|  |  |
| --- | --- |
| Assessment Title | Assignment (SAAirline Ticketing) |

## Competency Details

|  |  |
| --- | --- |
| Unit code/s and title/s | ICTPRG443 – Test software development |
| Qualification code/s and title/s | ICT40120 - Certificate IV in Information Technology |
| Business unit/Work group | BARTS/IT Studies |

## Instructions

|  |  |
| --- | --- |
| Method/s of assessment | Product (Create and Written) |
| Overview of assessment | This assessment will require you to conduct a unit testing for a given class library. Student will establish the test requirements, prepare a test plan, design the test cases, write the test scripts, execute the test scripts and record the test results. |
| Task/s to be assessed | This assessment will require you to complete the following tasks based on the SAAirlineTicketing scenario:   * Task 1 – Test requirements (team work & individual requirement report) * Task 2 – Test plan * Task 3 – Test case design * Task 4 – Test script and execution * Task 5 – Test results * Task 6 – Unit testing progress (team meeting and individual progress report) * Task 7 – Defect tracking |
| Time allowed | Refer to your schedule for submission dates |
| Location of assessment | Assessment can be completed anywhere with access to the resources required. (see Resources Required section below) |
| Decision making rules | To receive a satisfactory outcome for this assessment you must complete all parts correctly.  Word counts are provided as guidance only. |
| Assessment conditions | This assessment must be undertaken where the conditions replicate noise levels and interruptions that people typically experience working in the ICT industry.  This is unsupervised assessment and you may access any required resources.  The Task 1 involves team work and as well as individual work.  The Task 2 ~ 5 are individual work. |
| Resources required | To complete this assessment, you will require the following:   * Access to Learn with Internet access * Visual Studio * Code requiring testing – C# SAAirline Ticketing class library * Word processing software such as Microsoft Word. |
| Result notification and reassessment information | You will be provided feedback and the result for your assignment on TAFESA Learn. You will be and given the chance to resubmit with required corrections only once.  Refer to the TAFE SA assessment policy for more information <https://www.tafesa.edu.au/apply-enrol/before-starting/student-policies/assessment> |

**Assignment (SAAirline Ticketing)**

SAAirline provides airline service. They fly flights with economy or first class seats. A flight normally, for example, has economy class of 20 rows with each row has 8 seats across (i.e. 160 economy seats) and 5 rows with each row has 4 seats across (i.e. 20 first class seats).

Flight information and customer information are captured. Non-discount prices are set as:

First Class seat at $2,000.00

Economy class at $1,000.00

Customers are optional to join membership such as Global member or Asia Member. Some customers do not join membership at all. For those who joined, they can receive discount on their seats booked.

* GlobalWorld members receive 20% discount.
* AsiaWorld members receive 10% discount.
* Customers can book several seats. Seats availability is recorded. The system displayed the invoice of what the customer has booked.

The **SAAirline** system has an **Airline Ticketing application program** is developing in **C# class library** by their own software team and it is still in an incomplete state. The input and output screens are still in the developing stage and are not available for integration test nor user acceptance test. Those tests are out of the scope in this stage of development. The application only has several business classes in the class library which are available for testing. The SAAirline has contracted the **ITWorks Company Ltd**. to conduct **unit testing** of the **SAAirline Booking** **project** to ensure those business classes are working properly. You and your other team member are working as software testers in ITWorks. Your team has experience in using **NUnit testing framework**. Your testing team has a test manager called **Jake Brown**. The project starts from today. The contracted project is expected to be completed within 7 weeks. On week 5, the project is expected to be the milestone date for test progress report and test review.

The project has an urgency to complete due to the other development are lined up to be completed. Therefore any defect or failed test cases will have very high priority to get it fixed. All defect test cases must be redeveloped and re-tested with 2 weeks.

