CFS2160 Software Design and Development

Cinema Booking System

C.J.Fletcher U1863522

Table of Contents

Modelling – See Appendices

Use Case

I first created a Use Case diagram to identify features which I wished to focus on. After some consideration I decided to focus on the Employee aspect of a Cinema Booking System.

Class Diagram

I strived to implement Model-View-Controller architecture. This made it easy to make changes to my classes as I saw fit as my classes had 'low coupling'.

My class diagram changed the most throughout the development process. I have omitted my FXMLController classes and instead created a class called GUI Controller to symbolise them. This was to reduce clutter on the diagram.

My model Classes have attributes of Objects of other classes e.g. Showing has attributes of Film and Theatre. Ticket has attribute of a Showing. This made it very easy to get necessary information and reduced duplicating information amongst my classes.

Activity Diagram

I created an activity diagram to cover the main function of the Cinema Booking System, buying tickets and from the tickets, creating bookings. This assisted in better understanding how my system would flow.

Sequence Diagrams

My sequence diagrams show that whenever the system is used a relevant Controller method is called which, in turn, calls methods of other relevant Controller classes, which if necessary creates or updates my Model objects, keeping in line MVC architecture I wished to implement.

Initial development

I started by searching for a way to retrieve film data easily, rather than having to populate the data myself. I found http://www.omdbapi.com/ which returned the film data in Json format. In order to parse this information I had to learn about retrieving information from URLs and how to parse a Json formatted String into an object[2].

All of my controllers and model classes are stored in separate packages for easier navigation.

I store my films, by IMDB ID, in a .txt file and read through them line by line[2]. I store the rest of my data in Serialised .dat files. Upon starting my GUI development I wished to replicate Material Design UI (used on android) and so found a library that enabled me to do so[2].

Data Storage/Persistence

My application data persists between runs, with the use of Serialisation, however I am not sure the way I have implemented it is best practice.

If I were to implement this again look to implement a way to serialise all of my Controllers at the same time, into the same file to avoid multiple instances of objects or to implement a database. According to D.Miller (1999)[1] deserialisation creates hard copies of objects e.g. when serialising my ArrayList of Films and Showings (separately), the Film attribute of the Showing class would be a separate object from the serialised Film object. This meant, in between runs of my application, I couldn't use showingObject.getFilm() to reference the relevant Film object in my Film controller as it will have created a separate Film object.

Passing data between GUI Screens

In my Main I created a Static instance of my Model Controllers and also of my Basket, Showing & User classes, with getters and setters. This made it easy to update information across any GUI screen and hide certain buttons based on the current user i.e. Main.getBasket.add(itemToAdd) or Main.setCurrentUser(admin).

I'm uncertain if this is best practice for a JavaFX application but found this to be the best way to manage passing data from screen to screen.

Password storage

I currently do not hash the passwords at all and would like to implement password hashing in the future.

Testing

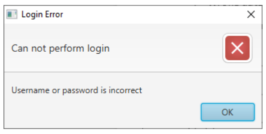
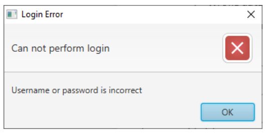
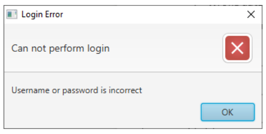
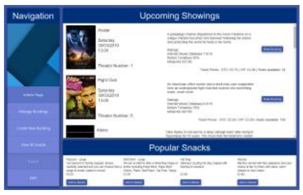
Throughout development I tested my methods by printing the outputs to screen and checking that the outcome is as I expected, I tested both with erroneous and correct data. I also did further tests, seen in the tables below, once I had finished my application

Furthermore I had three of my friends, two programming savvy and one not, to test my application for errors and for usability. I found a few bugs in my application this way and applied fixes as necessary.

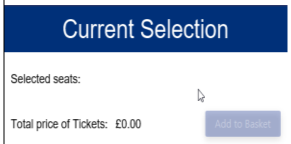
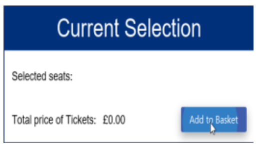
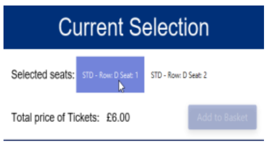
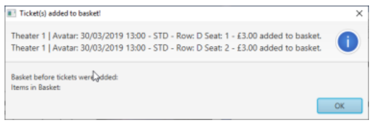
Legend

T – Correct data | F - Wrong data | E - Error message displayed

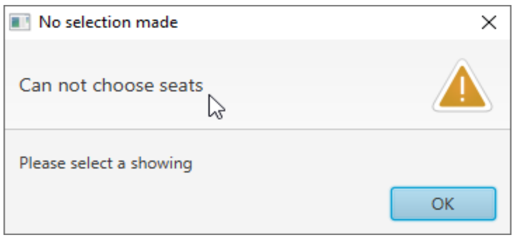
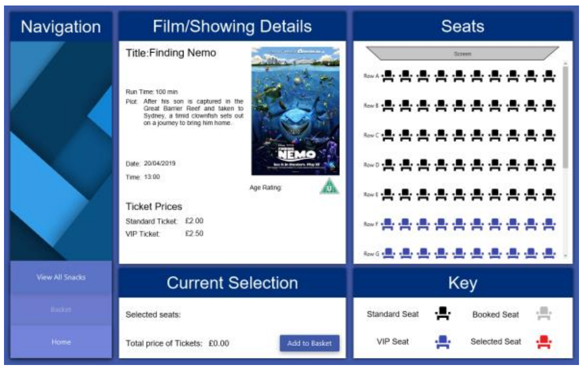
Login Test – Pressing login button

