Online Booking System-Cinema

**Project Report For**

**CS522: Software Quality Assurance and Test Automation**

**Group No: 1**

**Submitted By**

**Pallavi Botchu 13772**

**Rupa Poosapati Chandran 12941**

**Abstract**

**Objective**

The main objective of this project is to show how testing can be done on a Java Application. “Cinema Booking System” is a Java application which helps users to reserve tickets for movies. It has all the features which are expected in a generic reservation system. We show different test cases we came up with and how the results are. Depending on the results we are able to fix the bugs in the source code.

**Description**

The Java application provides the users different time slots and available seats for those slots. The user should be able to select the number of seats and select the seats from the available seats. The application will then calculate the total price of tickets. The prices of tickets are set at different levels so that you can support child, adult and over-aged prices. The application is able to handle different scenarios and it shows respective messages for user to easily navigate it.

**Scope**

The Java Application can be extended to support for multiple movies and multiple timeslots over different days. The design can be extended to be used on any online platform so that the end users can reserve the seats on the internet.

Table of Contents

[1. Introduction 4](#_Toc436782843)

[1.1. Document Overview 4](#_Toc436782844)

[1.2. Problem Definition 4](#_Toc436782845)

[1.3. Literature Survey 4](#_Toc436782846)

[1.4. Architecture Diagram 5](#_Toc436782847)

[2. Overview of test results 5](#_Toc436782848)

[2.1. Bugged Source Code 5](#_Toc436782849)

[2.2. Test Results 5](#_Toc436782850)

[2.3. Algorithm 9](#_Toc436782866)

[2.5. Enhanced Source Code 10](#_Toc436782867)

[3. Detailed Test Results 10](#_Toc436782868)

[3.1. Control Flow Graph 10](#_Toc436782869)

[3.2. Cause Effect Graph 11](#_Toc436782870)

[3.3. Decision Table 13](#_Toc436782871)

[3.4. Performance Matrix 14](#_Toc436782872)

[3.4.1 .Description 14](#_Toc436782873)

[3.4.3. Percentage of statement Coverage 16](#_Toc436782875)

[3.4.4. Percentage of equivalence Coverage 16](#_Toc436782876)

[3.4.5. Percentage of branch Coverage 16](#_Toc436782877)

[3.4.6. Percentage of Boundary value Analysis Coverage 17](#_Toc436782878)

[Appendix A 18](#_Toc436782879)

[Appendix B 19](#_Toc436782880)

[Appendix C 25](#_Toc436782881)

[Ticket class 25](#_Toc436782882)

# 1. Introduction

## 1.1. Document Overview

This document is the software test report of Cinema Booking System Software development project. It contains the results of tests, which were executed during the testing phase of the Java Application.

## 1.2. Problem Definition

This is a Java Application developed to buy tickets for movies on different time slots. The application provides a really easy way for the user to select the available seats. This involves saving of the selections from multiple users so that users have dynamic way of knowing what seats are available for him/her to choose. For this the application maintains a data base of reserved and unreserved seats for every show. For this project we verified different scenarios which user can encounter while booking a show. Through this process we are able to suggest multiple changes to the code to make it a better application.

## 1.3. Literature Survey

In earlier days people have to go early, wait in a queue till user get chance to buy ticket for a movie, if it’s a new and recent movie they may /may not have tickets to buy. Moreover ,need to buy what are remaining(no choice of choosing seats).But with this online service just open this application filling data ,selecting the timings with desired seats at last order with rejoice. Any user can reserve tickets from anywhere to anyone. No needs to wait till others come to confirm tickets

1. Choice of choosing different show timings with available seats at that time slot.
2. Updated information been generated, for instance the reserved seats are excluded from list every time.
3. This is simple online java application which makes user life easy and comfortable.
4. Users billing details, private information been protected from other users. Safe to use.
5. If insufficient data is filled during process, warning sign appears in dialog box in order to refill correct information.
6. Use can decide whether to proceed or stop till last step of process by showing confirmation.
7. There are error detection dialogs which appear during the process of booking.
8. One can delete database information once all shows are done for the day.

