Group Project Progress Report

Date: 11-Mar-2021
Course Code and Course Title (CRN Number): CEN4072 Software Testing (CRN 10421)

Software Application Title/Name: SWT - Airline Reservation System
Project Manager: Jana Grunewald
Group Members Name:
Jana Grunewald
Jeffry Munoz
Breanna Rhodes
Tamara Vergara
Project Phase (Milestone) Unit Testing
From Report Date: 25-Feb-2021
Report Date: 11-Mar-2021
Complete by (Milestone): 11-Mar-2021

Description of Group Project Milestone

Project Milestone Name: Unit Testing								
Requirement Description	Task ID	Task Description	Pass/Fail/Not Executed	Test Date	Responsible Primary Team Member	Responsible Secondary Team Member	Comment	Additional Comment
The system shall check for valid data entry when adding a new customer	01	Data validation when adding a customer to the database	Pass	6-Mar-2021	Jeffry Munoz	Jana Grunewald	When using invalid characters for the name fields the customer was added	The system displays the successfully added message regardless of what the input is. This issue was observed on all instances where user input was needed
The system shall be able to close pop-up windows	02	Close open dialog boxes after	Fail	8-Mar-2021	Jana Grunewald	Tamara Vergara	The provided menu did	This issue occurred on all

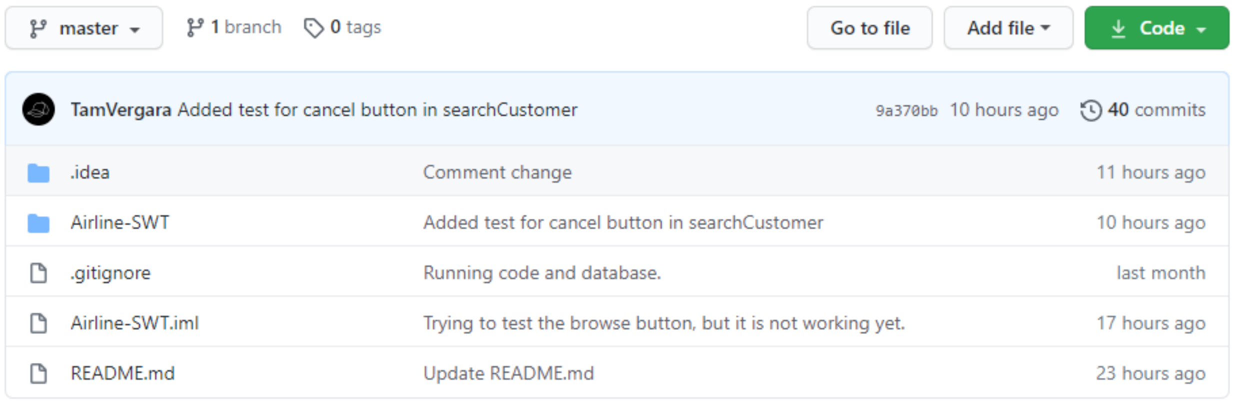
after completing tasks		adding/searching					not allow the user to close the task pop-up window	instances. All tasks opened a new window and none could be closed
Installing/setting up of testing tools (JUnit 5)	03	Add necessary JUnit dependencies and imports	Pass	7-Mar-2021	Breanna Rhodes	Jeffry Munoz	Successfully added JUnit to the project	Breanna helped Tamara and Jana get this set up properly
Running of Test Case number 1	04	Add and Run Test case number 1 to the project code	Pass	7-Mar-2021	Jeffry Munoz	Breanna Rhodes	Successfully added code for test case number 1	Jana created the test case for the cancel button
Create test case 1 for addCustomer class	05	Create the test case according to test plan for the input fields in the addCustomer class	Pass	6-Mar-2021	Jeffry Munoz	Breanna Rhodes	Successfully created parametrized tests for the required input fields	
Create test case number 2 for the addFlight class	06	Create the test case according to test plan for the input fields in the addFlight class	Pass	6-Mar-2021	Breanna Rhodes	Jeffry Munoz	Successfully created tests for the required input fields needed to add a flight to the Database	Jana created the test case for the cancel button

Create test case number 3-5 for the searchCustomer class	07	Create the test case according to the test plan for the input fields in the searchCustomer class	Pass	6-Mar-2021	Tamara Vergara	Jana Grunewald	Created test cases for the add, browse, find and cancel buttons. Successfully passed the negative testing portion while using decision tables.	Jana did the Browse button and Tamara did the find, cancel and add button with help from Jana on the add button.
Create test case for the login class	08	Create the test case according to the test plan for the input fields in the login class to authenticate users in database prior to using the application	Pass	8-Mar-2021	Breanna Rhodes	Jana Grunewald	Created a test case for the login screen using input validation. The username and password are checked via the MySQL database and if the input fails, it shows an error message to the user.	
Create presentation	09	Create the presentation for delivery	Pass	10-Mar-2021	Breanna Rhodes	Jeffry Munoz	We ensured the presenter could run	

		during class					all test cases need for the deliverable	
Create progress report documents	10	Create the progress report and deliverable documentation need to for the second deliverable	Pass	10-Mar-2021	Jeffry Munoz	Breanna Rhodes	Created the needed documentation and screenshots for the second deliverable along	
Finish the Deliverable 2 document	11	Update and finish the deliverable documentation needed for class.	Pass	10-March-2021	Tamara Vergara	Jana Grunewald	Added test cases 3-9 along with an updated screenshot of the test case coverage.	Everyone pitched in to help with the last few documentation edits
Create test case number 8 for the ticket class	12	Create the test case according to the test plan for the input fields in the ticket class	Pass	10-March-2021	Tamara Vergara	Jana Grunewald	Created a test case to determine if inputting letters into the price textfield produces an SQLException.	Jana created the test case for the cancel button
Create test case number for the ticketreport class	13	Create the test case according to the test plan for the display for the	Pass	10-March-2021	Jana Grunewald	Tamara Vergara	Created a test case to test the cancel button. The test for the	

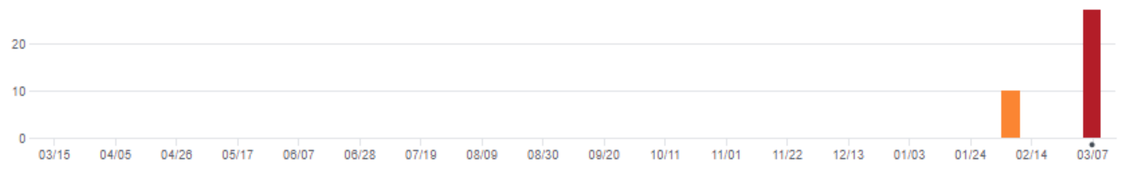
		ticketreport class					display is from the ticket class.	
Create test case number for the userCreation class	14	Create the test case according to the test plan for the input fields in the userCreation class	Pass	10-March-2021	Tamara Vergara	Jana Grunewald	Created a test case to determine if inputting an invalid input into the textfields produces an SQLException. It also tested if the cancel button was able to actually hide the userCreation page	Jana created the test case for the cancel button