Given the following items:

* Visual Studio Sales System project with C# class library source codes (Student files)
* ITWorks Test Plan template - customize the details as appropriate (Student files)
* The dependency diagram (Appendix A)
* The class diagram (Appendix B)

**Assessment schedule:**

* Attend **test requirement team meeting** on **Session 2**.
* Submit **Task 1 (individual Test Requirements report)** to LEARN on **Session 3**.
* Attend **test progress team meeting** on **Session 5**.
* Submit **Task 6 (Unit Testing Progress Report)** to LEARN on **Session 5** using the **ITWorks Unit Testing Progress Report.docx** file.
* Submit the following items in a **zip** file to LEARN on **Session 7**:
* The C# Test Project application with the test driver & all references to your class library being tested.
* The Test plan (add/modify details from the template plus the test cases)
* The word document with your answers from TASK 2 ~ 5 & 7. You must include the screen shots of all test cases, the summary result of NUnit Project & the defect tracking report.

Complete the following tasks:

**Task 1. (Test Requirements – team meeting and individual requirement report)**

* Oral communication (team meeting) on **Session 2**.
* Test Requirements Report (individual work) due by **Session 3**.

You need to form a team with no more than two members. The team members will meet in a workshop to determine the testing requirements, standards and terms.

ITWorks expects team members to follow the communication protocols which are listed in **ITWorks Communication Protocols.docx**. While the team is conducting the meeting, your lecturer will act as the IT Manager and to sign off the team members has met those protocols specified as the following:

* Raise questions to seek for clarification in the context of test requirements. Each member raises at lease three questions
* Listen and respond to the question being asked by the other team members (each member respond at least three questions)

Your test manager (impersonate by your lecturer) will observe your meeting that you collaborate with your team members. After the meeting each team member will document the outcome of the meeting individually using a Word document by answering the following questions:

1. What kind of testing is required for the SAAirline project? Note the given software is incomplete. It only has the .dll file available for this testing project.
2. Name and describe two roles involved in the SAAirline Ticketing testing project? Describe each of these two roles with no more than 30 words.
3. What testing tools would you use for conducting the type of test that you have chosen in Part a.
4. What are the benefits of the testing type that you have identified in Part c above? (Approx. 30 words)
5. Name the standards being used by the testing industry for the following testing aspects:

* Testing Concepts and Definition
* Testing Documentation
* Unit testing

**Task 2 (Test Plan)**

Given the case scenario of the SAAirline Ticketing testing project, **develop a test plan**. For completing the test documentation, you must use the provided **ITWorks Test Plan template** which is given as part of the student file. Customize the test plan to suit for your SAAirline Ticketing unit test project. **Add, delete** or specify **N/A** (as not appropriate) on any section or item if it is not suitable to the project. Make sure that you delete the explanation which are in red colour. Replace your own content and change the fonts in black colour. Rename the file so that the file name does not have the word “template”.

**Task 3 (Test Case Design)**

In the following table list all classes and their methods in the class library.

1. For each of the class constructor and methods, what are the minimum number test cases that you will generate based on either:

Equivalence Partitioning

or Boundary value analysis

test case design techniques.

In your answer, other than the number of test cases for the methods, you must also state the test case design techniques being used.

