Contents

[Section 1 – Plan, strategy & Documentation 3](#_Toc289291262)

[1.1 - 2.3.3\_01: Plan 4](#_Toc289291263)

[1.2 - 2.3.3\_02: Strategy 5](#_Toc289291264)

[Stage One 5](#_Toc289291265)

[Review Specification 5](#_Toc289291266)

[Stage Two 6](#_Toc289291267)

[Identify Resources and Resource Requirements 6](#_Toc289291268)

[Available Resources 7](#_Toc289291269)

[1.3 - 2.3.3\_03: Design Review 8](#_Toc289291270)

[Overview 8](#_Toc289291271)

[Minutes 8](#_Toc289291272)

[1.4 - 2.3.3\_04: Requirements 9](#_Toc289291273)

[1.5 - 2.3.3\_05: Resource Allocation 10](#_Toc289291274)

[Development 10](#_Toc289291275)

[Test Case Engineering 11](#_Toc289291276)

[Testing 11](#_Toc289291277)

[1.6 - 2.3.3\_06: Requirements Annotated 13](#_Toc289291278)

[1.7 - 2.3.3\_07: Additional Designs 14](#_Toc289291279)

[1.8 – 2.3.3\_08: Quality Assurance 15](#_Toc289291280)

[Section 2 - Standards 16](#_Toc289291281)

[2.1 – 2.3.1.1\_01: Coding Standards 17](#_Toc289291282)

[Commenting Style 17](#_Toc289291283)

[Variable Naming Style 17](#_Toc289291284)

[Function and Method Naming Style 18](#_Toc289291285)

[Brace and Indentation Formatting 19](#_Toc289291286)

[Example: 20](#_Toc289291287)

[2.2 – 2.3.1.1\_02: Testing Standards 22](#_Toc289291288)

[Section3 - Testing 23](#_Toc289291289)

[3.1 – 2.3.4\_01 Test Cases 24](#_Toc289291290)

[3.2 – 2.3.4\_02: Basis Path Tests 25](#_Toc289291291)

[3.3 – 2.3.4\_03: Unit Tests 32](#_Toc289291292)

[Unit Test – Seats Date: 30.03.11 – JUnit – Whitebox 32](#_Toc289291293)

[Unit Test – Booking Date: 30.03.2011 – JUnit – Whitebox 35](#_Toc289291294)

[Unit Test – FilmShowing Date: 30.03.2011 – JUnit – Whitebox 37](#_Toc289291295)

[Unit Test – Customer Date: 30.03.2011 – JUnit – Whitebox 41](#_Toc289291296)

[Unit Test – Staff Date: 30.03.2011 – JUnit – Whitebox 42](#_Toc289291297)

[Unit Test – Film Date: 30.03.2011 – JUnit – Whitebox 46](#_Toc289291298)

[Unit Test – Main Date: 30.03.2011 – JUnit – Whitebox 48](#_Toc289291299)

[3.4 – 2.3.4\_04: White Box Testing 50](#_Toc289291300)

[4.1 – 2.3.2\_01: Requirements TraciNG 51](#_Toc289291301)

[User Manual 57](#_Toc289291302)

[5.1 – 2.3.5\_01: User Manual 58](#_Toc289291303)

[Administrator User Guide 58](#_Toc289291304)

[Standard User Guide 59](#_Toc289291305)

# Section 1 – Plan, strategy & Documentation

## 1.1 - 2.3.3\_01: Plan

* Build quality assurance document
* Establish final set of requirements
  + These are based on the partner groups requirements and feedback from Kelvin
* Discuss and document on methods to implement the system
  + Language
  + IDE
* Establish who will do what
  + Developer
  + Analysts
  + Test Case Engineers
  + Testers
* Establish configuration management
  + Keep track of version control and changes
* Begin development of system
  + Unit tests
  + Quality Audits
  + Implementation reviews
  + Testing
    - White Box:
      * Basis path testing
    - Black Box:
      * Stress testing
* Build/Engineer test cases
  + Build tests from requirements
  + Visual/Flow charts
    - Testing logical flow through the system
  + Black box
  + White box
* Finish development
  + Alpha/Beta tests
  + Acceptance testing

## 1.2 - 2.3.3\_02: Strategy

### Stage One

### Review Specification

* Ensure entire team has a clear understanding of what is required.
* Establish a clear idea of how development is to proceed.
* Produce a final, clear document of all system requirements.
* Break down into assignable tasks.
  + To be delegated to correct team members.
* Ensure team members with tasks understand what is required of them.

### Stage Two

### Identify Resources and Resource Requirements

|  |  |
| --- | --- |
| Developers | Begin development:   * Unit test all methods as they are written. * Ensure work is kept to a standard defined in Quality Assurance. * Quality audits to be carried out periodically. |
| Test Engineers | Engineer test cases:   * Basis Path testing. * Test cases to match requirements. * Usability tests. * Acceptance tests. |
| Testers | Carry out tests developed by test engineers:   * Develop reports on test results. * Compare to expected results. |
| Analysts | Analyse test reports:   * Establish areas of improvement, * Ensure system is sticking to requirements. |
| Project Manager | Team management:   * Carry out quality audits. * Ensure team is adhering to quality strategy. |

### Available Resources

|  |  |
| --- | --- |
| Resource | Roles |
| David Russell | * Project Manager * Developer * Tester |
| Amanda Patterson | * Tester * Analyst |
| James Nightingale | * Developer * Tester |
| Scott Dennison | * Developer * Tester |

## 1.3 - 2.3.3\_03: Design Review

### Overview

Planned to meet at 2pm, Tuesday 8th of April in Octagon concourse. Aim is to exchange documents and establish which requirements in each specification are unneeded and if there are any immediate changes either group will make to the specification.

