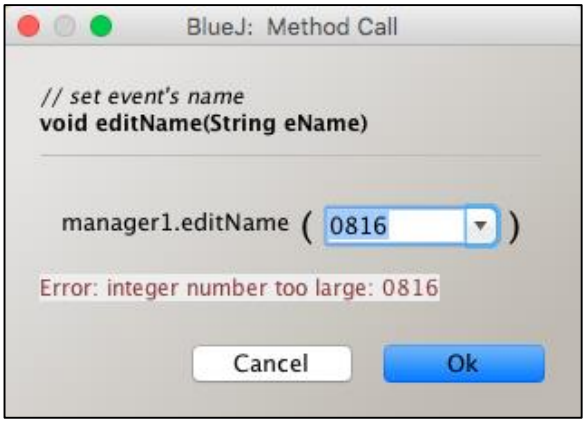


Figure 2: result for test no. 7

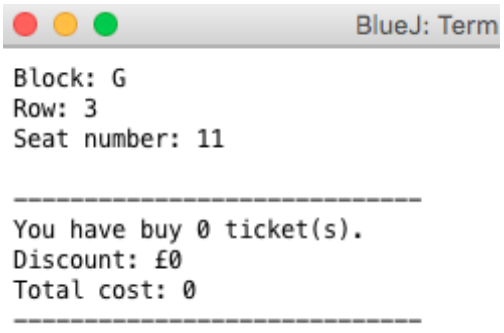


Figure 3: result for test no. 14, 21

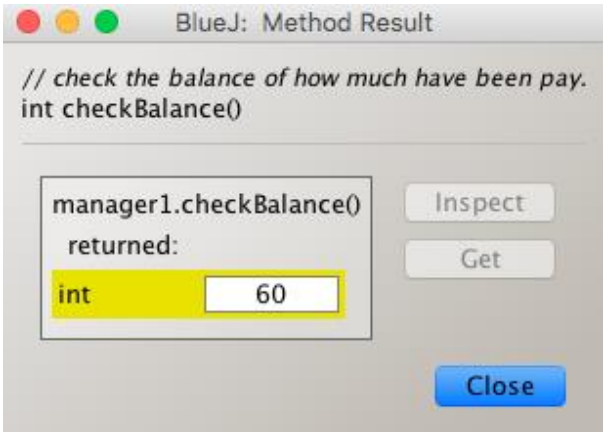


Figure 4: result for test no. 11

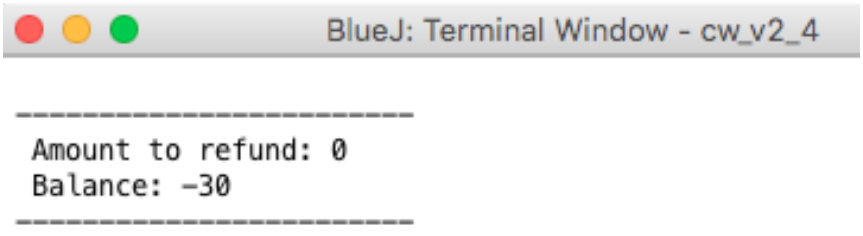


Figure 5: result for test no. 19

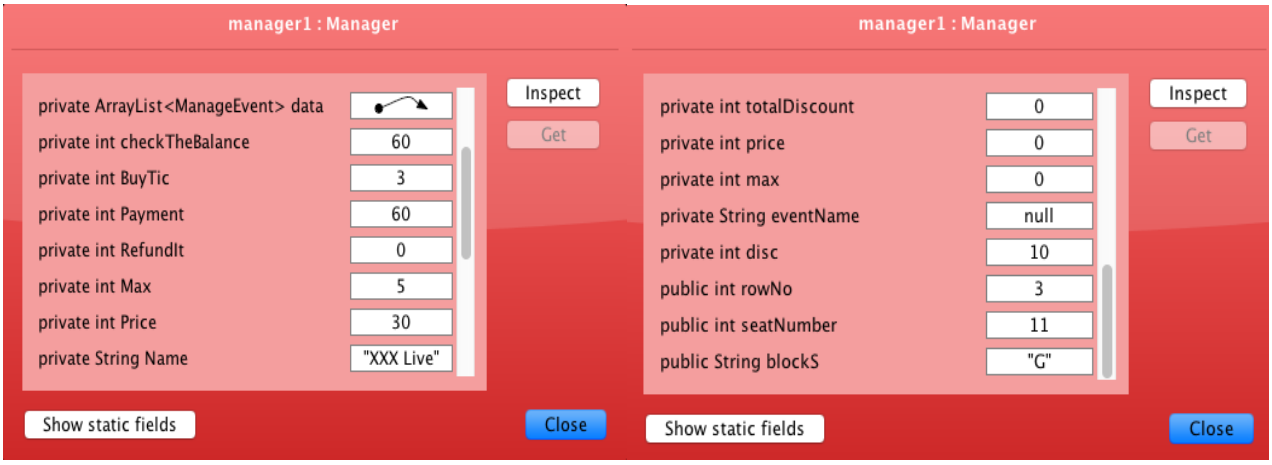


Figure 6: this shows the data that been stored in the manager class.

Implementation phase Appendix 1 Meeting Minutes References

Meeting 1 – 1st December 2015

Discussed about the sequence model outline what should be included on it, and have started on doing the sequence model and requirement statement.

Meeting 2 – 2nd December 2015

Continue working on the sequence model, as it is not done on last time. And also get the requirement statement task done.

Meeting 3 – 4th December 2015

We started discussing about the partial class diagram, the things we have talked about were what is it, what should be in it, and also how it will be look like, as we're not too sure the different between partial class diagram and class diagram. Temi take note on all those things to help him to create the diagram. After that we make a start on the actual system, which is create a project in Blue J and get the basic description about the project done in the ReadMe file.

Meeting 4– 8th December 2015

We did the pseudo code task in this meeting. The duration of meeting was more than an hour, but we didn't the pseudo code task done, because of we have to finishing off some other task in section a.

Meeting 5– 10th December 2015

We have this meeting on WhatsApp group chat to talk about the partial class diagram task. Because of Temi having some question about the task, so Elaine and Robert have tried to help him. Apart of giving idea and suggestion, also send some images of some drawing of simple diagram.

Meeting 6– 14th December 2015

In this long meeting, we have get the pseudo code task done, and start to do the coding for the project. The implementation does require a lot of time to do, because of always getting error when compile it after some new codes has added.

Meeting 7 – 15th December 2015