## 1.4. Architecture Diagram

GUI - Selections

Show 1

- Price

- Seat Selection

Child: $

Adult: $

OAP: $

Users

Price Retrieval

Seat Reserve

# 2. Overview of test results

## 2.1. Bugged Source Code

Bugged code is a biased code that causes the software to give inappropriate results which are more likely to be noticed and fixed. Appendix A contains buggy code on which testing is performed.

## 2.2. Test Results

We came up with the following test cases to completely test the application. We went through different scenarios which a normal user will go through during the booking of a show. We also came up with different combinations where the application has to throw proper messages for it to be intuitive for the end user.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TestCase** | **Title** | **Description** | **Expected** | **Observed** | **Pass/Fail** |
| TC1 | Availability of seats across transactions | Select 2 seats | Should show selected seats for next transaction | Display the selected seats for next transaction | Pass |
|  |  | Select 5 seats | Should show selected seats for next transaction | Display the selected seats for next transaction | Pass |
|  |  |  |  |  |  |
| TC2 | Availability of seats across various time slots | Select 2 seats at 1pm | Show selected seats filled at 1pm | Display filled seats at 1pm | Pass |
|  |  | Select 2 same seats at 3pm | Show selected seats filled at 3pm | Display filled seats at 3pm | Pass |
|  |  | Select 2 same seats at 5pm | Show selected seats filled at 5pm | Display filled seats at 5pm | Pass |
|  |  |  |  |  |  |
| TC3 | Max allowed tickets per transaction is 5 | Select 5seats for adults | Should show confirm button to proceed after selection | Display confirmation after selection of seats | Pass |
|  |  | Select 5seats for child | Should show confirm button to proceed after selection | Display confirmation after selection of seats | Pass |
|  |  | Select 5seats for old people | Should show confirm button to proceed after selection | Display confirmation after selection of seats | Pass |
|  |  |  |  |  |  |
| TC4 | Cant reserve more than available seats | Select 6 seats | Should show ticket quantity error | Displays warning with no enough  seats remaining | Pass |
|  |  |  |  |  |  |
| TC5 | Selection of seats for all tickets | 2seats out of 3tickets | Display seat selection error | Not all seats have been assigned warning appears | Pass |
|  |  | 3seats out of 5tickets | Display seat selection error | Not all seats have been assigned warning appears | Pass |
|  |  |  |  |  |  |
| TC6 | User cant select same seat for multiple tickets |  | Error should be shown | No Message is being shown. | Fail |
| TC7 | User is prompted before confirmation |  | Confirmation of next process should appear | Decision of confirmation of process displays | Pass |
|  |  |  |  |  |  |
| TC8 | User shouldn't be able to order without selecting any tickets |  | Seat selection error dialog box should appear | Not all seats have been assigned error displays | Pass |
|  |  |  |  |  |  |
| TC9 | Validity of price | 2 in adults | Once time, adults have chosen  calculated price should be shown in text box | Displays calculated amount after filling timings, how many adults | Pass |
|  |  | 2 in adults+1 child | Once time, how many adults, child have chosen calculated price should be shown in text box timings | Displays calculated amount after filling, how many adults, child | Fail |
|  |  | 1adult+1child+2oap | Once time, number of adults, child, oap have chosen  calculated price should be shown in text box | Displays calculated amount after filling particular | Fail |
|  |  |  |  |  |  |
| TC10 | Database reset | Yes | Once delete db button clicked should show note with y/n button | Displays Yes/no button to select delete all database operations performed until now | Pass |
|  |  | No | Once delete db button clicked should show note with y/n button | If click no there is no change in database transactions | Pass |

## 

## 2.3. Algorithm

If (timeCombo())

