PART C - DEVELOPMENT

Contents

C.1 Complexity Features **2**
C.1.1 Dynamic Arrays 2
C.1.2 Linked Lists 5
C.1.3 Sequential Access 7
C.1.4 Encapsulation and Modularity 10
C.1.5 Inheritance and Interfaces 12
C.1.6 Searching 14
C.1.7 JavaFX 17
C.1.8 JavaMail 21
C.1.9 iText PDF Generator 24
C.1.10 iText QR Code Generator 27

C.2 Bibliography **29**

C.1 Complexity Features

C.1.1 Dynamic Arrays

An array is a homogenous collection of elements of the same data type.[1] Java internally handles memory allocation: by declaring an array, memory space is dynamically allocated for values of a particular type.[2]

Arrays were used as the main model for the seats. Through button declaration each seat in the cinema floor plan was converted into an element within the array.

Appropriateness	Ingenuity

• Dynamic arrays allowed the addition and deletion of new elements/seats and presentation of booked seats, as requested by my client, Mickey Mouse.
• This allowed for the completion of Success Criteria 4.2-4.4.
	• The use of arrays:
	  ○ Allowed the ability to initialize an array easily in the method `initialiseArray(){}`, i.e., through `seats[index]=seatId;`
	  ○ Allowed random access of values enabling the use of Java dynamic arrays through `for` loops and `ArrayList` structures mentioned in [3].

[1] Vertica. 2022. *1 - What are Complex Data Types? | Vertica*. [online] Available at: <https://www.vertica.com/blog/complex-data-types-in-sql-1-what-are-they/> [Accessed 7 January 2022].
[2] Techopedia.com. 2022. *What is Array in Java? - Definition from Techopedia*. [online] Available at: <https://www.techopedia.com/definition/1143/array-java> [Accessed 7 January 2022].
[3] www.javatpoint.com. 2022. *Dynamic Array in Java - Javatpoint*. [online] Available at: <https://www.javatpoint.com/dynamic-array-in-java> [Accessed 7 January 2022].

Code

```
@FXML
private Button A1, A2, A3, A4, A5, A6, A7, A8, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2, C3, C4, C5, C6, C7, C8;

//Instantiates array variables
private Button[] seats = new Button[24];
public static boolean[] bookings = new boolean[24];
public static boolean[] booked;
```

@FXML Buttons that dictate the elements in each array.

Here we instantiate an array of Button and boolean values.

The dynamic boolean array is instantiated.

```
//Method initialises 'Button[] seats' array giving each index a unique seat ID
private void initialiseArray() {
    seats[0]=A1;
    seats[1]=A2;
    seats[2]=A3;
    seats[3]=A4;
    seats[4]=A5;
    seats[5]=A6;
    seats[6]=A7;
    seats[7]=A8;

    seats[8]=B1;
    seats[9]=B2;
    seats[10]=B3;
    seats[11]=B4;
    seats[12]=B5;
    seats[13]=B6;
    seats[14]=B7;
    seats[15]=B8;

    seats[16]=C1;
    seats[17]=C2;
    seats[18]=C3;
    seats[19]=C4;
    seats[20]=C5;
    seats[21]=C6;
    seats[22]=C7;
    seats[23]=C8;
}
```

We can simply initialise the 'seats' array by assigning each index, from 0 to 23 of the array to a seat identification number. For example, the first seat is "A1" and the last seat is "C8".

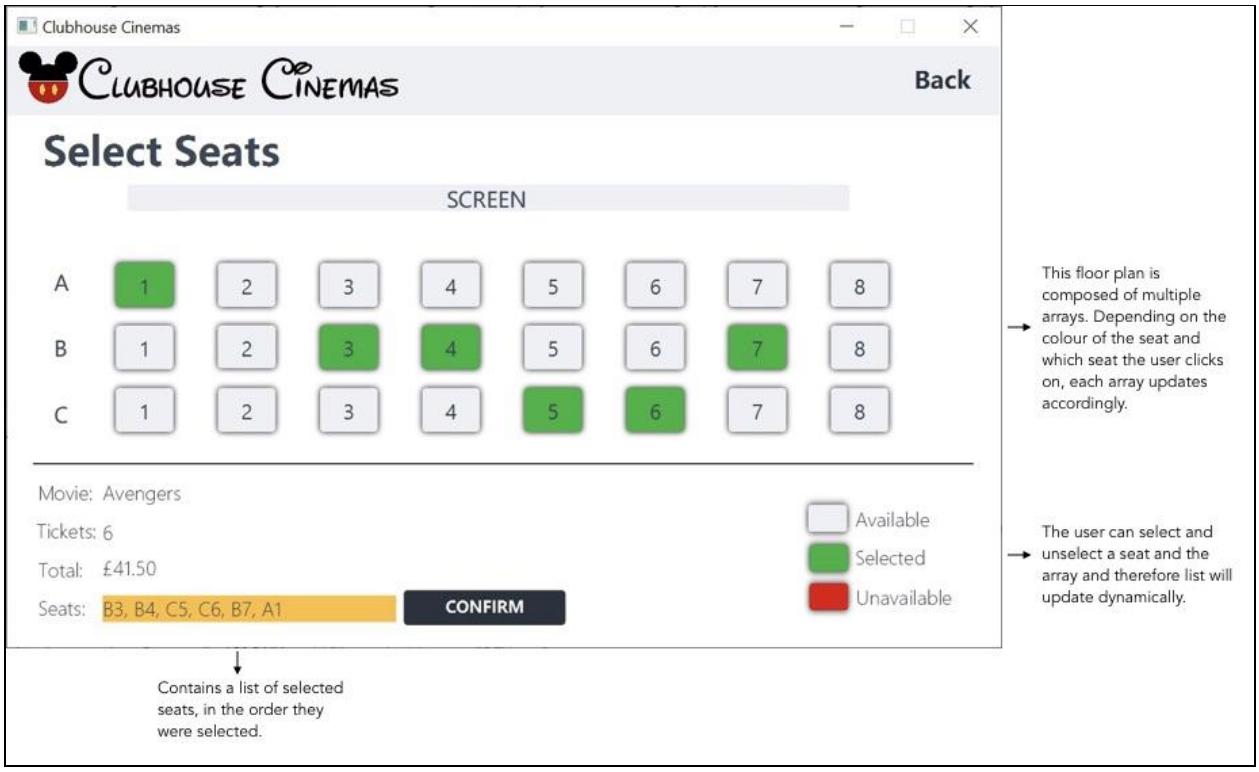
```
/*
 * Method 'setUpSeats()' constructs the visual outlook of the cinema floorplan giving seats a colour depending on availability and selection
 * available = #edf0f4  selected = #23b33b  unavailable = #e40606
 */
private void setUpSeats() {
    for(int i=0; i<seats.length; i++){ //FOR loop - iterates through arrays
        if(bookings[i]==false){ //IF the seat is available...
            seats[i].setStyle("-fx-background-color:  #edf0f4"); //Gives button a GREY colour
            int finalI1 = i;

            seats[i].setOnAction(event -> { //Using a lambda expression it acts as an action listener executing the code
                if(booked[finalI1]==false){ //IF the seat is selected...
                    //Checks user hasn't booked too many seats
                    if(numberOfSeats<maxSeats){
                        numberOfSeats++;
                        seats[finalI1].setStyle("-fx-background-color:  #23b33b"); //Gives button a GREEN colour
                        setBookedSeats(seats[finalI1], true);
                    }else { //Outputs error message
                        Alert alert = new Alert(AlertType.WARNING, "Error: Maximum seat limit reached!",
                                ButtonType.OK);
                        alert.showAndWait();
                        if (alert.getResult() == ButtonType.OK) {
                            return;
                        }
                    }
                }
                else if(booked[finalI1]==true){ //IF the user unselects a seat...
                    numberOfSeats--; //Subtracts 1 from the number of selected seats
                    seats[finalI1].setStyle("-fx-background-color:  #edf0f4");//Gives button a GREY colour
                    setBookedSeats(seats[finalI1], false);
                }
                popSeat(seats[finalI1]);
            });
        }else if(bookings[i]==true){ //IF the seat is unavailable...
            seats[i].setStyle("-fx-background-color:  #e40606"); //Gives button a RED colour
            int finalI = i;
            seats[i].setOnAction(event -> rotateButton(seats[finalI]));
        }
    }
}
```

Using an array allows us to use the for loop structure to create a custom list of elements. It iterates through the array using .length.

We then present each element in the form of colours, depending on the availability or selection of the seats.

The dynamic array booked[] allows us to use arrays, while not needing to change the array's size and parse through the array to check for the next empty element. This creates a simple checking tool for booked seats.

Implementation



C.1.2 Linked Lists

Linked List is an Abstract Data Type (ADT) that holds a collection of nodes, the nodes can be accessed sequentially.[4] In Java a linked list is implemented using the doubly linked list data structure to store its elements, containing two pointers.[5]

Linked lists were used in the application to aid the process of storing movie details in the variables `titles`, `startDates`, `endDates`, `time1`, `time2`, `time3` so that duplicate movies were not created by the employee.

Appropriateness	Ingenuity

• Linked lists within the program decreased the overall complexity, providing an easy solution to retrieving data.
• This allowed for the completion of Success Criteria 5.2.
	• The use of linked lists:
	  ○ Allowed the ability to append and delete values within the linked list, e.g. `.add()` and `.remove()`.
	  ○ As Java uses a doubly linked list structure, one can perform the above functions from the header or tail node.
	  ○ Easier to get a specific element in the linked list through `.get(int index)`. This allows for easier searching.
	  ○ There is also improved memory efficiency due to its dynamic size during run time.

