İZMİR UNIVERSITY OF ECONOMICS

SE 318
SOFTWARE VERIFICATION AND VALIDATION
SPRING 2022

CINE-TICKET

SELEN SONMEZ

ELÇIN KARADENIZLI

ARIF BATUHAN YILDIRIMOĞLU

SIMGE AYDOGAN

UNIT TEST DOCUMENT

Version *3.0*

06.06.2022

VERSION HISTORY

VERSION 1.0 (30.04.2022)
In this release, we have fulfilled the below requirements which have the highest priorities: Functional Requirements: -The system should allow customers to view the available seats and timings for a particular movie. -The system should provide customers a list of the movies which are being shown in the theater or to be released for a short while. -The system should allow customers to create their accounts with their email addresses and passwords. Non-Functional Requirements: -The system will include try-catch block to prevent users to see any errors. -The system will be served as a desktop application.
VERSION 2.0 (21.05.2022)
In this release, we have fulfilled the below requirements which have medium priorities: Functional Requirements: -The system should allow customers to make payment online using various payment gateways like Credit/Debit cards. -The system should allow the owner to keep track on available seats for a particular movie. -The system should allow customers to book the tickets online.
VERSION 3.0 (06.06.2022)
In this release, we have fulfilled the below requirements which have low priorities: Functional Requirements: -The system should allow customers to cancel their tickets. -The system should allow the owner to check the progress of ticket reservation. -The system should allow the owner to manage the entire system from a single portal. Non-Functional Requirements: -The system should provide information boxes which will appear next to each option to provide more clear understanding.

INTRODUCTION

1.1 PURPOSE OF THE TEST CASE DOCUMENT

The Test Case document documents the functional requirements of the Cine-Ticket test case. The intended audience is the project manager, project team, and testing team. Some portions of this document may on occasion be shared with the client/user and other stakeholder whose input/approval into the testing process is needed.
Cine-Ticket is a desktop application for a movie ticket booking system. Tickets for various movies can be booked using the online facilities. Real-time customer and reservation information is received from the database. There are several modules like registration, booking, payment, monitoring of movies and reservations depending on the user type which are integrated to fulfill the purpose.

1.2 CONSTRAINTS

Cine-Ticket is implemented using Java programming language. JUnit framework is used to test the written code on Eclipse.

2 UNIT TEST FRAMEWORK: *JUNIT*

JUnit is a unit testing framework for the Java programming language. JUnit promotes the idea of “first testing then coding”, which emphasis on setting up the test data for a piece of code which can be tested first and then can be implemented. This approach increases the productivity and stability of program code and reduces the time spent on debugging.

JUnit is an open-source framework that has several useful features. It provides annotations to identify test methods, assertions for testing expected results and test runners for running tests. JUnit being less complex results as taking much less time. JUnit tests can be run automatically and check their own results and provide immediate feedback. Also, JUnit allows you to organize tests into test suites which eases the process of running multiple tests. Test progress bars which are showed up after running the test allows you visualize the test results by turning into green after a successful test and otherwise, turns into red.

3 TEST CASES

Test Case 1	
Test Definition	
Test case 1 allows us to test whether the customer's login information like e-mail and password belongs to the correct customer.	
Input Value	
email : elcin@gmail.com, password: 123	
Expected Value	**Actual Value**
“elcin”	“elcin”
Result of Test Case	*successful*
Test Script	

```
public class TestCase1 extends LoginInterface{
        LoginInterface test = new LoginInterface();

        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test=null;
        }

        @Test
        public void positiveTest() {


                String email = "elcin@gmail.com";
                String password = "123";

                String customer = test.getCustomer(email, password).getName();
                assertEquals("elcin", customer);


        }
}
```

Test Case 2	
Test Definition	
Test case 2 allows us to test whether the movies are in the right movie theater.	
Input Value	
name : “Batman”	
Expected Value	**Actual Value**
2	**1**
Result of Test Case	***successful***
Test Script	

```
public class TestCase2 {

            MoviesInterface test = new MoviesInterface();

            @Before
            public void setUp() throws Exception {
            }

            @After
            public void tearDown() throws Exception {
                        test = null;
            }


            @Test
            public void negativeTest() {
                        int index = 0;
                        String name = "Batman";
                        for(int i = 0; i<test.getMovieNames().size(); i++) {

                                    if(test.getMovieNames().get(i).getText().equals(name)) {
                           index = i;
                         break;
                                    }

                        }
                         int theaterNo = Integer.parseInt(test.getMovieTheaterNo().get(index).getText());
                        assertNotEquals(2, theaterNo);

            }

}
```