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Expected Outcome	Error message	Error message	Error message	Home page shown
Actual Outcome	 Login Error Can not perform login Username or password is incorrect OK Error message	 Login Error Can not perform login Username or password is incorrect OK Error message	 Login Error Can not perform login Username or password is incorrect OK Error message	 Home page shown

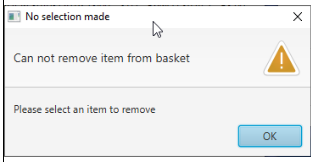
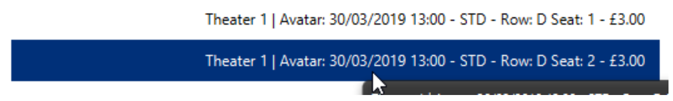
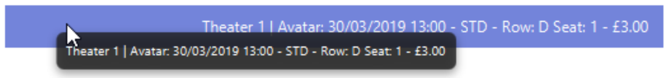
Pressing Add to Basket button on showing page

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Seats selected T/F	F	F	T	T
Checked customer age checkbox (T/F)	F	T	F	T
Expected Outcome	Add to basket button disabled	Add to basket button enabled, nothing on click	Add to basket button disabled	Added to basket alert shown
Actual Outcome	 Current Selection Selected seats: Total price of Tickets: £0.00 Add to basket button disabled	 Current Selection Selected seats: Total price of Tickets: £0.00 Add to basket button enabled, nothing on click	 Current Selection Selected seats: STD - Row: D Seat: 1 STD - Row: D Seat: 2 Total price of Tickets: £6.00 Add to basket button disabled	 Ticket(s) added to basket! Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: D Seat: 1 - £3.00 added to basket. Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: D Seat: 2 - £3.00 added to basket. Basket before tickets were added: Items in Basket: Added to basket alert shown

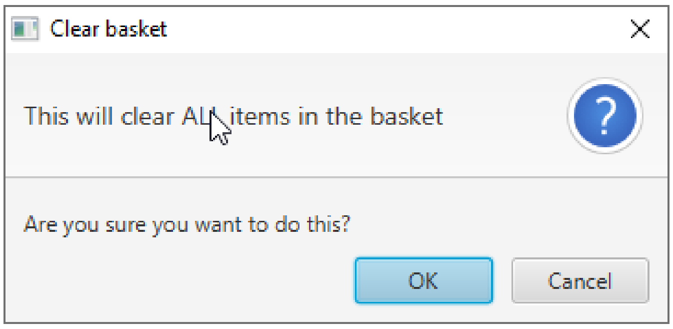
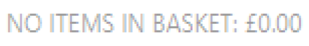
Selecting a showing – Pressing select seat button

Conditions	Rule 1	Rule 2
Showing selected (T/F)	F	T
Expected Outcome	Error message	Seats page shown
Actual Outcome	 No selection made Can not choose seats Please select a showing Error message	 Navigation Film/Showing Details Seats Screen Row A Row B Row C Row D Row E Row F Row G View All Snacks Basket Home Current Selection Key Add to Basket Title:Finding Nemo Run Time: 100 min Plot: After his son is captured in the Great Barrier Reef and taken to Sydney, a timid clownfish sets out on a journey to bring him home. Date: 20/04/2019 Time: 13:00 Age Rating: Ticket Prices Standard Ticket: £2.00 VIP Ticket: £2.50 Selected seats: Total price of Tickets: £0.00 Standard Seat Booked Seat VIP Seat Selected Seat Home page shown

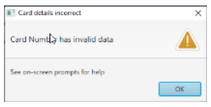
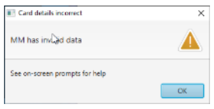
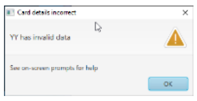
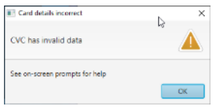
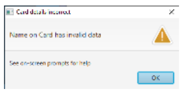
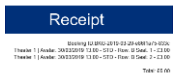
Basket - Remove selected item button pressed