Group Project Milestone Screenshot

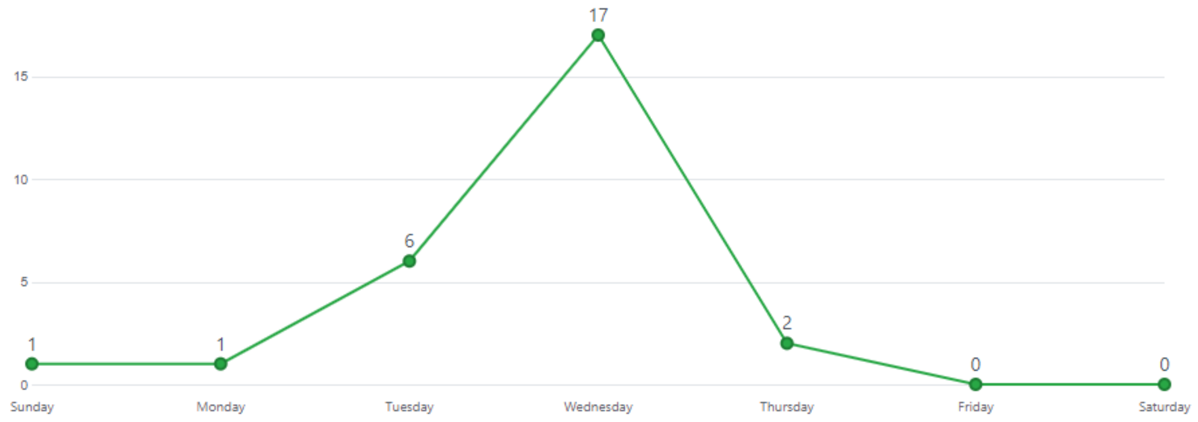


TamVergara Added test for cancel button in searchCustomer 9a370bb 10 hours ago 40 commits

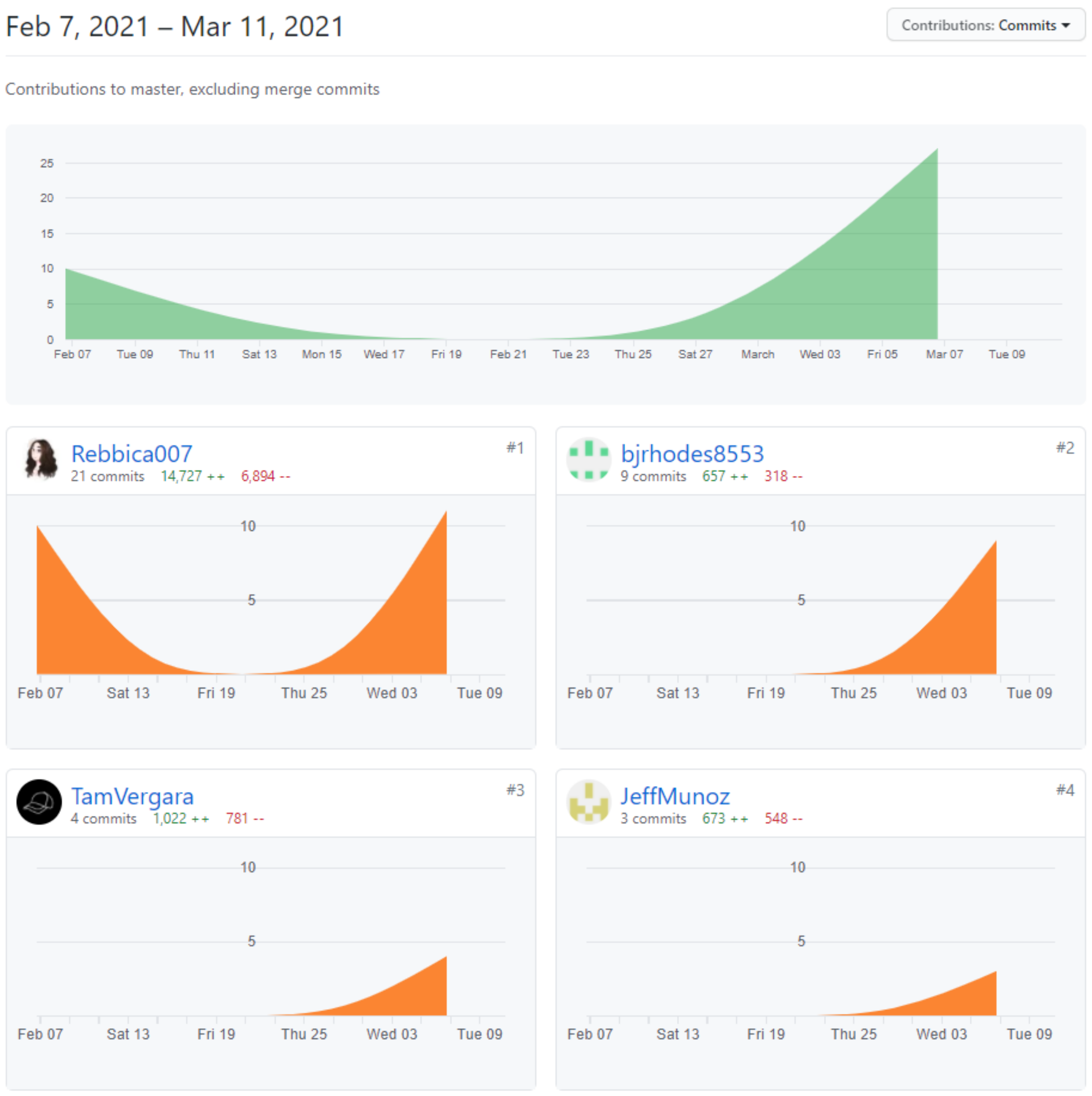
.idea	Comment change	11 hours ago
Airline-SWT	Added test for cancel button in searchCustomer	10 hours ago
.gitignore	Running code and database.	last month
Airline-SWT.iml	Trying to test the browse button, but it is not working yet.	17 hours ago
README.md	Update README.md	23 hours ago



20
10
0
03/15
04/05
04/26
05/17
06/07
06/28
07/19
08/09
08/30
09/20
10/11
11/01
11/22
12/13
01/03
01/24
02/14
03/07



15
10
5
0
1
1
6
17
2
0
0
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday



Group Project Meeting Minutes
Meeting Date: 2/26/21
Meeting Length: 47 minutes
Team Members present:
Jana Grunewald
Jeffry Munoz
Breanna Rhodes
Tamara Vergara
Summary: During this meeting, tasks were assigned for the completion of the second deliverable. It was decided that Jeffry, Breanna, and Tamara would each write a test case to match the example while Janna

added the needed tools. Breanna questioned the number of test cases needed. The others agreed to reach out for clarification the following week.

Meeting Date: 3/05/21
Meeting Length: 35 Minutes
Team Members present:
Jana Grunewald
Jeffry Munoz
Breanna Rhodes
Tamara Vergara
Summary: During this meeting, we discussed the project progress report. Tamara suggested that we split up the report evenly amongst the group members. Meanwhile, Janna suggested that each team member review one another's test case to ensure that it was clear and concise. Jeffry suggested that everyone run the test case code to ensure that the results could be replicated. Breanna mentioned the need for the presentation and who should prepare it and present it on the due date.

Additional Important Requirements:

1. Submit one (1) Microsoft Word Document or PDF on Canvas
    a. **Do <u>not</u> submit a zipped file (-10 points)**
2. Your progress report must include all information specified and the subsections/subcategories included in this template
    a. **<u>Missing or incorrect information will</u> result in a 10-point deduction from the overall project grade (-10 points)**
3. Progress Report File Naming Convention
    a. **Course Code-CRN-ProgressReport-ProjectPhaseName-Group#**
        i. e.g. CEN4072-10421-ProgressReport-UnitTestingPhase-Group7

Deliverable 2

Testing Plan

Test Case # 1
Black Box Testing: Equivalence Classes
Testing Type = Input Validation Testing

Testing Category : Requirement TypeFunctional	
Test Criteria#1	
Requirement# FUN-1:	The application shall allow users to add a unique customer record to the database by checking for valid input
Input:	● The user must enter a string for the following fields ○ First Name ■ Specifically, the user enters a string with only alphabetic characters (a-z, A-Z) such as Brenda ○ Last Name ■ Same as first name such as Rhodes ○ Address ■ Specifically, the user enters a string with only alphanumeric characters (a-z, A-Z, 0-9) such as 10501 FGCU Blvd S ○ Contact ■ A phone number only containing numeric characters (0-9) such as 2395901000 ● The user must provide a number for the following ○ Nic no ■ A number only containing alphanumeric characters (a-z, A-Z, 0-9) ○ PassportID ■ Same as Nic no Actual input listed in the parametrized collection bellow

Test method or procedure description:	For this Test equivalence classes were utilized to divide the valid strings for names and address with ones with invalid characters. The same method was used to differentiate between valid integers for the Nic No. and PassportID fields. This ensures that only the valid inputs are passed on to the database to create new customers.