[4] Medium. 2022. *#SideNotes — Linked List — Abstract Data Type and Data Structure*. [online] Available at: <https://lucasmagnum.medium.com/sidenotes-linked-list-abstract-data-type-and-data-structure-fd2f8276ab53> [Accessed 7 January 2022].

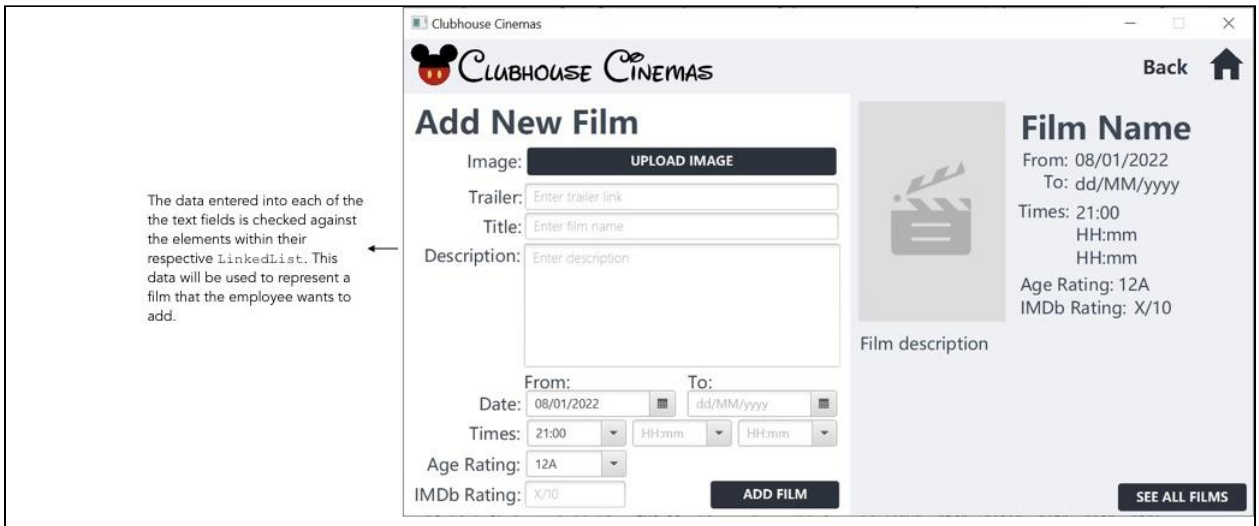
[5] GeeksforGeeks. 2022. *LinkedList in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/linked-list-in-java/> [Accessed 7 January 2022].

Code

```java
//Creates LinkedLists of type String for titles, startDates, endDates, time1, time2 and time3
public static LinkedList<String> titles=new LinkedList<String>();
public static LinkedList<String> startDates = new LinkedList<String>();
public static LinkedList<String> endDates = new LinkedList<String>();
public static LinkedList<String> time1=new LinkedList<String>();
public static LinkedList<String> time2 = new LinkedList<String>();
public static LinkedList<String> time3 = new LinkedList<String>();

//Accessing the LinkedLists from the Main class to use in the FilmManagement class
LinkedList<String> titles = Main.titles;
LinkedList<String> startDates = Main.startDates;
LinkedList<String> endDates = Main.endDates;
LinkedList<String> time1 = Main.time1;
LinkedList<String> time2 = Main.time2;
LinkedList<String> time3 = Main.time3;
```

Refers to the abstract data structure, `LinkedList`, used for retrieving film-specific information from the `films.txt` file.

Implementation



The data entered into each of the the text fields is checked against the elements within their respective `LinkedList`. This data will be used to represent a film that the employee wants to add.

When a non-duplicate film is added these details are added to the films.txt file.

The cycle then repeats with these details also being added to the LinkedLists to avoid future duplicates.

C.1.3 Sequential Access

Sequential access refers to the reading or writing of data records in sequential order, that is, one record after the other.[6] Within Clubhouse Cinemas, sequential access denotes the type of access used for sequential files, such as a text file.[7]

Sequential access was important throughout the program with it being utilised in reading/writing to and from files, and performing linear searches within files (see C.1.6).

Appropriateness	**Ingenuity**

• Sequential access through writing to files solved the problem of wasted memory for my client, Mickey Mouse. With sequential access data is added onto the end of the text files, meaning there are no gaps in the data.
• This allowed the completion of Success Criteria 2.1, 4.6.3, 5.3.1, 5.4.2.
	• The use of sequential access:
	  ○ Allows for a quick store of data that is easy to read in TXT files such as `bookings.txt`.
	  ○ Data can be easily read through the `.split()` function which places a distinguishable mark, a delimiter, between each data section.
	  ○ Allows for easy data sharing executed through the `exportBookings()` method which produces a saveable `bookings exported.txt` file.

[6] Webopedia. 2022. *What is Sequential Access? | Webopedia*. [online] Available at: <https://www.webopedia.com/definitions/sequential-access/> [Accessed 7 January 2022].
[7] Homeandlearn.co.uk. 2022. *What is a Text File*. [online] Available at: <https://www.homeandlearn.co.uk/NET/nets8p1.html> [Accessed 7 January 2022].

Code

```java
@FXML
private Button logoutBtn, manageFilmsBtn, exportBookingsBtn;

//Method 'exportBookings()' creates a saveable 'bookings exported.txt' file
public void exportBookings(ActionEvent event) throws IOException{

    //Employee chooses a folder in whcih they want to save the newly exported file
    DirectoryChooser folderChooser = new DirectoryChooser();
    folderChooser.setTitle("Select folder:");
    File defaultDirectory = new File(".");
    folderChooser.setInitialDirectory(defaultDirectory);
    File selectedDirectory = folderChooser.showDialog(null);

    //IF the employee clicks on cancel
    if (selectedDirectory == null) {
        return;
    }

    //Clearing the export file, in case it exists from before
    PrintWriter pw = new PrintWriter(new FileOutputStream(
            new File(selectedDirectory.toPath() + "/bookings exported.txt")));
    pw.close();

    //Creating the printwriter using the append option now
    pw = new PrintWriter(new FileOutputStream(
            new File(selectedDirectory.toPath() + "/bookings exported.txt"),
            true));

    //Creates FileReader and BufferedReader to be able to read the file
    FileReader fr = new FileReader("bookings.txt");
    BufferedReader br=new BufferedReader(fr);
    String line=br.readLine(); //Reads the first line of the file
    while(line!=null) { //WHILE loop to iterate through the lines of the file
        String[] data=line.split(";"); //Assings a semi-colon as the data delimiter
        pw.append(line); //Appends (adds) line to the 'bookings exported.txt' file
        line=br.readLine(); //Reads the next line
    }

    pw.close(); //Closes PrintWriter

    //Creates message of the completed task
    Alert alert = new Alert(AlertType.INFORMATION, "The bookings.txt file has been exported!",
            ButtonType.OK);
    alert.showAndWait();
    if (alert.getResult() == ButtonType.OK) {
        return;
    }
}
```

`@FXML Button exportBookingsBtn;` dictates when the export action should take place. When clicked on it excutes the `exportBookings()()` method.

Here the `PrintWriter` overwrites the file, essentially clearing it of any previous doings. This is so that the employee has access to the most update version of the bookings.

Adding the `true` condition onto the end of the file allows for quick, sequential data writing to happen.

The program sequentially reads for the file accessing each line one after the other in a regemented process, to then add the line to the new file.