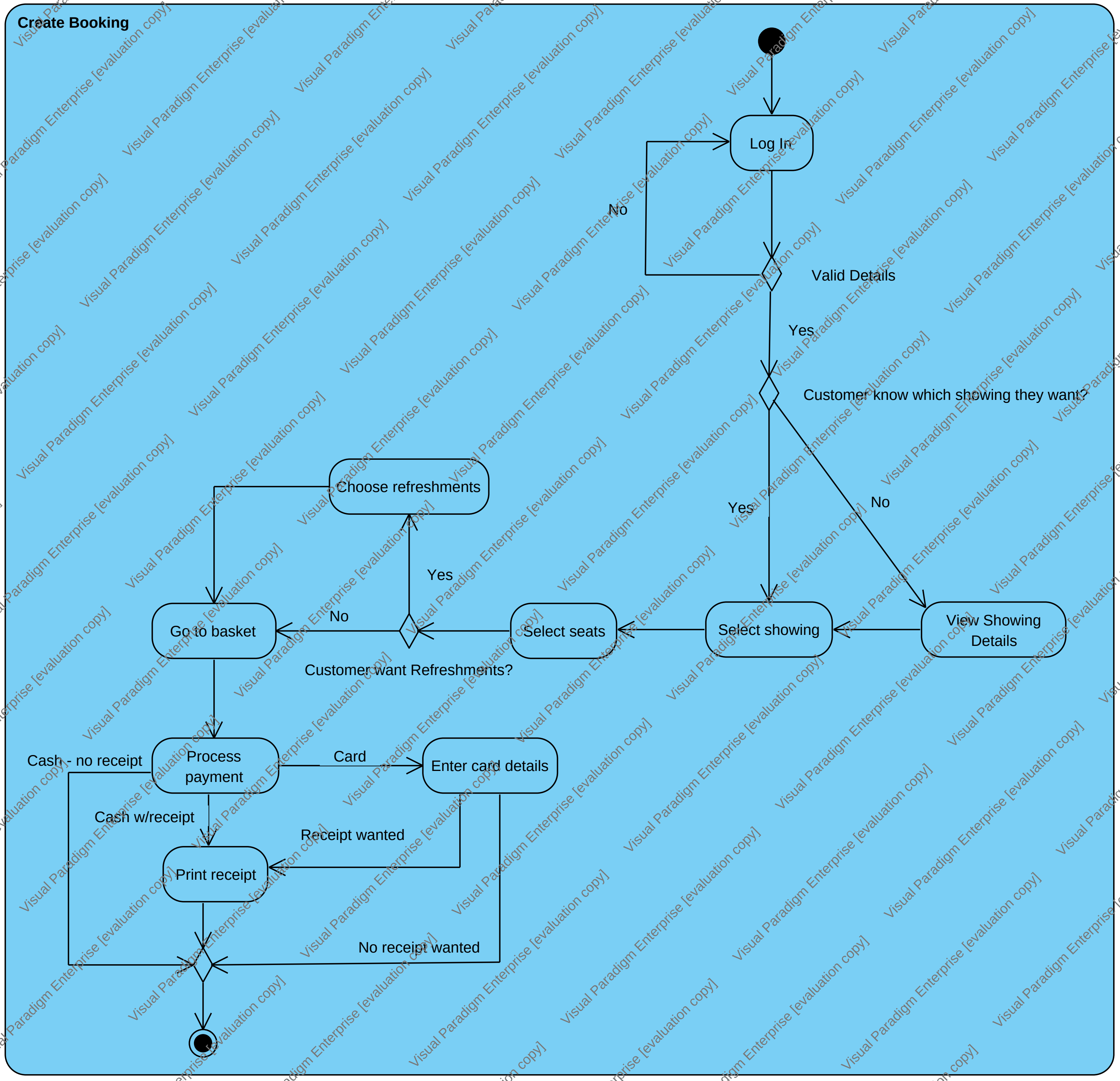
Conditions	Rule 1	Rule 2
Item selected(F/T)	F	T
Expected Outcome	No selection alert	Item removed from basket
Actual Outcome	 No selection made Can not remove item from basket Please select an item to remove OK No selection alert	 Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: D Seat: 1 - £3.00 Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: D Seat: 2 - £3.00 Item removed from basket  Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: D Seat: 1 - £3.00 Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: D Seat: 1 - £3.00