Test Method/Procedure (Code)	
	```
	@Test

	    //First test case will test the flight name being entered into the field.

	    public void testcase1() throws InterruptedException {

	        //Regex pattern made to reflect the inner boundaries of the string needed

	        Pattern pattern = Pattern.compile("^[a-zA-Z0-9]+$");

	        addCustomer frame = new addCustomer();

	        frame.setVisible(true);

	        //Created public getters in the addFlight class to access the components

	        JTextField firstNameTest = (JTextField) addCustomer.getTxtfirstname();

	        JTextField lastNameTest = (JTextField) addCustomer.getTxtlastname();

	        JTextField passPortIdTest = (JTextField) addCustomer.getTxtpassport();

	        JTextField nicTest = (JTextField) addCustomer.getTxtnic();

	        JTextArea addressTest = (JTextArea) addCustomer.getTxtaddress();

	        JTextField contactTest = (JTextField) addCustomer.getTxtcontact();

	        //This sets the string to the correct sting in the list

	        firstNameTest.setText(customerFirstName);
	```

```
        lastNameTest.setText(customerLastName);

        passPortIdTest.setText(passportId);

        contactTest.setText(contact);

        nicTest.setText(nic);

        addressTest.setText(address);

        //Allow the program to accept the text set to
the component

        sleep(2000);

        firstNameTest.postActionEvent();

        lastNameTest.postActionEvent();

        passPortIdTest.postActionEvent();

        contactTest.postActionEvent();

        nicTest.postActionEvent();

        //Test the Regex against the test case to see
if its within the boundaries.

        Matcher match =
pattern.matcher(firstNameTest.getText());

        boolean firstNameResult = match.find();

        Assert.assertTrue(expectedResult);

    }

@Parameterized.Parameters

    public static Collection inputStrings() {

        return Arrays.asList(new Object[][] {

                { "Brenda" ,"Rhodes","123s5",
"2395901000", "4072qwer","10501 FGCU Blvd S" , true
},

                { "%^&Bob","Builder","123s5",
"2395901000", "4072qwer","10501 FGCU Blvd S" , false
},

                { "Brenda", "Rhodes","^5",
"2395901000", "4072qwer","10501 FGCU Blvd S" , false
},
```

	```
	{ "Brenda", "Rhodes","545", "2395901&000", "4072qwer","10501 FGCU Blvd S" , false},

	{ "Brenda", "Rhodes","5", "2395901000", "4072!qwer","10501 FGCU Blvd S" , false},

	{"Brenda", "Rhodes","^5", "2395901000", "4072qwer","10501 FG*CU Blvd S" , false},

	{"", "","", "", "","", false},

	});

	}

	@Test

	public void testButton3() {

	addCustomer create = new addCustomer();

	create.setVisible(true);

	create.jButton3.doClick();

	Boolean isHidden = create.getVisibility();

	System.out.println("Button3 test should pass.");

	Assert.assertTrue(isHidden);

	}
	```
Expected Output	This should show the success message which indicates that the customer was added to the database only when valid characters are used
Test Result (Actual Output)	Success message displayed
Discrepancies	All customers were added to the database even when invalid characters were used for the indicated fields

Dependencies	N/A
Branch Coverage (%)	96%
Statement Coverage (%)	100%

Test Case # 2
Black Box Testing: Boundary Value Analysis
Testing Type = Positive Testing

Testing Category : Requirement Type: Functional	
Test Criteria#2	
Requirement# FUN-5:	The user shall be able to add a unique flight.
Input:	<ul><li>The user must enter a string for the following fields<ul><li>Flight Name<ul><li>Specifically, the user enters a string with alphabetic characters (a-z, A-Z), as long as it does not exceed 255 characters in length such as ‘JetBlue’</li></ul></li><li>Dep Time<ul><li>Specifically, the user enters a string leading with integers followed by the period (.) character or the colon character (:) with the option of ‘AM’ or ‘PM’ on the end of the string as</li></ul></li></ul></li></ul>

	long as the string does not exceed 255 characters, in the format hour:minutes, such as ‘8.00AM’ or ‘11:00’. ○ Arr Time ■ Specifically, the user enters a string leading with integers followed by the period (.) character or the colon character (:) with the option of ‘AM’ or ‘PM’ on the end of the string OR can be a single digit as long as the string does not exceed 255 characters, in the format hour:minutes, such as ‘8.00AM’ or ‘8’. ○ Flight Charge ■ Specifically, the user enters a string of integers as long as the string does not exceed 255 characters. ● The user must select from a drop down menu for the following: ○ Source ■ Specifically, from one of the following choices: ‘India’, ‘Srilanka’, ‘Uk’, ‘Usa’, ‘Canada’, ‘Chinna’. ○ Departure ■ Specifically, from one of the following choices: ‘India’, ‘Srilanka’, ‘Uk’, ‘Usa’, ‘Canada’, ‘Chinna’. ● The user must choose from a date picker for the following: ○ Date ■ Specifically the year, month and date of flight.
Test method or procedure description:	For the Boundary Test Analysis, classes were developed based on the lower and upper boundaries. Because the focus is positive testing, only test cases that produce a positive result will be used. The test cases that produce a positive result will be the test cases within the upper and lower boundaries.