### Minutes

14.00 – Everyone arrived but Scott – Text message says he’s running late and that he is on his way

14.05 - Decided to go ahead with exchange whilst waiting for Scott. Exchanged our documents with BHH. BHH does not have access to their assessed requirement specification. Began to go through our requirements with BHH, eliminating unneeded requirements.

14.30 – Scott arrives. Finished requirement exchange with partner group. As BHH do not have their assessed document, they go to find Kelvin to get access to their document.

14.40 – Waiting on Kelvin to finish tutorial for BHH document feedback. Discussed possible implementation with BHH.

14.50 – Amanda has to leave for an exam. Members of BHH group go to find Kelvin again.

14.55 – Richard (BHH) has to leave for lecture.

15.00 - BHH unable to locate Kelvin, agreed to email their assessed requirements ASAP.

19.07 – BHH email their scanned, assessed requirements. Evidence located: 2.5.5\_02CommLog02\_DesignReview

## 1.4 - 2.3.3\_04: Requirements

3.1.1 Log operator onto the system.

* All operators must login to the system with a username and password to access the system.

3.1.2 Record the operators’ bookings

* Record a log of the bookings by each operator.

3.1.3 Book Seats up to 6 hours before film

* Book up to 10 seats in one booking (1 row)
* Peak Times are fixed on Saturdays and Sundays between 12-4 and 6-11

3.1.4 Cancel Bookings

* Customer’s bookings can be cancelled via request of the operator, they require details of customer and film times and payment details for refund

3.1.5 View which seats are available

* Operators can view which seats are available.
* It is possible for customers to request which seats are available for booking.

3.1.6 Different prices for premium seats

* Rows H and I are premium seats

3.1.7 Able to set prices of premium seats as a percentage increase of the costs for standard seating

3.1.8 Store film information

* File name and age rating for each film as text information

3.1.9 Check customer is correct age for viewing the film

* System function, enter date of birth of customer when booking film, check against age rating of film and if age of customer is equal to or greater than the age rating of film then booking can be completed

3.2.1 Personal information not to be kept for more than one year on database

* No credit card information is to be kept of customer

3.5.1 Software colour scheme to be red, white and green

3.5.2 Minimise number of steps to complete booking to keep it as simple as possible

## 1.5 - 2.3.3\_05: Resource Allocation

### Development

|  |  |
| --- | --- |
| Language: | * Java |
| Resources: | * Scott Dennison. * James Nightingale. * David Russell. |
| Strategy: | * Initially develop functionality with Command Line. * Implement to web interface once fully functional. |
| Version Control and Repository: | * github.com/staffs-ppsp |
| Allocation: | * Scott: Booking and Film classes. * James: Staff and Customer classes (adding Person interface). * Dave: FilmShowing and Seats classes. * Scott: Web interface * James & Dave: Java back end functionality |
| Quality Assurance: | * Unit testing on methods as code is written. * Adhering to quality assurance document. * Testing carried out by Amanda. |

### Test Case Engineering

|  |  |
| --- | --- |
| Resources: | * Amanda Patterson. |
| Strategy: | * Writing test cases to ensure program meets the specification. * Writing test cases to ensure program is fit for purpose. * Audit tests to ensure coding meets agreed standards. |

### Testing

|  |  |
| --- | --- |
| Resources: | * Amanda Patterson. * David Russell. * James Nightingale. * Scott Dennison. |
| Strategy: | * Adhere to written test cases and testing standards to ensure usability. * Allocate specific testing to team members. |

#### White Box

|  |  |
| --- | --- |
| Resources: | * David Russell. * James Nightingale. |
| Strategy: | * Construct basis path test diagrams from code. * Follow logical paths from code and ensure flow is correct. * Construct unit tests. * Ensure objects are constructed and referenced correctly. |

#### Black Box

|  |  |
| --- | --- |
| Resources: | * Scott Dennison * Amanda Patterson |
| Strategy: | * Prepare test data. * Subject system to test data. * Record and analyse results. * Determine results are acceptable or within established boundaries. |

#### Acceptance Testing

|  |  |
| --- | --- |
| Resources: | * Amanda Patterson. * David Russell. * James Nightingale. * Scott Dennison. |
| Strategy: | * Prepare test data which ensures the implementation meets requirements and functions as expected. |

## 1.6 - 2.3.3\_06: Requirements Annotated

3.1.3 Book Seats up to 6 hours before film

* Book up to 10 seats in one booking (1 row)  
  The interface that is shown does not allow this
* Peak Times are fixed on Saturdays and Sundays between 12-4 and 6-11  
  ? Don't understand?

3.1.4 Cancel Bookings

* Customer’s bookings can be cancelled via request of the operator, they require details of customer and film times and payment details for refund  
  There is no interface designed for this

3.1.8 Store film information

* File name and age rating for each film as text information  
  there is no mention of being able to ADD films! Nor an interface

3.5.1 Software colour scheme to be red, white and green  
Please can we discuss this with both the other team and the customer.

3.5.2 Minimise number of steps to complete booking to keep it as simple as possible.

## 1.7 - 2.3.3\_07: Additional Designs

# NewDesigns.png

## 1.8 – 2.3.3\_08: Quality Assurance

The following points must be adhered to throughout the project.

Code

* Variables named correctly.
* Place holders for code.
* Layout is industry standard/indentation.
* Comments to briefly describe functionality.
* Easy to understand flow of code.
* Separate into blocks/ classes.
* Correctly aligned vertically.
* Correct versioning.
* Variables declared properly.
* Easy to maintain, test and debug.
* Easy to fix, modify.
* Easy to read and understand by others - commented/ documented.