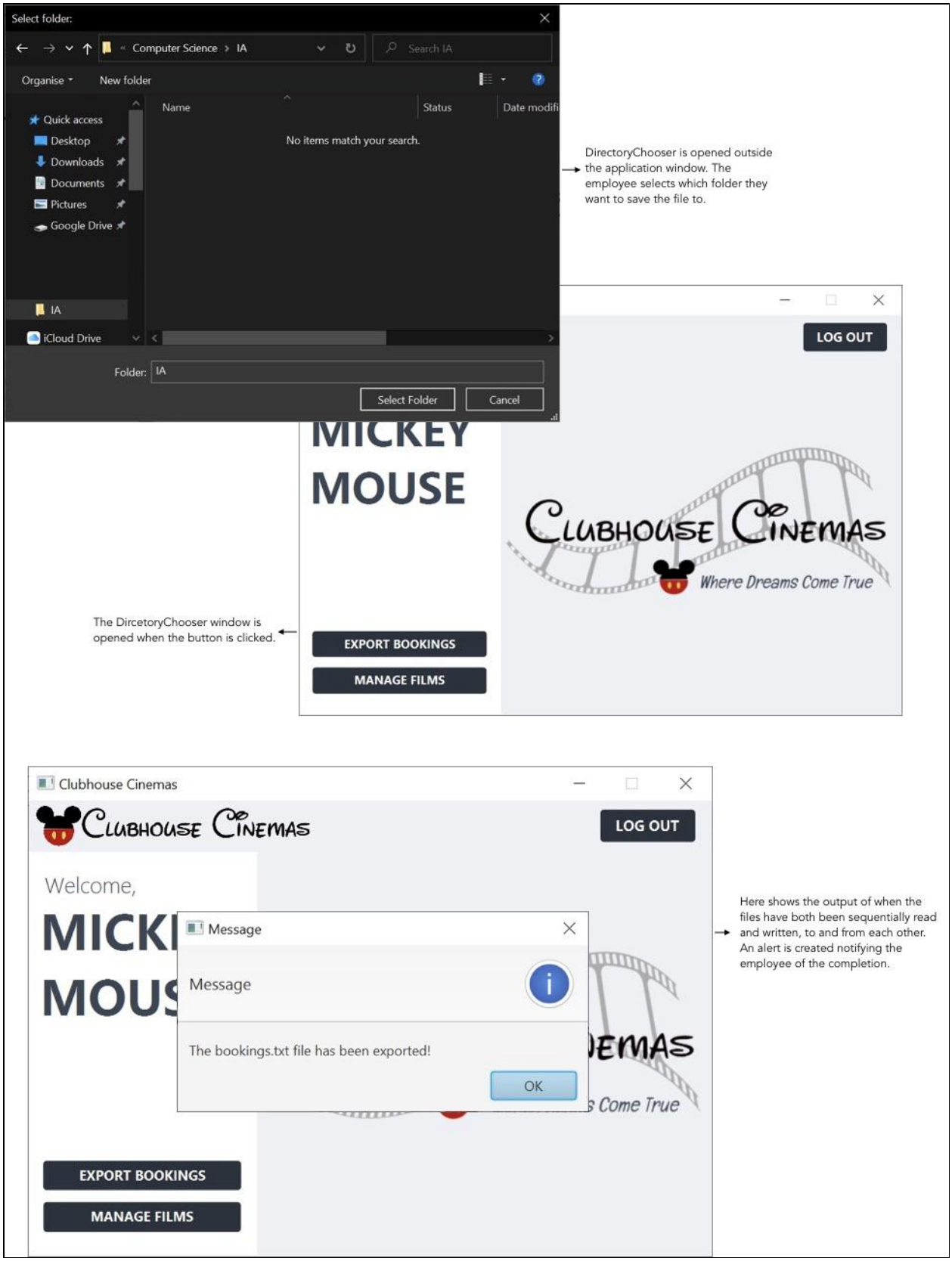
Snapshot of `bookings.txt` file, this is the file to be read from and exported.

bookings.txt

```
1 booked;Ruben;Odamo;Avengers;06/01/2022;17:00;C4, C5, B1, B2;Yes;CLUBH1809
2 booked;Ruben;Odamo;Avengers;08/01/2022;13:00;C4, C5;Yes;CLUBH6361
3 booked;Ruben;Odamo;Avengers;08/01/2022;17:00;B4, B5;Yes;CLUBH5290
```

Each line in the file contains semi-colons, which act as a delimiter for the data points. This is so the data is readable.

Implementation



C.1.4 Encapsulation and Modularity

Encapsulation refers to the mechanism of limiting direct public access to components of an object class and providing access to those through public behaviours (methods).[8] This goes hand in hand with modularity which is the process of decomposing a program into a set of modules (methods) in order to reduce the overall complexity of the problem.[9]

Encapsulation and modularity was used throughout the application. For example using private instance variables in the `BookingHistoryItem.java` class in different window controllers via public methods.

Appropriateness	**Ingenuity**

• Encapsulation and modularity proved extremely useful throughout, resolving the issue of keeping data retrieved from the TXT file database or user local/private to that class, thus providing additional protection to the user's data (see Success Criteria 4.6.3 and 5.4).
• Encapsulation and modularity allowed the completion of Success Criteria 1.1-5.4.
	• The uses of encapsulation and modularity:
	  ○ Allowed for the privatisation of a user's booking details, which is used for booking confirmation and when viewing the `bookings.txt` file.
	  ○ It is achieved through the variables `status`, `firstName`, `lastName`, `film`, `date`, `time`, `seats`, `vip`, `idNumber`, all of type String.
	  ○ All the variables have getter and setter methods in order to provide public access, eg. `getStatus();` `BookingHistoryItem();`
	  ○ Modularity increased the efficiency of program development, whilst also increasing the overall readability of the program.

[8] Sumo Logic. 2022. *What is Encapsulation in OOP? | Sumo Logic*. [online] Available at: <https://www.sumologic.com/glossary/encapsulation/> [Accessed 7 January 2022].
[9] Tutorialspoint.com. 2022. *OOAD - Object Oriented Principles*. [online] Available at: <https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_principles.htm> [Accessed 7 January 2022].

Code

```java
//Initialises the variables
private String status, firstName, lastName, film, date, time, seats, vip, idNumber;

//Setter
public BookingHistoryItem (String status, String firstName, String lastName, String film, String date, String time, String seats, String idNumber, String vip) {

    this.status = status;
    this.firstName = firstName;
    this.lastName = lastName;
    this.film = film;
    this.date = date;
    this.time = time;
    this.seats = seats;
    this.vip = vip;
    this.idNumber = idNumber;
}

//Getter
public String getStatus() {

    return status;
}

//Getter
public String getFirstName() {

    return firstName;
}

//Getter
public String getLastName() {

    return lastName;
}

//Getter
public String getFilm() {

    return film;
}

//Getter
public String getDate() {

    return date;
}

//Getter
public String getTime() {

    return time;
}

//Getter
public String getSeats() {

    return seats;
}

//Getter
public String getVip() {

    return vip;
}

//Getter
public String getIdNumber() {

    return idNumber;
}
```

Refers to the getters and setters methods created for the private instance variables within `BookingHistoryItem.java`.

Setters - create method to set the variable from outside the class.

This is ingenious as it encapsulates the variables within the class, ensuring only access by methods.

Getters - creates methods to return the variables from outside the class.

```java
//Initialises the variables
@FXML
private TableView<BookingHistoryItem> table;
//Table categories are initialised
@FXML
private TableColumn<BookingHistoryItem, String> status, firstName, lastName, film, date, time, seats, vip, idNumber;

@Override
public void initialize(URL location, ResourceBundle resources) {
//(...)

    // specifying how to populate the columns of the table
    status.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("status"));
    firstName.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("firstName"));
    lastName.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("lastName"));
    film.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("film"));
    date.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("date"));
    time.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("time"));
    seats.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("seats"));
    vip.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("vip"));
    idNumber.setCellValueFactory(new PropertyValueFactory<BookingHistoryItem, String>("idNumber"));

    //Creates an 'ObservableList' of the object type 'BookingHistoryItem'
    ObservableList<BookingHistoryItem> list = FXCollections.observableArrayList();

    //TRY-CATCH Block
    try {
        //Creates FileReader and BufferedReader to be able to read the file
        FileReader fr = new FileReader("bookings.txt");
        BufferedReader br=new BufferedReader(fr);
        String line=br.readLine(); //Reads the first line of the file
        while(line!=null) { //WHILE loop to iterate through the lines of the file
            String[] data=line.split(";"); //Assings a semi-colon as the data delimiter
            //Adds the object to the list
            list.add(new BookingHistoryItem(data[0],data[1],data[2],data[3],data[4],
                    data[5],data[6],data[7],data[8]));
            line=br.readLine(); //Reads the next line
        }
        fr.close(); //Closes FileReader
    } catch (IOException e) {
        e.printStackTrace();
    }

    table.setItems(list); //Adds the list to the table
}
```

This code highlights the utilising of the encapsulated variables. In this case from `BookingManagement.java`.

C.1.5 Inheritance and Interfaces

Inheritance within Java OOP refers to the process whereby one object inherits the properties of another object, implied by the Java keywords `extends` and `implements`.[10] Interfaces are a collection of abstract methods. A class implements an interface, inheriting the interface's abstract methods.[11]

Inheritance and interfaces is used throughout the application program, with it being primarily used to extend controller classes to their corresponding model class, whilst also allowing for new implementation to increase the features of a class.

Appropriateness	**Ingenuity**

• Inheritance allows for the connection of sub-classes to their corresponding parent classes, which allows for reusability of code segments within the program.
• This is shown through the implementation of the interface Initializable. Doing so results in the execution of certain checks and communication with program files. This allowed the completion of Success Criteria 3.1-3.3, 4.6.1, 5.2-5.4.1.
	• The use of inheritance:
	  ○ Allowed for sub-classes e.g. `Exception`, to be extended to the parent class e.g. `InvalidFilmInputException`.
	  ○ This allowed for a custom unchecked exception through stating: `class InvalidFilmInputException extends Exception {}`
	• The implementation of interfaces:
	  ○ The use of the `Initializable` interface which is implemented in the inheritance of method: `public void initialize(URL location, ResourceBundle resources){}`

[10] Ib.compscihub.net. 2022. [online] Available at: <https://ib.compscihub.net/wp-content/uploads/2018/07/D.2.2.pdf> [Accessed 7 January 2022].
[11] Tutorialspoint.com. 2022. *Java - Interfaces*. [online] Available at: <https://www.tutorialspoint.com/java/java_interfaces.htm> [Accessed 7 January 2022].

Code

```
//public class which implements the interface Initializable
public class BookingManagement implements Initializable{
// (...)

//InvalidFilmInputException class (subclass) inherits the attributes and methods from the Exception class (superclass)
class InvalidFilmInputException extends Exception {
// (...)
```

Shows the implementation of an interface, `Initializable`, used throughout all controller classes.