The first thing we did on this day is continuing the implementation, since the other tasks were done and Temi was not arriving yet in the early. After Temi arrived, Robert and Elaine also show and explained the things had done before he arrives. Temi did give some suggestion on the system too. After that Temi then start to show us about what he has done for the partial class diagram, and we review the work together.

Meeting 8 – 16th December 2015

On this day, we're focus on the implementation, as there are some problems on the method. We all sat together to try to figure out the problem, and also tried to find to solution for it. But unfortunately, we can't get the problem solved, so we have to leave it since we don't have much time left before the deadline.

Meeting 9 – 17th & 18th December 2015

This is the last meeting for the assignment. After we have done the testing on this day, we then do a final check on everything and finishing off the report as well. After that all done, the report do let all team member to have a look, when we all agree on the work, it is ready to be submit.

Implementation phase Appendix 2 Meetings Record

Meeting no.	Date	Attendees	Topics Discussed	Duration
1.	1st December 2015	Robert, Elaine and Temi	Sequence Model Part 1	2 Hours
2.	2nd December 2015	Robert and Elaine	Sequence Model Part 2	45 mins
3.	4th December 2015	Robert and Elaine	Partial class diagram and actual system	1 Hour 30 mins
4.	8th December 2015	Robert and Elaine	Pseudo code	1 Hour 45 mins
5.	10th December 2015	Robert, Elaine and Temi	Partial class diagram	2 hours
6.	14th December 2015	Robert, Elaine and Temi	Finish off the pseudo code task, and start implementing the actual system.	3 Hour 30 mins
7.	15th December 2015	Robert, Elaine and Temi	Implementing the actual system. Also partial class diagram explanations and finalizing them.	3 Hours
8.	16th December 2015	Robert, Elaine and Temi	Implementing the actual system.	3 Hours
9.	17th December 2015	Robert, Elaine and Temi	Testing the actual system, finishing the whole report and final check for everything have done.	3 Hours
10.	18th December 2015	Robert, Elaine and Temi		1 Hour