# Project Assessment: Develop an object-oriented application

## Criteria

### Unit code, name and release number

ICTPRG413 - Use a library or pre-existing components (1)

ICTPRG430 - Apply introductory object-oriented language skills (1)

### Qualification/Course code, name and release number

ICT40518 - Certificate IV in Programming (1)

## Student details

### Student number

336558344

### Student name

Alex Smith

## Assessment Declaration

* This assessment is my original work and no part of it has been copied from any other source except where due acknowledgement is made.
* No part of this assessment has been written for me by any other person except where such collaboration has been authorised by the assessor concerned.
* I understand that plagiarism is the presentation of the work, idea or creation of another person as though it is your own. Plagiarism occurs when the origin of the material used is not appropriately cited. No part of this assessment is plagiarised.

### Student signature and Date

Alex Smith 07/06/2021

[Submitted electronically]

Version: 20200120

Date created: 1 October 2019

Date modified: 20 January 2020

For queries, please contact:

Technology and Business Services SkillsPoint

Ultimo

© 2020 TAFE NSW, Sydney  
RTO Provider Number 90003 | CRICOS Provider Code: 00591E

This assessment can be found in the: [Learning Bank](https://share.tafensw.edu.au/share/access/searching.do?doc=%3Cxml%2F%3E&in=P7ac4831b-430a-4b8d-8b56-f7b32ed5b9cf&q=&type=standard&sort=rank&dr=AFTER)

The contents in this document is copyright © TAFE NSW 2020, and should not be reproduced without the permission of the TAFE NSW. Information contained in this document is correct at time of printing: 7 June 2021. For current information please refer to our website or your teacher as appropriate.

## Assessment instructions

Table 1 Assessment instructions

| Assessment details | Instructions |
| --- | --- |
| **Assessment overview** | The objective of this assessment is to assess your knowledge and performance as required to apply introductory object-oriented language skills and use a library or pre-existing components |
| **Assessment Event number** | 1 of 3 |
| **Instructions for this assessment** | This is a project-based assessment and will be assessing you on your knowledge and performance of the unit.  This assessment is in six parts:  Review and clarify requirements  Design the application  Develop the application  Test the application  Evaluation  Perform handover.  The assessment also contains:   * Observation Checklist 1-2 * Assessment Feedback.   **Check the Assessment instructions and the Observation Checklists to ensure that you’ve covered all the required tasks.** |
| **Submission instructions** | On completion of this assessment, you are required to upload it or hand it to your assessor for marking.  Ensure you have written your name at the bottom of each page of your assessment.  Submit the following documents for each part:   * Part 1: Review and clarify requirements   + You will be observed by your assessor * Part 2: Design the application   + Project design documentation report * Part 3: Develop the application   + Completed application incorporating reusable component   + XML documentation * Part 4: Test the application   + Test data document * Part 5: Evaluation   + Evaluation report * Part 6: Perform handover   + You will be observed by your assessor.   It is important that you keep a copy of all electronic and hardcopy assessments submitted to TAFE and complete the assessment declaration when submitting the assessment. |
| **What do I need to do to achieve a satisfactory result?** | To achieve a satisfactory result for this assessment all questions must be answered correctly and all items in the Observation Checklists must be marked Satisfactory. |
| **Assessment conditions** | Assessment conditions will be safe and replicate the workplace. Noise levels, production flow, interruptions and time variances must be typical of those experienced in the programming and software field of work.  Assessment may be undertaken in normal classroom conditions, which is assumed to be noisy and similar to workplace conditions, or within the workplace. This may include phones ringing, people talking and other interruptions. |
| **What do I need to provide?** | * USB drive or other storage method with enough free space to save work to. |
| **What will the assessor provide?** | * An integrated software development environment (IDE) * Applications for software development (Visual Studio suite) * User requirements outlined in scenario * Reuse library within Visual Studio * Technical requirements outlined in scenario and assessment * Organisational conventions outlined in scenario * User for consultation * Documents in the zipped folder Cl\_Programming\_1\_AE\_Pro\_1of3\_SR1.zip |
| **Due date and time allowed** | Indicative time to complete assessment:   * In class: Four hours * Out of class: 16 hours. |
| **Assessment location** | Parts 1 and 6 will be completed in the classroom. All other parts will be a combination of in and out of the classroom.  The student may access their referenced text, learning notes and other resources. |
| **Supervision** | Part of this assessment is an unsupervised, take-home assessment. Your assessor may ask for additional evidence to verify the authenticity of your submission and confirm that the assessment task was completed by you. |
| **Reasonable adjustment** | If you have a permanent or temporary condition that may prevent you from successfully completing the assessment event(s) in the way described, you should talk to your assessor about ‘reasonable adjustment’. This is the adjustment of the way you are assessed to take into account your condition, which must be approved BEFORE you attempt the assessment. |
| **Assessment feedback, review or appeals** | In accordance with the TAFE NSW policy *Manage Assessment Appeals,* all students have the right to appeal an assessment decision in relation to how the assessment was conducted and the outcome of the assessment. Appeals must be lodged within **14 working days** of the formal notification of the result of the assessment.  If you would like to request a review of your results or if you have any concerns about your results, contact your Teacher or Head Teacher. If they are unavailable, contact the Student Administration Officer.  Contact your Head Teacher for the assessment appeals procedures at your college/campus. |