A comprehensive list of coding standards can be found in index 2.3.1\_QualityDocuments.

The produced software must also meet the following standards.

Product

* Use of requirement’s or program specification.
* No bugs.
* Completed product.
* Documentation.

# Section 2 - Standards

## 2.1 – 2.3.1.1\_01: Coding Standards

### Commenting Style

As the coding is to be a collaborative effort, developers must adhere to the commenting style standards defined here. This is so other developers who may need to read, update or remove code will be able to understand with ease the functionality of methods, algorithms or data structures in the code. Comments must not impact the formatting conventions, for example, comments must not make altering blocks of code unnecessarily task heavy.

#### Conventions:

* Javadoc must include Author, version and java version.
* Javadoc all functions and methods.
* Variable declarations are to be grouped with a comment above.
* Comments at the side of variables to describe their use. However this must not conflict with formatting conventions described above.
* Comments must be descriptive without being too large and avoiding ‘walls of text’.
* Comments covering few lines (1-2 lines) may use the standard // feature.
* Larger comments which cover more than a few lines must be grouped with the /\* \*/ feature for code clarity and neatness.

### Variable Naming Style

Variables must be named in accordance to the standards defined below. Again, the collaborative nature of the project, there must be a standard to variable declaration used by all developers for the ease of altering and reading others’ code. Variables must avoid being ambiguous for ease of referencing, passing and casting.

#### Conventions:

* Declaring variables ‘on the fly’ must be avoided, with the exception of for loops, and declared at the top of classes and functions.
* All variables declared must use title case, as seen in the examples below
* Underscores must be avoided in variable declaration.
* Underscores must be used to denote parameter variables.
* The exception to these standards is for loops, which may use single variable names, for example; i, j, k etc.
* Arrays and data structures must be declared as such: arrValues. Followed by a comment which describes what data types are in the array.
* Variables must follow the 3 (Boolean types break this exception) Hungarian notation.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Correct Notation** | **Parameter** |
| Integer | intIntegerVar | \_intIntegerVar |
| Double | dblDoubleVar | \_dblDoubleVar |
| Boolean | boolBooleanVar | \_boolBooleanVar |
| String | strStringVar | \_strStringVar |
| Char | chrCharVar | \_chrCharVar |
| Long | lngLongVar | \_lngLongVar |
| Float | fltFloatVar | \_fltFloatVar |
| Short | shtShortVar | \_shtShortVar |
| Byte | bytByteVar | \_bytByteVar |
| Object | objObjectVar | \_objObjectVar |

### Function and Method Naming Style

Again, correctly named functions and methods are essential for code clarity and ease of modification for other developers. The names must give a hint as to what the method or function does, this is for ease of calls to the method or function in code.

#### Conventions:

* As with variables, names must use title case with the first word in lowercase regardless of the circumstance.
* Underscores must be avoided, unless in the case of parameter variables.

### Brace and Indentation Formatting

For code clarity and neatness, all developers must adhere to the following conventions. Code clarity is essential when working on a collaborative project and poor formatting is inexcusable, as all IDE’s have customisable formatting options.

#### Indentation Conventions:

* Declarations within class bodies must be indented.
* Statements within methods/constructors, blocks, switch and case statements must be indented.
* Vertical lines must be adhered to in the case of line breaks and comments occurring after variables.

#### Brace Conventions:

* Braces must follow the declaration on the same line.
* Catch/finally blocks must begin on a new line after the brace.
* Else if blocks must also being on a new line after the brace.

### Example:

*/\*\**

*\* @author Jeff Jones*

*\* @version 1.01*

*\* @since 1.6*

*\*/*

class MyClass {

*// Variable Declaration*

int intHouseNumber; *// Number of the house*

double dblGrams; *// Weight in grams*

*/\**

*Constructor for MyClass*

*I will make*

*this comment span*

*several lines.*

*\*/*

public MyClass() {

super.MySuperClass;

}

*// switch statement for something*

switch (intHouseNumber) {

case 1:

return true;

break;

case 2:

return false;

break;

default:

}

*/\*\**

*\* @param \_intHouseNum passed house number*

*\*/*

public void setHouseNum(int \_intHouseNum) {

intHouseNumber = \_intHouseNum;

}

}

## 2.2 – 2.3.1.1\_02: Testing Standards

# Section3 - Testing

## 3.1 – 2.3.4\_01 Test Cases

|  |  |  |
| --- | --- | --- |
| **Case** | **Testing Requirements** | **Test Case Completed** |
| 01.001 | To access the system the user must enter their username and password |  |
| 01.002 | To access the admin privileges the user must enter the admin username and password |  |
| 02.001 | Bookings recorded |  |
| 02.002 | Log kept of bookings made by the operator |  |
| 03.001 | Bookings not allowed less than 6 hours before film |  |
| 03.002 | A user must be able to select up to 10 seats to be booked at a time |  |
| 03.003 | Peak times are to be set between specified hours |  |
| 04.001 | User must be able to cancel film bookings at request of customer |  |
| 04.002 | Details required for cancellation: Customer, film times, payment details |  |
| 05.001 | User is able to view the seats currently available for booking in a showing |  |
| 06.001 | User able to see which seats are premium seats, Rows H and I |  |
| 06.002 | Price difference in place for booking premium seats |  |
| 07.001 | User able to add: new film, with details: Film name and age rating |  |
| 07.002 | User able to edit film information currently available |  |
| 08.001 | System function, user must enter age of customer booking film, this is checked against age rating of film, if age of customer is equal to or greater than the film rating then booking can be completed. |  |