Db.FullDataBaseGeneration(selectedTime); //Get the database for the time slot

timeArray=db.AvailableSeatsArrayReturn(SelectedTime); //Fetch array of available seats

if(adultQuantityCombo() || childQuantityCombo() ||oapQuantityCombo())

seatCombo1.addItem(seatArrayList.get(z)); )); // Display ticket for seat selection

totalString=getTotal();// Total price of tickets

if(totalCount>5)

tooManyTickets()// Cannot reserve more than 5 tickets

if(totalCount>availableSeats())

notEnoughSeats()// Cannot reserve more than available seats

if(totalCount>0) // If total tickets is 0, then we won’t display order button

showOrderButton()

if(orderButtonClick())

showDialogBox() // Shows user whether he/she wants to buy tickets or not.

if(checkSeatsForTickets())

notSelectedAllSeats()// User didn’t select seats for every ticket

if(duplicat SeatsForTickets())// User selected the same seat for multiple tickets

duplicateSeats()

reserveSeats() // Reserve Seats in the Database

confirmTickets() // Confirm that user bought tickets

if(resetButtonClick())

if(OptionPane.Yes)

resetDataBase()

## 2.5. Enhanced Source Code

Enhanced code is the final code after bug fix of critical, major and medium issues. Appendix C contains bug free or enhanced code after testing, debugging and fixing the bugs.

# 3. Detailed Test Results

## 3.1. Control Flow Graph

## 

A: if(timeCombo())

B: Db.FullDataBaseGeneration(selectedTime);

timeArray=db.AvailableSeatsArrayReturn(SelectedTime);

C:if(adultQuantityCombo() || childQuantityCombo() ||oapQuantityCombo())

D: seatCombo1.addItem(seatArrayList.get(z)); )); // Display ticket for seat selection

totalString=getTotal();// Total price of tickets

E: if(totalCount>5)

H: tooManyTickets()// Cannot reserve more than 5 tickets

F: if(totalCount>availableSeats())

I: notEnoughSeats()// Cannot reserve more than available seats

G:  if(totalCount>0) // If total tickets is 0, then we won’t display order button

J: showOrderButton()

L: if(orderButtonClick())

M: showDialogBox() // Shows user whether he/she wants to buy tickets or not.

N: if(checkSeatsForTickets())

P: notSelectedAllSeats()// User didn’t select seats for every ticket

O: if(duplicat SeatsForTickets())// User selected the same seat for multiple tickets

Q: duplicateSeats()

R: reserveSeats() // Reserve Seats in the Database

confirmTickets() // Confirm that user bought tickets

S: if(resetButtonClick())

T: if(OptionPane.Yes)

U: resetDataBase()

## 

## 3.2. Cause Effect Graph

A:adult

B:child

C:old aged person

D:price & availability

A:count>5

B:count>available seats

C:ticket=0

D:cannot reserve

A:selected all seats

B:duplicate seats

C:error

A:ticket

B:order

^

v

~

~

^

^

=>

v

~

v

~

~

~

~

~

~

=>

~

## 

## 3.3. Decision Table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s.no | operator | condition | Step1 | Step2 | Step3 | Step4 | Step5 | Step6 | Step7 | Step8 |
| 1 | OR( V) | A | T | T | T | T | F | F | F | F |
|  |  | B | T | T | F | F | T | T | F | F |
|  |  | C | T | F | T | F | T | F | T | F |
|  |  | outcome | T | T | T | T | T | T | T |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | operator | condition | Step1 | Step2 | Step3 | Step4 | Step5 | Step6 | Step7 | Step8 |
| 2 | OR(V) | A | T | T | T | T | F | F | F | F |
|  |  | B | T | T | F | F | T | T | F | F |
|  |  | C | T | F | T | F | T | F | T | F |
|  |  | outcome | T | T | T | T | T | T | T |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | operator | condition | Step1 | Step2 | Step3 | Step4 | Step5 | Step6 | Step7 | Step8 |
| 3 | OR(V) | A(~) | T | T | F | F |  |  |  |  |
|  |  | B | T | F | T | F |  |  |  |  |
|  |  | outcome | T | T | T |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | operator | condition | Step1 | Step2 | Step3 | Step4 | Step5 | Step6 | Step7 | Step8 |
| 4 | If and only if(=>) | A | T | F |  |  |  |  |  |  |
|  |  | outcome | T |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

