Book A Movie Ticket - JUnit Test

**Grade settings**: Maximum grade: 100  
**Disable external file upload, paste and drop external content**: Yes  
**Run**: Yes **Evaluate**: Yes  
**Automatic grade**: Yes

***BookAMovie*** is an automated Movie ticket booking system. Using the application, the management wants to take reports based on the tickets booked for the movies screened by them.

They have developed an application for the above purpose. The details of the various functions supported by the system are provided in this case study.

You are required to write Junit test case and check the correctness of the application developed.

**Functional Requirements :**

The application has the below classes and methods implemented.

You are provided with a model class MovieTicket

**Component Specification : MovieTicket  (Model Class)**

|  |  |  |
| --- | --- | --- |
| **Type(Class)** | **Attributes** | **Methods** |
| MovieTicket | int ticketId  String movieName  int screenNumber  int numberOfSeats  String circle  LocalDate showDate | Necessary getters and setters are provided.  A six argument constructor in the order ticketId, movieName, screenNumber, numberOfSeats, circle and showDate  is provided.  Also the equals and toString methods are overridden. |

Here the circle  can take a value either “King” or “Queen”.  King circle represents the first five rows in a screen and the rest rows falls in Queen circle. Depending on the circle the ticket cost is calculated.

**Component Specification :  InvalidMovieTicketException  (This class inherits the Exception Class)**

|  |  |
| --- | --- |
| **Type(Class)** | **Methods** |
| InvalidMovieTicketException | Provided with a single argument constructor – InvalidMovieTicketException(String message) |

You are also provided with a utility class BookAMovie with business methods.

**Component Specification : BookAMovie (Utility Class)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Attributes** | **Methods** |
| Book a Movie Ticket | BookAMovie | List<MovieTicket> movieTicketList | Getter and setter for the movieTicketList are provided. |

The below are the requirements  implemented in the Utility class for which JUnit test cases are to be written and tested.

**Component Specification:  BookAMovie  (Utility Class)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validating the circle | BookAMovie | public boolean validateCircle(String circle) | Validate the circle.. If valid return true else this method should throw a userdefined exception | Throw a user defined exception “InvalidMovieTicketException”   if the circle is invalid with a message  “Invalid circle”.  Circle is valid if its value is “King” or “Queen” (case insensitive) |
| Book a movie ticket | BookAMovie | public boolean addMovieTicket(int ticketId, String movieName, int screenNumber, int numberOfSeats, String circle, LocalDate showDate) | This method should validate the circle. If circle is valid add to the movieTicketList. Else throw a user defined exception. | If circle is invalid, throw a user defined exception “InvalidMovieTicketException” if the movie ticket is not added. |
| View Movie Ticket based on ticket id. | BookAMovie | public MovieTicket viewMovieTicketById(int   ticketId) | This method should retrieve the MovieTicket that holds the ticketId passed as parameter. If MovieTicket object exists return that object, Else thow a user defined exception. | Throw a user defined exception “InvalidMovieTicketException”   if a movie ticket does not exist with a message  “Invalid movie ticket id” |

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Methods** | **Responsibilities** |
| View Tickets booked based on screen number | BookAMovie | public List<MovieTicket> viewMovieTicketByScreen(int screenNumber) | This method takes the screenNumber as argument. It should return the list of movie tickets that are booked for the screen number passed as argument. |
| View Tickets booked Movie wise. | BookAMovie | public Map<String,List<MovieTicket>>  viewTicketsMovieWise() | This method should return the list of MovieTicket for each movie name.  To do this, iterate the movieTicketList and return a Map that holds movieName as key and the list of MovieTicket objects pertaining to that movieName as values. |
| View total seats booked details screen wise within a period | BookAMovie | public Map<Integer,Integer> viewScreenWiseTotalSeatsBooked ( LocalDate fromDate, LocalDate toDate) | This method takes fromDate and toDate as parameter. It should find the total number of seats  booked under each screen within from date and to date and populate the map with key as screen number and value as total number of seats booked.  To take the total tickets booked take the sum of numberOfSeats from the MovieTicket objects. |

You need to write Junit test for the BookAMovie class.