Test Case 3	
Test Definition	
Test case 3 allows us to test whether the movie “Batman is in the right time	
Input Value	
name = “Batman”	
Expected Value	**Actual Value**
“11.00”	**“11.00”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase3 {

        MoviesInterface test = new MoviesInterface();

        @Before
        public void setUp() throws Exception {

        }

        @After
        public void tearDown() throws Exception {

        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Batman";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                   index = i;
                  break;
                        }

                }
                 String movieTime = test.getMovieTimes().get(index).getText();

                assertEquals("11.00", movieTime);

        }
```

Test Case 4	
Test Definition	
Test case 4 allows us to check from the database whether the seats have been taken.	
Input Value	
movieTheater = 1, seatNumber = 1	
Expected Value	**Actual Value**
true	**false**
Result of Test Case	***successful***
Test Script	

```
public class TestCase4 {
	DatabaseConnection db = new DatabaseConnection();
	movieTheater =1;
	seatNumber = 1;
	@Before
	public void setUp() throws Exception {
	}

	@After
	public void tearDown() throws Exception {
		db = null;
	}

	@Test
	public void negativeTest() {

		Boolean isTaken = db.getSeats(movieTheater).get(seatNumber-1);
		assertNotEquals(true,isTaken);
	}

}
```

Test Case 5	
Test Definition	
Test case 5 allows us to check from the database whether the seats are empty.	
Input Value	
movieTheater = 1, seatNumber = 2	
Expected Value	**Actual Value**
false	**false**
Result of Test Case	***successful***
Test Script	

```
public class TestCase5 {
        DatabaseConnection db = new DatabaseConnection();
        movieTheater =1;
        seatNumber = 2;

        @Before
        public void setUp() throws Exception {

        }

        @After
        public void tearDown() throws Exception {
                db = null;
        }

        @Test
        public void positiveTest() {

                Boolean isTaken = db.getSeats(movieTheater).get(seatNumber -1);
                assertEquals(false,isTaken);
        }
```

Test Case 6	
Test Definition	
Test case 6 allows us to check if an already existed movie can be inserted into database or not	
Input Value	
movie(“name”, “time”, 6, “price”)	
Expected Value	**Actual Value**
true	**false**
Result of Test Case	***successful***
Test Script	

```
public class TestCase6 {
	Movies movie = null;
	Boolean isNull = false;

	@Before
	public void setUp() throws Exception {


	}

	@After
	public void tearDown() throws Exception {

		DatabaseConnection.deleteMovies(movie.getName());
	}



	//CHECKS IF AN ALREADY EXISTED MOVIE CAN BE INSERTED INTO DATABASE OR NOT
	@Test
	public void negativeTest() {
		movie = DatabaseConnection.insertMovies("name","time" , 6, "price");

		if(movie == null) {
			isNull = true;
		}
		assertNotEquals(true,isNull);

	}

}
```

Test Case 7	
Test Definition	
Test case 7 allows us to check if a movie that is not placed in database can be deleted or not	
Input Value	
Movie(“movieName”, “movieTime”, 5, “moviePrice”)	
Expected Value	**Actual Value**
true	**true**
Result of Test Case	***successful***
Test Script	

```
public class TestCase7 {
        AddMovieInterface addMovie = new AddMovieInterface();
        DeleteMovieInterface deleteMovie = new DeleteMovieInterface();
        DatabaseConnection db = new DatabaseConnection();
        int size = 0;

        @Before
        //To check multiple times, the movie we wanted to delete is inserted before the deletion in case of
        //preventing errors or wrong results
        public void setUp() throws Exception {
                addMovie.insertMovies("movieName","movieTime" , 5, "moviePrice");
                for(int i = 0; i<5; i++) {
                        System.out.println(db.getMovieNamesForCombobox()[i]);
                }
                size = db.getMovieNamesForCombobox().length;
        }