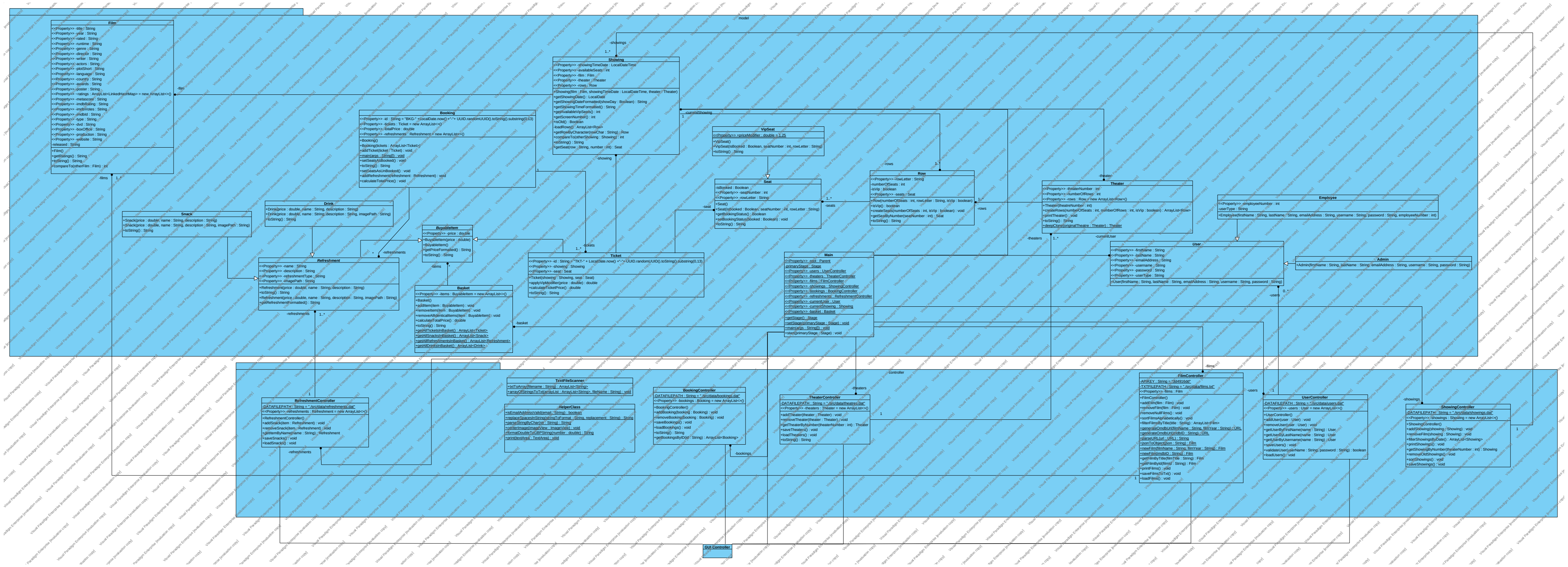
Basket – Clear all button pressed

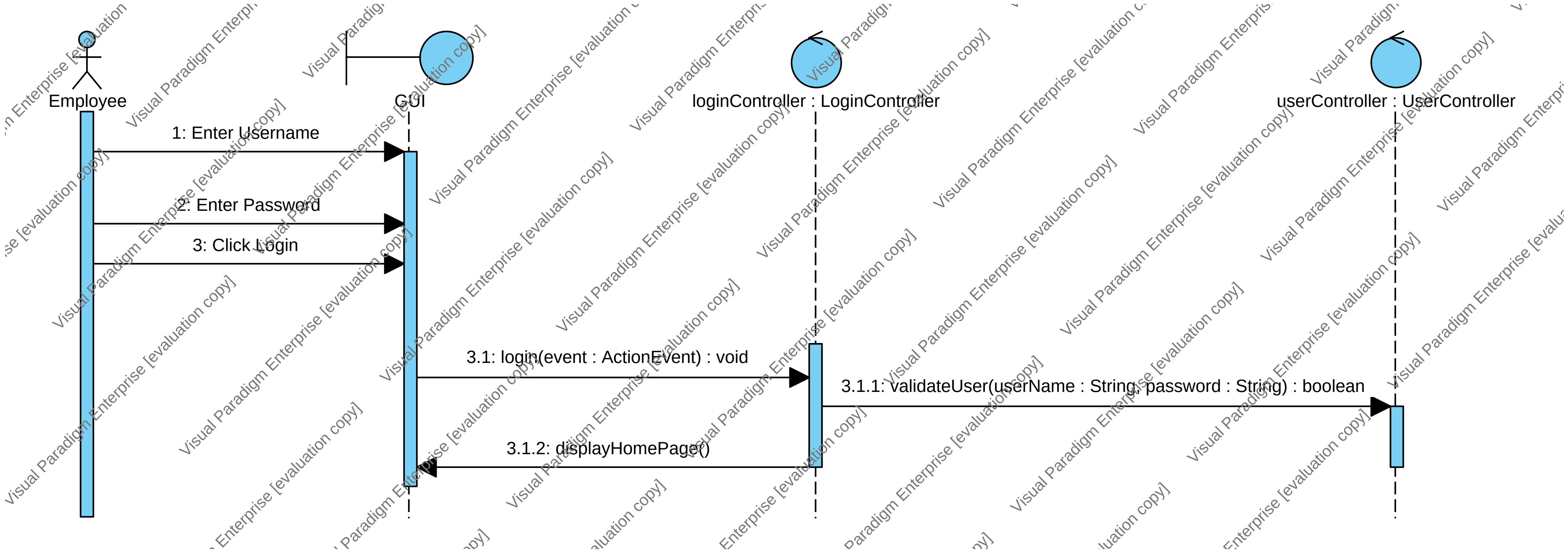
Conditions	Rule 1	Rule 2
Items in basket (F/T)	F	T
Expected Outcome	Nothing	Are you sure prompt Followed by clear basket if yes.
Actual Outcome	Nothing	Are you sure prompt  Clear basket This will clear ALL items in the basket Are you sure you want to do this? OK Cancel Followed by clear basket if yes.  NO ITEMS IN BASKET: £0.00