## 3.2 – 2.3.4\_02: Basis Path Tests

|  |  |  |
| --- | --- | --- |
| Test Case # | Flow Graph | Code |
| 01.001 |  | if (loginAdminFunction() == 1) {  //CASE BODY  }  else {  break;  }  public static int loginUserFunction() {  System.out.println("User Name:");  strInputUserName = kybd.next();  kybd.reset();  System.out.println("Password:");  strInputPassword = kybd.next();    if (strInputUserName.equalsIgnoreCase(strUsername) && strInputPassword.equalsIgnoreCase(strPassword)) {    System.out.println("Success");  intAttemptCtr = 0; // Reset the attempt counter  return 1;  } else {  intAttemptCtr++;  System.out.println("\nUnsuccessful Attempt #" + intAttemptCtr);  if (intAttemptCtr == 3) {  System.out.println("3 Consecutive Unsuccessful Attempts.. Exiting Program");  return escape = 2;  }  return 3;  }  } |

|  |  |  |
| --- | --- | --- |
| 01.002 |  | if (loginAdminFunction() == 1) {  //CASE BODY  }  else {  break;  }  public static int loginAdminFunction() {  System.out.println("User Name:");  strInputUserName= kybd.next();  kybd.reset();  System.out.println("Password:");  strInputPassword = kybd.next();  if (strInputUserName.equalsIgnoreCase(strAdminUsername)  && strInputPassword.equalsIgnoreCase(strAdminPassword)){  System.out.println("Success");  intAttemptCtr = 0; // Reset the attempt counter  return 1;  } else {  intAttemptCtr++;  System.out.println("\nUnsuccessful Attempt #" + intAttemptCtr);  if (intAttemptCtr == 3) {  System.out.println("3 Consecutive Unsuccessful Attempts.. Exiting Program");  return escape = 2;  }  }  return 3;  } |

|  |  |  |
| --- | --- | --- |
| 08.001 |  | // Call to method  public static boolean advancedBooking() throws Exception {  // Add a customer  // Make a booking  // Add a showing to the booking  if (customer.getIntAge() > newBooking.getFilmShowing().getIntRating()) {  System.out.println("Customer Under Age");  return false;  } else {  // Add booking and choose seats return true;  } |

|  |  |  |
| --- | --- | --- |
| 03.002 |  | // CREATE CUSTOMER OBJECT  customers.addCustomer(customer);  Booking newBooking = new Booking(customer, stf1);  System.out.println("Available Showings");  showings.showShowings();  System.out.println("ID Of Showing");  kybd.reset();  intCase = kybd.nextInt();  newBooking.setIntShowingID(showings.getByID(intCase));  newBooking.setIntFilmRating(showings.getByRating(intCase));  newBooking.setDblTotalPrice(showings.getByPrice(intCase));  if (customer.getIntAge() < newBooking.getIntFilmRating()) {  System.out.println("Customer Under Age");  return false;  } else {  Seats seatBooking = new Seats(newBooking);  System.out.println("Number of Seats: ");  intCase = kybd.nextInt();  newBooking.setDblTotalPrice(newBooking.getDblTotalPrice() \* intCase);  if (intCase > 10 || intCase <= 0) {  System.out.println("Exceeds maximum number of seats per booking");  return false;  }  if (intCase <= 0) {  System.out.println("Number of seats must be greater than 0");  return false;  }  seatBooking.setIntNumSeats(intCase);  System.out.println("Row: (A - I)");  strCase = lineReader.readLine();  switch (strCase.toUpperCase().charAt(0)) {  case 'A':  seatBooking.setObjRow(Row.A);  break;  // ETC  case 'I':  seatBooking.setObjRow(Row.I);  newBooking.setDblTotalPrice(newBooking.getDblTotalPrice() \* 1.5);  break;  default:  System.out.println("Invalid Seat");  return false;  }  System.out.println("Column: (0 - 15)");  intCase = kybd.nextInt();  switch (intCase) {  case 0:  seatBooking.setObjColumn(Column.J);  break;  // ETC  default:  System.out.println("Invalid Seat");  return false;  } bookings.addBooking(newBooking);  screen1.addSeat(seatBooking);  System.out.println(newBooking);  System.out.println(screen1);  return true;  }  } |

|  |  |
| --- | --- |
|  | public static boolean addFilmFunction() {  Film newFilm = new Film();  System.out.println("Film Title:");  try {  newFilm.setStrFilmName(lineReader.readLine());  } catch (IOException e) {  System.out.println("Invalid Entry");  }  System.out.println("Film Rating:");  newFilm.setIntRating(kybd.nextInt());  films.addFilm(newFilm);  FilmShowing showing = new FilmShowing();  showing.addFilm(newFilm);  return true;  } |
|  | public void addSeat(Seats \_objSeats) throws Exception {  try {  for (int i = 0; i < \_objSeats.intNumSeats; i++) {  if (arrSeats[\_objSeats.objRow.getRowNum()][\_objSeats.objColumn.getColNum() + i] == 0) {arrSeats[\_objSeats.objRow.getRowNum()][\_objSeats.objColumn  .getColNum() + i] = \_objSeats.objBooking  .getIntBookingID();  bookings.add("Booking ID: "  + \_objSeats.objBooking.getIntBookingID() + "\n"  + "Row: " + \_objSeats.objRow.getRowNum() + 1 + "\n"  + "Column: " + \_objSeats.objColumn);  } else  throw new Exception();  }  // } else {  // System.out.println("Unable to book at this time");  // }  } catch (Exception e) {  System.out.println("Seat Already Booked: " + \_objSeats.objRow  + \_objSeats.objColumn);  } |