|  |  |  |
| --- | --- | --- |
| Classes | Methods | No of test cases |
| Flight (e.g.) | **Constructor:**  Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) i.e. possible integers -ve, 0, +ve è 3\*3\*3\*3= 81  **Using Using Equivalence Partitioning**  **Methods of the class:**  getFirstClass() **Using Using Equivalence Partitioning**  getEconomy() **Using Using Equivalence Partitioning**  setEconomy(ArrayList economy) **Using Using Equivalence Partitioning**  setFirstClass(ArrayList firstClass) **Using Using Equivalence Partitioning** | 81  1  1  1  1 |
| Activity | Activity(Flight theFlight) **Equivalence Partitioning**  bookSeats(int priceCode, int number, Customer theCustomer) **Equivalence Partitioning** 2\*3  Invoice getCustomerBooking(Customer cust) **Equivalence Partitioning** | 1  6  1 |
| AsiaWorld  Member | Getdiscount()**Using Equivalence Partitioning** | 1 |
| GlobalWorld  Member | Getdiscount()**Using Equivalence Partitioning** | 1 |
| Customer | Customer(int memberType, String firstName, String lastName, String creditNumber, String creditType, String expiry) **Using Using Equivalence Partitioning**  getCreditNumber()**Using Equivalence Partitioning**  getLastName() **Using Equivalence Partitioning**  getFirstName() **Using Equivalence Partitioning**  getCreditType() **Using Equivalence Partitioning**  getExpiry() **Using Equivalence Partitioning**  getMemberType() **Using Equivalence Partitioning**  setCreditNumber(String creditNumber) **Using Equivalence Partitioning**  setLastName(String lastName) **Using Equivalence Partitioning**  setFirstName(String firstName) **Using Equivalence Partitioning**  setCreditType(String creditType) **Using Equivalence Partitioning**  setExpiry(String expiry) **Using Equivalence Partitioning**  setMemberType(int memberType) **Using Equivalence Partitioning** | 3  1  1  1  1  1  1  1  1  1  1  1  3 |
| Invoice | Invoice(int priceCode, Customer theCust, int rowNum, int startSeatNum, int seatsBooked) **Using Equivalence Partitioning**  2\*1\*3\*3\*3  getNumberOfSeats()**Using Equivalence Partitioning**  getPriceCode()**Using Equivalence Partitioning**  getTheCust()**Using Equivalence Partitioning**  getStartSeatNum()**Using Equivalence Partitioning**  getRowNum()**Using Equivalence Partitioning**  setNumberOfSeats(int numberOfSeats) **Using Equivalence Partitioning**  setPriceCode(int priceCode) Boundary value analysis  -1,0,1,2  setTheCust(Customer theCust) **Using Equivalence Partitioning**  setRowNum(int rowNum) **Using Equivalence Partitioning**  setStartSeatNum(int startSeatNum) **Using Equivalence Partitioning**  getDiscount(int memberCode) **Using Equivalence Partitioning**  getCharge(int priceCode) **Using Equivalence Partitioning** | 54  1  1  1  1  1  3  4  1  3  3  1  1 |
| Seat | Seat(int numAvail, int code)3\*2 **Using Equivalence Partitioning**  Boolean bookSeats(int num)3 **Using Equivalence Partitioning**  getPriceCode()**Using Equivalence Partitioning**  getLastBooked()**Using Equivalence Partitioning**  getCurrentSeat()**Using Equivalence Partitioning** | 6  3  1  1  1 |
|  | Total no of test cases: | 199 |

1. Since there are too many test cases to fully test the application, you are only required to design 10 test cases in total.

For the 10 test cases, you must include:

* Four test cases must be related with the methods from the invoice class, Seat class or Activity class. (e.g. the calculations of seats, discount & price code).

Methods from the **Invoice** class:

* setNumberOfSeats(int numberOfSeats)
* setPriceCode(int priceCode)
* setTheCust(Customer theCust)
* setRowNum(int rowNum)
* setStartSeatNum(int startSeatNum)
* getDiscount(int memberCode) e.g. *Asia World or Global world members with correct discount*
* getCharge(int priceCode)

Methods from the **Seat** class:

* bookSeats(int num)

Method from **Activity** class:

* bookSeats (int priceCode, int number, Customer theCustomer)
* getCustomerBooking(Customer cust)
* One test case on null (e.g. IsNull) value. e.g. no customer name
* One test case on same object (i.e. AreSame) *e.g. Test the Invoice in getting a customer (Create a customer & invoice objects, set customer to existing object. i.e. Sales Customer)*
* Two test cases to test for exceptions. (e.g. argument exception, argumentNullException, argumentOutOfRangeException etc.)
* Two others as you prefer.