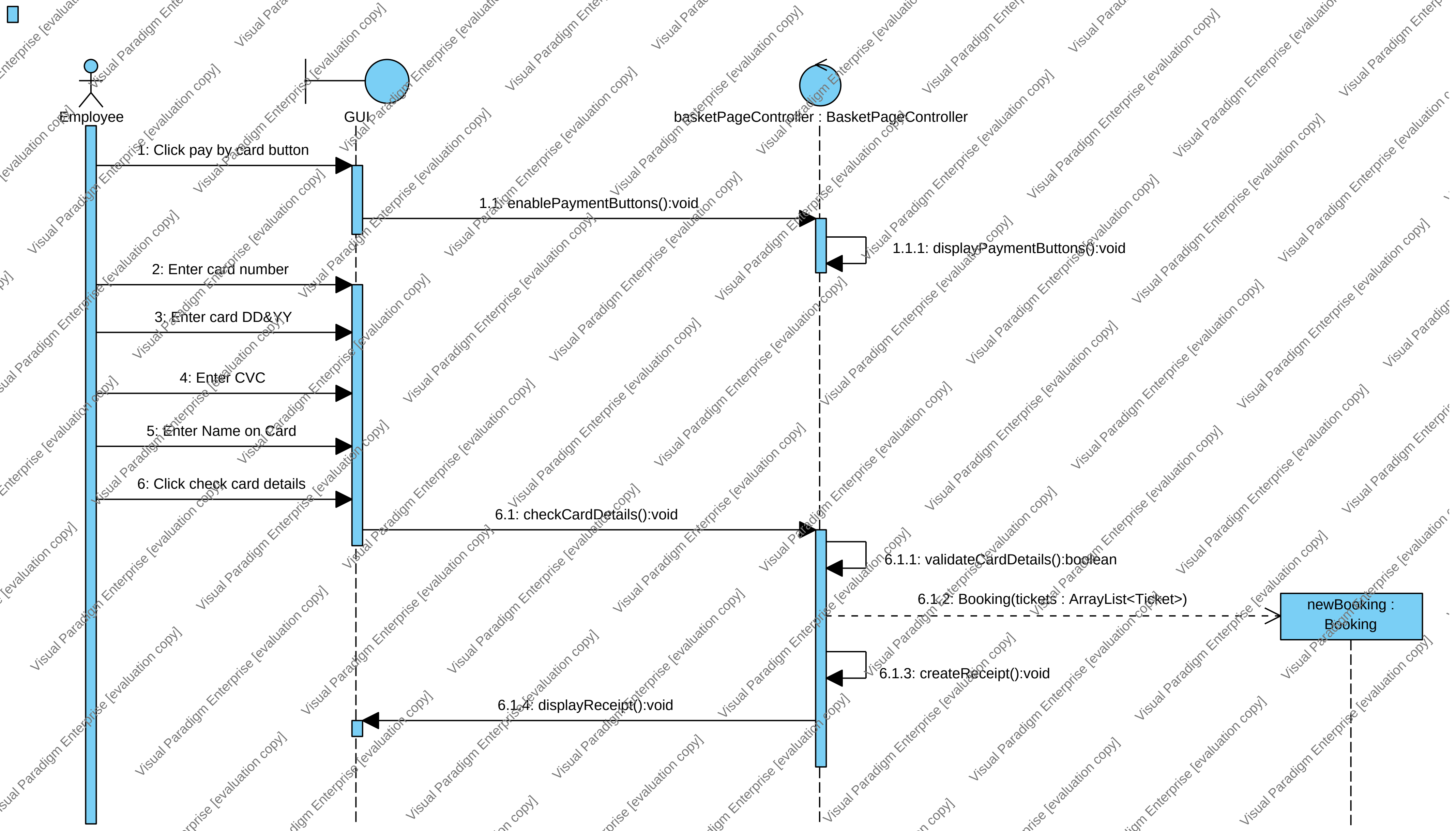
Paying with card

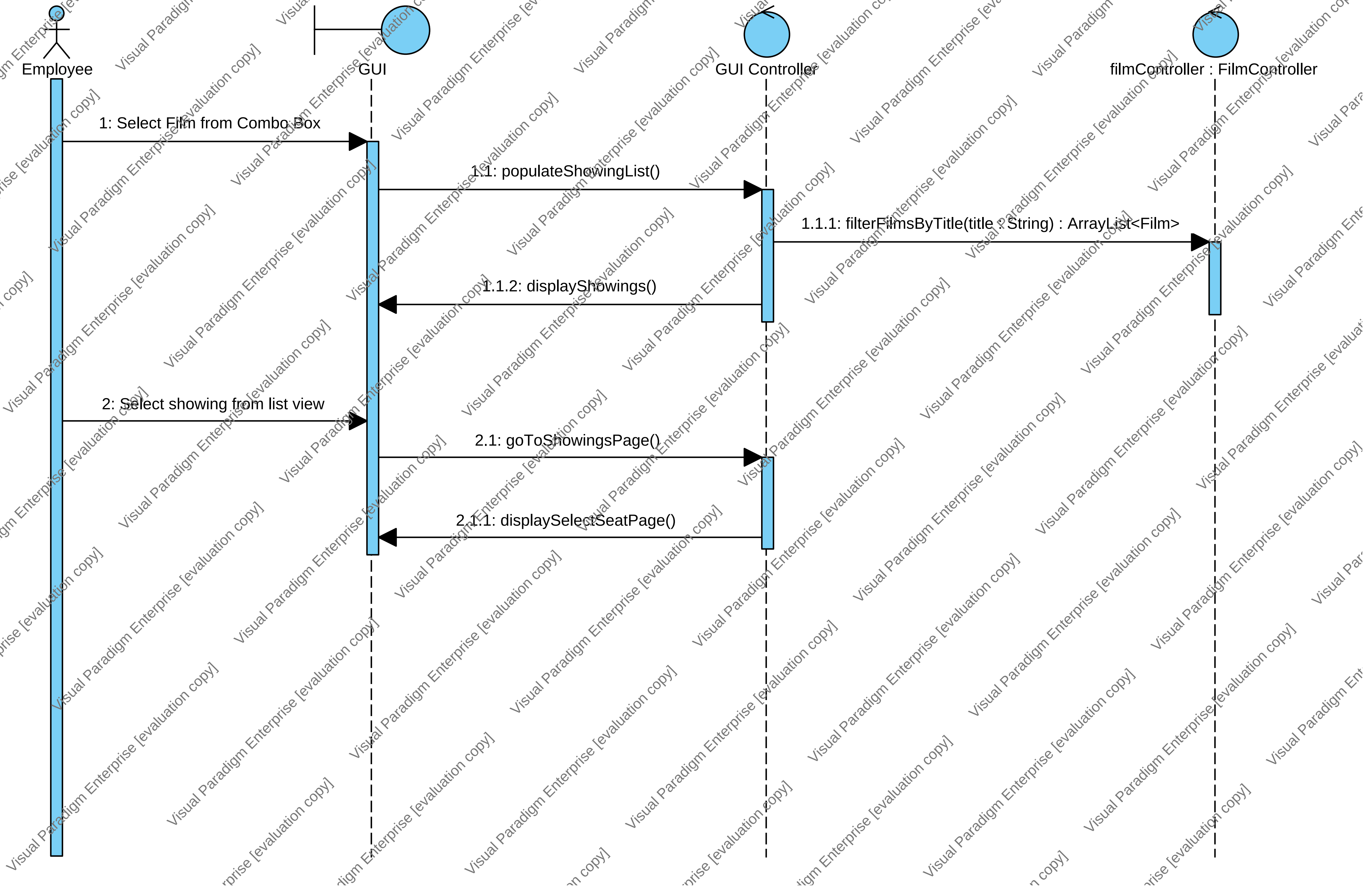
Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Card Number(T/F)	F	T	T	T	T	T
Card Name(T/F)	F	F	T	T	T	T
Card MM (T/F)	F	F	F	T	T	T
Card YY (T/F)	F	F	F	F	T	T
Card CVC	F	F	F	F	F	T
Expected Outcome	Error message	Error message	Error message	Error message	Error message	Receipt created
Actual Outcome	Error message  Card details incorrect Card Number has invalid data See on-screen prompts for help OK	Error message  Card details incorrect MM has invalid data See on-screen prompts for help OK	Error message  Card details incorrect YY has invalid data See on-screen prompts for help OK	Error message  Card details incorrect CVC has invalid data See on-screen prompts for help OK	Error message  Card details incorrect Name on Card has invalid data See on-screen prompts for help OK	Receipt created  Receipt Booking ID:BKG-2019-03-20-[illegible] Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: B Seat: 1 - £3.00 Theater 1 | Avatar: 30/03/2019 13:00 - STD - Row: B Seat: 2 - £3.00 Total: £6.00