Refers to `InvalidFilmInputException` extending `Exception`, which is an example of inheritance.

```
package application;

import java.net.URL;
import java.util.ResourceBundle;

/**
 * public interface Initializable
 *
 * Controller initialization interface.
 * <p>
 * <em>NOTE</em> This interface has been superseded by automatic injection of
 * <code>location</code> and <code>resources</code> properties into the
 * controller. {@link FXMLLoader} will now automatically call any suitably
 * annotated no-arg <code>initialize()</code> method defined by the controller.
 *  It is recommended that the injection approach be used whenever possible.
 *  @since JavaFX 2.0
 */

/**
 * Called to initialize a controller after its root element has been
 * completely processed.
 *
 * @param location
 * The location used to resolve relative paths for the root object, or
 * <tt>null</tt> if the location is not known.
 *
 * @param resources
 * The resources used to localize the root object, or <tt>null</tt> if
 * the root object was not localized.
 */
@Override
public void initialize(URL location, ResourceBundle resources) {
//(...)
```

The initializable interface implements the `initialize()`, which is essential throughout due to it allowing for code to be executed when a particular window is opened.

`@Override` annotation functions as a method modifier. The `initialize()()` method overrrides the existing supertype method public interface Initializable.

This allows the compiler to catch errors, thus reducing mistakes and increasing the code maintainability.

C.1.6 Searching

Searching is the process of finding a particular element in a list.[12] Searching in Clubhouse Cinemas is done through the implementation of a linear search algorithm on a TXT file. Linear searching describes the process of sequentially moving through a data set looking for a matching value.[13]

Searching is important throughout the program with it being present in user authentication, finding film and booking information and adding new bookable films to the user view.

Appropriateness	**Ingenuity**

• Searching through TXT files allowed for the fundamental problem of my client Mickey Mouse, of retrieving and validating data to be solved.
• This allowed the completion of Success Criteria 1.2, 3.2, 5.1, 5.2.
	• The use of searching:
	  ○ Allowed for authentication of users through searching through TXT file `Registration details.txt`.
	  ○ Allows for error handling through the `throws IOException` added to the method. The user's inputted `loginEmail` and `loginPswd` can be authenticated to check if it is in the file.
	  ○ For the adding of new films, searching ensures the film to be added by the employee is unique in every field, throwing the `InvalidFilmInputException` otherwise.

[12] Medium. 2022. *An Simplified Explanation of Linear Search*. [online] Available at: <https://medium.com/karuna-sehgal/an-simplified-explanation-of-linear-search-5056942ba965> [Accessed 7 January 2022].
[13] www.javatpoint.com. 2022. *Linear Search - javatpoint*. [online] Available at: <https://www.javatpoint.com/linear-search> [Accessed 8 January 2022].

Code

```java
//Method 'checkLogin()' checks the login details of the user
@SuppressWarnings("unused")
private void checkLogin() throws IOException{
    Main m = new Main();

    //IF the user is an employee the program transitions to the 'Employee Home.fxml' page
    if(email.getText().toString().equals("clubhousecinemas@gmail.com") && password.getText().toString().equals("123456")) {
        Main.setEmployeeMode(true);
        m.changeScene("Employee Home.fxml");
    }
    //ELSE IF any of the entry fields are empty an error message is displayed
    else if(email.getText().isEmpty() || password.getText().isEmpty()) {
        if(email.getText().isEmpty() && password.getText().isEmpty()){
            invalidEntry.setText("*Please enter email and password.*");
        }
        else if(email.getText().isEmpty()) {
            invalidEntry.setText("*Please enter email.*");
        }
        else if(password.getText().isEmpty()) {
            invalidEntry.setText("*Please enter password.*");
        }

    }
    //ELSE checks the user's details against the values in the 'Registration details.txt' file
    else {
        //Retrieves the user inputted details from the text fields and stores them in variables
        String loginEmail = email.getText();
        String loginPswd = password.getText();

        //Stores the file path in the String variable 'filepath'
        String filepath = "C:\\Users\\ruben\\OneDrive\\Documents\\Ruben School"
                + "\\Year 12\\Computer Science\\IA\\Registration details.txt";


        //Creates FileReader and BufferedReader to be able to read the file
        BufferedReader br = new BufferedReader(new FileReader(filepath));
        //Creates LinkedLists to store email and passwords
        LinkedList<String> emails = new LinkedList<String>();
        LinkedList<String> passwords = new LinkedList<String>();
        String line = br.readLine(); //Reads the first line of the file
        while (line != null) { //WHILE loop to iterate through the lines of the file
            String[] data =line.split(";"); //Assings a semi-colon as the data delimiter
            //Adds data values to their respective lists
            String emailData = data[2];
            String pswdData = data[3];
            emails.add(emailData);
            passwords.add(pswdData);
            line = br.readLine(); //Reads the next line
        }
        br.close(); //Closes BufferedReader


        //Iterartes through LinkedLists to authenticate the email and password
        int userPos = 0;
        for(int i=0;i<emails.size();i++) {
            if(emails.get(i).equals(loginEmail) && passwords.get(i).equals(loginPswd)) {
                userPos = i;
                currentUser = loginEmail;
                m.changeScene("View Films.fxml");
            }
            else {
                invalidEntry.setText("*Invalid email or password*");
            }
        }

    }
}
```

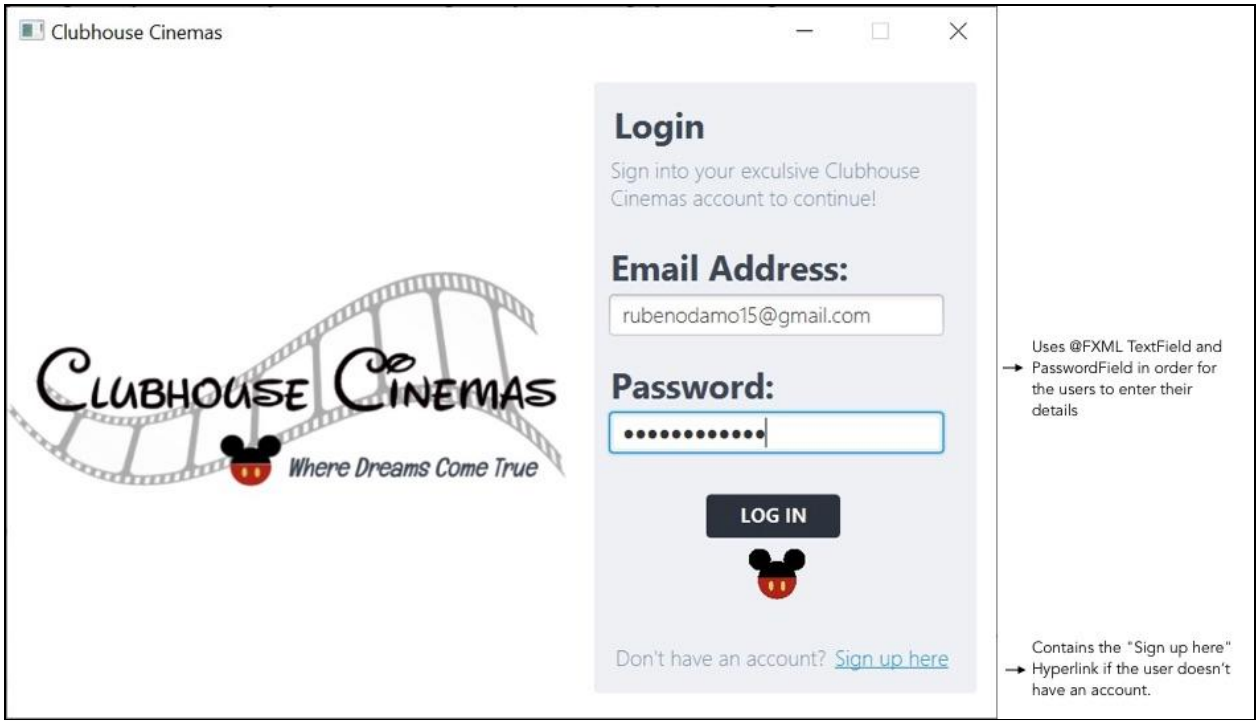
This method `checkLogin()` performs the authentication checks on the user input, within the `Login.fxml` page.

Here is the search "key" denoting the values the program is looking for within the file and thus the `LinkedLists`.

Refers to an example of searching where the program searches for the input email and password.

Ingenuity is shown through the use of a `for` loop, a recursion technique, in order to search whether the inputted information is correct.

Implementation



C.1.7 JavaFX

JavaFX is a Java library that allows for the creation of graphical user interfaces (GUI) in the development of certain applications.[14] For this application, JavaFX was used to dictate the interaction and communication between the Clubhouse Cinema GUI and the user actions, thus allowing for unique data representation based on user input. SceneBuilder[15] was also employed to create each window as well.

Appropriateness	Ingenuity

• The application desired by my client, Mickey Mouse, required multiple graphical components in order for the user to interact with the application (see Success Criteria 1.1-5.4).
• JavaFX provided specific components such as a DatePicker, Buttons and a Menu Bar.
• Custom design elements such as dynamic buttons and integrated transitions from the MenuBar to and from FXML windows were made possible as a result.
• Overall JavaFX allowed the completion of Success Criteria 1.1-5.4.
	• The uses of JavaFX:
	  ○ Provided dynamic buttons, with colour changing and movement capabilities when clicked. This was done using the `import javafx.scene.control.Button;` and `import javafx.animation.*;` within lambda action listener expressions.
	  ○ The import of JavaFX allows for the colour change `.setStyle();` , angle change `.setByAngle();` and position change `.setToX();`
	  ○ Also supports integrated transitions between FXML windows via the custom `changeScene();` method.
	  ○ Provides content-specific components which simplify the overall user experience.

[14] Openjfx.io. 2022. *JavaFX*. [online] Available at: <https://openjfx.io/> [Accessed 7 January 2022].
[15] Oracle.com. 2022. *JavaFX Scene Builder Information*. [online] Available at: <https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html> [Accessed 7 January 2022].