**Testing Scenarios:**

You are provided with a class  “BookAMovieTest”  to do this testing.

**Note :**

              To perform testing the movieTicketList should contain objects of MovieTicket.

              To do this, in BookAMovieTest  class you are provided with a setup method.  Use this method to initialize the  movieTicketList  attribute in BookAMovie class.  Create few objects for  MovieTicket,  populate a list with these objects and set the movieTicketList to this list using the setMovieTicketList method in BookAMovie class.

The below are the test methods  to be implemented in BookAMovieTest  class.

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidateCircleWhenKing | This method should test the validateCircle method when a valid circle “king” is passed as parameter to this method. |
| test12ValidateCircleWhenQueen | This method should test the validateCircle method when a valid circle “queen” is passed as parameter to this method. |
| test13ValidateCircleWhenInvalid | This method should test the validateCircle method when an invalid circle  is passed to this method.  validateCircle method is expected  to throw InvalidMovieTicketException when circle is invalid.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test14AddMovieTicketForValidCircle | This method should test the addMovieTicket method when  valid circle is provided for the MovieTicket.  Test for the success scenario of addition of MovieTicket object into the list. |
| test15AddMovieTicketForInvalidCircle | This method should test the addMovieTicket method when  invalid circle is provided for the MovieTicket. In this case, addMovieTicket method is expected to throw InvalidMovieTicketException.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test16ViewMovieTicketByIdForValidId | This method should test the viewMovieTicketById method when a ticketId is passed as parameter exists in the movieTicketList. |
| test17ViewMovieTicketByIdForInvalidId | This method should test  the viewMovieTicketById method when a ticketId is passed as parameter does not exist in the movieTicketList.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test18ViewMovieTicketByScreen | This method should test the correctness of  viewMovieTicketByScreen method.  Perform testing for the correctness of the list returned. |
| test19VewTicketsMovieWise | This method should test the correctness of viewTicketsMovieWise method.  Perform testing for the correctness of the map returned. |
| test20ViewScreenWiseTotalSeatsBooked | This method should test the viewScreenWiseTotalSeatsBooked method.  Perform testing for the correctness of the map returned. |

Implement the test methods and provide the needed annotation to all the methods in BookAMovieTest class.

Also provide the required annotation, so that the test methods are executed in ascending order of the test method names.

You are provided with a Main class with the main method to check the correctness of the test methods written in BookAMovieTest class.

Having completed writing the test methods, uncomment the code in Main class and execute the main method.

### **Automatic evaluation**[**[-]**](javascript:void(0);)

**Proposed grade: 100.0 / 100**  
**Result Description**  
[[-]](javascript:void(0);)**Grading and Feedback**

*Writing JUnit for a Utility class - 60.0 / 60.0(Success)*

*Test Coverage - 40 / 40.0(Success)*

Test Case Passed

#### **BookMovieTicket/src/BookAMovie.java**

1 *import* java.time.LocalDate;

2 *import* java.util.ArrayList;

3 *import* java.util.LinkedHashMap;

4 *import* java.util.List;

5 *import* java.util.Map;

6

7 // System Under Test

8 *public* *class* BookAMovie {

9

10 List<MovieTicket> movieTicketList = *new* ArrayList<>();

11

12 *public* List<MovieTicket> getMovieTicketList() {

13 *return* movieTicketList;

14 }

15

16 *public* *void* setMovieTicketList(List<MovieTicket> movieTicketList) {

17 *this*.movieTicketList = movieTicketList;

18 }

19

20 *public* *boolean* validateCircle(String circle) throws InvalidMovieTicketException {

21 *if* (circle.equalsIgnoreCase("king") || circle.equalsIgnoreCase("queen"))

22 *return* *true*;

23 *else*

24 *throw* *new* InvalidMovieTicketException("Invalid circle");