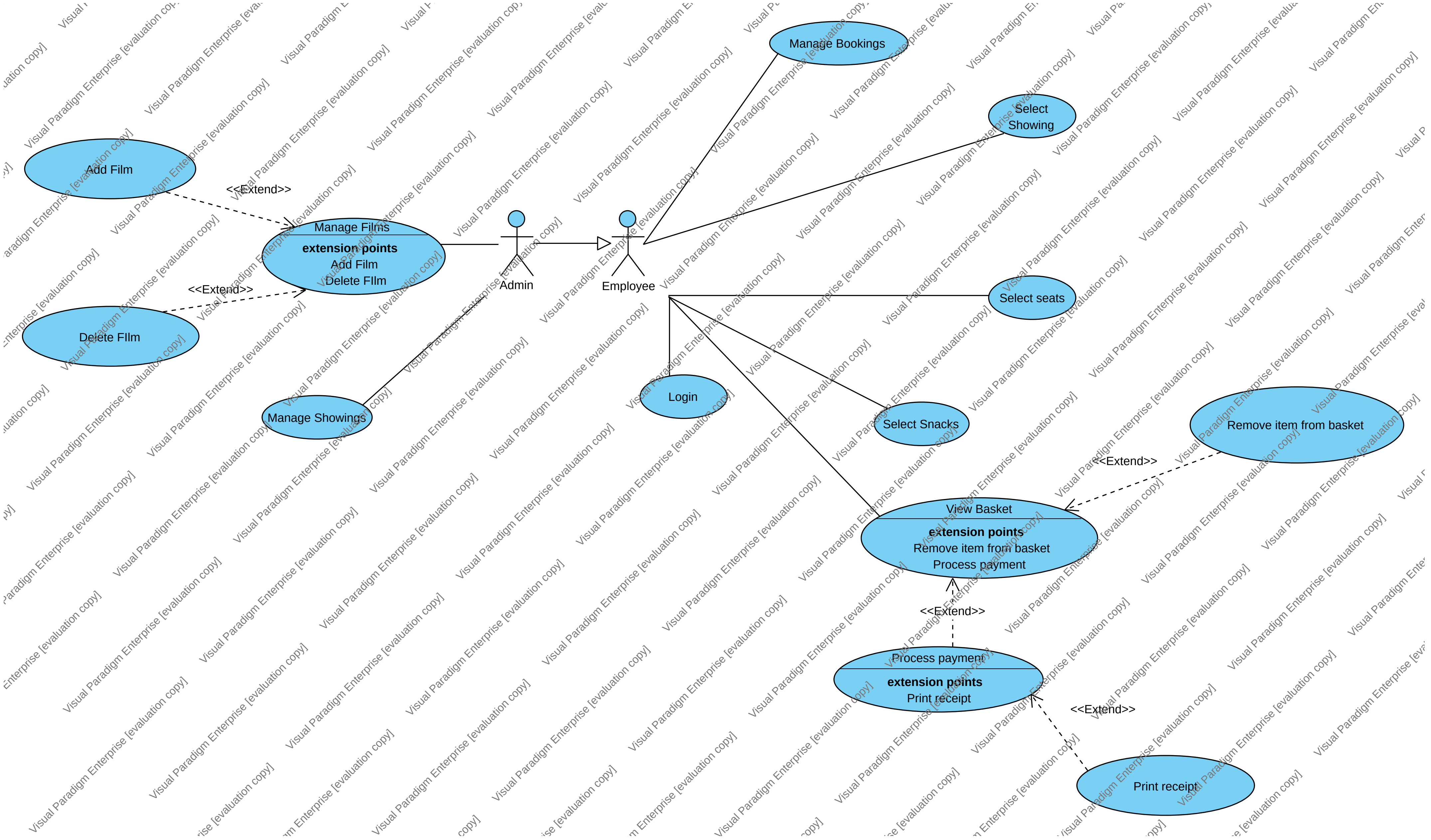

Create Booking
Log In
No
Valid Details
Yes
Customer know which showing they want?
Yes
No
View Showing Details
Select showing
Select seats
Customer want Refreshments?
Yes
No
Choose refreshments
Go to basket
Process payment
Card
Enter card details
Cash - no receipt
Cash w/receipt
Receipt wanted
Print receipt
No receipt wanted