        @After
        public void tearDown() throws Exception {
        deleteMovie.deleteMovies(deleteMovie.getMovieNameToBeDeleted("movieName"));
                addMovie = null;
                deleteMovie = null;
        }
        //CHECKS IF A MOVIE THAT IS NOT PLACED IN THE DATABASE CAN BE DELETED OR NOT
        @Test
        public void positiveTest() {
                deleteMovie.deleteMovies(deleteMovie.getMovieNameToBeDeleted("movieName"));
                int deletedSize = db.getMovieNamesForCombobox().length;

                System.out.println(size);
                assertTrue(deletedSize != size);

        }
```

Test Case 8	
Test Definition	
Test case 8 allows us to check if the card that belongs to customer and the card information of the customer that stored in the database is same	
Input Value	
Email : “selen@gmail.com”, Password : 123	
Expected Value	**Actual Value**
Memory address of the card (Card@49ec71f8)	**Memory address of the card (Card@49ec71f8)**
Result of Test Case	***successful***
Test Script	

```
public class TestCase8 {

	LoginInterface login = new LoginInterface();
	private Card card;
	private Card card2;
	private Card card3;

	private Customer customer;

	@Before
	public void setUp() throws Exception {

		customer = login.getCustomer("selen@gmail.com","123");
		card2 = DatabaseConnection.getCardInfoOfCustomer("selen@gmail.com");
		customer.setCard(card2);
	}

	@After
	public void tearDown() throws Exception {

	}

	@Test
	public void positiveTest() {

		card =customer.getCard();

		assertSame(card, card2);

	}
}
```

Test Case 9	
Test Definition	
Test case 9 allows us to check if the proceed button at movie theater scene does what is intended to do	
Input Value	
TheaterID = 1, seatNumber=7	
Expected Value	**Actual Value**
SeatStatus.PROCESSING	**SeatStatus.PROCESSING**
Result of Test Case	***successful***
Test Script	

```
public class TestCase9 {

        CinemaSeatsDesign c = new CinemaSeatsDesign();
        SeatButton seatButton;
        int theaterID = 1;
        nt seatNumber = 7;

        @Before
        public void setUp() throws Exception {
                for(Map.Entry <Integer,ArrayList<SeatButton>> m :
c.getSeatsForEachMovieTheater().entrySet()) {
                        if(m.getKey()==theaterID) {
                                seatButton = m.getValue().get(seatNumber-1);
                                m.getValue().get(seatNumber-1).setStatus(SeatStatus.EMPTY);
                        }
                }
                }



        @After
        public void tearDown() throws Exception {
        }

        @Test
        public void positiveTest() {

                seatButton.doClick();

                assertEquals(seatButton.getStatus(), SeatStatus.PROCESSING);
        }
}
```

Test Case 10	
Test Definition	
Test case 10 allows us to check if the taken seat seems as taken when a seat is taken	
Input Value	
theaterID = 1, otherSeatNumber = 2	
Expected Value	**Actual Value**
true	**false**
Result of Test Case	***successful***
Test Script	

```
public class TestCase10 {

        DatabaseConnection db = new DatabaseConnection();
        int theaterID = 1;
        int seatNumber = 3;
        int otherSeatNumber =2;

        @Before
        //Seat 3 is taken before the test
        public void setUp() throws Exception {
                db.TakeSeat(theaterID, seatNumber);
        }

        @After
        //Seat 3 made available after the test
        public void tearDown() throws Exception {

                db.UpdateSeats(theaterID, seatNumber);
        }

        @Test
        //Checking if the seat 3 is not taken
        public void negativeTest() {

                        db.getSeats(theaterID).get(otherSeatNumber-1);

                        assertNotEquals(db.getSeats(theaterID).get(otherSeatNumber-1), false);
        }

}
```

Test Case 11	
Test Definition	
Test case 11 allows us to check if a registered customer can login when the password was wrong	
Input Value	
Password = “123”	
Expected Value	**Actual Value**
true	**true**
Result of Test Case	***successful OR***
Test Script	

```
public class TestCase11 {
	LoginInterface login = new LoginInterface();
	boolean isPasswordTrue;
	String password;

