|  |  |
| --- | --- |
| Assessment Title | Practical Test (CarTowing) |

## Competency Details

|  |  |
| --- | --- |
| Unit code/s and title/s | ICTPRG433 – 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 carry out software testing based on the CarTowing scenario. You are required to define 8 test cases, develop and execute test script using the NUnit and fix testing framework. You also need to record the test results, fix the bugs and re-test the modified code |
| Task/s to be assessed | This assessment will require you to complete the following tasks   * Task 1 – Define Test Cases * Task 2 – Develop test scripts and execute tests * Task 3 – Record Test Results * Task 4 – Manage Defects |
| Time allowed | 2 Hours  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 a supervised assessment and you may access any required resources.  This is not group work and must be completed as an individual. |
| Resources required | To complete this assessment, you will require the following:   * Access to Learn with Internet access * Learn resources * Visual Studio * 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> |

ITWorks Software Company Ltd. got a contract to conduct the unit testing for the Car Towing program library. Given a TowingCostCalculator source codes, as the tester you are required to do:

* Prepare your test cases.
* Write a NUnit test driver to do unit testing on the codes provided
* Record the test results
* Manage the test defects

Car Towing business scenario:

A tow truck operator charges vehicle that needs towing based on the following factors:

* call out fee i.e. more expensive for a more heavy vehicle.
* how heavy the vehicle i.e. more expensive for trucks are fully loaded.
* how far they need to tow i.e. need more fuel for more distance for more heavy vehicle
* is the vehicle park inside a covered area i.e. harder to access if under cover.
* member or not i.e. give discount if subscribed as member

The Towing Cost of a vehicle is based on the following algorithm.

|  |  |
| --- | --- |
| Reason for charging | Charge method |
| Call out fee | Vehicle weight:   * Below or equal to 1500 kg 🡪 $100.00 * Above 1500 kg & below or equal to 5000 kg 🡪 $200.00 * Above 5000 kg 🡪 $300.00 |
| Distance | Distance towed & vehicle weight:   * Free for the first 10 km * For vehicle less than or equal to 1500 kg 🡪 $5.00 / km * For vehicle between 1500 kg & below or equal to 5000 kg 🡪 $10.00 / km * For vehicle above 5000 kg 🡪 $15.00 / km |
| After hours surcharge | Time required the service:   * Normal hours (NH) 🡪 $0.00 * After hours (AH) 🡪 $50.00 * Public holidays (PH) 🡪 $100.00 |
| Under cover area or not | Under cover or not?   * If under cover 🡪 $20.00 * Not under cover 🡪 $0.00 |
| Member discount | Member or not?   * If member 🡪 30% off * If not member 🡪 no discount |