Develop unit test cases and document the test case design in the following table. For each test case, fill the following information:

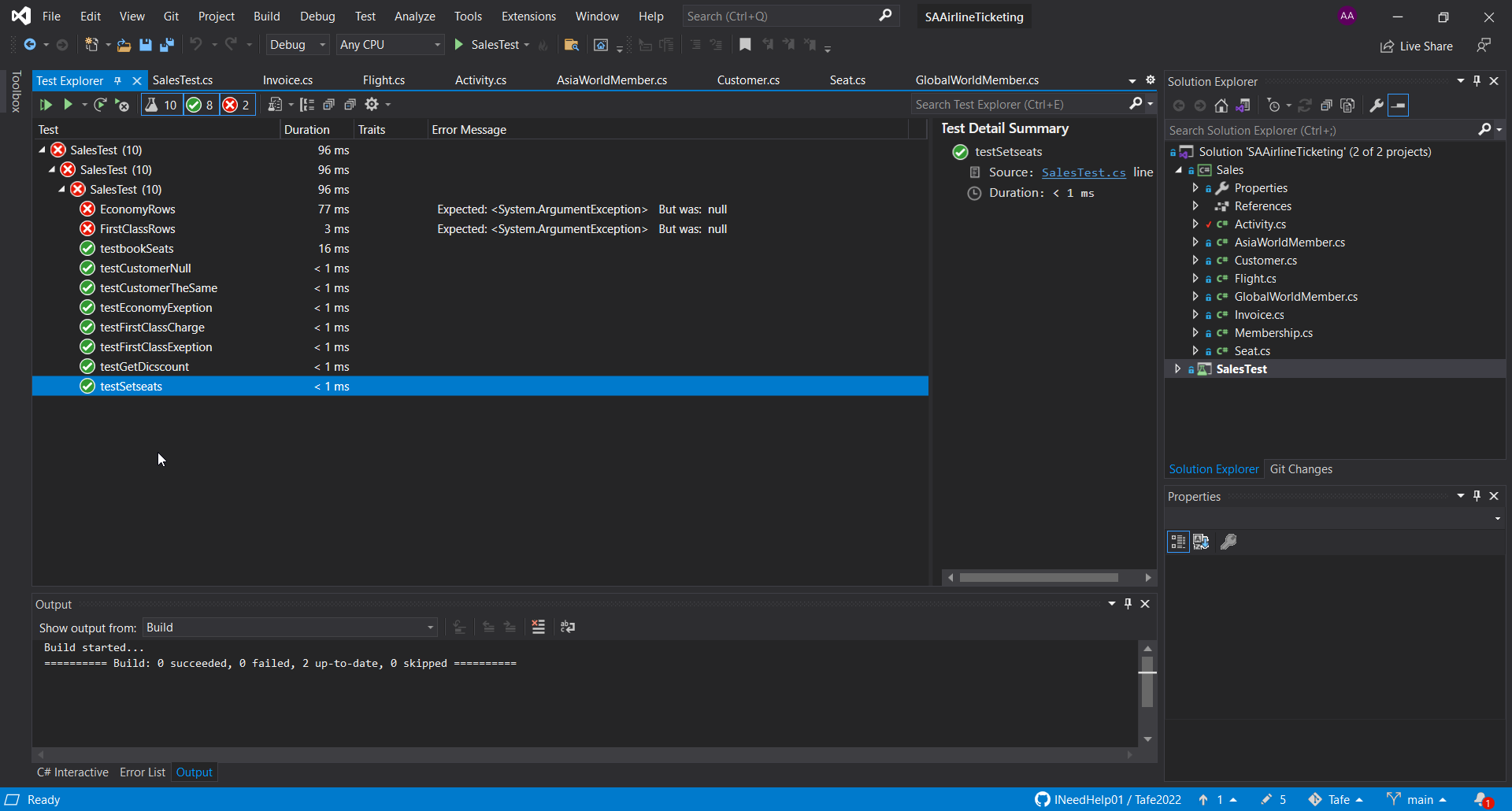
* Test case id – all test cases must be numbered.
* Test case description – summarise the test case with a few words.
* Reason to test in the business context – explain why you want to do the test.
* Data input – what object need to construct and what input parameters used.
* Test case technique used (e.g. specify boundary value analysis or equivalence class partition).
* Expected result – The tester calculates the expected result according to business process.
* Date/Time/Duration – The date and time the test is carried out and how long. The duration includes time spent the planning, design and execution.
* Resources required – include tester, test manager or product owner or delegate.