## 3.4. Performance Matrix

### 3.4.1 .Description

Test Cases passed VS Failed

a) Number of Test cases planned VS executed

b) Number of Test cases passed VS failed

### 

**Defects Severity and Status**

**3.4.2. Types of Tests Performed**

**Unit Testing:** Each part of the program is isolated and to show individual parts are correct

**Stress Testing:** System is stressed beyond its specifications to check how and when it fails. Performed under heavy load like putting large number beyond storage capacity or database load

**Usability Testing:** User-friendliness check. Application flow is tested. New users understand the application easily during process. Proper guide whenever user stuck at any point. Basically system navigation is checked in this testing.

**Acceptance Testing:** Testing is done to verify if system meets the customer specified requirements. User or customer does testing whether to accept application.

**Functional Testing:** This testing ignores the internal parts and focus on the output as per the requirements or not. Black-box testing geared to functional requirements of an application.

### 3.4.3. Percentage of statement Coverage

This is also known as line coverage or segment coverage. It covers only true conditions. This is calculated based on the number of statements covered through the positive test inputs.

Therefore 100% of the statement coverage is achieved based on testcases for different input data of information.

### 

### 3.4.4. Percentage of equivalence Coverage

In this method the input domain data is divided into different equivalence classes. This is typically used to reduce the total number of test cases to a finite set of testable test cases, still covering maximum requirements.

### 3.4.5. Percentage of branch Coverage

Branch coverage is a requirement that for each branch in the program, each branch have been executed at least once during testing. This is calculated based on the number of possible paths covered through the testing.

The possible paths from the control flow graph are:-

ABCDEHKLMNPRSTUV ABCDEHKLMOQRSTUV ABCDFIKLMNPRSTUV

ABCDFIKLMOQRSTUV ABCDGJKLMNPRSTUV ABCDGJKLMOQRSTUV

Based on test cases the total number of branches covered =6

Total number of branches=6

Therefore 100% of the branch coverage is achieved based on testcases.

### 3.4.6. Percentage of Boundary value Analysis Coverage

Since most application errors occurs at the boundaries of input domain, boundary analysis technique is used to identify errors at boundaries rather than finding those exit in center of input domain.

It allows 5 but not 6

Boundary values are 5 and 6

Therefore 100% of boundary value coverage is analyzed on below test cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Max allowed tickets per transaction is 5 | Select 5seats for adults | Should show confirm button to proceed after selection | Display confirmation after selection of seats | Pass |
|  | Select 5seats for child | Should show confirm button to proceed after selection | Display confirmation after selection of seats | Pass |
|  | Select 5seats for old people | Should show confirm button to proceed after selection | Display confirmation after selection of seats | Pass |
|  |  |  |  |  |
| Cant reserve more than available seats | Select 6 seats | Should show ticket quantity error | Displays warning with no enough  seats remaining | Pass |

# Appendix A

Bugged source code

***public class Ticket {***

***private int SeatPrice;***

***private static int child = 8;***

***private static int adult = 10;***

***private static int oap = 8;***

***public Ticket(int SeatPrice) {***

***this.SeatPrice = SeatPrice;***

***}***

***public static int getSeatPrice(Integer c, String s) {***

***if (s.equals("adult")) {***

***return c \* 10;***

***}***

***if (s.equals("child")) {***

***return 0; (don’t calculate seat price for child)***

***}***

***if (s.equals("oap")) {***

***return c \* 8;***

***}***

***return 0;***

***}***

***public void setSeatPrice(int SeatPrice) {***

***this.SeatPrice = SeatPrice;***

***}***

***public static String calculateStringTotal(int i) {***

***String str = Integer.toString(i);***

***return str;***

***}***

***}***

# NewMain Code:

for (int i = 0; i < proceedArray.length; i++) {

for (int p = 0; p < proceedArray.length; p++) {

if (i != p) { // if the two compared objects have the same seat number...