	@Before
	public void setUp() throws Exception {

	}

	@After
	public void tearDown() throws Exception {
	}

	@Test
	public void positiveTest() {
		password = "123";
		//A customer has already registered with this email and the password was 123
		//Checking if he can log into the system
		if(login.getCustomer("uzunbayir.serhat@ieu.edu.tr",password)!=null) {
			isPasswordTrue = true;
		}
		else {
			isPasswordTrue = false;
		}
		assertEquals(true, PasswordTrue);

	}
}
```

Test Case 12	
Test Definition	
Test case 12 allows us to check if the customers and the movie tickets that they booked seem to system owner reservation list are in correct order	
Input Value	
Email = elcin@gmail.com	
Expected Value	**Actual Value**
Morbius	**Morbius**
Result of Test Case	***successful***
Test Script	

```
public class TestCase12 {
	SystemOwnerReservationsInterface sori = new SystemOwnerReservationsInterface();
	String movieName;
	@Before
	public void setUp() throws Exception {
	}

	@After
	public void tearDown() throws Exception {
	}

	@Test
	public void positiveTest() {
		//This customer booked a ticket for Morbius
		int index = 0;
		String email = "elcin@gmail.com";

		for(int i = 0; i<sori.getCustomerMails().size(); i++) {

			if(sori.getCustomerMails().get(i).getText().equals(email)) {
				index = i;
				break;
			}

		}
		movieName = sori.getMovieNames().get(index).getText();
		assertEquals("Morbius",movieName);

	}
}
```

Test Case 13	
Test Definition	
Test case 13 allows us to check if the customers and the seat numbers that they booked seem to system owner reservation list are in correct order	
Input Value	
Email = siimgeaydogan@gmail.com	
Expected Value	**Actual Value**
3	**5**
Result of Test Case	***successful***
Test Script	

```
public class TestCase13 {
	SystemOwnerReservationsInterface sori = new SystemOwnerReservationsInterface();
	int seatNumber;
	@Before
	public void setUp() throws Exception {
	}
	@After
	public void tearDown() throws Exception {
	}

	@Test
	public void negativeTest() {
		//This customer booked a ticket for seat 5
		int index = 0;
		String email = "siimgeaydogan@gmail.com";

		for(int i = 0; i<sori.getCustomerMails().size(); i++) {

			if(sori.getCustomerMails().get(i).getText().equals(email)) {
				index = i;
				break;
			}
		}
		seatNumber = Integer.parseInt(sori.getSeatNumber().get(index).getText());
		assertNotEquals(3,seatNumber);
	}

}
```

Test Case 14	
Test Definition	
Test case 14 allows us to test whether the movie "Dune" is in the right movie theater at system owner movie interface.	
Input Value	
name = “Dune”	
Expected Value	**Actual Value**
3	**3**
Result of Test Case	***successful***
Test Script	

```
public class TestCase14 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Dune";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                  index = i;
                 break;
                        }

                }
                 int theaterNo = Integer.parseInt(test.getMovieTheaterNo().get(index).getText());
                assertEquals(3, theaterNo);

        }
}
```

Test Case 15	
Test Definition	
Test case 15 allows us to test whether the movie "Dune" is in the right time at system owner movie interface.	
Input Value	
name = “Dune”	
Expected Value	**Actual Value**
“11.00”	**“11.00”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase15 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Dune";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                          index = i;
                         break;
                        }

                }
                String movieTime = test.getMovieTimes().get(index).getText();
                assertEquals("11.00", movieTime);



        }
}
```

Test Case 16	
Test Definition	
Test case 16 allows us to test whether the movie "Eiffel" has the right price at system owner movie interface.	
Input Value	
name = “Eiffel”	
Expected Value	**Actual Value**
“$5”	**“$5”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase16 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Eiffel";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                    index = i;
                  break;
                        }

                }
                String price = test.getMoviePrices().get(index).getText();
                assertEquals("$5", price);



        }
}
```

Test Case 17	
Test Definition	
Test case 17 allows us to test whether the movie "Batman" has the right price at system owner movie interface.	
Input Value	
name = “Batman”	
Expected Value	**Actual Value**
“$5”	**“$5”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase17 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Batman";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                  index = i;
                 break;
                        }

                }
                String price = test.getMoviePrices().get(index).getText();
                assertEquals("$5", price);



        }
}
```