`

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Cases Design** | | | | | | | |
| **Test case id**  **TC xxx** | **Name of method being tested e.g. class.method().**  **No need to include the parameters in this column.** | **Reason to test in business context** | **Data input (Constructor used and the Method to be tested include the parameters)** | **Test Case Design Technique used** | **Date/time**  **&**  **Duration (hours)** | **Resources Required**  **(Name of tester, hardware & software needed)** | **Expected result** |
| TC 1 | Invoice.setNumberOfSeats(int numberOfSeats) | To make sure that invoice seat selection is working for customers for correct payment | Constructors:  customerMain = new Sales.Customer(1, "john", "Smith", "4560361423329093", "Visa", "03/2026");  invoiceMain = new Sales.Invoice(0, customerMain, 1, 4, 1);  invoiceMain.setNumberOfSeats(2);  Assert.AreEqual(2, invoiceMain.getNumberOfSeats()); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | 2 |
| TC 2 | Invoice.getDiscount(int memberCode) | Ensure that discounts customers have work | Constructors:  customerMain = new Sales.Customer(1, "john", "Smith", "4560361423329093", "Visa", "03/2026");  invoiceMain = new Sales.Invoice(0, customerMain, 1, 4, 1);  Assert.AreEqual(0.9, invoiceMain.getDiscount(1)); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | 0.9 |
| TC 3 | Seat.bookSeats(int num) | To ensure that seat booking deducts seats correctly | Constructors:  seatMain = new Sales.Seat(100, 0);  seatMain.bookSeats(5);  Assert.AreEqual(6, seatMain.getCurrentSeat());  Assert.AreEqual(1, seatMain.getLastBooked()); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | 6  1 |
| TC 4 | customerNull.getFirstName() | To test if a null name is passed into the method | Constructors:  Sales.Customer customerNull = new Sales.Customer(1, null, "Smith", "4560361423329093", "Visa", "03/2026");  Assert.IsNull(customerNull.getFirstName()); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | Is Null |
| TC 5 | Customer.Customer(int memberType, String firstName, String lastName, String creditNumber, String creditType, String expiry) | Tests if identical customers are the same and that the customer is linked to an invoice | Constructor:  invoiceMain = new Sales.Invoice(0, customerMain, 1, 4, 1);  customerMain = new Sales.Customer(1, "john", "Smith", "4560361423329093", "Visa", "03/2026");  Assert.AreSame(invoiceMain.getTheCust(), customerMain); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | invoiceMain.getTheCust() |
| TC 6 | Sales.Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Making sure that flights have an exception when economy seats are entered as 0 | Constructor:  Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(10, 0, 15, 10)); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | Exception |
| TC 7 | Sales.Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Making sure that flights have an exception when first class seats are entered as 0 | Constructors:  Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(10, 10, 15, 0)); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | Exception |
| TC 8 | Sales. Invoice(int priceCode, Customer theCust, int rowNum, int startSeatNum, int seatsBooked) | Making sure charges give the expected result | Constructors:  invoiceMain = new Sales.Invoice(0, customerMain, 1, 4, 1);  Assert.AreEqual(2000, invoiceMain.getCharge(1)); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | 2000 |
| TC 9 | Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Making sure there is an exception when economy rows has no rows as there can not be seats without rows | Constructors:    Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(0, 10, 10, 10)); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | exception |
| TC 10 | Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Making sure there is an exception when first class rows has no rows as there can not be seats without rows | Constructors:  Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(10, 10, 0, 10)); | Equivalence | 15/11/2022 | Andre Alexandrov  PC  Visual Studio  Microsoft.NET.Test.Sdk  NUnit  NUnit3TestAdapter | Exception |

**TASK 4. (Test Script and Execution)**

1. In the Visual Studio, create a test driver to test the class library using the NUnit test framework. Implement the unit tests. For each test cases designed in TASK 3, you are required to implement the algorithm in the test scripts/procedures (i.e. construct the appropriate objects with the designed test input parameters). You need to work out the manually for the expected results.
2. Execute your test cases. Submit a screen shot that all test cases have been executed.