if (proceedArray[i].equals(proceedArray[p])) { //...and is not a zero (this is a default value, not a seat number)

// then call an error stating duplicate seats have been allocated

if (proceedArray[i] != 0 || proceedArray[p] != 0) {

duplicateSeats();

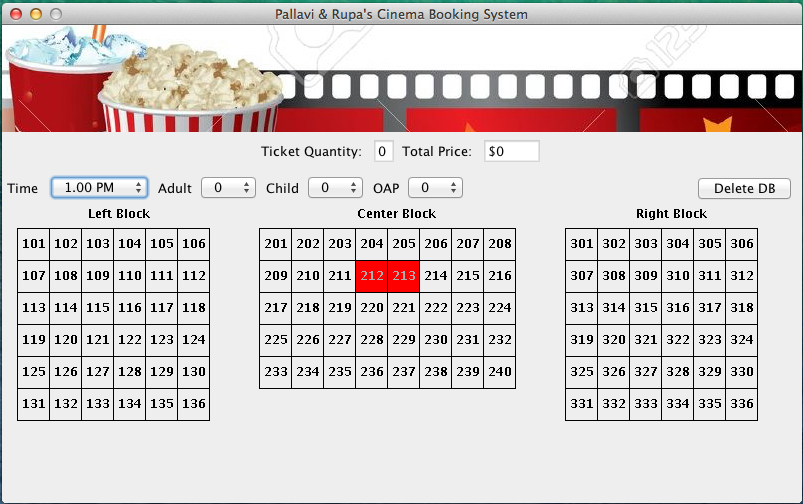
return 0;