Test Method/Procedure (Code)	`@Test`
	`/*`
	`This test was constructed to test the input of the user as it is entered into the Swing GUI for creating a new flight, and stores the input into the flight table in the database by clicking the button.`
	`*/`
	`    public void testUserInput() throws InterruptedException {`
	`        //Make the addFlight frame visible for the test`
	`        addflight frame = new addflight();`

```
        frame.setVisible(true);



//Regex pattern made to reflect the inner boundaries of the string
needed.

        Pattern namePattern = Pattern.compile("[A-Za-z]");

        Pattern datePattern =
Pattern.compile("^([0-9]{4}[-/]?((0[13-9]|1[012])[-/]?(0[1-9]|[12][0-9
]|30)|(0[13578]
|1[02])[-/]?31|02[-/]?(0[1-9]|1[0-9]|2[0-8]))|([0-9]{2}(([2468][048]|[
02468][48])|[13579][26])|([13579][26]|[02468][048]|0[0-9]|1[0-6])00)[-
/]?02[-/]?29)$");

        Pattern timePattern =
Pattern.compile("([01]?[0-9]|2[0-3]):[0-5][0-9](.*AM|PM.*)");

        Pattern chargePattern = Pattern.compile("[0-9]+");



//A String Array of the elements in the combo box is made to
initialize the boundary.

        String[] expResult = {"India", "Srilanka", "Uk", "Usa",
"Canada", "Chinna"};



//Created public getters in the addFlight class to access the Swing
components.

        JTextField nameTest = (JTextField) addflight.getTxtName();

        JComboBox<String> sourceTest =
(JComboBox<String>)addflight.getTxtsource();

        JComboBox<String> departTest =
(JComboBox<String>)addflight.getTxtdepart();

         JDateChooser dateTest = (JDateChooser)addflight.getTxtdate();

        JTextField departTimeTest =
(JTextField)addflight.getTxtdtime();

        JTextField arrTimeTest =
(JTextField)addflight.getTxtarrtime();

         JTextField chargeTest =
(JTextField)addflight.getTxtflightcharge();
```

```
//Initialize the booleans that will be used to test the Source and
Departure combo boxes.

        boolean testSourceContains = false;

        boolean testDepartContains = false;


//A loop iterates through the array and adds each item to the combo
box selection component.

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

            sourceTest.setSelectedItem(expResult[i]);

            departTest.setSelectedItem(expResult[i]);

            sleep(200);

//Once the component selects the item, it is tested to see if it exist
in the combo box.

            testSourceContains =
expResult[i].equals(sourceTest.getSelectedItem());

            testDepartContains =
expResult[i].equals(departTest.getSelectedItem());

//If the item is in the combobox then the selection made by the user
was within bounds.

        }

/*

Because a Swing GUI cannot be accessed during testing, the component
is set to the test value manually.

The test value for the name of the flight is within the Regex value
and is expected to pass.

*/

        nameTest.setText("yellowSky");


//The test time being set is within the regex bounds for time, and is
expected to pass
```

```
        departTimeTest.setText("11:00AM");

        arrTimeTest.setText("08:33PM");

//The test date selected for the flight is the current date.

        dateTest.setCalendar(Calendar.getInstance());


//The test charge for the flight is within regex bounds and is
expected to pass.

        chargeTest.setText("14000");


//After the test value is set, the program sleeps so it can accept the
value before moving on.

        sleep(2000);

        nameTest.postActionEvent();

        departTimeTest.postActionEvent();

        arrTimeTest.postActionEvent();

        chargeTest.postActionEvent();


//The date needs to be formatted the same way as the database has
dates stored.

        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

        String Date =
dateFormat.format(((JDateChooser)addflight.getTxtdate()).getDate());


//Matcher objects are created for each of the fields that have Regex
boundaries.

        Matcher nameMatch = namePattern.matcher(nameTest.getText());

        Matcher dateMatch = datePattern.matcher(Date);

        Matcher departTimeMatch =
timePattern.matcher(departTimeTest.getText());

        Matcher arrTimeMatch =
timePattern.matcher(arrTimeTest.getText());
```

```
        Matcher chargeMatch =
chargePattern.matcher(chargeTest.getText());


//The results of the test inputs against the Regex patterns are
stored.

        boolean nameResult = nameMatch.find();

        boolean departTimeResult = departTimeMatch.find();

        boolean dateResult = dateMatch.find();

        boolean arrTimeResult = arrTimeMatch.find();

        boolean chargeResult = chargeMatch.find();


//A boolean array holds all the results of the test inputs.

        boolean[] testResults = {

                nameResult,

                testSourceContains,

                testDepartContains,

                dateResult,

                departTimeResult,

                arrTimeResult,

                chargeResult};


        boolean allTestPassed = false;

/*

Each test input is tested for its validity. If a test input is false,
then the loop will break and the test will fail.

*/

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

            if (testResults[i] == true) {

                allTestPassed = true;

            } else {
                allTestPassed = false;

                break;

            }

        }

//Assert determines the validity of the test.

        Assert.assertTrue(allTestPassed);


//This button's action stores the newly created flight into the
database.

        addflight.jButton1.doClick();

    }

   @Test

       public void testButton2() {

           // Create instance of the searchCustomer class

           addflight details = new addflight();

           details.setVisible(true);

           details.jButton2.doClick();

           Boolean isHidden =details.getVisibility();

           System.out.println("Button2 test should pass.");

           Assert.assertTrue(isHidden);

      }
```

Expected Output	When the user enters their information into the text fields, a success message should populate if the user enters information within the boundaries. A button will be pressed after the flight information is added, and the information will be added to the database. If all the information is correctly entered, a pop-up will appear stating the flight was created.

Test Result (Actual Output)	All test cases within boundaries were successful, and the flight was successfully added to the database.
Discrepancies	No discrepancies were discovered, all test case inputs were expected to pass.
Dependencies	N/A
Branch Coverage (%)	95%
Statement Coverage (%)	100%

Test Case # 3
Black Box Testing: Decision Tables
Testing Type = Negative Testing

Testing Category : Requirement Type: Functional	
Test Criteria#3	
Requirement# FUN-19:	The application shall allow users to upload images to their profile.
Input:	• The user must choose a photo from a file chooser for the following: ○ Photo

	■ Specifically, the user must choose an invalid photo extension.

Test method or procedure description:	We used a decision table to determine test case 3. If we included the entire table it would take up a lot of space, so we're only going to include the rules that produced exceptions, and rules that we expected to throw exceptions, but didn't.
	
	L = letters (eg. ABC)
	
	N = numbers (eg. 123)
	
	S = special characters (eg. ♥)
	
	T = selected
	
	F = unselected / empty
	
	Pc = correct path
	
	Pi = incorrect path

CONDITIONS	RULE 1	RULE 2	RULE 3	RULE 4	RULE 5	RULE 6
Find Text	S	F	F	F	F	F
First Name	L	L	L	L	N	F
Last Name	L	L	L	L	N	F
Nic No	N	N	N	N	L	F
Passport ID	N	N	N	N	L	F
Address	L	L	L	L	N	F
Male	T	T	T	T	T	F
Female	F	F	F	T	F	F
Contact	N	L	N	N	N	F
Image Path	Pc	Pc	Pi	Pc	Pc	F
Button3 (Update)	F	T	T	T	T	T
Button 4 (Find)	T	F	F	F	F	F

ACTIONS						
Update Registration				T	T	T
SQLException	T					
ParseException		T				
IOException			T			

Test Method/Procedure (Code)

```
 @Test

  /* This test was constructed to test if the customer was valid, then

  * testing if the image file exists. The customer must be valid in
order

  * for this test to work and the path must not exist since this is
being

  * tested using the negative testing technique.

  */

  public void testButton1IOException() throws InterruptedException {


    // Create instance of the searchCustomer class

    searchCustomer search = new searchCustomer();

    search.setVisible(true);

    // Make sure the customer is valid, if it is empty, make it a
valid customer.

    JTextField findText = (JTextField) search.getFindText();

    if (findText.getVisibleRect().isEmpty()){

      findText.setText("CS001");

    }

    else {

      findText.getText();
```

	`    }`
	
	`    // Access the label used to display image`
	
	`    JLabel photoText = (JLabel) search.getPhotoText();`
	
	`    // Enter an invalid input (no image path) to trigger IOException`
	
	`    // when the popup screen shows, enter photo.png in the file name then click okay.`
	
	`    // this forms an IOException`
	
	`    photoText.setText("C:/Owner/Desktop/photo.png");`
	
	`    // Give program 2 seconds to accept text`
	
	`    sleep(2000);`
	
	`    // Click button1 (Browse) to attempt a search with the invalid input`
	
	`    searchCustomer.jButton1.doClick();`
	
	`    // In searchCustomer IOException catch, button1IsIOExceptionThrown set to true`
	
	`    // This was work around since I wasn't able to get`
	
	`    // jButton1ActionPerformed to throw an exception for some reason`
	
	`    // If invalid input, and SQL not thrown, test fails`
	
	`    // If invalid input, and SQL thrown, test passes`
	
	`    Assert.assertTrue(searchCustomer.button1IsIOExceptionThrown);`
	
	`    System.out.println("Button 1 IO exception was caught");`
	
	`  }`

Expected Output	The user enters an invalid photo path which should return an IO Exception.
Test Result (Actual Output)	All test cases were successful, and nothing was changed in the database.