25 }

26

27 *public* *boolean* addMovieTicket(*int* ticketId, String movieName, *int* screenNumber, *int* numberOfSeats, String circle,

28 LocalDate showDate) throws InvalidMovieTicketException {

29 *if* (validateCircle(circle)) {

30 movieTicketList.add(*new* MovieTicket(ticketId, movieName, screenNumber, numberOfSeats, circle, showDate));

31 *return* *true*;

32 } *else*

33 *throw* *new* InvalidMovieTicketException("Invalid circle");

34 }

35

36 *public* MovieTicket viewMovieTicketById(*int* ticketId) throws InvalidMovieTicketException {

37 *for* (MovieTicket ticketObj : movieTicketList) {

38 *if* (ticketObj.getTicketId() == ticketId)

39 *return* ticketObj;

40 }

41 *throw* *new* InvalidMovieTicketException("Invalid movie ticket id");

42 }

43

44 *public* List<MovieTicket> viewMovieTicketByScreen(*int* screenNumber) {

45 List<MovieTicket> temp = *new* ArrayList<MovieTicket>();

46

47 *for* (MovieTicket ticketObj : movieTicketList) {

48 *if* (ticketObj.getScreenNumber() == screenNumber)

49 temp.add(ticketObj);

50 }

51 *return* temp;

52 }

53

54 *public* Map<String, List<MovieTicket>> viewTicketsMovieWise() {

55

56 Map<String, List<MovieTicket>> temp = *new* LinkedHashMap<String, List<MovieTicket>>();

57

58 *for* (MovieTicket ticketObj : movieTicketList) {

59 String movieName = ticketObj.getMovieName();

60 *if* (!temp.containsKey(movieName))

61 temp.put(movieName, *new* ArrayList<>());

62 temp.get(movieName).add(ticketObj);

63 }

64 *return* temp;

65

66 }

67

68 *public* Map<Integer, Integer> viewScreenWiseTotalSeatsBooked(LocalDate fromDate, LocalDate toDate) {

69

70 Map<Integer, Integer> temp = *new* LinkedHashMap<Integer, Integer>();

71

72 *for* (MovieTicket ticketObj : movieTicketList) {

73

74 *if* (ticketObj.getShowDate().compareTo(fromDate) >= 0 && ticketObj.getShowDate().compareTo(toDate) <= 0) {

75 *int* screenNumber = ticketObj.getScreenNumber();

76 *if* (!temp.containsKey(screenNumber))

77 temp.put(screenNumber, 0);

78 temp.put(screenNumber, temp.get(screenNumber) + ticketObj.getNumberOfSeats());

79 }

80

81 }

82 *return* temp;

83

84 }

85

86 }

87

#### **BookMovieTicket/src/BookAMovieTest.java**

1 *import* *static* org.junit.Assert.\*;

2

3 *import* java.time.LocalDate;

4 *import* java.util.ArrayList;

5 *import* java.util.List;

6 *import* java.util.Map;

7

8 *import* org.junit.Before;

9 *import* org.junit.FixMethodOrder;

10 *import* org.junit.Test;

11 *import* org.junit.runners.MethodSorters;

12

13 @FixMethodOrder(MethodSorters.NAME\_ASCENDING)

14 *public* *class* BookAMovieTest {

15

16 *static* BookAMovie bookAMovie;

17

18 @Before

19 *public* *void* setUp() throws Exception {

20 bookAMovie = *new* BookAMovie();

21 // Try to create few MovieTicket objects and add to a list.