# 3.3 – 2.3.4\_03: Unit Tests

### Unit Test – Seats Date: 30.03.11 – JUnit – Whitebox

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Data** | **Expected** | **Actual** | **Cause** | **Action Taken** |
| 0 | static Seats screen1 = new Seats("Screen 1");  static Staff stf1 = new Staff("James", "Nightingale", 20, "07123283823", 456, "Manager", "jn1", "lalalala");  static Film flm1 = new Film("Paul", 15);  static FilmShowing fs1 = new FilmShowing(flm1, "27/03/2011", screen1, 6.99);  static Customer cst1 = new Customer("Dave", "Russell", 21, "07533475113");  static Booking booking = new Booking(cst1, stf1, fs1); Seats seats = new Seats(booking); |  |  |  |  |
| 1 | assertNotNull("Booking ID is null", seats.getIntBookingID()); | Pass | Pass | N/A | None taken |
| 2 | assertNotNull("Showing ID is null", seats.getIntShowingID()); | Pass | Pass | N/A | None taken |
| 3 | assertNotNull("Customer ID is null", seats.getIntCustomerID()); | Pass | Pass | N/A | None taken |
| 4 | assertNotNull("Screen is null", screen1.getStrScreen()); | Pass | Pass | N/A | None taken |
| 5 | assertNotNull("Price is null", seats.getDoubleTotalPrice()); | Pass | Pass | N/A | None taken |
| 6 | assertSame("Screens do not match", screen1.getStrScreen(), fs1.getStrScreen()); | Pass | Pass | N/A | None taken |
| 7 | int test;  assertSame("Return error", seats.setIntNumSeats(2), test = seats.setIntNumSeats(2));  assertSame("Number of seats do not match", seats.setIntNumSeats(2),seats.getIntNumSeats()); | Pass | Pass | N/A | None taken |
| 8 | assertNotNull("Number of seats is null", seats.getIntNumSeats()); | Pass | Pass | N/A | None taken |
| 9 | assertTrue("Row is unrecognised", seats.setObjRow(Row.A)); | Pass | Pass | N/A | None taken |
| 10 | seats.setObjRow(Row.A);  assertNotNull(seats.getObjRow()); | Pass | Pass | N/A | None taken |
| 11 | assertTrue("Column is unrecognised", seats.setObjColumn(Column.J)); | Pass | Pass | N/A | None taken |
| 12 | seats.setObjColumn(Column.L);  assertNotNull("Column has not been set", seats.getObjColumn()); | Pass | Pass | N/A | None taken |
| 13 | try {  assertTrue("Adding has failed - Returned false", screen1.addSeat(seats));  } catch (Exception e) {  fail("Adding has failed - Exception caught");  } | Pass | Pass | N/A | None taken |
| 14 | screen1.addSeat(newSeats);  assertNotNull(screen1.toString()); | Pass | Pass | N/A | None taken |
| 15 | assertNotNull("Get date/time failed",seats.getCurrentDateTime()); |  |  |  |  |

### Unit Test – Booking Date: 30.03.2011 – JUnit – Whitebox

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Data** | **Expected** | **Actual** | **Cause** | **Action Taken** |
| 0 | Seats screen1 = new Seats("Screen 1");  Booking booking = new Booking("Bookings");  Staff stf1 = new Staff("James", "Nightingale", 20, "07123283823", 456, "Manager", "jn1", "lalalala");  Film flm1 = new Film("Paul", 15);  FilmShowing fs1 = new FilmShowing(flm1, "27/03/2011", screen1, 6.99);  Customer cst1 = new Customer("Dave", "Russell", 21, "07533475113");  Booking staffBooking = new Booking(stf1);  Booking sncBooking = new Booking(cst1, stf1);  Booking fullBooking = new Booking(cst1, stf1, fs1)  Booking testBooking; |  |  |  |  |
| 1 | assertNotNull(testBooking = new Booking()); | Pass | Pass | N/A | None taken |
| 2 | assertNotNull(staffBooking.toString()); | Pass | Pass | N/A | None taken |
| 3 | assertNotNull(testBooking = new Booking(stf1)); | Pass | Pass | N/A | None taken |
| 4 | assertNotNull(testBooking = new Booking(cst1,stf1)); | Pass | Pass | N/A | None taken |
| 5 | assertNotNull(testBooking = new Booking(cst1, stf1, fs1)); | Pass | Pass | N/A | None taken |
| 6 | assertNotNull(staffBooking.getIntBookingID());  assertNotSame(staffBooking.getIntBookingID(), sncBooking.getIntBookingID()); | Pass | Pass | N/A | None taken |
| 7 | assertNotNull(fullBooking.getDblTotalPrice()); | Pass | Pass | N/A | None taken |
| 8 | assertNotNull(staffBooking.getIntStaffID()); | Pass | Pass | N/A | None taken |
| 9 | assertNotNull(fullBooking.getIntShowingID()); | Pass | Pass | N/A | None taken |
| 10 | assertNotNull(fullBooking.getIntCustomerID()); | Pass | Pass | N/A | None taken |
| 11 | assertNotNull(fullBooking.getIntFilmRating()); | Pass | Pass | N/A | None taken |
| 12 | assertNotNull(fullBooking.getShowingTime()); | Pass | Pass | N/A | None taken |
| 13 | booking.addBooking(fullBooking);  assertNotNull(booking); | Pass | Pass | N/A | None taken |
| 14 | assertNotNull(fullBooking.generateBookingID()); | Pass | Pass | N/A | None taken |
| 15 | assertNotNull(fullBooking.getDateTime()); | Pass | Pass | N/A | None taken |
| 16 | assertNotNull(fullBooking.toString()); | Pass | Pass | N/A | None taken |