Code

changeScene()

```
//Method 'changeScene()' allows any FXML window to be opened, that is integrated within the application window
public void changeScene(String fxml) throws IOException {
    Parent pane = FXMLLoader.load(getClass().getResource(fxml)); //Defines the variable 'pane' of the FXML window wanted
    stg.getScene().setRoot(pane); //Sets the stage as the new pane
    stg.sizeToScene(); //Resizes the stage
}
```

This method `changeScene()` takes the parameter reading of the `FXML` file name, and loads it to the `Stage stg`.

This is ingenious due to it providing maintainability to the code and direct, integrated transitions to windows.

Refers to the code which executes the integrated transitions between windows.

Example use of changeScene()

```
//When the 'Sign up here' Hyperlink is pressed it transitions to the 'Sign Up.fxml' page
public void goToReg(ActionEvent event) throws IOException{
    Main m = new Main();
    m.changeScene("Sign Up.fxml");
}

//Method 'checkLogin()' checks the login details of the user
@SuppressWarnings("unused")
private void checkLogin() throws IOException{
    Main m = new Main();

    //IF the user is an employee the program transitions to the 'Employee Home.fxml' page
    if(email.getText().toString().equals("clubhousecinemas@gmail.com") && password.getText().toString().equals("123456")) {
        Main.setEmployeeMode(true);
        m.changeScene("Employee Home.fxml");
    }
    // (...)
```

This is an example use of the method `changeScene()`. It shows when the user is authenticated and or when a button is clicked, it opens a new `FXML` page.

```
@FXML
private DatePicker selectedDate;

@FXML
void initialize() throws IOException {
    selectedFilm = Main.getSelectedFilmTitle(); //Assigns the film chosen by the user to the 'selectedFilm' variable

    //Initialises the ObservableLists
    ObservableList<String> times = FXCollections.observableArrayList();
    ObservableList<String> numberOfTickets = FXCollections.observableArrayList("0","1","2","3","4","5","6","7","8");
    ObservableList<String> validation = FXCollections.observableArrayList("0");

    //Creates 'String' variables for 'endDate' and 'age'
    String endDate = "";
    String age = "";

    //Creates FileReader and BufferedReader to be able to read the file
    FileReader fr = new FileReader("films.txt");
    BufferedReader br=new BufferedReader(fr);
    String line=br.readLine(); //Reads the first line of the file
    while(line!=null) { //WHILE loop to iterate through the lines of the file
        String[] data=line.split(";"); //Assings a semi-colon as the data delimiter
        //Finds data associated with the chosen film and assigns the film details to corresponding variables
        if(data[0].equals(selectedFilm))
        {
            selectedFilmTitle.setText(data[0]);
            age = data[8];
            times = FXCollections.observableArrayList(data[5],data[6],data[7]);
            endDate = data[4];
        }
        line=br.readLine(); //Reads the next line
    }

    //Formats the DatePicker so the user cannot pick a date before today's  date or after the end date
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
    LocalDate today = LocalDate.now();
    LocalDate end = LocalDate.parse(endDate, formatter);
    final Callback<DatePicker, DateCell> dayCellFactory =
            new Callback<DatePicker, DateCell>() {
                @Override
                public DateCell call(final DatePicker selectedDate) {
                    return new DateCell() {
                        @Override
                        public void updateItem(LocalDate item, boolean empty) {
                            super.updateItem(item, empty);

                            if (item.isBefore(today) || item.isAfter(end)) {
                                    setDisable(true);
                                    setStyle("-fx-background-color: #ffc0cb;");
                            }
                        }
                    };
                }
            };
    selectedDate.setDayCellFactory(dayCellFactory);
    selectedDate.setValue(LocalDate.now());
```

Refers to the code which created the `DatePicker` for the `TicketBooking.fxml` window.

`DateTimeFormatter`, formats all dates including the one shown on the `DatePicker`, in the format "dd/MM/yyyy".

This is ingenious as it sets a readable format which is understood by the user. This format may also allow sorting in the future.

This code provides ingenuity by blocking dates previous to the current date and dates after the end screening date. These blocked dates cannot be selected.

→ Refers to the code which sets up the cinema floor plan.

```java
/*
 * Method 'setUpSeats()' constructs the visual outlook of the cinema floorplan giving seats a colour depending on availability and selection
 * available = #edf0f4  selected = #23b33b  unavailable = #e40606
 */
private void setUpSeats() {
    for(int i=0; i<seats.length; i++){ //FOR loop - iterates through arrays
        if(bookings[i]==false){ //IF the seat is available...
            seats[i].setStyle("-fx-background-color:  #edf0f4"); //Gives button a GREY colour
            int finalI1 = i;

            seats[i].setOnAction(event -> { //Using a lambda expression it acts as an action listener executing the code
                if(booked[finalI1]==false){ //IF the seat is selected...
                    //Checks user hasn't booked too many seats
                    if(numberOfSeats<maxSeats){
                        numberOfSeats++;
                        seats[finalI1].setStyle("-fx-background-color:  #23b33b"); //Gives button a GREEN colour
                        setBookedSeats(seats[finalI1], true);
                    }else { //Outputs error message
                        Alert alert = new Alert(AlertType.WARNING, "Error: Maximum seat limit reached!",
                                ButtonType.OK);
                        alert.showAndWait();
                        if (alert.getResult() == ButtonType.OK) {
                            return;
                        }
                    }
                }
                else if(booked[finalI1]==true){ //IF the user unselects a seat...
                    numberOfSeats--; //Subtracts 1 from the number of selected seats
                    seats[finalI1].setStyle("-fx-background-color:  #edf0f4");//Gives button a GREY colour
                    setBookedSeats(seats[finalI1], false);
                }
                popSeat(seats[finalI1]);
            });
        }else if(bookings[i]==true){ //IF the seat is unavailable...
            seats[i].setStyle("-fx-background-color:  #e40606"); //Gives button a RED colour
            int finalI = i;
            seats[i].setOnAction(event -> rotateButton(seats[finalI]));
        }
    }
}

 //Method 'rotateButton()' performs the animation functions when a seat button is selected
 public void rotateButton(Button btn){
     if(rotatedpane ==false){
         rotatedpane =true;
         RotateTransition rt=new RotateTransition(Duration.millis(60),btn);
         rt.setByAngle(45); //Sets the angle the button will rotate by
         rt.setCycleCount(2); //Sets the number of cycles in the animation
         rt.setAutoReverse(true); //Sets auto reverse flag to make animation run back and forth
         rt.play(); //Executes the button rotation

         //When animation is complete, returns the button back to original position
         rt.setOnFinished(event -> { //Using a lambda expression it acts as an action listener executing the code
             RotateTransition rt2=new RotateTransition(Duration.millis(60),btn);
             rt2.setByAngle(-45);
             rt2.setCycleCount(2);
             rt2.setAutoReverse(true);
             rt2.play();
             rt2.setOnFinished(event1 -> rotatedpane =false);
         });
     }
 }

 //Method 'popSeat()' performs the animation functions when a seat button is selected
 private void popSeat(Button btn) {
     ScaleTransition st = new ScaleTransition(Duration.millis(200), btn);
     st.setToX(1.2); //Sets the final X position of the button
     st.setToY(1.2); //Sets the final X position of the button
     st.setRate(1.5); //Sets the speed of the animation
     st.setCycleCount(1); //Sets the number of cycles in the animation
     st.play(); //Executes the button scalling


     st.setOnFinished(event -> { //Using a lambda expression it acts as an action listener executing the code
         ScaleTransition st2 = new ScaleTransition(Duration.millis(200), btn);
         st2.setToX(1);
         st2.setToY(1);
         st2.setRate(1.5);
         st2.setCycleCount(1);
         st2.play();
     });
 }
```

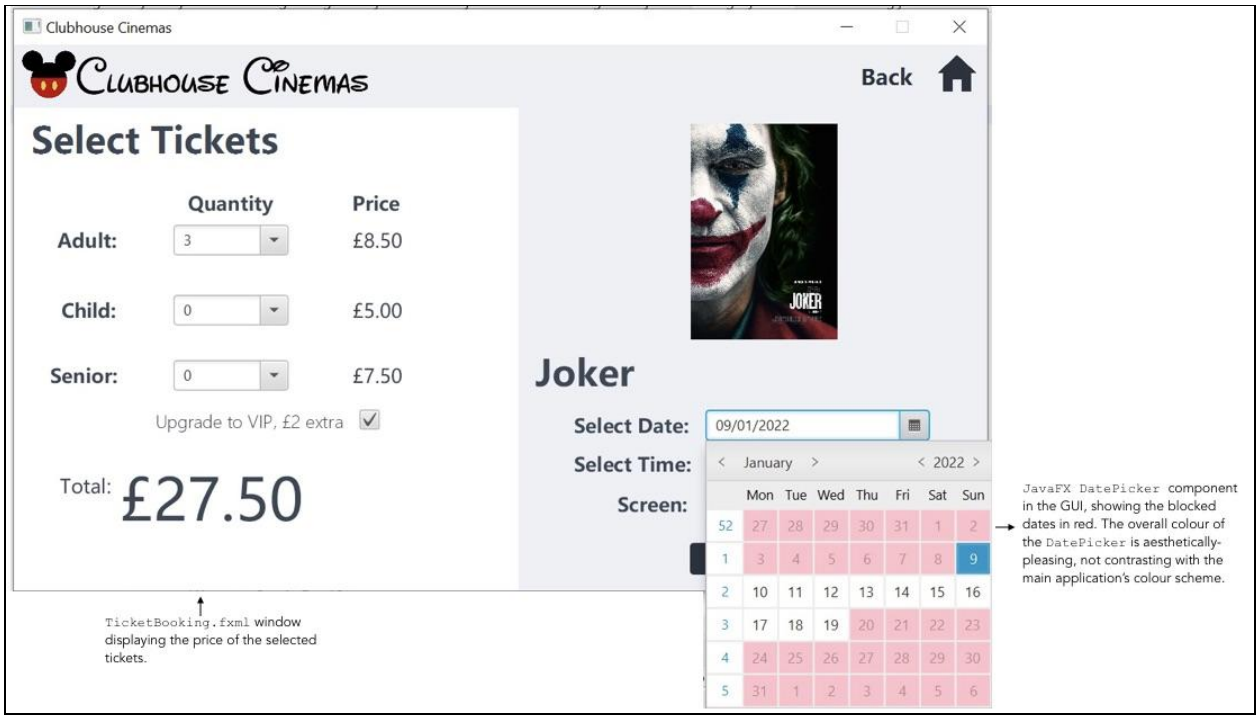
Uses lambda expression which is used as an action listener, executing the relevant code for when a seat button is selected.