## Specific task instructions

### Scenario

You have been contracted as a software developer for an IT company to develop an application that calculates payroll data for one of their clients.

#### User requirements

* The application needs to import a comma delimited (.csv) file for payroll data and calculate the gross, net and tax amounts for each employee.
* The comma delimited file contains data for two types of employees:
  + Residents (employees who live in Australia)
  + Working holiday (employees who are visiting Australia and on a working holiday).
* The method for calculating tax is different for each type of employee; the program must account for this and calculate the tax accurately according to the employee type (see section on calculating tax below).
* Once the pay amounts have been calculated, the employee ID, gross, net and tax amounts for each employee must be written to:
  + a comma delimited file (.csv) using the naming convention {DateTime.Now.Ticks}-records.csv
    - For example: ***637050122993045755-export.csv***
    - For more information on ticks visit the web page [DateTime.Ticks Property](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.ticks?view=netframework-4.8).
  + the console window.
* The functionality for writing the computed pay values to the console and a .csv file must be implemented using a **reusable component.**
* The IT company prides itself on providing high-quality software to customers and requires that unit tests be created for testing by:
  + importing the data from the comma delimited values file and loading the data into a list of objects
  + checking the correctness of the values calculated for the gross, net and tax amounts of each employee.

#### Calculating tax

The *TaxCalculator* class has two methods, one for calculating resident tax and one for calculating working holiday tax.

##### Resident tax

A resident’s tax amount is calculated based on the value of gross pay using values in **Table 2: Resident tax scale**. For example, if the gross amount is 652 (which is greater than 361 and less than or equal to 932), the coefficient values (A and B) are 0.3477 and 44.2476. These values are then used to calculate the tax amount using the formula **Tax** = **A** multiplied by **Gross** minus **B.**

In the example of a gross amount of 652, this would be calculated as:

**Tax = 0.3477 \* 652 - 44.2476**, which results in the tax amount of **182.4528**.

Table : Resident tax scale

|  |  |  |  |
| --- | --- | --- | --- |
| Gross (>) | Gross (<=) | A | B |
| -1 | 72 | 0.19 | 0.19 |
| 72 | 361 | 0.2342 | 3.213 |
| 361 | 932 | 0.3477 | 44.2476 |
| 932 | 1380 | 0.345 | 41.7311 |
| 1380 | 3111 | 0.39 | 103.8657 |
| 3111 | 999999 | 0.47 | 352.7888 |

##### Working holiday tax

The tax for a working holiday employee is based on the tax scale as shown in **Table 3: Working holiday tax scale,** which is evaluated by using the sum of an employee’s year to date pay (how much they have been paid during the financial year) and the gross pay (for the current pay).