Test Case 18	
Test Definition	
Test case 18 allows us to test whether the movie "Morbius" has the right price at system owner movie interface.	
Input Value	
name = “Morbius”	
Expected Value	**Actual Value**
“$5”	**“$5”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase18 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Morbius";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                   index = i;
                  break;
                        }

                }
                String price = test.getMoviePrices().get(index).getText();
                assertEquals("$5", price);


        }
}
```

Test Case 19	
Test Definition	
Test case 19 allows us to test whether the movie "Dune" has the right price at system owner movie interface.	
Input Value	
name = “Dune”	
Expected Value	**Actual Value**
“$5”	**“$5”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase19 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Dune";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                   index = i;
                  break;
                        }

                }
                String price = test.getMoviePrices().get(index).getText();
                assertEquals("$5", price);


        }
}
```

Test Case 20	
Test Definition	
Test case 20 allows us to test whether the movie "Morbius" is in the right time at system owner movie interface.	
Input Value	
name = “Morbius”	
Expected Value	**Actual Value**
“11.00”	**“11.00”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase20 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Morbius";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                  index = i;
                 break;
                        }

                }
                String movieTime = test.getMovieTimes().get(index).getText();
                assertEquals("11.00", movieTime);


        }
}
```

Test Case 21	
Test Definition	
Test case 21 allows us to test whether the movie "Eiffel" is in the right time at system owner movie interface.	
Input Value	
name = “Eiffel”	
Expected Value	**Actual Value**
“11.00”	**“11.00”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase21 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Eiffel";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                   index = i;
                  break;
                        }

                }
                String movieTime = test.getMovieTimes().get(index).getText();
                assertEquals("11.00", movieTime);



        }
}
```

Test Case 22	
Test Definition	
Test case 22 allows us to test whether the movie "Batman" is in the right time at system owner movie interface.	
Input Value	
name = “Batman”	
Expected Value	**Actual Value**
“11.00”	**“11.00”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase22 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Batman";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                   index = i;
                  break;
                        }

                }
                String movieTime = test.getMovieTimes().get(index).getText();
                assertEquals("11.00", movieTime);


        }
}
```

Test Case 23	
Test Definition	
Test case 23 allows us to test whether the movie "Batman" is in the right movie theater at system owner movie interface.	
Input Value	
name = "Batman"	
Expected Value	**Actual Value**
1	**1**
Result of Test Case	***successful***
Test Script	

```
public class TestCase23 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Batman";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                  index = i;
                 break;
                        }

                }
                 int theaterNo = Integer.parseInt(test.getMovieTheaterNo().get(index).getText());
                assertEquals(1, theaterNo);

        }
}
```

Test Case 24	
Test Definition	
Test case 24 allows us to test whether the movie "Morbius" is in the right movie theater at system owner movie interface.	
Input Value	
name = “Morbius”	
Expected Value	**Actual Value**
4	**4**
Result of Test Case	***successful***
Test Script	

```
public class TestCase24 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Morbius";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                  index = i;
                 break;
                        }

                }
                 int theaterNo = Integer.parseInt(test.getMovieTheaterNo().get(index).getText());
                assertEquals(4, theaterNo);

        }
}
```

Test Case 25	
Test Definition	
Test case 25 allows us to test whether the movie "Eiffel" is in the right movie theater at system owner movie interface.	
Input Value	
name = “Eiffel”	
Expected Value	**Actual Value**
2	**2**
Result of Test Case	***successful***
Test Script	

```
public class TestCase25 {

        SystemOwnerMovieInterface test = new SystemOwnerMovieInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                String name = "Eiffel";
                for(int i = 0; i<test.getMovieNames().size(); i++) {

                        if(test.getMovieNames().get(i).getText().equals(name)) {
                   index = i;
                  break;
                        }

                }
                 int theaterNo = Integer.parseInt(test.getMovieTheaterNo().get(index).getText());
                assertEquals(2, theaterNo);

        }
}
```

Test Case 26	
Test Definition	
Test case 26 allows us to test the name of the movie that shown in right movie theater 1 according to the customer movie interface.	
Input Value	
theaterID = 1	
Expected Value	**Actual Value**
“Batman”	**“Batman”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase26 {

        MoviesInterface test = new MoviesInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                int theaterID = 1;
                for(int i = 0; i<test.getMovieTheaterNo().size(); i++) {

                        if(Integer.parseInt(test.getMovieTheaterNo().get(i).getText()) == theaterID) {
                   index = i;
                 break;
                        }

                }
                 String movieName = test.getMovieNames().get(index).getText();
                assertEquals("Batman", movieName);

        }
}
```