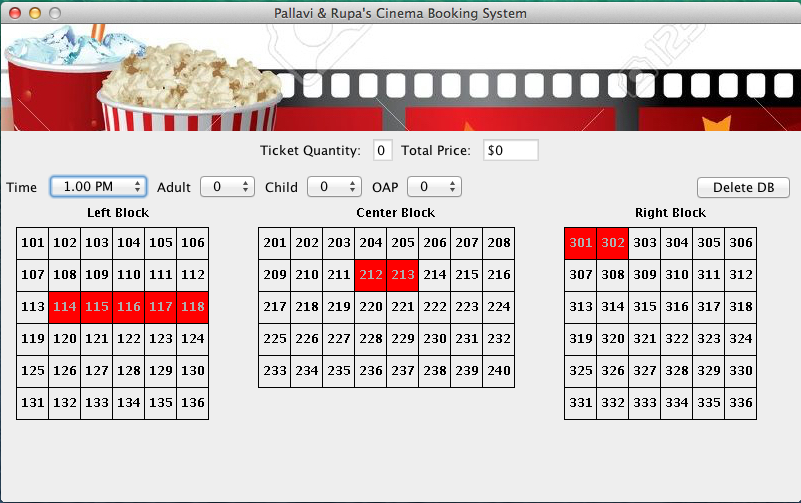
} } } }

# Appendix B

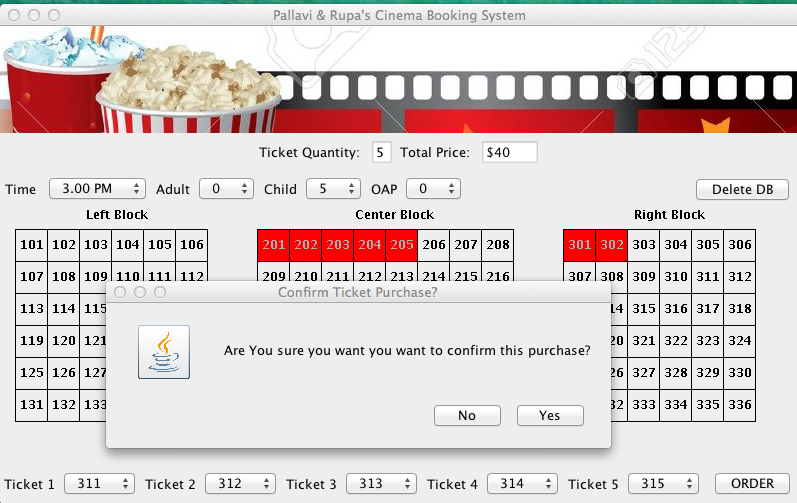
Test driver: Test cases with results

testcase1: Test the availability of seats across transaction

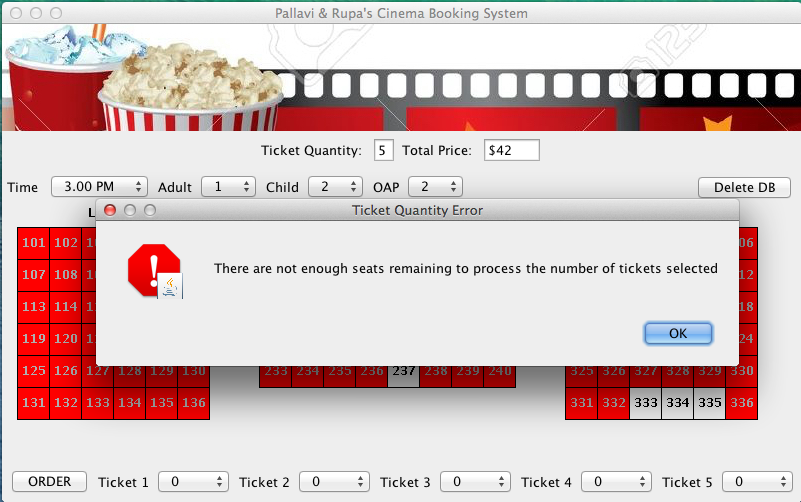
 1.1 select 2 seats

 2.1 select 2 seats at 1pm

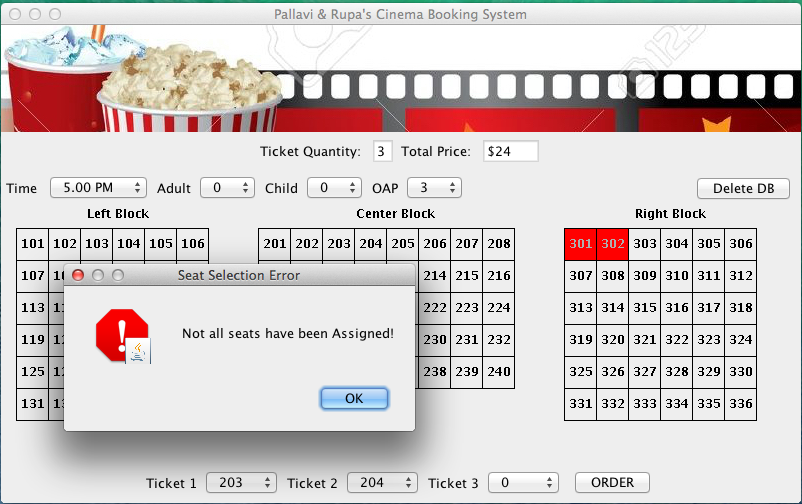
testcase3: Test the maximum allowed tickets per transaction is 5