For example, if the total of an employee’s year to date pay is 47,938 and the gross amount is 418, the total gross amount is 47938 + 418 = 48356 (which is greater than 37000 and less than or equal to 90000). This value is then used to determine the tax rate, which would be 0.32 (in this example).

The tax is then calculated by multiplying the gross amount (not the total gross amount) by the tax rate using the formula **Tax = gross \* rate**.

In this example, this would be calculated as:

**Tax = 418 \* 0.32**, which results in the tax amount of **133.76.**

Table : Working holiday tax scale

|  |  |  |
| --- | --- | --- |
| Gross (>) | Gross (<=) | Rate |
| -1 | 37000 | 0.15 |
| 37000 | 90000 | 0.32 |
| 90000 | 180000 | 0.37 |
| 180000 | 9999999 | 0.45 |

#### Organisational development principles and practices

* The IT company uses an iterative approach to software development. Each stage must be checked as it is completed, with issues corrected in previous stages as necessary.
* All coding must comply with the coding standards as described at [C# Coding Standards and Naming Conventions](https://www.dofactory.com/reference/csharp-coding-standards).
* All methods must have internal XML documentation as illustrated below in **Figure 1 - C# Internal XML comments (example)**; further information is available at [Recommended Tags for Documentation Comments (C# Programming Guide)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/xmldoc/recommended-tags-for-documentation-comments).

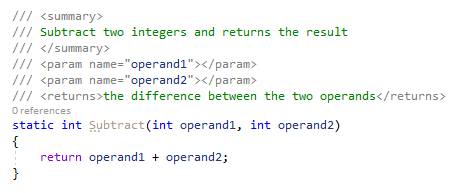


Figure 1 - C# Internal XML comments (example)

* All tests must be documented using the *Software testing documentation* template (Software testing documentation.docx).

**You must follow the listed Organisational development principles and practices in the appropriate stages of the project.**

Download and unzip the resource folder (Cl\_Programming\_1\_AE\_Pro\_1of3\_SR1.zip) for files referred to within the assessment.

## Part 1: Review and clarify requirements

Meet with the client in a role play (10-15 minutes) to review and clarify the user requirements.

Your assessor will observe the role play and complete **Observation Checklist 1**.

Role play participants:

* **Client** – Your assessor will arrange for another person to participate as the client.
* **Software Developer** (this is you).

Ensure that you include the following in your role play:

1. Review and clarify the user requirements with the client, asking questions if necessary to ensure that you have correctly understood what is needed.

**Please ask me some questions about the project**

1. **Which platforms will you need this application to run on?**
2. **How will you use the calculator?**
3. **Where does the data come from?**
4. **What features does the calculator need to have?**

## Part 2: Design the application

Using the information you have gathered through research and discussion with the client, create the project design documentation by including the following tasks in a word-processed report.

### Task 1

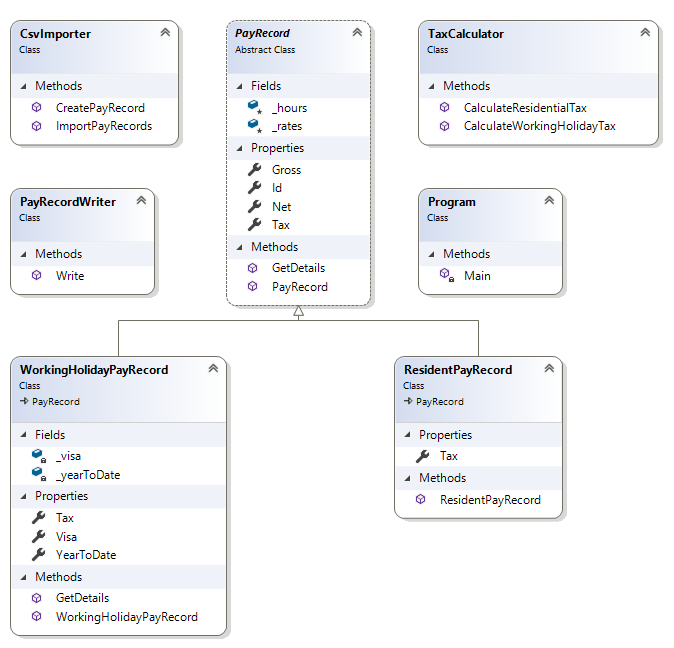
**Table 4 - Classes** contains the information needed to plan and determine the application design. Create a class diagram of this information using software, MS Paint or even pen and paper. Ensure that the classes, relationships, methods and properties are captured in the diagram.