The method setUpSeats(), also shown in C.1.1 shows how the .setStyle() function can be used to change seat colour.

This is ingenious as it suggests the reusability of the codes, because one seat button is able to hold and display many colours.

Further lambda expression which executes when the animation is complete. This is ingenious as it can be used to make a structural GUI format correction when a rotation/scale animation is complete.

Implementation



C.1.8 JavaMail

JavaMail is an API (application programming interface) that is used to compose, write and read emails.[16] It provides a platform-independent and protocol-independent framework to build mail and messaging applications.[17]

Specifically, for this application JavaMail used the Simple Mail Transfer Protocol (SMTP)[18] mechanism to deliver booking confirmation emails, to allow for the interaction and communication between the main bookings database and the user. This ensured that every user would have a customised, coherent, PDF booking receipt in their inbox.

Appropriateness	**Ingenuity**

• The application requested by my client, Mickey Mouse, required a unique, saveable message for the user to receive after booking (see Success Criteria 4.6.2).
• JavaMail provided mail transfer capabilities through a button click.
• It enabled custom design elements, like a user-tailored email message and unique PDF booking receipt to be sent.
• Overall JavaMail allowed for the completion of Success Criteria 4.6.2.
	• The uses of JavaMail:
	  ○ Provided a well structured message content whilst also abstracting the specific details behind mail transfer. This was done using the `import javax.mail.internet.MimeMessage;` and `import javax.mail.internet.MimeBodyPart;` within a try catch block.
	  ○ Also allows for secure integrated mail transfer within the `sendEmail(){}` method, which is provided with the `import javax.mail.PasswordAuthentication;`
	  ○ JavaMail provides email labelling where an email can be attributed to a `String type` and sent using the `.send();` function.

[16] www.javatpoint.com. 2022. *Java Mail Tutorial- javatpoint*. [online] Available at: <https://www.javatpoint.com/java-mail-api-tutorial> [Accessed 7 January 2022].
[17] Oracle.com. 2022. *JavaMail API*. [online] Available at: <https://www.oracle.com/java/technologies/javamail.html> [Accessed 7 January 2022].
[18] GeeksforGeeks. 2022. *Simple Mail Transfer Protocol (SMTP) - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/simple-mail-transfer-protocol-smtp/> [Accessed 7 January 2022].

Code

SendEmail.java class and sendEmail() method

```java
public class SendEmail { //Controller class which sends an email to the user

    //Method 'sendEmail()' creates a bookign confirmation email and sends it to the user
    static void sendEmail(String recipient, String type) {

        //Initialises 'username' and 'password' variables that cannot be reassigned
        final String username = "clubhousecinemas@gmail.com";
        final String password = "clubhousecinemas123";

        /*JavaMail Properties is used to set in the session objects and to create the session object.
         * SMTP - Simple Mail Transfer Protocol is used*/
        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        //Creates 'Session' object that provides access to JavaMail Protocols
        Session session = Session.getInstance(props,new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username, password);
            }
        });

        //Creates ByteArrayOutputStream and sets it to null
        ByteArrayOutputStream outputStream = null;

      //Store the message content in the 'String' variable 'content'
      String content = "Hello " + Confirmation.name + ",\n\n" +
              "Thank you for choosing Clubhouse Cinemas. Your booking for the film " + Main.getSelectedFilmTitle() + " has been confirmed."
              + " Please, keep this email with the PDF receipt as proof of your booking.\n"
              + "\nLooking forward to seeing you on " + Confirmation.finalDate +", at "
              + Confirmation.finalTime +
              "!\n\nEnjoy the film!\n\nMickey Mouse\nCEO of Clubhouse Cinemas";

      try { //TRY-CATCH Block
          //Creates a 'MimeBodyPart' and adds the message content to it
          MimeBodyPart textBodyPart = new MimeBodyPart();
          textBodyPart.setText(content);

          //Assigns 'outputStream' to a new ByteArrayOutputStream
          outputStream = new ByteArrayOutputStream();
          writePdf(outputStream); //Calls the 'writePdf()' method
          byte[] bytes = outputStream.toByteArray(); //Creates a byte array called 'bytes'

          //Creates a link between the application and the pdf so data can be written to it
          DataSource dataSource = new ByteArrayDataSource(bytes, "application/pdf");
          //Creates a 'MimeBodyPart' and adds the pdf to it
          MimeBodyPart pdfBodyPart = new MimeBodyPart();
          pdfBodyPart.setDataHandler(new DataHandler(dataSource));
          pdfBodyPart.setFileName("BookingConfirmation.pdf");

          //Creates a 'MimeMultipart' and adds the 'textBodyPart' and 'pdfBodyPart' to it
          MimeMultipart mimeMultipart = new MimeMultipart();
          mimeMultipart.addBodyPart(textBodyPart);
          mimeMultipart.addBodyPart(pdfBodyPart);

          //Creates a new message setting the sender and reciever addresses
          Message message = new MimeMessage(session);
          message.setFrom(new InternetAddress("clubhousecinemas@gmail.com"));
          message.setRecipient(Message.RecipientType.TO, new InternetAddress(recipient));

          if (type.equals("confirmation")) { //IF the email is a "confirmation" email
              //Set the subject and message content
              message.setSubject(Confirmation.bookingId+" - Booking Confirmation for "+Main.getSelectedFilmTitle());
              message.setContent(mimeMultipart);
          }

          Transport.send(message); //Sends email

      } catch (Exception e) {
          throw new RuntimeException(e);

      } finally {
          //cleans off outputStream
          if(null != outputStream) {
              try { outputStream.close(); outputStream = null; }
              catch(Exception ex) { }
          }
      }
  }
```

Refers to the code which sends the confirmation email.

SMTP `Properties` are used to deliver the email in a reliable, coherent and consistent fashion.

Here shows the use of the `java.mail.Authenticator()`. Utilising this is ingenious as it allows for secure, integrated mail transfer. This means the users bookings a guaranteed to only reach the user's email from clubhousecinemas@gmail.com.

This depicts the message content template that is personalised with the users information.

This is ingenious as it conveys the personal experience with booking a film. It attributes the email to the user and only the user.

Here shows the email labelling system. The confirmation email is labelled as "confirmation".

The email is then sent using `.send()`.

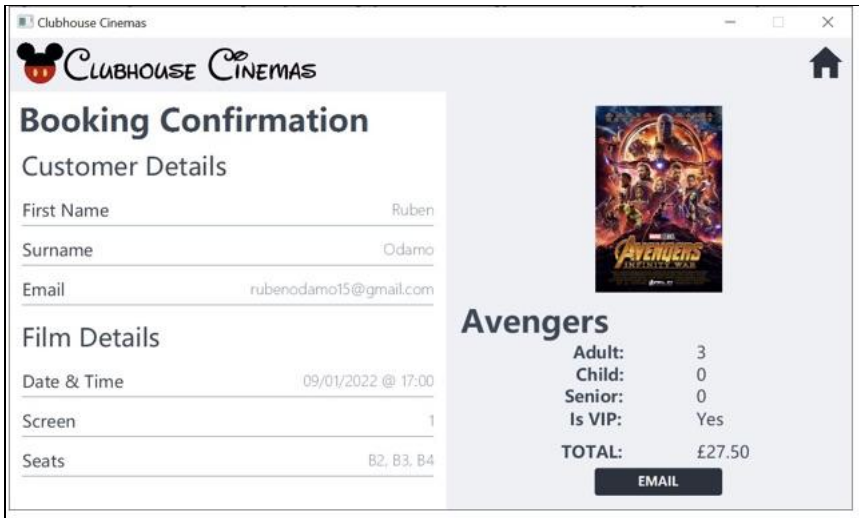
This shows the ingenuity as other email 'types' can be created depending on the scenario. This allows for future communications between Clubhouse Cinemas and the user to be organised.