Discrepancies	There were no discrepancies since the test cases were meant to pass by putting incorrect input into the fields.
Dependencies	<ul><li>Button 1 is dependent on Button 4 since to change a photo, the user must already exist.</li><li>MySQL Database</li></ul>
Branch Coverage (%)	70%
Statement Coverage (%)	82%

Test Case # 4
Black Box Testing: Decision Tables
Testing Type = Negative Testing

Testing Category : Requirement Type: Functional	
Test Criteria#4	
Requirement# FUN-7:	The application shall allow users to update their profile.
Input:	<ul><li>The user must enter an invalid String for the following:<ul><li>contactText<ul><li>Specifically, the user enters a string of letters (i.e. ABC)</li></ul></li></ul></li></ul>

Test method or procedure description:	We used a decision table to determine test case 4. If we included the entire table it would take up a lot of space, so we're only going to include the rules that produced exceptions, and rules that we expected to throw exceptions, but didn't.
	L = letters (eg. ABC)
	N = numbers (eg. 123)
	S = special characters (eg. ♥)
	T = selected
	F = unselected / empty
	Pc = correct path
	Pi = incorrect path

CONDITIONS	RULE 1	RULE 2	RULE 3	RULE 4	RULE 5	RULE 6
Find Text	S	F	F	F	F	F
First Name	L	L	L	L	N	F
Last Name	L	L	L	L	N	F
Nic No	N	N	N	N	L	F
Passport ID	N	N	N	N	L	F
Address	L	L	L	L	N	F
Male	T	T	T	T	T	F
Female	F	F	F	T	F	F
Contact	N	L	N	N	N	F
Image Path	Pc	Pc	Pi	Pc	Pc	F
Button3 (Update)	F	T	T	T	T	T
Button 4 (Find)	T	F	F	F	F	F
ACTIONS						
Update				T	T	T

Registration						
SQLException	T					
ParseException		T				
IOException			T			

Test Method/Procedure (Code)

```
  @Test
  /* This test was constructed to test if the customer was valid, then
   * testing if the contact text can be updated using letters.
   * The customer must be valid in order for this test to work. The
customer
   * contact text will not update if it is letters.
   */
  public void testButton2SQLException() throws InterruptedException {
    // Create instance of the searchCustomer class
    searchCustomer search = new searchCustomer();
    search.setVisible(true);
    JTextField findText = (JTextField) search.getFindText();
    if (findText.getVisibleRect().isEmpty()){
      findText.setText("CS001");
    }
    else {
      findText.getText();
    }

    // Access the textfield used to search
    JTextField contactText = (JTextField) search.getContactText();
    // Enter an invalid input (letters) to trigger SQLException
```

	`    contactText.setText("ABC");`
	
	`    // Give program 2 seconds to accept text`
	
	`    sleep(2000);`
	
	`    // Click button2 (Update) to attempt a search with the invalid input`
	
	`    searchCustomer.jButton2.doClick();`
	
	`    // In searchCustomer SQLException catch, button2IsSQLThrown set to true`
	
	`    // This was work around since I wasn't able to get`
	
	`    // jButton2ActionPerformed to throw an exception for some reason`
	
	`    // If invalid input, and SQL not thrown, test fails`
	
	`    // If invalid input, and SQL thrown, test passes`
	
	`    Assert.assertTrue(searchCustomer.button2IsSQLThrown);`
	
	`    System.out.println("Button2 SQLException test should pass.");`
	
	`  }`

Expected Output	The user enters alphabet characters (a-z, A-Z) into the contactText field which should return a SQL Exception.
Test Result (Actual Output)	All test cases were successful, and nothing was changed in the database.
Discrepancies	There were no discrepancies since the test cases were meant to pass by putting incorrect input into the fields.
Dependencies	● Button 2 is dependent on Button 4 since to update a profile, the user must already exist. ● MySQL Database

Branch Coverage (%)	70%
Statement Coverage (%)	82%

Test Case # 5
Black Box Testing: Decision Tables
Testing Type = Negative Testing

Testing Category : Requirement Type: Functional	
Test Criteria#5	
Requirement# FUN-3:	The application shall allow users to search for a customer stored in the database using their customer ID.
Input:	● The user must enter an invalid String for the following: ○ Customer ID ■ Specifically, the user enters a string with ♥ as its input.

Test method or procedure description:	We used a decision table to determine test case 5. If we included the entire table it would take up a lot of space, so we're only going to include the rules that produced exceptions, and rules that we expected to throw exceptions, but didn't.
	
	L = letters (eg. ABC)
	
	N = numbers (eg. 123)

S = special characters (eg. ♥)

T = selected

F = unselected / empty

Pc = correct path

Pi = incorrect path

CONDITIONS	RULE 1	RULE 2	RULE 3	RULE 4	RULE 5	RULE 6
Find Text	S	F	F	F	F	F
First Name	L	L	L	L	N	F
Last Name	L	L	L	L	N	F
Nic No	N	N	N	N	L	F
Passport ID	N	N	N	N	L	F
Address	L	L	L	L	N	F
Male	T	T	T	T	T	F
Female	F	F	F	T	F	F
Contact	N	L	N	N	N	F
Image Path	Pc	Pc	Pi	Pc	Pc	F
Button3 (Update)	F	T	T	T	T	T
Button 4 (Find)	T	F	F	F	F	F
ACTIONS						
Update Registration				T	T	T
SQLException	T					
ParseException		T				
IOException			T			

Test Method/Procedure (Code)

```
 @Test

  /* This test was constructed to test if the customer was valid, then

   * testing if the customer exists. The customer file is 2 letters followed

   * by 3 numbers.

   */

  public void testButton4SQLException() throws InterruptedException {

    // Create instance of the searchCustomer class

    searchCustomer search = new searchCustomer();

    search.setVisible(true);

    // Access the textfield used to search

    JTextField findText = (JTextField) search.getFindText();

    // Enter an invalid input (a special character) to trigger SQLException

    findText.setText("♥");

    // Give program 2 seconds to accept text

    sleep(2000);

    // Click button4 (Find) to attempt a search with the invalid input

    searchCustomer.jButton4.doClick();

    // In searchCustomer SQLException catch, button4IsSQLThrown set to true

    // This was work around since I wasn't able to get

    // jButton4ActionPerformed to throw an exception for some reason

    // If invalid input, and SQL not thrown, test fails

    // If invalid input, and SQL thrown, test passes

    Assert.assertTrue(searchCustomer.button4IsSQLThrown);

    System.out.println("Button4 SQLException test should pass.");

  }
```

	`@Test`
	`  public void testButton3() {`
	`    searchCustomer search = new searchCustomer();`
	`    search.setVisible(true);`
	`    search.jButton3.doClick();`
	`    Boolean isHidden = search.getVisibility();`
	`    System.out.println("Button1 test should pass");`
	`    Assert.assertTrue(isHidden);`
	`  }`

Expected Output	The user enters an invalid character (i.e., ♥) for the customerId which should return a SQL Exception.
Test Result (Actual Output)	All test cases were successful, and nothing was changed in the database.
Discrepancies	There were no discrepancies since the test cases were meant to pass by putting incorrect input into the fields.
Dependencies	● MySQL Database
Branch Coverage (%)	70%
Statement Coverage (%)	82%

Test Case # 6

Black Box Testing: Equivalence Classes
Testing Type = Positive Testing

Testing Category : Requirement Type: Functional	
Test Criteria#6	
Requirement# FUN-4:	The application shall allow users to view a table of all tickets.
Input:	● The user must select from a drop down menu for: ○ Source ■ Specifically, the user must choose to fly to "India," "Srilanka," "Uk," "Usa," "Canada," or "Chinna" ○ Departure ■ Specifically, the user must choose to fly from "India," "Srilanka," "Uk," "Usa," "Canada," or "Chinna" ○ Class ■ Specifically, the user must choose between "Economy" and "Business." ● The user must enter a string for: ○ Customer ID ■ Specifically, the user enters a string with 2 alphabetic characters (a-z, A-Z) and 3 digits (0-9) as long as it does not exceed 255 characters. ○ Price ■ Specifically, the user must enter a string with digits (0-9) as long as it does not exceed 255 characters. ○ Seats ■ Specifically, the user must enter a string with digits (0-9) as long as it does not exceed 255 characters.
Test method or procedure description:	For this test, input validation testing was used by completing the "book flight" section first, then opening the "flight report." The information was split into the correct categories and is displayed with the information filled out on the "book

	flight" page. Tested the cancel button to make sure the display was hidden via mouse click.