**TASK 5. (Test Results)**

**Test Cases Report.** Execute the test cases if they have completed the test case designed. Record the test results or state the current status after the execution of the test cases. Record the Actual Result and specify whether the test is “Pass” or “Fail”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases Result as at: 06/12/2022** | | | | **Test Case Result:**  **Fill these only after the execution of the tests** | |
| **Test case id**  **TC xxx** | **Name of method being tested e.g. class.method()** | **Data input (include the parameters)** | **Expected result** | **Actual result** | **Pass / Fail or Not executed yet** |
| TC 1 | Invoice.setNumberOfSeats(int numberOfSeats) | invoiceMain.setNumberOfSeats(2);  Assert.AreEqual(2, invoiceMain.getNumberOfSeats()); | 2 | 2 | pass |
| TC 2 | Invoice.getDiscount(int memberCode) | Assert.AreEqual(0.9, invoiceMain.getDiscount(1)); | 0.9 | 0.9 | pass |
| TC 3 | Seat.bookSeats(int num) | seatMain.bookSeats(5);  Assert.AreEqual(6, seatMain.getCurrentSeat());  Assert.AreEqual(1, seatMain.getLastBooked()); | 6, 1 | 6,1 | pass |
| TC 4 | customerNull.getFirstName() | Sales.Customer customerNull = new Sales.Customer(1, null, "Smith", "4560361423329093", "Visa", "03/2026");  Assert.IsNull(customerNull.getFirstName()); | Is Null | Is Null | pass |
| TC 5 | Customer.Customer(int memberType, String firstName, String lastName, String creditNumber, String creditType, String expiry) | Assert.AreSame(invoiceMain.getTheCust(), customerMain); | invoiceMain.getTheCust() | invoiceMain.getTheCust() | pass |
| TC 6 | Sales.Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(10, 0, 15, 10)); | Argument exception | Argument exception | pass |
| TC 7 | Sales.Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(10, 10, 15, 0)); | Argument exception | Argument exception | pass |
| TC 8 | Sales. Invoice(int priceCode, Customer theCust, int rowNum, int startSeatNum, int seatsBooked) | Assert.AreEqual(2000, invoiceMain.getCharge(1)); | 2000 | 2000 | pass |
| TC 9 | Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(0, 10, 10, 10)); | Argument exception | null | fail |
| TC 10 | Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) | Sales.Flight flightTest;  Assert.Throws<ArgumentException>(() => flightTest = new Sales.Flight(10, 10, 0, 10)); | Argument exception | null | fail |

Note: Copy test cases from **Task 4** and fill the following columns:

* **Actual result** - result must be returned from the execution of the method.
* **Pass or Fail** – after the test execution, the test result must be either pass or fail.
* **Not Executed yet** – These test cases are behind in the test schedule.

For submission:

Capture one screen shot for each test cases that are failed in the test execution. e.g. if you have two test cases failed, you need to screen shots. In each screen shot, you must show both the expected result and actual result. Note: The expected result is in the test script. The actual result is in the Test Detail Summary of the Test Explorer.

Test Case X. Test result screen shot.

Test Case Y. Test result screen shot.

etc…

**Task 7. (Defect Tracking)**

Fill the following table for the test cases that are failed in your test cases. If you do not have a fail test case, generate one deliberately, therefore it is testing to fail.

Defect id (Dxxx) is related with the test case id (TCxxx). e.g. D5 is related with TC5.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Defect ID    D xxx | Priority  (low 1 ~ high 3) | Severity  (low 1 ~ high 3) | Created by team member | Created Date | Team member Assigned to | Expected Resolved Date (Exp. in two weeks) | Status (See possible options below) |
| D009 | 2 | 3 | Andre Alexandrov | 6/12/2022 |  | In one week | open |
| D010 | 2 | 3 | Andre Alexandrov | 6/12/2022 |  | In one week | open |
|  |  |  |  |  |  |  |  |

Provide a screen shot for each failed test case (i.e. Shown the test case result) that needs tracking.

**Note:**

**Created date:** use the data in session 7

**Possible Status:**

**New** - Initial status for a new test case. Indicates that the test case has been created but is not yet executed.

**Fixed** – The defect code has been fixed by the development team and open for testing.

**Open** – A test case has been designed and approved for testing.

**Reopen** – A test case may have been closed or blocked and re-stated to execute the test case again.

**Duplicate** -  Add test cases with the same basic information, save time by duplicating an existing test case and then editing the new test case.