model
controller
Film
<<Property>> -title : String
<<Property>> -year : String
<<Property>> -rated : String
<<Property>> -runtime : String
<<Property>> -genre : String
<<Property>> -director : String
<<Property>> -writer : String
<<Property>> -actors : String
<<Property>> -plotShort : String
<<Property>> -language : String
<<Property>> -country : String
<<Property>> -awards : String
<<Property>> -poster : String
<<Property>> ~ratings : ArrayList<LinkedHashMap> = new ArrayList<>()
<<Property>> -metascore : String
<<Property>> -imdbRating : String
<<Property>> -imdbVotes : String
<<Property>> -imdbId : String
<<Property>> -type : String
<<Property>> -dvd : String
<<Property>> -boxOffice : String
<<Property>> -production : String
<<Property>> -website : String
-released : String
+Film()
+getRatings() : String
+toString() : String
+compareTo(otherFilm : Film) : int
-film
-films
1..*
Showing
<<Property>> -showingTimeDate : LocalDateTime
<<Property>> -availableSeats : int
<<Property>> -film : Film
<<Property>> -theater : Theater
<<Property>> -rows : Row
+Showing(film : Film, showingTimeDate : LocalDateTime, theater : Theater)
+getShowingDate() : LocalDate
+getShowingDateFormatted(showDay : Boolean) : String
+getShowingTimeFormatted() : String
+getAvailableVipSeats() : int
+getScreenNumber() : int
+isOld() : Boolean
-loadRows() : ArrayList<Row>
+getRowByCharacter(rowChar : String) : Row
+compareTo(otherShowing : Showing) : int
+toString() : String
+getSeat(row : String, number : int) : Seat
-showings
1..*
-showing
-currentShowing
1
Booking
<<Property>> -id : String = "BKG-" +LocalDate.now() +"-"+ UUID.randomUUID().toString().substring(0,13)
<<Property>> -tickets : Ticket = new ArrayList<>()
<<Property>> -totalPrice : double
<<Property>> -refreshments : Refreshment = new ArrayList<>()
+Booking()
+Booking(tickets : ArrayList<Ticket>)
+addTicket(ticket : Ticket) : void
+main(args : String[]) : void
+setSeatsAsBooked() : void
+toString() : String
+setSeatsAsUnBooked() : void
+addRefreshment(refreshment : Refreshment) : void
+calculateTotalPrice() : void
1
VipSeat
<<Property>> +priceModifier : double = 1.25
+VipSeat()
+VipSeat(isBooked : Boolean, seatNumber : int, rowLetter : String)
+toString() : String
Seat
-isBooked : Boolean
<<Property>> -seatNumber : int
<<Property>> -rowLetter : String
+Seat()
+Seat(isBooked : Boolean, seatNumber : int, rowLetter : String)
+getBookingStatus() : Boolean
+setBookingStatus(booked : Boolean) : void
+toString() : String
-seat
-seats
1..*
Row
<<Property>> -rowLetter : String
-numberOfSeats : int
-isVip : boolean
<<Property>> -seats : Seat
+Row(numberOfSeats : int, rowLetter : String, isVip : boolean)
+isVip() : boolean
+createSeats(numberOfSeats : int, isVip : boolean) : void
+getSeatByNumber(seatNumber : int) : Seat
+toString() : String
-rows
1..*
Theater
<<Property>> -theaterNumber : int
<<Property>> -numberOfRows : int
<<Property>> -rows : Row = new ArrayList<Row>()
+Theater(theatreNumber : int)
+createRows(numberOfSeats : int, numberOfRows : int, isVip : boolean) : ArrayList<Row>
+printTheater() : void
+toString() : String
+deepClone(originalTheatre : Theater) : Theater
-theater
-theaters
1..*
-currentUser
Employee
<<Property>> -employeeNumber : int
-userType : String
+Employee(firstName : String, lastName : String, emailAddress : String, username : String, password : String, employeeNumber : int)
User
<<Property>> -firstName : String
<<Property>> -lastName : String
<<Property>> -emailAddress : String
<<Property>> -username : String
<<Property>> -password : String
<<Property>> -userType : String
+User(firstName : String, lastName : String, emailAddress : String, username : String, password : String)
0..*
-users
Admin
+Admin(firstName : String, lastName : String, emailAddress : String, username : String, password : String)
Snack
+Snack(price : double, name : String, description : String)
+Snack(price : double, name : String, description : String, imagePath : String)
+toString() : String
Drink
+Drink(price : double, name : String, description : String)
+Drink(price : double, name : String, description : String, imagePath : String)
+toString() : String
BuyableItem
<<Property>> -price : double
+BuyableItem(price : double)
+BuyableItem()
+getPriceFormatted() : String
+toString() : String
-refreshments
*
-items
*
Refreshment
<<Property>> -name : String
<<Property>> -description : String
<<Property>> -refreshmentType : String
<<Property>> -imagePath : String
+Refreshment(price : double, name : String, description : String)
+toString() : String
+Refreshment(price : double, name : String, description : String, imagePath : String)
+getRefreshmentFormatted() : String
-refreshments
1..*
Basket
<<Property>> -items : BuyableItem = new ArrayList<>()
+Basket()
+addItem(item : BuyableItem) : void
+removeItem(item : BuyableItem) : void
+removeAllIdenticalItems(item : BuyableItem) : void
+calculateTotalPrice() : double
+toString() : String
+getAllTicketsInBasket() : ArrayList<Ticket>
+getAllSnacksInBasket() : ArrayList<Snack>
+getAllRefreshmentsInBasket() : ArrayList<Refreshment>
+getAllDrinksInBasket() : ArrayList<Drink>
-basket
-tickets
1..*
Ticket
<<Property>> -id : String = "TKT-" + LocalDate.now() +"-"+ UUID.randomUUID().toString().substring(0,13)
<<Property>> -showing : Showing
<<Property>> -seat : Seat
+Ticket(showing : Showing, seat : Seat)
+applyVipModifier(price : double) : double
+calculateTicketPrice() : double
+toString() : String
Main
<<Property>> -root : Parent
-primaryStage : Stage
<<Property>> -users : UserController
<<Property>> -theaters : TheaterController
<<Property>> -films : FilmController
<<Property>> -showings : ShowingController
<<Property>> -bookings : BookingController
<<Property>> -refreshments : RefreshmentController
<<Property>> -currentUser : User
<<Property>> -currentShowing : Showing
<<Property>> -basket : Basket
+getStage() : Stage
+setStage(primaryStage : Stage) : void
+main(args : String[]) : void
+start(primaryStage : Stage) : void
1
RefreshmentController
-DATAFILEPATH : String = "./src/data/refreshments.dat"
<<Property>> -refreshments : Refreshment = new ArrayList<>()
+RefreshmentController()
+addSnack(item : Refreshment) : void
+removeSnack(item : Refreshment) : void
+getItemByName(name : String) : Refreshment
+saveSnacks() : void
+loadSnacks() : void
-refreshments
TextFileScanner
+txtToArray(filename : String) : ArrayList<String>
+arrayOfStringsToTxt(arrayList : ArrayList<String>, fileName : String) : void
HelperClass
+isEmailAddressValid(email : String) : boolean
+replaceSpacesInString(stringToFormat : String, replacement : String) : String
+parseStringByChar(str : String) : String
+centerImage(imageView : ImageView) : void
+formatDoubleToGBPString(number : double) : String
+print(textArea : TextArea) : void
BookingController
-DATAFILEPATH : String = "./src/data/bookings.dat"
<<Property>> -bookings : Booking = new ArrayList<>()
+BookingController()
+addBooking(booking : Booking) : void
+removeBooking(booking : Booking) : void
+saveBookings() : void
+loadBookings() : void
+toString() : String
+getBookingsByID(id : String) : ArrayList<Booking>
-bookings
-theaters
TheaterController
-DATAFILEPATH : String = "./src/data/theatres.dat"
<<Property>> -theaters : Theater = new ArrayList<>()
+addTheater(theater : Theater) : void
+removeTheater(theater : Theater) : void
+getTheaterByNumber(theaterNumber : int) : Theater
+saveTheaters() : void
+loadTheaters() : void
+toString() : String
1
-films
FilmController
-APIKEY : String = "3d4916dd"
-TXTFILEPATH : String = "./src/data/films.txt"
<<Property>> -films : Film
+FilmController()
+addFilm(film : Film) : void
+removeFilm(film : Film) : void
+removeNullFilms() : void
+sortFilmsAlphabetically() : void
+filterFilmsByTitle(title : String) : ArrayList<Film>
+generateOmdbUrl(filmName : String, filmYear : String) : URL
+generateOmdbUrl(imdbID : String) : URL
+parseURL(url : URL) : String
+jsonToObject(json : String) : Film
+newFilm(filmName : String, filmYear : String) : Film
+newFilm(imdbID : String) : Film
+getFilmByTitle(filmTitle : String) : Film
+getFilmById(filmId : String) : Film
+printFilms() : void
+saveFilmsToTxt() : void
+loadFilms() : void
1
-users
1
UserController
-DATAFILEPATH : String = "./src/data/users.dat"
<<Property>> -users : User = new ArrayList<>()
+UserController()
+addUser(user : User) : void
+removeUser(user : User) : void
+getUserByFirstName(name : String) : User
+getUserByLastName(name : String) : User
+getUserByUsername(name : String) : User
+saveUsers() : void
+validateUser(userName : String, password : String) : boolean
+loadUsers() : void
-showings
ShowingController
-DATAFILEPATH : String = "./src/data/showings.dat"
<<Property>> -showings : Showing = new ArrayList<>()
+ShowingController()
+addShowing(showing : Showing) : void
+removeFilm(showing : Showing) : void
+filterShowingsByDate() : ArrayList<Showing>
+printShowings() : void
+getShowingByNumber(theaterNumber : int) : Showing
+removeOldShowings() : void
+sortShowings() : void
+saveShowings() : void
1
GUI Controller