### Unit Test – FilmShowing Date: 30.03.2011 – JUnit – Whitebox

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Test Data** | | | | **Expected** | **Actual** | **Cause** | **Action Taken** |
| 0 | Seats screen1 = new Seats("Screen 1");  FilmShowing showings = new FilmShowing("Showings");  Film flm1 = new Film("Paul", 15);  FilmShowing testFS;  FilmShowing fs1 = new FilmShowing(flm1, "27/03/2011", screen1, 6.99); | | | |  |  |  |  |
| 1 | showings.addFilmShowing(fs1);  System.out.println(fs1); | | | | Pass | Pass | N/A | None taken |
| 2 | assertNotNull(testFS = new FilmShowing(flm1)); | | | | Pass | Pass | N/A | None taken |
| 3 | assertNotNull(testFS = new FilmShowing("Showings")); | | | | Pass | Pass | N/A | None taken |
| 4 | assertNotNull(testFS = new FilmShowing(flm1, "21/03/2009", screen1, 4.99)); | | | | Pass | Pass | N/A | None taken |
| 5 | showings.addFilmShowing(fs1);  showings.showShowings(); | | | | Pass | Pass | N/A | None taken |
| 6 | assertNotNull(fs1.getIntShowingID()); | | | | Pass | Pass | N/A | None taken |
| 7 | fs1.setStrDate("21/03/2009"); | Pass | Fail | java.lang.IllegalArgumentException:  Cannot format given Object as a Date | Tasked developer to fix method. | | | |
| 8 | assertNotNull(fs1.getStrDate()); | | | | Pass | Pass | N/A | None taken |
| 9 | assertNotNull(fs1.getStrScreen()); | | | | Pass | Pass | N/A | None taken |
| 10 | fs1.setStrScreen("Screen 2");  System.out.println(fs1.toString());  assertNotNull(fs1.getStrScreen()); | | | | Pass | Pass | N/A | None taken |
| 11 | assertNotNull(fs1.getDblPrice()); | | | | Pass | Pass | N/A | None taken |
| 12 | fs1.setIntPrice(7.99);  System.out.println(fs1); | | | | Pass | Pass | N/A | None taken |
| 13 | showings.addFilmShowing(fs1); | | | | Pass | Pass | N/A | None taken |
| 14 | showings.showShowings(); | | | | Pass | Pass | N/A | None taken |
| 15 | showings.addFilmShowing(fs1);  assertNotNull(showings.getByID(2));  System.out.println(showings.getByID(2)); | | | | Pass | Pass | N/A | None taken |
| 16 | showings.addFilmShowing(fs1);  assertNotNull(showings.getByRating(2));  System.out.println(showings.getByRating(2)); | | | | Pass | Pass | N/A | None taken |
| 17 | showings.addFilmShowing(fs1);  assertNotNull(showings.getByPrice(2));  System.out.println(showings.getByPrice(2)); | | | | Pass | Pass | N/A | None taken |
| 18 | testFS = new FilmShowing();  assertNotNull(testFS.generateFSID()); | | | | Pass | Pass | N/A | None taken |
| 19 | assertNotNull(fs1.getDateTime()); | | | | Pass | Pass | N/A | None taken |

### Unit Test – Customer Date: 30.03.2011 – JUnit – Whitebox

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Data** | **Expected** | **Actual** | **Cause** | **Action Taken** |
| 0 | Customer customers = new Customer("Customers");  Customer testArray;  Person testCust;  Customer cust1 = new Customer("Dave", "Russell", 21, "01232 123123");  Customer cust2 = new Customer(); |  |  |  |  |
| 1 | assertNotNull(testCust = new Customer()); | Pass | Pass | N/A | None taken |
| 2 | assertNotNull(testCust = new Customer("James", "Nightingale", 20, "01232 123123")); | Pass | Pass | N/A | None taken |
| 3 | assertNotNull(testArray = new Customer("Customers")); | Pass | Pass | N/A | None taken |
| 4 | assertNotNull(cust1.getCustomerID()); | Pass | Pass | N/A | None taken |
| 5 | customers.addCustomer(cust1); | Pass | Pass | N/A | None taken |
| 6 | assertNotNull(cust2.generateCustomerID()); | Pass | Pass | N/A | None taken |
| 7 | System.out.println(cust1); | Pass | Pass | N/A | None taken |