**Invalid** - Write a negative test case by selecting the invalid input data or undesired user behaviour that you acquire by performing positive test case steps.

**Won’t fix** – Due to requirement may have changed, the test case no longer contribute to the part of the integrated code any more.

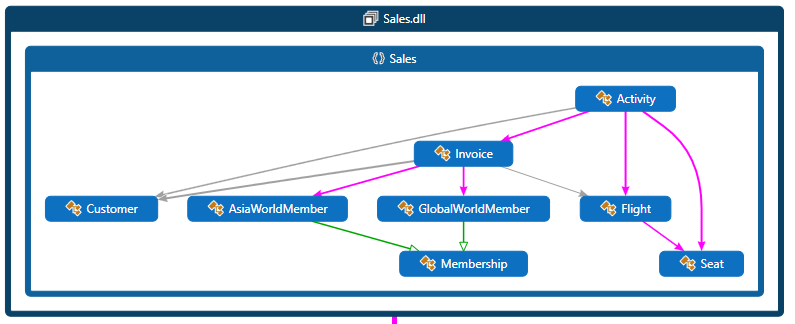
**Need more information** – The defected code may have complex logic that need more information to clarify the algorithm of the unit code.

**Suspended** – The test case is suspended due to a portion of testing activities are suspended e.g. critical path deadline is met and no more delivery is acceptable or holiday shuts down for both development and testing.