 3.1 select 5 seats for child

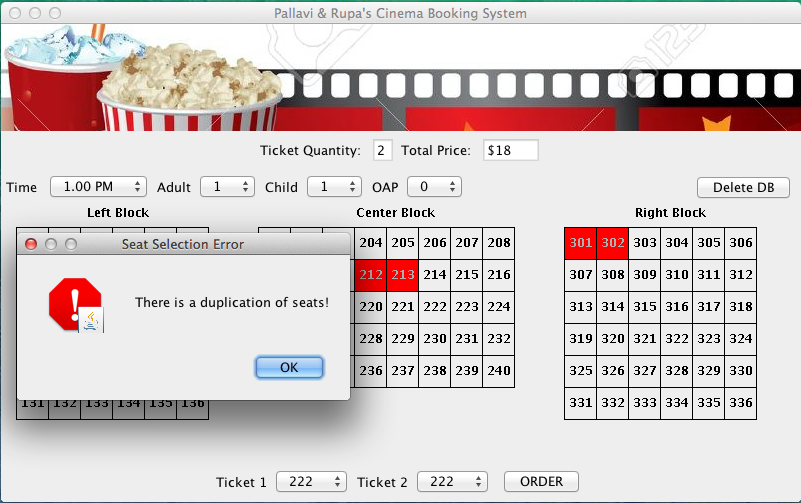
testcase4: Test that one cannot reserve more seats than that of available seats

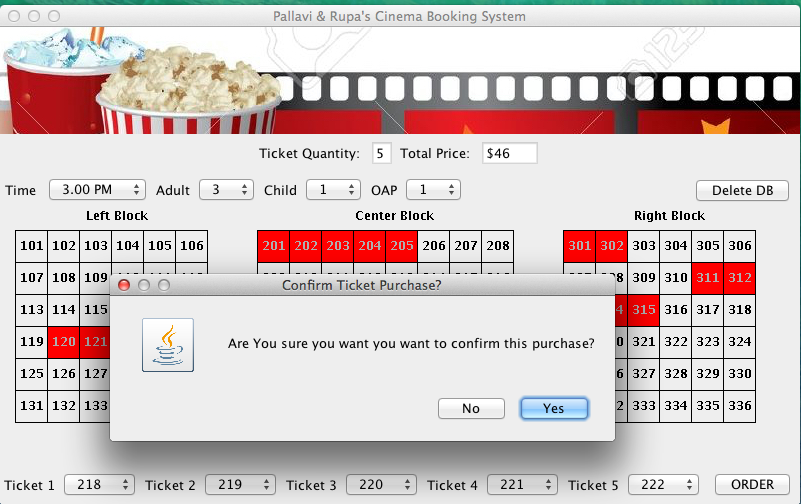
 4.1 select 5 seats

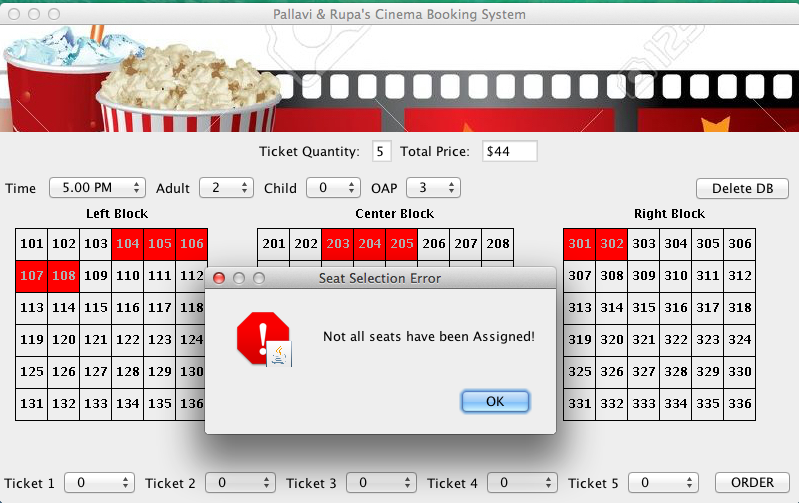
testcase5: Test that one has made the selection of seats for all the tickets