### Unit Test – Staff Date: 30.03.2011 – JUnit – Whitebox

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Data** | **Expected** | **Actual** | **Cause** | **Action Taken** |
| 0 | Staff staff = new Staff("Staff");  Staff testStaff;  Staff stf1 = new Staff("Amanda", "Patterson", 21, "01010 929292", "Head Honcho", "ap1", "ap1"); |  |  |  |  |
| 1 | assertNotNull(stf1.getStrFirstName()); | Pass | Pass | N/A | None taken |
| 2 | stf1.setStrFirstName("Manda");  assertNotNull(stf1.getStrFirstName()); | Pass | Pass | N/A | None taken |
| 3 | assertNotNull(stf1.getStrLastName()); | Pass | Pass | N/A | None taken |
| 4 | stf1.setStrLastName("patterson");  assertNotNull(stf1.getStrLastName()); | Pass | Pass | N/A | None taken |
| 5 | assertNotNull(stf1.getIntAge()); | Pass | Pass | N/A | None taken |
| 6 | stf1.setIntAge(22);  assertNotNull(stf1.getIntAge()); | Pass | Pass | N/A | None taken |
| 7 | assertNotNull(stf1.getStrTelephone()); | Pass | Pass | N/A | None taken |
| 8 | stf1.setStrTelephone("01010 124123");  assertNotNull(stf1.getStrUsername()); | Pass | Pass | N/A | None taken |
| 9 | testStaff = new Staff();  assertNotNull(testStaff); | Pass | Pass | N/A | None taken |
| 10 | testStaff = new Staff("Staff");  assertNotNull(testStaff); | Pass | Pass | N/A | None taken |
| 11 | testStaff = new Staff("Scott", "Dennison", 20, "01010 231231", "Crew Member", "sd1", "sd1"); | Pass | Pass | N/A | None taken |
| 12 | staff.addStaff(stf1); | Pass | Pass | N/A | None taken |
| 13 | staff.viewStaff(); | Pass | Pass | N/A | None taken |
| 14 | assertNotNull(stf1.getStaffID()); | Pass | Pass | N/A | None taken |
| 15 | assertNotNull(stf1.generateStaffID()); | Pass | Pass | N/A | None taken |
| 16 | assertNotNull(stf1.getStrRole()); | Pass | Pass | N/A | None taken |
| 17 | stf1.setStrRole("THE BOSS");  assertNotNull(stf1.getStrRole());  System.out.println(stf1.getStrRole()); | Pass | Pass | N/A | None taken |
| 18 | assertNotNull(stf1.getStrUsername()); | Pass | Pass | N/A | None taken |
| 19 | stf1.setStrUsername("ap\_m1");  assertNotNull(stf1.getStrUsername());  System.out.println(stf1.getStrUsername()); | Pass | Pass | N/A | None taken |
| 20 | assertNotNul(stf1.getStrPassword()); | Pass | Pass | N/A | None taken |
| 21 | stf1.setStrPassword("password");  assertNotNull(stf1.getStrPassword());  System.out.println(stf1.getStrPassword()); | Pass | Pass | N/A | None taken |
| 22 | System.out.println(stf1); | Pass | Pass | N/A | None taken |

### Unit Test – Film Date: 30.03.2011 – JUnit – Whitebox

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Data** | **Expected** | **Actual** | **Cause** | **Action Taken** |
| 0 | Film films = new Film("Films");  Film testFilm;  Film newFilm2 = new Film("Harry Potter and the Mysterious Case of Missing Work", 15); |  |  |  |  |
| 1 | assertNotNull(testFilm = new Film("Films")); | Pass | Pass | N/A | None taken |
| 2 | assertNotNull(testFilm = new Film()); | Pass | Pass | N/A | None taken |
| 3 | assertNotNull(testFilm = new Film("The Exorcist", 18)); | Pass | Pass | N/A | None taken |
| 4 | films.addFilm(newFilm2);  assertNotNull(films); | Pass | Pass | N/A | None taken |
| 5 | assertNotNull(newFilm2.getIntFilmID()); | Pass | Pass | N/A | None taken |
| 6 | newFilm2.setStrFilmName("The Exorcist 2");  assertNotNull(newFilm2.getStrFilmName());  System.out.println(newFilm2.getStrFilmName()); | Pass | Pass | N/A | None taken |
| 7 | assertNotNull(newFilm2.getStrFilmName()); | Pass | Pass | N/A | None taken |
| 8 | newFilm2.setIntRating(15);  assertNotNull(newFilm2.getIntRating());  System.out.println(newFilm2.getIntRating()); | Pass | Pass | N/A | None taken |
| 9 | assertNotNull(newFilm2.getIntRating()); | Pass | Pass | N/A | None taken |
| 10 | System.out.println(newFilm2); | Pass | Pass | N/A | None taken |
| 11 | assertNotNull(newFilm2.generateFilmID()); | Pass | Pass | N/A | None taken |

### Unit Test – Main Date: 30.03.2011 – JUnit – Whitebox

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Data** | **Expected** | **Actual** | **Cause** | **Action Taken** |
| 0 | 1 – admin, admin  2 – y, 1.8  3 – user, user  4 – Harry Potter, 15  5 – Dave, Russell, 21, 01231 123132, 2  6 - 2 |  |  |  |  |
| 1 | assertNotNull(main\_package.Main.loginAdminFunction()); | Pass | Pass | N/A | None taken |
| 2 | assertNotNull(main\_package.Main.premiumSeatPrice()); | Pass | Pass | N/A | None taken |
| 3 | assertNotNull(main\_package.Main.loginUserFunction()); | Pass | Pass | N/A | None taken |
| 4 | assertNotNull(main\_package.Main.addFilmFunction()); | Pass | Pass | N/A | None taken |
| 5 | try {  assertTrue(main\_package.Main.advancedBooking());  } catch (Exception e) {  fail("Will fail regardless. Has no showings to reference");  } | Fail | Fail | N/A | None taken |
| 6 | try {  assertTrue(main\_package.Main.makeBooking());  } catch (Exception e) {  fail("Will fail regardless. Has no showings to reference");  } | Fail | Fail | N/A | Test method separately |