## 

**Task 1. Define Test Cases**

a. Given the test data in the following table, for each of the test case, calculate the **Expected Result**. Complete the table with **Reasons for the Test** for test case 5 & 6.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Test Input Data | | | | | | | Expected Result |
|  | Reg no | Vehicle weight (Boundary value analysis) | Distance  (Boundary value analysis) | After Hour (Equivalence Partitioning) | Parking under cover (Equivalence Partitioning) | Member?  (Equivalence Partitioning) | Reason for the test case | Total Towing Charge |
| Test Case 1 | UPS-136 | 1,200 kg  e.g. $100.00 | 8 km  e.g. $0.00 | Normal hours  (NH)  e.g. $0.00 | Not under cover  e.g. $0.00 | Not a member  e.g. - $0.00 | Weight <= 1500 kg  Distance <= 10 km  NH  Not Covered  Not member | $100.00 (given) |
| Test Case 2 | VSS-338 | 5,000 kg  e.g. $200.00 | 20 km  e.g. $100.00 | After hours  (AH)  e.g. $50.00 | Is under cover  e.g. $20.00 | Is a member  e.g. - $111.00 | Weight <= 5000 kg  Distance > 10 km  AH  Covered  Is member | $259 (given) |
| Test Case 3 | BB-013 | 1000 kg | 10 km | Public holidays  (PH) | Not under cover | Is a member | Weight <= 1500 kg  Distance <= 10 km  PH  Not Covered  Is member | $100.00 (call out) + $0.00 (distance) + $100.00 (public holiday) + $0.00 (not under cover) - $60.00 (member discount) = **$140.00** |
| Test Case 4 | AA-216 | 1500 kg | 40 km | Normal hours  (NH) | Is under cover | Is a member | Weight <= 1500 kg  Distance > 10 km  NH  Covered  Is member | $100.00 (call out) + $150.00 (distance) + $0.00 (normal hours) + $20.00 (under cover) - $81.00 (member discount) = **$189.00** |
| Test Case 5 | RTB-369 | 5,500 kg | 30 km | Public holidays  (PH) | Not under cover | Not a member | Weight > 5,000 kg  Distance > 10 km  PH  Not Covered  Not member | $300.00 (call out) + $300.00 (distance) + $100.00 (public holiday) + $0.00 (not under cover) = **$700.00** |
| Test Case 6 | VVF-128 | 6,000 kg | 30km | After hours (AH) | Is under cover | Is a member | Weight > 5,000 kg  Distance > 10 km  AH  Covered  Is member | $300.00 (call out) + $300.00 (distance) + $50.00 (after hours) + $20.00 (under cover) - $201.00 (member discount) = **$469.00** |

b. In the following table, design two more test cases. The design of the test data must comply with the concept of **Equivalence Partitioning** and **Boundary Value Analysis**. For each of the test case design, make sure that the test case has a unique reason for the test. No **reason for the test** has the same reason as those in the Task 1a.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Test Input Data | | | | | | | Expected Results |
|  | Reg no | Vehicle weight | Distance | After Hour | Parking under cover | Member? | Reason for the test | Total Towing Charge |
| Test Case 7 |  | 1,600 kg | 15 km | Normal Hours (NH) | Is under cover | Not a member | Weight <= 5000 kg  Distance > 10 km  NH  Covered  Not member | $200.00 (call out) + $50.00 (distance) + $0.00 (after hours) + $20.00 (under cover) = **$270.00** |
| Test Case 8 |  | 5,000 kg | 50 km | Public Holidays (PH) | Is not under cover | Is a member | Weight <= 5000 kg  Distance > 10 km  PH  Not covered  Is member | $200.00 (call out) + $400.00 (distance) + $100.00 (after hours) + $0.00 (under cover) - $210.00 (member discount) = **$490.00** |

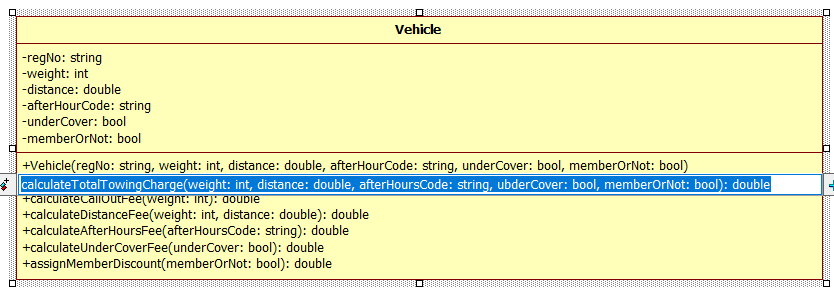
### Task 2. Design Test Cases

Given the **CalculateTowingCostApp – Incorrect** C# student file, you are required to do unit testing for the **Vehicle class** using the NUnit testing framework. (Note: In the given C# solution, create a new project as class library of .Net Core for the NUnit test driver.)

Write the NUnit test script to incorporate the eight test cases that you have designed in Task 1 a & b to test the **calculateTotalTowingCharge( )** method.