5.1. Two seats out of 3tickets

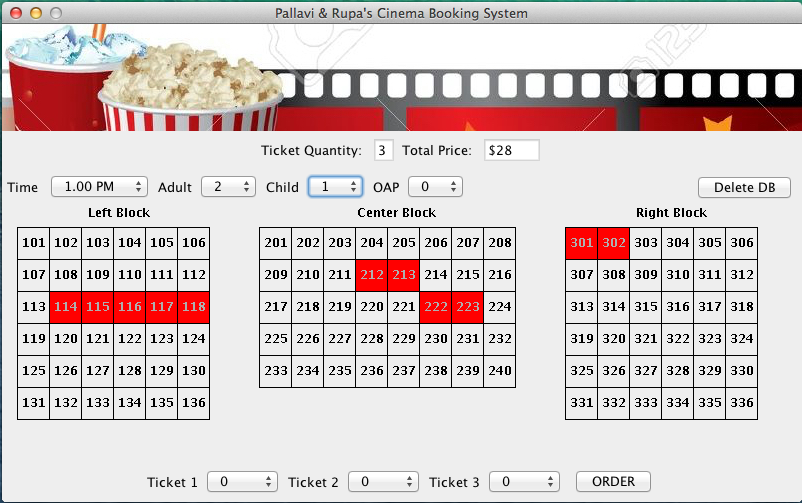
testcase6: Test that user can’t select same seat for multiple tickets

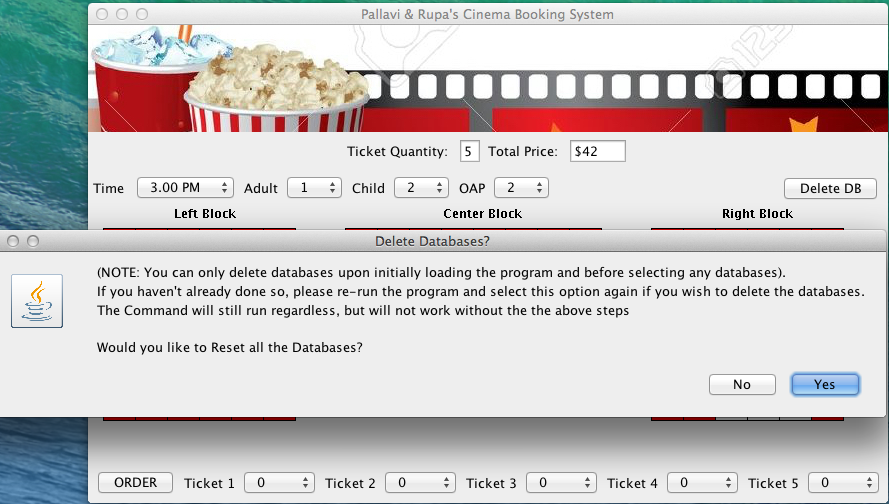
testcase7: Test that user is prompted before the confirmation

testcase8: Test that user shouldn't be able to order without selecting any ticket



testcase9: Test the validity of price

testcase10: Test the database reset

click yes/no???? if no ->no change ; if no then no change ;if yes then

# Appendix C

Enhanced source code

# Ticket class

***public class Ticket {***

***private int SeatPrice;***

***private static int child = 8;***

***private static int adult = 10;***

***private static int oap = 8;***

***public Ticket(int SeatPrice) {***

***this.SeatPrice = SeatPrice; }***

***public static int getSeatPrice(Integer c, String s) {***

***if (s.equals("adult")) {***

***return c \* 10; }***

***if (s.equals("child")) {***

***return c \* 8; }***

***if (s.equals("oap")) {***

***return c \* 8; }***

***return 0; }***

***public void setSeatPrice(int SeatPrice) {***

***this.SeatPrice = SeatPrice; }***

***public static String calculateStringTotal(int i) {***

***String str = Integer.toString(i);***

***return str;***

***}***