22 // Set that list to movieTicketList in BookAMovie using setMovieTicketList

23 // method

24 List<MovieTicket> movieList = *new* ArrayList<MovieTicket>();

25 movieList.add(*new* MovieTicket(1000, "Goblin", 10, 50, "king", LocalDate.of(2020, 1, 20)));

26 movieList.add(*new* MovieTicket(1001, "Death Note", 10, 50, "queen", LocalDate.of(2020, 1, 22)));

27 movieList.add(*new* MovieTicket(1002, "Steins Gate", 10, 50, "king", LocalDate.of(2020, 1, 24)));

28 movieList.add(*new* MovieTicket(1003, "Fullmetal Alchemist", 10, 50, "queen", LocalDate.of(2020, 1, 26)));

29 movieList.add(*new* MovieTicket(1004, "Baki", 10, 50, "king", LocalDate.of(2020, 1, 28)));

30 bookAMovie.setMovieTicketList(movieList);

31 }

32

33 @Test

34 *public* *void* test11ValidateCircleWhenKing() throws InvalidMovieTicketException {

35

36 // test the validateCircle method when a valid circle king is provided

37 assertTrue(bookAMovie.validateCircle("king"));

38 }

39

40 @Test

41 *public* *void* test12ValidateCircleWhenQueen() throws InvalidMovieTicketException {

42

43 // test the validateCircle method when a valid circle queen is provided.

44 assertTrue(bookAMovie.validateCircle("queen"));

45 }

46

47 @Test(expected = InvalidMovieTicketException.*class*)

48 *public* *void* test13ValidateCircleWhenInvalid() throws InvalidMovieTicketException {

49

50 // test the validateCircle method when an invalid circle is passed to this

51 // method

52 bookAMovie.validateCircle("jack");

53 }

54

55 @Test

56 *public* *void* test14AddMovieTicketForValidCircle() throws InvalidMovieTicketException {

57

58 // test the addMovieTicket method when valid circle is provided for the

59 // MovieTicket

60 assertTrue(bookAMovie.addMovieTicket(1005, "Hospital Playlist", 10, 50, "king", LocalDate.of(2020, 1, 27)));

61 }

62

63 @Test(expected = InvalidMovieTicketException.*class*)

64 *public* *void* test15AddMovieTicketForInvalidCircle() throws InvalidMovieTicketException {

65

66 // test the addMovieTicket method when invalid circle is provided for the

67 // MovieTicket

68 bookAMovie.addMovieTicket(1005, "Hospital Playlist", 10, 50, "jack", LocalDate.of(2020, 1, 27));

69 }

70

71 @Test

72 *public* *void* test16ViewMovieTicketByIdForValidId() throws InvalidMovieTicketException {

73

74 // test the viewMovieTicketById method when a ticketId is passed as parameter

75 // exists in

76 // the movieTicketList

77 assertTrue(bookAMovie.viewMovieTicketById(1000).getTicketId() == 1000);

78 }

79

80 @Test(expected = InvalidMovieTicketException.*class*)

81 *public* *void* test17ViewMovieTicketByIdForInvalidId() throws InvalidMovieTicketException {

82

83 // test the viewMovieTicketById method when a ticketId is passed as parameter

84 // does not exist in the movieTicketList

85 bookAMovie.viewMovieTicketById(900);

86 }

87

88 @Test

89 *public* *void* test18ViewMovieTicketByScreen() {

90

91 // test the viewMovieTicketByScreen method

92 assertFalse(bookAMovie.viewMovieTicketByScreen(10).isEmpty());

93 }

94

95 @Test

96 *public* *void* test19VewTicketsMovieWise() {

97

98 // test the viewTicketsMovieWise method

99 assertFalse(bookAMovie.viewTicketsMovieWise().isEmpty());

100 }

101

102 @Test

103 *public* *void* test20ViewScreenWiseTotalSeatsBooked() {

104

105 // test the viewScreenWiseTotalSeatsBooked method

106 assertFalse(bookAMovie.viewScreenWiseTotalSeatsBooked(LocalDate.of(2020, 1, 20), LocalDate.now()).isEmpty());

107 }

108

109 }

110

#### **BookMovieTicket/src/InvalidMovieTicketException.java**

1 @SuppressWarnings("serial")

2 *public* *class* InvalidMovieTicketException *extends* Exception {

3

4 *public* InvalidMovieTicketException(String message) {

5 *super*(message);