Example use of sendEmail()

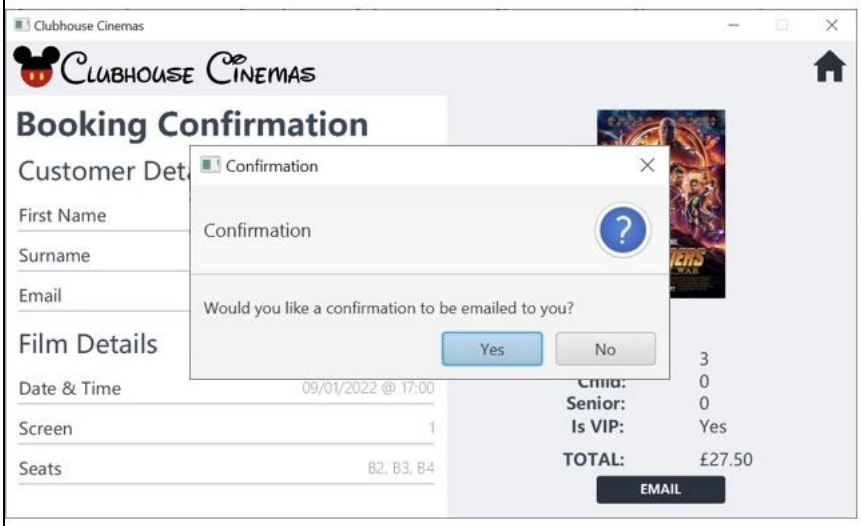
```java
    //Method 'emailConfirmation()' is executed when the user clicks on the 'EMAIL' buttoon
    public void emailConfirmation(ActionEvent event) throws IOException{
        //Outputs confirmation alert
        Alert alert = new Alert(AlertType.CONFIRMATION, "Would you like a confirmation to be emailed to you?",
                ButtonType.YES, ButtonType.NO);
        alert.showAndWait();

        //IF the user selects "yes" a booking confirmation email is sent
        if(alert.getResult() == ButtonType.YES) {
            SendEmail.sendEmail(Login.getCurrentUser(), "confirmation");
            alert.close();
        }
        //ELSE the alert is closed
        else {
            alert.close();
            return;
        }
    }
```

This is an example use of the `sendEmail()` method, where when the user presses the "EMAIL" button, it sends a booking confirmation email.

Implementation

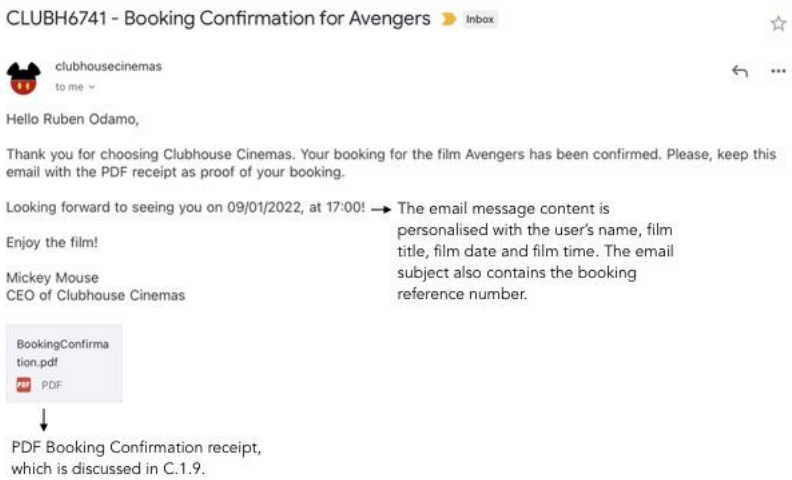


Displays the users booking confirmation.



When the user clicks on the "EMAIL" button, a confirmation alert appears requesting for a yes/no answer for an email.

This is ingenious as it makes the user aware an email is to be sent, so that a spam mailing list is not created.



Example booking confirmation email sent to my inbox.

The email message content is personalised with the user's name, film title, film date and film time. The email subject also contains the booking reference number.

PDF Booking Confirmation receipt, which is discussed in C.1.9.

C.1.9 iText PDF Generator

The Portable Document Format (PDF) is a file format for presenting data that is independent of application software, hardware, and operating systems. Each PDF file contains a description of a fixed-layout flat document, including text, fonts, graphics, and other data required to display it.[19] iText[20] is a Java PDF library that allows the creation of PDF documents, such as the `BookingConformation.pdf`, which is sent to a Clubhouse Cinema user via email.

For this application iText was used in conjunction with JavaMail (C.1.8) to ensure that the user had access to an external copy of their booking details outside of the main application.

Appropriateness	**Ingenuity**

• The application requested by my client, Mickey Mouse, required personalised user experiences for each booking, organised via employee actions (see Success Criteria 4.6, 5.4).
• iText PDF Generator allowed me to solve my clients problem by connecting each individual booking to a unique booking confirmation receipt, which is PDF based.
• This allowed the completion of Success Criteria 4.6.2, 4.6.3.
	• The uses of iText PDF Generator:
	  ○ Allowed for flexible PDF development with the `import com.itextpdf.text.pdf.PdfWriter;`, `import com.itextpdf.text.Document;` and `import com.itextpdf.text.Paragraph;`.
	  ○ The above ensured that within the `writePdf(){}` method, different elements such as text and images can be added to the PDF using the `.add();` function.
	  ○ Personalisation was also implemented through the `import com.itextpdf.text.Font;` and `import com.itextpdf.text.FontFactory;`.
	  ○ These provided a variety of modern fonts to make the PDF unique to Clubhouse Cinemas.

[19] Tutorialspoint.com. 2022. *iText - Overview*. [online] Available at: <https://www.tutorialspoint.com/itext/itext_overview.htm> [Accessed 7 January 2022].
[20] iTextpdf. 2022. *The Leading PDF Library for Developers | iText*. [online] Available at: <https://itextpdf.com/en> [Accessed 8 January 2022].

772 words

Code

```java
import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.Font;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.Image;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.BarcodeQRCode;
import com.itextpdf.text.pdf.PdfWriter;
```

The imports required for the PDF generation.

Refers to the code that generates the `BookingConfirmation.pdf`.

The method `writePdf()` is executed within the `SendEmail.java` class.

```java
//Method 'writePdf' creates the 'BookingConfirmation.pdf' to be emailed to the user
public static void writePdf(OutputStream outputStream) throws Exception {
    //Creates fonts to be used in the PDF
    Font title = FontFactory.getFont(FontFactory.HELVETICA, 36f, Font.BOLD);
    Font subtitle = FontFactory.getFont(FontFactory.HELVETICA, 16f, Font.BOLD);
    Font italics = FontFactory.getFont(FontFactory.HELVETICA, 12f, Font.ITALIC);
```

This is ingenious as the fonts can be reused for other PDF creations.

```java
    //Creates a new document and writes it to the outputStream so it can be sent
    Document document = new Document();
    PdfWriter.getInstance(document, outputStream);

    document.open(); //Opens the document
```

Here shows the assigning of different font styles to different variables. This allows for the PDF to be personalised in various styles. Having different fonts ensures the `BookingConfirmation.pdf` is more readable and clear to the user.

```java
    //Adds document details
    document.addTitle("Booking Confirmation PDF");
    document.addSubject("Receipt PDF");
    document.addAuthor("Mickey Mouse");
    document.addCreator("Clubhouse Cinemas");
```

This shows the addition of document details, e.g. `.addTitle()`. This is ingenious as aside from the email the user can also tell where the document came from.

```java
    //Creates new paragraph and adds the content of the receipt to the paragraph
    Paragraph paragraph = new Paragraph();
    paragraph.add(new Chunk("Your Booking Receipt\n\n", title));
    paragraph.add(new Chunk("Booking ID: "+Confirmation.bookingId+"\n"+
                            "Cinema: Clubhouse Cinemas"+"\n\n"+
                            "Film: "+Main.getSelectedFilmTitle()+"\n"+
                            "Screen: "+TicketBooking.screenNum+"\n"+
                            "Date: "+Confirmation.finalDate+"\n"+
                            "Time: "+Confirmation.finalTime+"\n"+
                            "Tickets: "+TicketBooking.adultTickets+" x Adult, "+ TicketBooking.childTickets+" x Child, "+
                            TicketBooking.seniorTickets+" x Senior"+"\n"+
                            "Seats: "+SeatBooking.userSeats+"\n"+
                            "is VIP: "+Confirmation.vipConf+"\n\n"+
                            "Total Payment: "+"£"+String.format("%.2f", TicketBooking.total)+"\n\n"));
```

This depicts the PDF content template that is personalised with the users information.

This is ingenious as it conveys the personal experience with booking a film. It attributes the receipt to the user and only the user.

```java
    paragraph.add(new Chunk("Face Covering Information\n", subtitle));
    paragraph.add(new Chunk("Please note that face coverings are optional in English cinemas. "
            + "They are mandatory in Scotland and Wales for guests over the age of 11, except those exempt for health reasons. "
            + "If you do not have a face covering when arriving at the cinema, in order to gain admission you will be required to "
            + "purchase one for £1.\n\n",italics));
    paragraph.add(new Chunk("Ticket Collection Information\n", subtitle));
    paragraph.add(new Chunk("There are several ways to gain admission to your film:\n"
            + "- Show this email on your smartphone to the usher to scan the barcode. And make your payment.\n"
            + "- Alternatively, please print out this confirmation, bring it with you to the cinema, and present it to the usher to"
            + " scan the barcode below for admittance, or show this email on your smartphone. And make your payment.", italics));

    //Adds the paragraph content to the document
    document.add(paragraph);

    //Creates a QR code that when scanned shows "Valid Booking - bookingID"
    BarcodeQRCode barcodeQRCode = new BarcodeQRCode("Valid Booking - "+Confirmation.bookingId, 1000, 1000, null);
    Image codeQrImage = barcodeQRCode.getImage(); //Generates QR code image
    codeQrImage.scaleAbsolute(200, 200); //Resizes QR code
    document.add(codeQrImage); //Adds the QR code to the PDF
```

`.add()` demonstastes the ease of adding elements to the PDF.

```java
    //Creates an Image and assigns the Clubhouse Cinemas logo to it
    Image img = Image.getInstance("C:\\Users\\ruben\\OneDrive\\Documents\\Ruben School\\Year 12\\Computer Science\\IA\\ClubhouseCinemas Logo.png");
    img.scaleAbsolute(312f, 129.5f); //Resizes image
    img.setAbsolutePosition(4, 22); //Positions image on the PDF
    document.add(img); //Adds image to PDF

    document.close(); //Closes the document
}
```

Implementation

Your Booking Receipt

→ Shows custom booking receipt, with all of the user's booking details.