Test Method/Procedure (Code)	`@Test`
	`  public void testButton1() {`
	`   // Create instance of the ticketreport class`

	```
	        ticketreport report = new ticketreport();

	        report.setVisible(true);

	        report.jButton1.doClick();

	        Boolean isHidden =report.getVisibility();

	        System.out.println("Button1 test should pass.");

	        Assert.assertTrue(isHidden);


	    }
	```
Expected Output	The table should fill out with what the user inputs.
Test Result (Actual Output)	The test cases passed. The table displays the information the user added from the “book flight” section.
Discrepancies	If something is not filled out, it fills out the table with the name of that field (i.e., jLabel18).
Dependencies	● ticket.java ● MySQL Database
Branch Coverage (%)	100%
Statement Coverage (%)	94%

Test Case # 7
Black Box Testing: Equivalence Classes
Testing Type = Negative Testing

Testing Category : Requirement Type: Functional	
Test Criteria#7	
Requirement# FUN-2:	The application shall allow users to add a unique user record to the database.
Input:	<ul><li>The user must enter a string for:<ul><li>FirstName<ul><li>Specifically, the user enters a string with alphabetic characters (a-z, A-Z) as long as it does not exceed 255 characters.</li></ul></li><li>LastName<ul><li>Specifically, the user enters a string with alphabetic characters (a-z, A-Z) as long as it does not exceed 255 characters.</li></ul></li><li>User Name<ul><li>Specifically, the user enters a string with alphabetic characters (a-z, A-Z) as long as it does not exceed 255 characters.</li></ul></li><li>Password<ul><li>Specifically, the user enters a string with alphabetic characters (a-z, A-Z) as long as it does not exceed 255 characters.</li></ul></li></ul></li></ul>
Test method or procedure description:	For this test, negative testing was used by filling out each area and clicking add. Tested the cancel button to make sure the display was hidden via mouse click.

`Test Method/Procedure (Code)`	`// Testing SQLException for Button 1 (Add)`
	
	`@Test`

```
  public void testButton1SQLException() throws InterruptedException
{

   // Create instance of the userCreation class

   userCreation creation = new userCreation();

   creation.setVisible(true);

   // Access the textfield used to fill the first name

   // It doesn't have to be the first name, I am just using it for
this test

   JTextField fNameText = (JTextField) creation.getFNameText();

   // Enter an invalid input (a special character) to trigger
SQLException

   fNameText.setText("♥");

   // Give program 2 seconds to accept text

   sleep(2000);

   // Click button1 (Add) to attempt to add a user with the invalid
input

   creation.jButton1.doClick();

   // In userCreation SQLException catch, button1IsSQLThrown set to
true

   // This was work around since I wasn't able to get

   // jButton1ActionPerformed to throw an exception for some reason

   // If invalid input, and SQL not thrown, test fails

   // If invalid input, and SQL thrown, test passes

    Assert.assertTrue(creation.button1IsSQLThrown);

   System.out.println("Button1 SQLException test should pass.");

  }

   @Test

  public void testButton2() {
```

	`// Create instance of the searchCustomer class`
	`userCreation cancel = new userCreation();`
	`cancel.setVisible(true);`
	`cancel.jButton2.doClick();`
	`Boolean isHidden =cancel.getVisibility();`
	`System.out.println("Button2 test should pass.");`
	`Assert.assertTrue(isHidden);`
	`}`

Expected Output	The user will be added to the database when the “Add” button is pressed.
Test Result (Actual Output)	The test cases passed. The user is added to the database after the “Add” button is pressed.
Discrepancies	All the fields do not need to be filled out for the user to be added to the database.
Dependencies	N/A
Branch Coverage (%)	100%
Statement Coverage (%)	94%

Test Case # 8
Black Box Testing: Equivalence Classes

Testing Type = Negative/Positive Testing

Testing Category : Requirement Type: Functional	
Test Criteria#8	
Requirement# FUN-12:	The application shall allow users to book a ticket based on price
Input:	<ul><li>The user must select from a drop down menu for:<ul><li>Source<ul><li>Specifically, the user must choose to fly to “India,” “Srilanka,” “Uk,” “Usa,” “Canada,” or “Chinna”</li></ul></li><li>Departure<ul><li>Specifically, the user must choose to fly from “India,” “Srilanka,” “Uk,” “Usa,” “Canada,” or “Chinna”</li></ul></li><li>Class<ul><li>Specifically, the user must choose between “Economy” and “Business.”</li></ul></li></ul></li><li>The user must enter a string for:<ul><li>Customer ID<ul><li>Specifically, the user enters a string with 2 alphabetic characters (a-z, A-Z) and 3 digits (0-9) as long as it does not exceed 255 characters.</li></ul></li><li>Price<ul><li>Specifically, the user must enter a string with digits (0-9) as long as it does not exceed 255 characters.</li></ul></li><li>Seats<ul><li>Specifically, the user must enter a string with digits (0-9) as long as it does not exceed 255 characters.</li></ul></li></ul></li></ul>
Test method or procedure description:	For this test, negative and positive testing were used by putting a correct character in the “Customer Id” field and putting incorrect alphabetical strings in for the “Price” field. Tested the cancel button to make sure the display was hidden via mouse click.

Test Method/Procedure (Code)	
	```
	// Tests SQLException for Button 1 (Book)

	  @Test

	  public void testButton1SQLException() throws InterruptedException
	{

	   // Create instance of the ticket class

	   ticket myTicket = new ticket();

	   myTicket.setVisible(true);

	   // Access the textfield used to fill in the price

	   JTextField priceText = (JTextField) myTicket.getPriceText();

	   // Enter an invalid input (ABC) to trigger SQLException

	   priceText.setText("ABC");

	   // Give program 2 seconds to accept text

	    sleep(2000);

	   // Click button1 (Book) to attempt to book the flight with the
	invalid input

	   myTicket.jButton1.doClick();

	   // In ticket SQLException catch, button1IsSQLThrown set to true

	   // This was work around since I wasn't able to get

	   // jButton1ActionPerformed to throw an exception for some reason

	   // If invalid input, and SQL not thrown, test fails

	   // If invalid input, and SQL thrown, test passes

	    Assert.assertTrue(myTicket.button1IsSQLThrown);

	   System.out.println("Button1 SQLException test should pass.");

	  }



	// Testing invalid values entered into Customer ID
	```

```
   @Test

   public void testCustomerIDInput() throws InterruptedException {

    // Acceptable values

    Pattern pattern = Pattern.compile("[A-Za-z]");

    ticket myTicket = new ticket();

    myTicket.setVisible(true);

    // Get the textfield

    JTextField customerIDText = (JTextField)
 myTicket.getCustomerIDText();

    // Give textfield invalid value

    customerIDText.setText("♥");

    // Give it 2 seconds to accept the value

    sleep(2000);

    customerIDText.postActionEvent();

    // Look for the invalid value within the list of valid values

    Matcher match = pattern.matcher(customerIDText.getText());

    // If it fails to find it, then the test passes

    boolean result = match.find();

    Assert.assertTrue(!result);

   }
```

Expected Output	The program will throw an error if an incorrect character is given for customer id and will throw an error if price is a non-numerical number.
Test Result (Actual Output)	The test cases passed. Incorrect values were spotted and threw exceptions.