### \*\*Hints: use Assert.AreEqual() in the test driver.

### The Vehicle class has the following constructor and operation with its signatures.



// Constructor for Vehicle

public Vehicle(string regNo, int weight, double distance, string afterHourCode, bool underCover, bool memberOrNot)

// Calculate Total Towing Charge

public double calculateTotalTowingCharge(int weight, double distance, string afterHoursCode, bool underCover, bool memberOrNot)

\*\*Hints: use Assert.AreEqual() in the test driver. The class name is called **Vehicle**.

* + - The weight is expected to be entered in int.
    - The distance is expected to be entered as double.
    - The “after hours code” is a string and is expected to be either “NH”, “AH” or “PH”
    - The “under cover” and “member or not” are bool data type and is expected to enter value like true or false.
    - The “total towing charge” is calculated and displayed with two decimal places.

For submission:

You must submit the C# solution with the NUnit test driver.

You must capture the screen shots to show all test cases have been executed.

For each failed test cases, capture the screen shots to show the actual result and expected result. For example, if you have 4 test case failed, you need to have 4 screen captured for submission.

**Screenshots:**

**All Test Cases:  
A screenshot of a computer

Description automatically generated**

**Failed Test Cases:  
Test Case 2:  
A screenshot of a computer

Description automatically generated  
Test Case 4:  
A screenshot of a computer

Description automatically generated  
Test Case 6:  
A screenshot of a computer

Description automatically generated**

**Test Case 7:  
A screenshot of a computer

Description automatically generated**

**Task 3.** Record Test Results

Record your test results accordingly in the table below. Note: some test cases may fail and some may pass.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Write your Expected Results here**  **(i.e. Calculated by hand)** | **Run the test cases and record the Actual Results here** | **Test Results**  **(Pass or Fail ?)** |
| Test Case 1 | $100.00 (Given) | $100 (Given) | Pass |
| Test Case 2 | $259.00 (Given) | $245 (Given) | Fail |
| Test Case 3 | $140.00 | $140.00 | Pass |
| Test Case 4 | $189.00 | $105.00 | Fail |
| Test Case 5 | $700.00 | $700.00 | Pass |
| Test Case 6 | $469.00 | $455.00 | Fail |
| Test Case 7 | $270.00 | $250.00 | Fail |
| Test Case 8 | $490.00 | $490.00 | Pass |

**Task 4.** Manage defects

Duplicated **CalculateTowingCostApp – Incorrect** C# program and call it **CalculateTowingCostApp – Fixed.** The incorrect source code of **Vehicle** class is shown in the Appendix. The following shows some **hints** of the errors in the program codes. Locate the errors that caused your test cases to fail.

Error 1. (The Vehicle constructor missing UnderCover property)

Error 2. (calculateTotalTowingCharge() method – UnderCover spelled uppercase)

Error 3a. (In the calculateCallOutFee() method, if (weight <= 1500) issues)

Error 3b. (In the calculateCallOutFee() method, if (weight > 5000 ~~&& weight <= 6000~~) issues)

Error 4a. (In calculateDistanceFee() method, if (weight > 5000 ~~&& weight <= 6000~~) issues)

Error 5. (In calculateUnderCoverFee() method, if (UnderCover == false) UnderCover spelled uppercase)

a.  Fix the errors by modifying the source codes. Execute the tests again.

For submission: Capture the **screen shots** to show the test cases has been executed.

Submit the **both** incorrect and fixed program to Moodle for marking.