Employee
GUI
loginController : LoginController
userController : UserController
1: Enter Username
2: Enter Password
3: Click Login
3.1: login(event : ActionEvent) : void
3.1.1: validateUser(userName : String, password : String) : boolean
3.1.2: displayHomePage()


Employee
GUI
basketPageController : BasketPageController
1: Click pay by card button
1.1: enablePaymentButtons():void
1.1.1: displayPaymentButtons():void
2: Enter card number
3: Enter card DD&YY
4: Enter CVC
5: Enter Name on Card
6: Click check card details
6.1: checkCardDetails():void
6.1.1: validateCardDetails():boolean
6.1.2: Booking(tickets : ArrayList<Ticket>)
newBooking : Booking
6.1.3: createReceipt():void
6.1.4: displayReceipt():void


Employee
GUI
GUI Controller
filmController : FilmController
1: Select Film from Combo Box
1.1: populateShowingList()
1.1.1: filterFilmsByTitle(title : String) : ArrayList<Film>
1.1.2: displayShowings()
2: Select showing from list view
2.1: goToShowingsPage()
2.1.1: displaySelectSeatPage()


Manage Bookings
Select Showing
Add Film
<<Extend>>
Manage Films
extension points
Add Film
Delete FIlm
Admin
Employee
<<Extend>>
Delete FIlm
Select seats
Login
Manage Showings
Select Snacks
Remove item from basket
<<Extend>>
View Basket
extension points
Remove item from basket
Process payment
<<Extend>>
Process payment
extension points
Print receipt
<<Extend>>
Print receipt

2 – Libraries used

1. GOOGLE.GSON-2.8.5 - https://github.com/google/gson

2. APACHE.commons-io-2.6 - https://commons.apache.org/

3. Credit card validation – APACHE.commons-validator-1.6 - https://commons.apache.org/

4. JavaFX Material Design styled buttons - jfoenix-8.0.8 - https://github.com/jfoenixadmin/JFoenix

Bibliography

1 - Miller, D. (1999). Java Tip 76: An alternative to the deep copy technique. Retrieved from https://www.javaworld.com/article/2077578/java-tip-76--an-alternative-to-the-deep-copy-technique.html