## 3.4 – 2.3.4\_04: White Box Testing

IT GOES HERE

# 4.1 – 2.3.2\_01: Requirements TraciNG

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Req ID** | **Requirement** | **Description** | **Method which satisfies requirement** | **Class** | **Location** | **Method Description** | **Test** |
| 3.1.1 | Log operator onto the system | All operators must login to the system with a username and password to access the system | public static boolean loginAdmin() and public static boolean loginUser() | Main | Line 194 - 223 | Authentication is done programatically using static strings, and equalsignorecase method of the string class. An unsecure method, however it fits the requirements |  |
| 3.1.2 | Record the operator's bookings | Record a log of the bookings by each operator. | public Booking(int,int,int,int,double,String) | Booking | Line 30 - 38 | Booking constructor includes the staff ID of the member of staff who logged the booking. The booking information is stored in a collection and can be extracted. |  |
| 3.1.3 | Book seats up to 6 hours before the film | Book up to 10 seats in one booking (1 row) Peak Times are fixed on Saturdays and Sundays between 12-4 and 6-11 | public void addSeats(Seats) | Seats | 154 - 176 | Method checks if the current date/time is not equal to the date/time of the showing and proceeds. Method then enters a for loop which specifies how many times to loop based on the number of seats the customer wishes to book. The method checks that the specified seat is empty, if not it throws and error which is caught and an error specifying that the desired seat is already booked. The method then populates the position and loops until the loop is satisfied. |  |
| 3.1.4 | Cancel Bookings | Customer’s bookings can be cancelled via request of the operator, they require details of customer and film times and payment details for a refund | NYI | N/A | N/A | N/A | N/A |
| 3.1.5 | View which seats are available | Operators can view which seats are available. It is possible for customers to request which seats are available for booking. | public String toString() and public void listByBooking() | Seats | Line 210 - 230 | toString() method enters a loop which prints out the lseats row by row, an empty seat - denoted by [ ] is printed if the seat is empty. And a [X] is printed if the seat is taken. The listByBooking() method loops through the collection of bookings with an enhanced for loop and prints all the bookings from all screens by booking ID and seat reference |  |
| 3.1.6 | Different prices for premium seats | Rows H and I are premium seats | NYI | N/A | N/A | N/A | N/A |
| 3.1.7 | Able to set prices of premium seats as a percentage increase of the costs for standard seating |  | NYI | N/A | N/A | N/A | N/A |
| 3.1.8 | Store film information | File name and age rating for each film as text information | public Film() and public void addFilm(Film) | Film | Line 18 - 32 | Constructor public Film() constructs a Film object with the attributes specified: Film title and rating. Method addFilm adds the film object to a collection. |  |
| 3.1.9 | Check customer is correct age for viewing the film | System function, enter date of birth of customer when booking film, check against age rating of film and if age of customer is equal to or greater than the age rating of film then booking can be completed | public static boolean advancedBooking() throws Exception | Main | Line 243 - 285 | When an advanced booking is made, the operator must enter the customers' details into the system. The booking is then made using the customers details. If the rating of the showing is greater than the customers' age, the booking is rejected. | N/A |
| 3.2.1 | Personal information not to be kept for more than one year on database | No credit card information is to be kept of customer | NYI | N/A | N/A | N/A | N/A |
| 3.5.1 | Software colour scheme to be red, white and green |  |  |  |  |  |  |
| 3.5.2 | Minimise number of steps to complete booking to keep it as simple as possible |  |  |  |  |  |  |

# User Manual

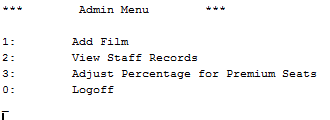
## 5.1 – 2.3.5\_01: User Manual

### Administrator User Guide

Default admin username: admin

Default admin password: admin

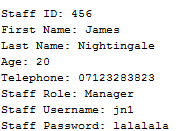
* To log in as admin, enter 1 from main menu, this will take you to username and password input screens.



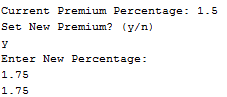
* To add a film select option one to bring you to the next menu, this will ask you to enter a film name and age rating.



* To look at staff records enter option 2 at the admin menu and will display the details of staff.



* As the admin it is also possible to change the percentage of the price increase from standard seats to premium seats. It will display the current percentage and then ask if you would like to change it, if you select yes it will ask you for the new percentage.



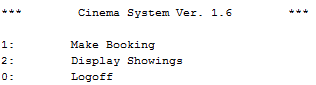
* To log out of the admin account press 0 at the main menu.

### Standard User Guide

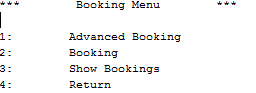
Default user username: user

Default user password: user

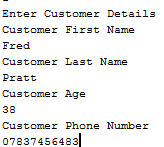
* To log in as user, enter 2 from main menu, this will take you to username and password input screens.



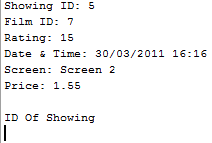
* To make a new booking select option 1 to go to the booking menu.



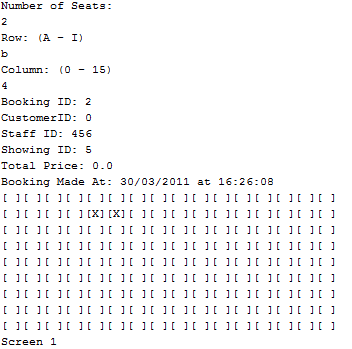
* To make an advanced booking select option 1, this will take you to another screen which will require you to input your name, age and telephone number.



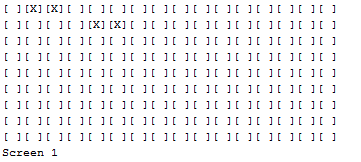
* After you have entered your details a list of film showings will appear and you will be asked to enter the ID of the showing you wish to book.



* You will then be asked to enter the number of seats you wish to book and select which row and column you wish to have the seats in. It will then display the total price of the booking and then a visual display of the seats being booked with date and time of when the booking was made with the screen number.



* To complete a normal booking you must select booking from the main menu by entering 2, the process is the same as advanced booking however you will not be required to enter the customer details.
* In order to see the bookings that have already been made it is possible to select the show bookings option from the main menu, this will display the seats that have been booked for each screen of the cinema.



* It is then possible to logoff the system by entering 0 at the main menu, from here you can select to log back in as admin or user.