**Closed** – A test case is closed when there will be no more work on it.

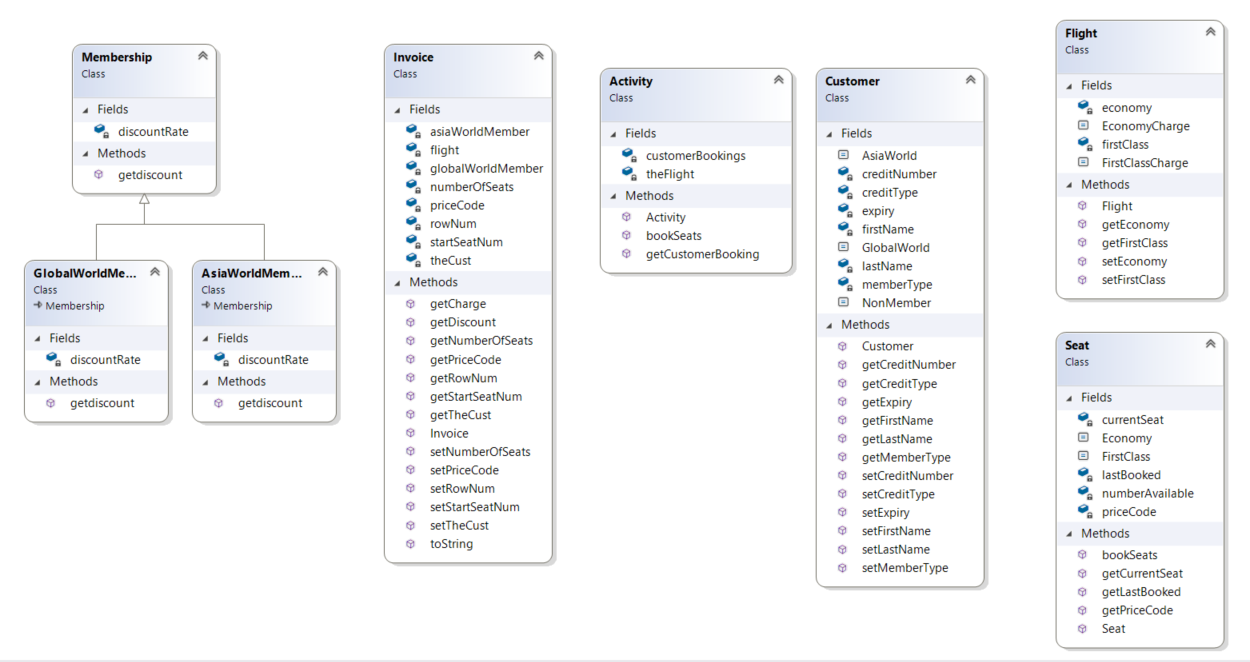
**Appendix A:**

**Class dependency diagram:**



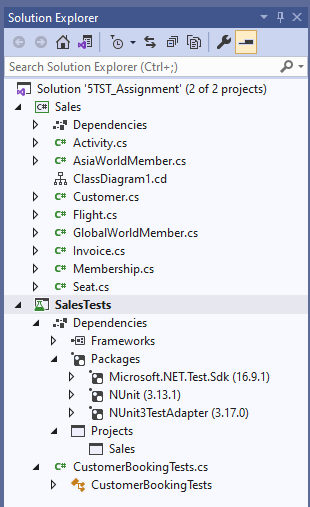
**Appendix B.**

**Class Diagram:**

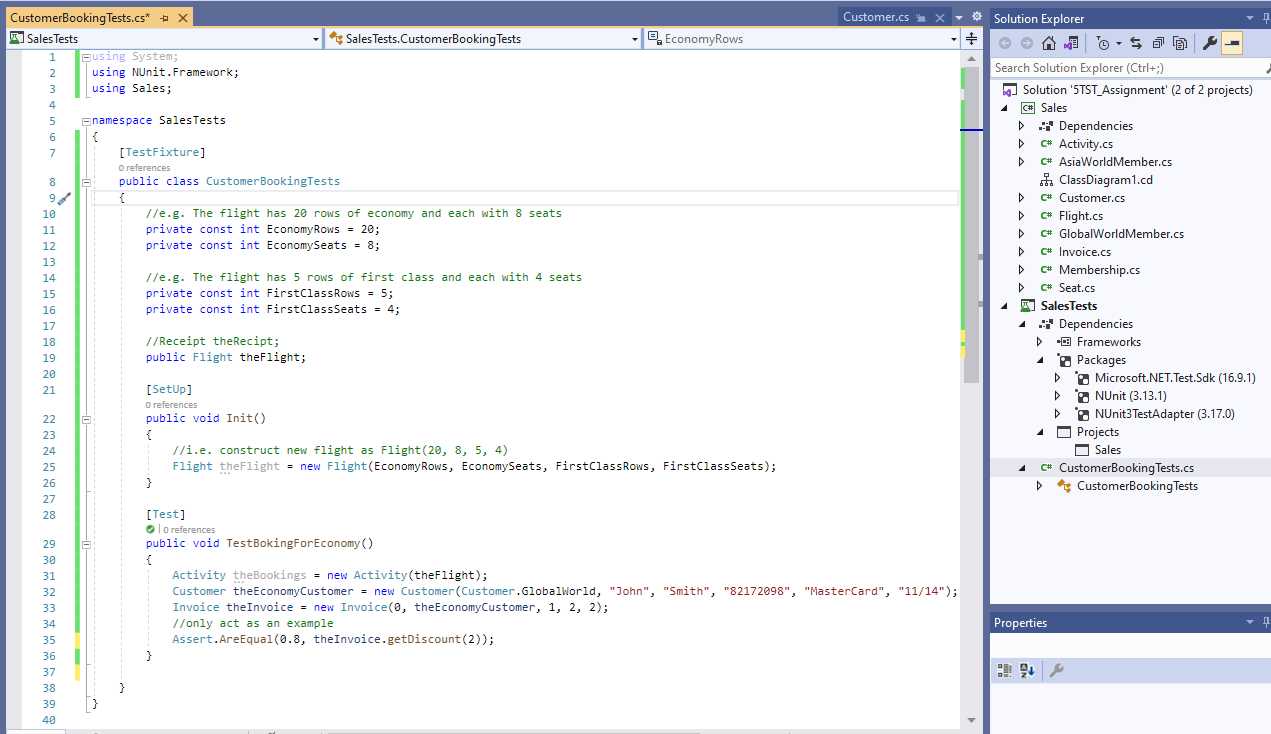


**Appendix C:**

**Showed project references and reference packages:**



**A test fixture example:**



Sample of an execution of a test case.