Booking ID: CLUBH6741
Cinema: Clubhouse Cinemas

Film: Avengers
Screen: 1
Date: 09/01/2022
Time: 17:00
Tickets: 3 x Adult, 0 x Child, 0 x Senior
Seats: B2, B3, B4
is VIP: Yes

Total Payment: £27.50

Face Covering Information

Please note that face coverings are optional in English cinemas. They are mandatory in Scotland and Wales for guests over the age of 11, except those exempt for health reasons. If you do not have a face covering when arriving at the cinema, in order to gain admission you will be required to purchase one for £1.

Ticket Collection Information

There are several ways to gain admission to your film:
- Show this email on your smartphone to the usher to scan the barcode. And make your payment.
- Alternatively, please print out this confirmation, bring it with you to the cinema, and present it to the usher to scan the barcode below for admittance, or show this email on your smartphone. And make your payment.

→ QR Code to be scanned by usher, explained in C.1.10.

Clubhouse Cinemas
Where Dreams Come True

C.1.10 iText QR Code Generator

A Quick Response code (QR code) is a two-dimensional bar code that contains a matrix of small squares in which information is stored.[21] iText[22] is a Java PDF library that allows the creation of PDF documents. Within this QR codes can be generated and placed upon the newly created PDF.

For this application iText was used in conjunction with JavaMail (C.1.8) to ensure that the employee can further validate a user's booking through an imaging device.

Appropriateness	**Ingenuity**

• The application requested by my client, Mickey Mouse, required a fast secure reliable way of confirming a booking (see Success Criteria 4.6, 4.7, 5.4).
• iText QR Code Generator allowed me to solve my clients problem by connecting each individual booking to a unique QR code, which when scanned provides the employee with a validity feature.
• This allowed the completion of Success Criteria 4.6.2.
	• The uses of iText QR Code Generator:
	  ○ Provides a clear scannable code with the `import com.itextpdf.text.pdf.BarcodeQRCode;` and `import com.itextpdf.text.Image;`.
	  ○ The above ensured the QR code can be well placed, with flexible sizing, within the PDF using `.scaleAbsolute(int, int);` and `.add();` functions.
	  ○ Allows for an extra layer of validation that is quick and error-free, with the unique code being generated based on the `Confirmation.bookinId` value.
	  ○ Data can also be stored efficiently within the QR code, with a large amount of data being able to be stored in a limited space.

[21] www.javatpoint.com. 2022. *Generating QR Code in Java - Javatpoint*. [online] Available at: <https://www.javatpoint.com/generating-qr-code-in-java> [Accessed 7 January 2022].
[22] iTextpdf. 2022. *The Leading PDF Library for Developers | iText*. [online] Available at: <https://itextpdf.com/en> [Accessed 8 January 2022].

Code

```
import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.Font;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.Image;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.BarcodeQRCode;
import com.itextpdf.text.pdf.PdfWriter;

//Creates a QR code that when scanned shows "Valid Booking - bookingID"
BarcodeQRCode barcodeQRCode = new BarcodeQRCode("Valid Booking - "+Confirmation.bookingId, 1000, 1000, null);
Image codeQrImage = barcodeQRCode.getImage(); //Generates QR code image
codeQrImage.scaleAbsolute(200, 200); //Resizes QR code
document.add(codeQrImage); //Adds the QR code to the PDF
```

Refers to the code that generates the QR code with the `BookingConfirmation.pdf`.

The imports required for the QR code generation.

Here shows the resizing of the QR so it is big enough to scan. This is ingenious as it minimises user input on the QR code readability.

`.add()` demonstastes the ease of adding elements to the PDF.

Implementation

Your Booking Receipt

Shows custom booking receipt, with all of the user's booking details, explained in C.1.10.

Booking ID: CLUBH6741
Cinema: Clubhouse Cinemas

Film: Avengers
Screen: 1
Date: 09/01/2022
Time: 17:00
Tickets: 3 x Adult, 0 x Child, 0 x Senior
Seats: B2, B3, B4
is VIP: Yes

Total Payment: £27.50

Face Covering Information

Please note that face coverings are optional in English cinemas. They are mandatory in Scotland and Wales for guests over the age of 11, except those exempt for health reasons. If you do not have a face covering when arriving at the cinema, in order to gain admission you will be required to purchase one for £1.

Ticket Collection Information

There are several ways to gain admission to your film:
- Show this email on your smartphone to the usher to scan the barcode. And make your payment.
- Alternatively, please print out this confirmation, bring it with you to the cinema, and present it to the usher to scan the barcode below for admittance, or show this email on your smartphone. And make your payment.

QR Code to be scanned by usher produces a "Vaild Booking" message.

This is ingenious as it adds an extra layer of validation to the booking, further ensuring it is unique. In addition to this it improves the efficiency of the booking validation process, making it quicker for the employee to verify bookings.

Clubhouse Cinemas
Where Dreams Come True

C.2 Bibliography

GeeksforGeeks. 2022. *LinkedList in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/linked-list-in-java/> [Accessed 7 January 2022].

GeeksforGeeks. 2022. *Simple Mail Transfer Protocol (SMTP) - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/simple-mail-transfer-protocol-smtp/> [Accessed 7 January 2022].

Homeandlearn.co.uk. 2022. *What is a Text File*. [online] Available at: <https://www.homeandlearn.co.uk/NET/nets8p1.html> [Accessed 7 January 2022].

Ib.compscihub.net. 2022. [online] Available at: <https://ib.compscihub.net/wp-content/uploads/2018/07/D.2.2.pdf> [Accessed 7 January 2022].

iTextpdf. 2022. *The Leading PDF Library for Developers | iText*. [online] Available at: <https://itextpdf.com/en> [Accessed 8 January 2022].

Medium. 2022. *An Simplified Explanation of Linear Search*. [online] Available at: <https://medium.com/karuna-sehgal/an-simplified-explanation-of-linear-search-5056942ba965> [Accessed 7 January 2022].

Medium. 2022. *#SideNotes — Linked List — Abstract Data Type and Data Structure*. [online] Available at: <https://lucasmagnum.medium.com/sidenotes-linked-list-abstract-data-type-and-data-structure-fd2f8276ab53> [Accessed 7 January 2022].

Openjfx.io. 2022. *JavaFX*. [online] Available at: <https://openjfx.io/> [Accessed 7 January 2022].

Oracle.com. 2022. *JavaFX Scene Builder Information*. [online] Available at: <https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html> [Accessed 7 January 2022].

Oracle.com. 2022. *JavaMail API*. [online] Available at: <https://www.oracle.com/java/technologies/javamail.html> [Accessed 7 January 2022].

Sumo Logic. 2022. *What is Encapsulation in OOP? | Sumo Logic*. [online] Available at: <https://www.sumologic.com/glossary/encapsulation/> [Accessed 7 January 2022].

Techopedia.com. 2022. *What is Array in Java? - Definition from Techopedia*. [online] Available at: <https://www.techopedia.com/definition/1143/array-java> [Accessed 7 January 2022].

Tutorialspoint.com. 2022. *Java - Interfaces*. [online] Available at: <https://www.tutorialspoint.com/java/java_interfaces.htm> [Accessed 7 January 2022].

Tutorialspoint.com. 2022. *OOAD - Object Oriented Principles*. [online] Available at: <https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_principles.htm> [Accessed 7 January 2022].

Tutorialspoint.com. 2022. *iText - Overview*. [online] Available at: <https://www.tutorialspoint.com/itext/itext_overview.htm> [Accessed 7 January 2022].

Vertica. 2022. *1 - What are Complex Data Types? | Vertica*. [online] Available at: <https://www.vertica.com/blog/complex-data-types-in-sql-1-what-are-they/> [Accessed 7 January 2022].

Webopedia. 2022. *What is Sequential Access? | Webopedia*. [online] Available at: <https://www.webopedia.com/definitions/sequential-access/> [Accessed 7 January 2022].

www.javatpoint.com. 2022. *Dynamic Array in Java - Javatpoint*. [online] Available at: <https://www.javatpoint.com/dynamic-array-in-java> [Accessed 7 January 2022].

www.javatpoint.com. 2022. *Generating QR Code in Java - Javatpoint*. [online] Available at: <https://www.javatpoint.com/generating-qr-code-in-java> [Accessed 7 January 2022].

www.javatpoint.com. 2022. *Java Mail Tutorial- javatpoint*. [online] Available at: <https://www.javatpoint.com/java-mail-api-tutorial> [Accessed 7 January 2022].

www.javatpoint.com. 2022. *Linear Search - javatpoint*. [online] Available at: <https://www.javatpoint.com/linear-search> [Accessed 8 January 2022].