Discrepancies	Flight gets added without all the information filled in.
Dependencies	N/A
Branch Coverage (%)	66%
Statement Coverage (%)	81%

Test Case # 9
Black Box Testing: Equivalence Classes
Testing Type = Input Validation Testing

Testing Category : Requirement Type: Functional	
Test Criteria#9	
Requirement# FUN-6:	The application shall allow users to log into the application using their unique username and password.
Input:	● The user must enter a string for: ○ Username ■ Specifically, the user enters a string with alphabetic characters (a-z, A-Z) and/or digits (0-9) as long as it does not exceed 255 characters. ○ Password

	■ Specifically, the user enters a string with alphabetic characters (a-z, A-Z) and/or digits (0-9) as long as it does not exceed 255 characters.
Test method or procedure description:	Input validation testing was used by adding a static username and password. This allowed us to test the user input and allow the user to login or display a message saying the username/password is incorrect.

Test Method/Procedure (Code)	`@Test`
	`public void testLogin() throws InterruptedException {`
	`Login frame = new Login();`
	`frame.setVisible(true);`
	
	`JTextField username = Login.getUsername();`
	`JTextField password =  Login.getPassword();`
	
	`username.setText("usernameTest");`
	`password.setText("passwordTest");`
	`sleep(2000);`

```
         username.postActionEvent();

        password.postActionEvent();



        Pattern namePattern = Pattern.compile("[A-Za-z]");

        Matcher usernameMatch =
namePattern.matcher(username.getText());

        Matcher passwordMatch =
namePattern.matcher(password.getText());



        boolean nameResult = usernameMatch.find();

        boolean passResult = passwordMatch.find();

        boolean allResults = false;



        if(nameResult == true && passResult == true){

            allResults = true;

         }

         else{allResults = false;}




        Assert.assertTrue(allResults);



        Login.jButton1.doClick();

        Login.jButton2.doClick();

   }



   @Test
```

```
public void testMain() {

      Login frame = new Login();

     frame.setVisible(true);

      if (frame.isActive() == true) {

         Assert.assertTrue(true);

      } else {

         Assert.assertFalse(false);

      }

}


@Test

public void testGetUsername(){

         JTextField user = Login.getUsername();

         user.setText("user");

         Assert.assertEquals("user", user.getText());


      }






@Test

public void testGetPassword() {

     JTextField pass = Login.getPassword();

     pass.setText("pass");
```

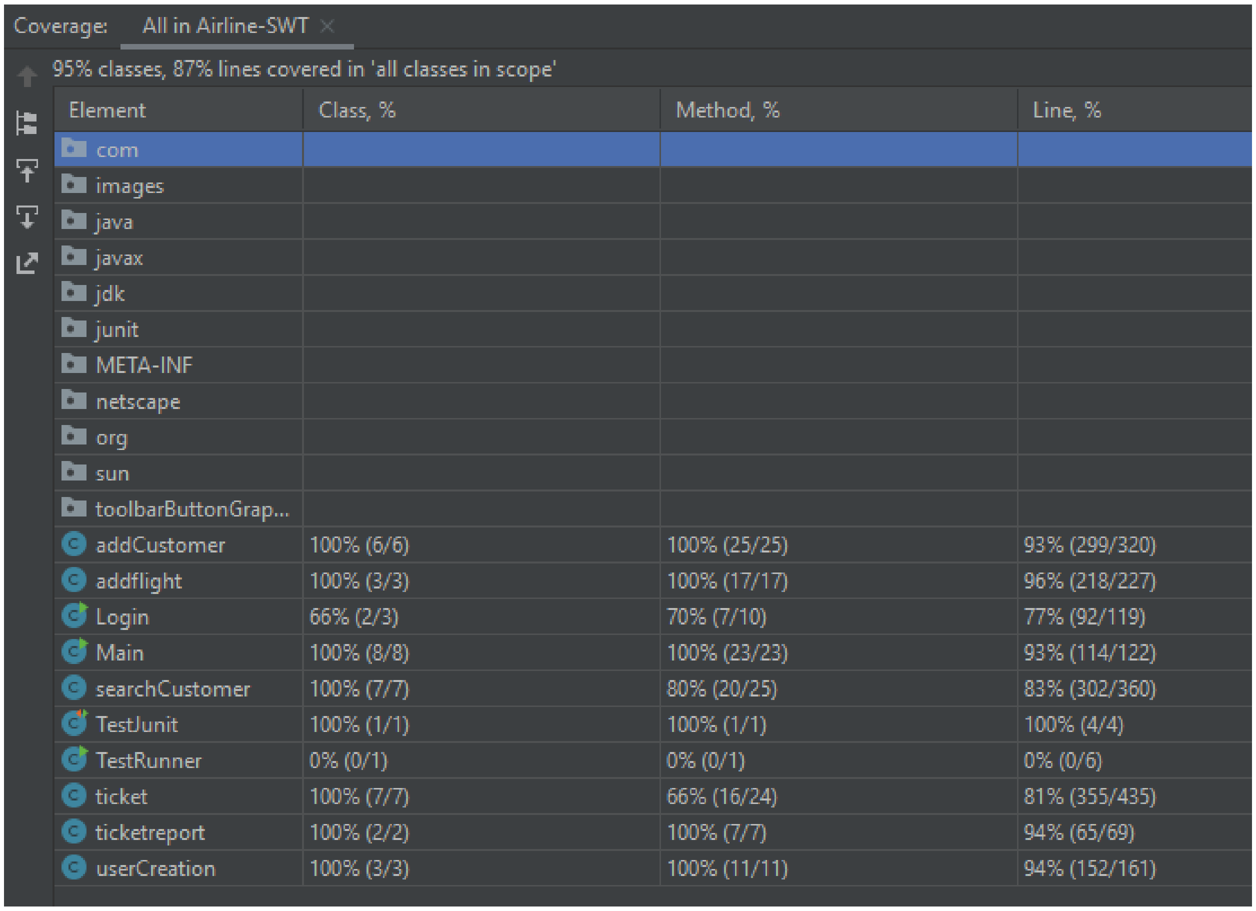
	`Assert.assertEquals("pass", pass.getText());`
	
	`}`

Expected Output	The input will fail because the username and password is not in the database.
Test Result (Actual Output)	The test passed. The input fails and displays a message.
Discrepancies	
Dependencies	• MySQL Database
Branch Coverage (%)	70%
Statement Coverage (%)	77%

Summary of Unit Testing Results:

1. There are 19 functional requirements and 11 non-functional requirements
2. 39 Test cases were written for the requirements
3. 32 test cases pass for functional requirements and 3 passes for nonfunctional requirements
4. 7 out of 39 fail
    a. For the functional requirements, the test case regarding the image failed but only during a certain set of circumstances. If the image is too large in size it will not allow the user to upload it. However, there is no error message that displays when doing so. This can be easily fixed by adding a line of code that specifies the size of the images that will be uploaded.