Test Case 27	
Test Definition	
Test case 27 allows us to test the name of the movie that shown in right movie theater 2 according to the customer movie interface.	
Input Value	
theaterID = 2	
Expected Value	**Actual Value**
“Eiffel”	**“Eiffel”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase27 {

        MoviesInterface test = new MoviesInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void positiveTest() {
                int index = 0;
                int theaterID = 2;
                for(int i = 0; i<test.getMovieTheaterNo().size(); i++) {

                        if(Integer.parseInt(test.getMovieTheaterNo().get(i).getText()) == theaterID) {
                  index = i;
                 break;
                        }

                }
                 String movieName = test.getMovieNames().get(index).getText();
                assertEquals("Eiffel", movieName);


        }
}
```

Test Case 28	
Test Definition	
Test case 28 allows us to test the name of the movie that shown in right movie theater 3 according to the customer movie interface.	
Input Value	
theaterID = 3	
Expected Value	**Actual Value**
“Dune”	**“Batman”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase28 {

        MoviesInterface test = new MoviesInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void negativeTest() {
                int index = 0;
                int theaterID = 3;
                for(int i = 0; i<test.getMovieTheaterNo().size(); i++) {

                        if(Integer.parseInt(test.getMovieTheaterNo().get(i).getText()) == theaterID) {
                   index = i;
                  break;
                        }

                }
                 String movieName = test.getMovieNames().get(index).getText();
                assertNotEquals("Batman", movieName);

        }

}
```

Test Case 29	
Test Definition	
Test case 29 allows us to test the name of the movie that shown in right movie theater 4 according to the customer movie interface.	
Input Value	
theaterID = 4	
Expected Value	**Actual Value**
“Dune”	**“Morbius”**
Result of Test Case	***successful***
Test Script	

```
public class TestCase29 {

        MoviesInterface test = new MoviesInterface();
        @Before
        public void setUp() throws Exception {
        }

        @After
        public void tearDown() throws Exception {
                test = null;
        }

        @Test
        public void negativeTest() {
                int index = 0;
                int theaterID = 4;
                for(int i = 0; i<test.getMovieTheaterNo().size(); i++) {

                        if(Integer.parseInt(test.getMovieTheaterNo().get(i).getText()) == theaterID) {
                  index = i;
                 break;
                        }

                }
                 String movieName = test.getMovieNames().get(index).getText();
                assertNotEquals("Dune", movieName);

        }

}
```

Test Case 30	
Test Definition	
Test case 30 allows us to check if a customer can cancel their tickets properly	
Input Value	
Email = “selen@gmail.com”, movieName = “Eiffel”	
Expected Value	**Actual Value**
false	**false**
Result of Test Case	***successful***
Test Script	

```
public class TestCase30 {
        DatabaseConnection db = new DatabaseConnection();



        @Before
        public void setUp() throws Exception {
                //Customer cancels the ticket before the test
                DatabaseConnection.cancelTicket("selen@gmail.com","Eiffel", 1);


        }

        @After
        public void tearDown() throws Exception {
                //Customer books the same ticket after the test
                DatabaseConnection.TakeSeat(2, 1);
                DatabaseConnection.ReserveTickets("selen@gmail.com", "Eiffel", 1);
        }

        @Test
        public void positiveTest() {
                //Checking if customer could cancel his/her ticket
                assertEquals(DatabaseConnection.getSeats(2).get(0), false);


        }
```

4. CONCLUSION

This test case document indicates that Cine-Ticket application functions properly according to the functional requirements. There are 10 functional requirements that includes capability of what both customer and system owner can do at this application. Total of 30 test case each one is composed of one positive and one negative test are performed. Test cases have been performed on modules such as payment, registration, login, monitoring of reservations and movie lists etc. All 30 test case are gathered under a test suite which allows you to run multiple tests cases at once. Input value, expected value and actual value of each test case with the definition and the result of each tests case is reported at the related tables. As a result, all test cases have been successful.