6 }

7 }

8

9

#### **BookMovieTicket/src/Main.java**

1 *import* org.junit.runner.JUnitCore;

2 *import* org.junit.runner.Result;

3 *import* org.junit.runner.notification.Failure;

4

5 *public* *class* Main {

6 *public* *static* *void* main(String args[]) {

7

8 // Uncomment these lines after completing the JUnit Code

9 Result result = JUnitCore.runClasses(BookAMovieTest.*class*);

10

11 *if* (result.getFailureCount() == 0) {

12 System.out.println("No Failures");

13 } *else* {

14 *for* (Failure failure : result.getFailures())

15

16 {

17 System.out.println(failure.toString());

18 }

19 }

20 System.out.println("Result " + result.wasSuccessful());

21 }

22 }

23

#### **BookMovieTicket/src/MovieTicket.java**

1 *import* java.time.LocalDate;

2 *import* java.time.format.DateTimeFormatter;

3

4 *public* *class* MovieTicket {

5

6 *private* *int* ticketId;

7 *private* String movieName;

8 *private* *int* screenNumber;

9 *private* *int* numberOfSeats;

10 *private* String circle;

11 *private* LocalDate showDate;

12

13 *public* MovieTicket() {

14

15 }

16

17 *public* MovieTicket(*int* ticketId, String movieName, *int* screenNumber, *int* numberOfSeats, String circle,

18 LocalDate showDate) {

19 *this*.ticketId = ticketId;

20 *this*.movieName = movieName;

21 *this*.screenNumber = screenNumber;

22 *this*.numberOfSeats = numberOfSeats;

23 *this*.circle = circle;

24 *this*.showDate = showDate;

25 }

26

27 *public* *int* getTicketId() {

28 *return* ticketId;

29 }

30

31 *public* *void* setTicketId(*int* ticketId) {

32 *this*.ticketId = ticketId;

33 }

34

35 *public* String getMovieName() {

36 *return* movieName;

37 }

38

39 *public* *void* setMovieName(String movieName) {

40 *this*.movieName = movieName;

41 }

42

43 *public* *int* getScreenNumber() {

44 *return* screenNumber;

45 }

46

47 *public* *void* setScreenNumber(*int* screenNumber) {

48 *this*.screenNumber = screenNumber;

49 }

50

51 *public* *int* getNumberOfSeats() {

52 *return* numberOfSeats;

53 }

54

55 *public* *void* setNumberOfSeats(*int* numberOfSeats) {

56 *this*.numberOfSeats = numberOfSeats;

57 }

58

59 *public* String getCircle() {

60 *return* circle;

61 }

62

63 *public* *void* setCircle(String circle) {

64 *this*.circle = circle;

65 }

66

67 *public* LocalDate getShowDate() {

68 *return* showDate;

69 }

70

71 *public* *void* setShowDate(LocalDate showDate) {

72 *this*.showDate = showDate;

73 }

74

75 *public* *boolean* equals(Object o) {

76 MovieTicket m = (MovieTicket) o;

77 *if* (*this*.ticketId == m.ticketId && *this*.movieName.equals(m.movieName))

78 *return* *true*;

79 *else*

80 *return* *false*;

81 }

82

83 *public* String toString() {

84

85 String date = showDate.format(DateTimeFormatter.ofPattern("dd-MM-yy"));

86 *return* "Ticket ID " + ticketId + " Movie Name " + movieName + " Screen " + screenNumber + " No.of seats "

87 + numberOfSeats + " Circle " + circle + " Show Date " + date;

88 }

89

90 }

91

## Grade

Reviewed on Thursday, 11 March 2021, 3:29 AM by Automatic grade  
**Grade** 100 / 100  
**Assessment report**  
[[-]](javascript:void(0);)**Grading and Feedback**

*Writing JUnit for a Utility class - 60.0 / 60.0(Success)*

*Test Coverage - 40 / 40.0(Success)*

Test Case Passed