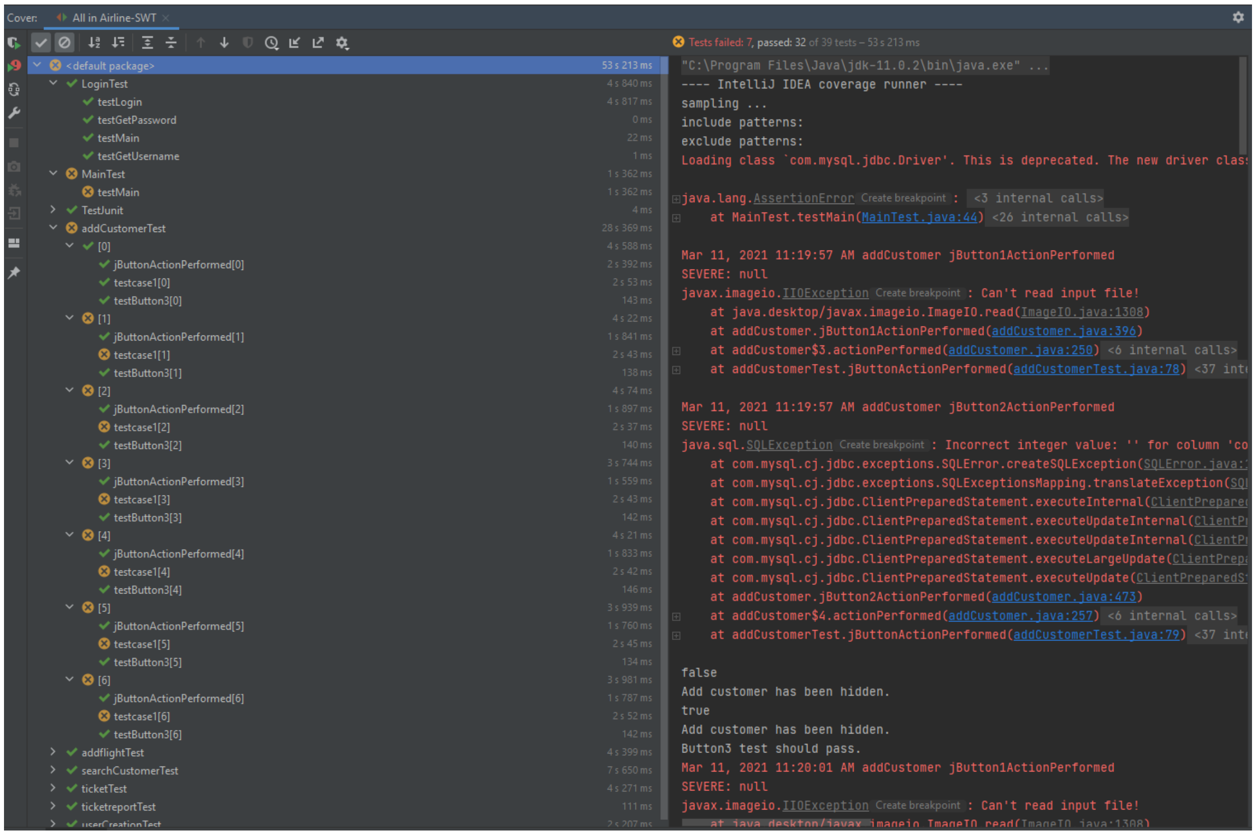
5. Screenshot of coverage



Coverage: All in Airline-SWT

95% classes, 87% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
com			
images			
java			
javax			
jdk			
junit			
META-INF			
netscape			
org			
sun			
toolbarButtonGrap...			
addCustomer	100% (6/6)	100% (25/25)	93% (299/320)
addflight	100% (3/3)	100% (17/17)	96% (218/227)
Login	66% (2/3)	70% (7/10)	77% (92/119)
Main	100% (8/8)	100% (23/23)	93% (114/122)
searchCustomer	100% (7/7)	80% (20/25)	83% (302/360)
TestJunit	100% (1/1)	100% (1/1)	100% (4/4)
TestRunner	0% (0/1)	0% (0/1)	0% (0/6)
ticket	100% (7/7)	66% (16/24)	81% (355/435)
ticketreport	100% (2/2)	100% (7/7)	94% (65/69)
userCreation	100% (3/3)	100% (11/11)	94% (152/161)



Cover: All in Airline-SWT

Tests failed: 7, passed: 32 of 39 tests – 53 s 213 ms

```
<default package>                          53 s 213 ms
  LoginTest                                 4 s 840 ms
    testLogin                               4 s 817 ms
    testGetPassword                              0 ms
    testMain                                    22 ms
    testGetUsername                              1 ms
  MainTest                                  1 s 362 ms
    testMain                                1 s 362 ms
  TestJunit                                      4 ms
  addCustomerTest                          28 s 369 ms
    [0]                                     4 s 588 ms
      jButtonActionPerformed[0]             2 s 392 ms
      testcase1[0]                           2 s 53 ms
      testButton3[0]                            143 ms
    [1]                                      4 s 22 ms
      jButtonActionPerformed[1]             1 s 841 ms
      testcase1[1]                           2 s 43 ms
      testButton3[1]                            138 ms
    [2]                                      4 s 74 ms
      jButtonActionPerformed[2]             1 s 897 ms
      testcase1[2]                           2 s 37 ms
      testButton3[2]                            140 ms
    [3]                                     3 s 744 ms
      jButtonActionPerformed[3]             1 s 559 ms
      testcase1[3]                           2 s 43 ms
      testButton3[3]                            142 ms
    [4]                                      4 s 21 ms
      jButtonActionPerformed[4]             1 s 833 ms
      testcase1[4]                           2 s 42 ms
      testButton3[4]                            146 ms
    [5]                                     3 s 939 ms
      jButtonActionPerformed[5]             1 s 760 ms
      testcase1[5]                           2 s 45 ms
      testButton3[5]                            134 ms
    [6]                                     3 s 981 ms
      jButtonActionPerformed[6]             1 s 787 ms
      testcase1[6]                           2 s 52 ms
      testButton3[6]                            142 ms
  addflightTest                             4 s 399 ms
  searchCustomerTest                        7 s 650 ms
  ticketTest                                4 s 271 ms
  ticketreportTest                              111 ms
  userCreationTest                          2 s 207 ms
```

```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude patterns:
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver clas

java.lang.AssertionError Create breakpoint :  <3 internal calls>
    at MainTest.testMain(MainTest.java:44) <26 internal calls>

Mar 11, 2021 11:19:57 AM addCustomer jButton1ActionPerformed
SEVERE: null
javax.imageio.IIOException Create breakpoint : Can't read input file!
    at java.desktop/javax.imageio.ImageIO.read(ImageIO.java:1308)
    at addCustomer.jButton1ActionPerformed(addCustomer.java:396)
    at addCustomer$3.actionPerformed(addCustomer.java:250) <6 internal calls>
    at addCustomerTest.jButtonActionPerformed(addCustomerTest.java:78) <37 inte

Mar 11, 2021 11:19:57 AM addCustomer jButton2ActionPerformed
SEVERE: null
java.sql.SQLException Create breakpoint : Incorrect integer value: '' for column 'co
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:
    at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQ
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPrepare
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPr
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPr
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeLargeUpdate(ClientPrepa
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdate(ClientPreparedS
    at addCustomer.jButton2ActionPerformed(addCustomer.java:473)
    at addCustomer$4.actionPerformed(addCustomer.java:257) <6 internal calls>
    at addCustomerTest.jButtonActionPerformed(addCustomerTest.java:79) <37 inte

false
Add customer has been hidden.
true
Add customer has been hidden.
Button3 test should pass.
Mar 11, 2021 11:20:01 AM addCustomer jButton1ActionPerformed
SEVERE: null
javax.imageio.IIOException Create breakpoint : Can't read input file!
    at java.desktop/javax.imageio.ImageIO.read(ImageIO.java:1308)
```