**Screenshot:  
  
A close up of a computer screen

Description automatically generated**

b. After executing the test, record your test results accordingly in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Expected Results**  **(i.e. Calculated by hand)** | **Actual Results** | **Test Results**  **(Pass or Fail ?)** |
| Test Case 1 | $100.00 | $100.00 | Pass |
| Test Case 2 | $259.00 | $259.00 | Pass |
| Test Case 3 | $140.00 | $140.00 | Pass |
| Test Case 4 | $189.00 | $189.00 | Pass |
| Test Case 5 | $700.00 | $700.00 | Pass |
| Test Case 6 | $469.00 | $469.00 | Pass |
| Test Case 7 | $270.00 | $270.00 | Pass |
| Test Case 8 | $490.00 | $490.00 | Pass |

## Appendix: The given Vehicle class has the following program codes. There are errors in it.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace TowingCostCalculator

{

public class Vehicle

{

public string RegNo { get; set; }

public int Weight { get; set; }

public double Distance { get; set; }

public string AfterHoursCode { get; set; }

public bool UnderCover { get; set; }

public bool MemberOrNot { get; set; }

public double TotalTowingCharge { get; set; }

private double lTotalTowingCharge;

private double lCallOutFee;

private double lDistanceFee;

private double lAfterHoursFee;

private double lUnderCoverFee;

private double lDiscount;

// Constructor for Vehicle

public Vehicle(string regNo, int weight, double distance, string afterHoursCode, bool underCover, bool memberOrNot)

{

RegNo = regNo;

Weight = weight;

Distance = distance;

AfterHoursCode = afterHoursCode;

MemberOrNot = memberOrNot;

TotalTowingCharge = 0.0;

}

// Calculate Total Towing Charge

public double calculateTotalTowingCharge(int weight, double distance, string afterHoursCode, bool underCover, bool memberOrNot)

{

lCallOutFee = this.calculateCallOutFee(weight);

lDistanceFee = this.calculateDistanceFee(weight, distance);

lAfterHoursFee = this.calculateAfterHoursFee(afterHoursCode);

lUnderCoverFee = this.calculateUnderCoverFee(UnderCover);

lDiscount = this.assignMemberDiscount(memberOrNot);

//lDiscount = this.assignMemberDiscount(memberOrNot);

lTotalTowingCharge = Math.Round((lCallOutFee + lDistanceFee + lAfterHoursFee + lUnderCoverFee) \* (1 - lDiscount), 2);

return lTotalTowingCharge;

}

// Calculate call out fee

private double calculateCallOutFee(int weight)

{

if (weight < 1500)

{

lCallOutFee = 100.00;

}

else if (weight > 1500 && weight <= 5000)

{

lCallOutFee = 200.00;

}

else if (weight > 5000 && weight <= 6000)

{

lCallOutFee = 300.00;

}

return lCallOutFee;

}

// Calculate distance fee

private double calculateDistanceFee(int weight, double distance)

{

if (distance == 0)

{

throw new ArgumentException("Distance cannot be zero");

}

else

{

if (distance <= 10.0)

{

lDistanceFee = 0.00;

}

else if (distance > 10.0)

{

if (weight <= 1500)

{

lDistanceFee = (distance - 10.0) \* 5.00;

}

else if (weight > 1500 && weight <= 5000)

{

lDistanceFee = (distance - 10.0) \* 10.00;

}

else if (weight > 5000 && weight <= 6000)

{

lDistanceFee = (distance - 10.0) \* 15.00;

}

}

lDistanceFee = Math.Round(lDistanceFee, 2);

return lDistanceFee;

}

}

// Calculate after hours fee

private double calculateAfterHoursFee(string afterHoursCode)

{

if (afterHoursCode == "NH")

{

lAfterHoursFee = 0;

}

else if (afterHoursCode == "AH")

{

lAfterHoursFee = 50.00;

}

else if (afterHoursCode == "PH")

{

lAfterHoursFee = 100.00;

}

return lAfterHoursFee;

}

// Calculate uder cover fee

private double calculateUnderCoverFee(bool underCover)

{

if (UnderCover == false)

{

lUnderCoverFee = 0.00;

}

else

{

lUnderCoverFee = 20.00;

}

return lUnderCoverFee;

}

// Assign member discount

private double assignMemberDiscount(bool memberOrNot)

{

if (memberOrNot == false)

{

lDiscount = 0.00;

}

else

{

lDiscount = 0.3;

}

return lDiscount;

}

}

}