For more information on class diagrams read the web page [UML Class Diagram Tutorial with Examples](https://www.guru99.com/uml-class-diagram.html).

Table 4 - Classes

| # | Class members |
| --- | --- |
| **Program** | **Methods**   * Main   + Parameters     - args : string[]   + Returns : void |
| **TaxCalculator** | **Methods**   * CalculateResidentialTax   + Modifiers:     - public, static   + Parameters     - gross : double   + Returns: double * CalculateWorkingHolidayTax   + Modifiers:     - public, static   + Parameters     - gross : double, yearToDate : double   + Returns: double |
| **CsvImporter** | **Methods**   * ImportPayRecords   + Modifiers:     - public, static   + Parameters     - file : string   + Returns: List<PayRecord> * CreatePayRecord   + Modifiers     - public , static   + Parameters     - id : int, hours : double[], rates : double[], visa : string, yearToDate : string   + Returns: PayRecord |
| **PayRecord** | **Fields**   * \_hours : double[] (protected) * \_rates : double[] (protected)   \* both \_hours and \_rates are used as a parallel arrays to store the hours and rate for each shift, both fields are initialled by arguments passed in to the constructor  **Constructor**   * Parameters   + id : int, hours : double[], rates : double[]   **Methods**   * GetDetails   + Modifiers:     - public, virtual   + Returns: string   **Properties**   * Id : int   + public getter / private setter * Gross : double   + public getter / no setter   + derived value calculated from the sum of (hours multiplied by rate) for all shifts. * Tax : double   + public abstract getter / no setter   + derived value calculated from the appropriate TaxCalculator method based on the type of pay record (resident or working holiday) * Net : double   + public getter / no setter   + derived value calculated from Gross minus Tax (Net = Gross - Tax) |
| **ResidentPayRecord** | \*Inherits from PayRecord, overrides abstract property Tax from base class.  **Constructor**   * Parameters   + id : int, hours : double[], rates : double[]   + (calls base constructor) |
| **WorkingHolidayPayRecord** | \*Inherits from PayRecord, overrides abstract property Tax and GetDetails method from base class.  **Constructor**   * Parameters   + id : int, hours : double[], rates : double[], visa : int, yearToDate : int (calls base constructor)   **Properties**   * Visa : int   + public getter / private setter * YearToDate : int   + public getter / private setter |
| **PayRecordWriter** | Class that contains a static method for writing the calculated pay records to a comma delimited file (and optionally console)  **Methods**   * Write   + Modifiers:     - public, static * Parameters   + file : string, records : List<PayRecord>, writeToConsole : bool (optional) * Returns: void |

**Place class diagram here**



### Task 2

The required application will be run on desktop computers, however the client would like to know what would be required if they wanted to convert it into an app.

**Identify some systems (this may include IDEs, operating systems) and devices that have the potential to meet their future needs. You can use a table similar to below.**

|  |  |
| --- | --- |
| Systems | Devices |
| iOS | iPhone |
| Android | Android smartphones |
| Linux | Various |
| Web-based | Various |
|  |  |

### Task 3

Identify and evaluate suitable reusable components that have the desired functionality for this project.

1. Analyse the project design (Part 2 Task 1) and the functionality required for the application to determine the types of reusable components needed.

There is a need for method to read data from csv file and write to csv file => working with files.

Identify two options for suitable reusable components to evaluate, including the repositories where they are located.

CsvHelper

LumenWorksCsvReader

Establish suitable criteria to use to evaluate the reusable components, including cost.

What is the cost?

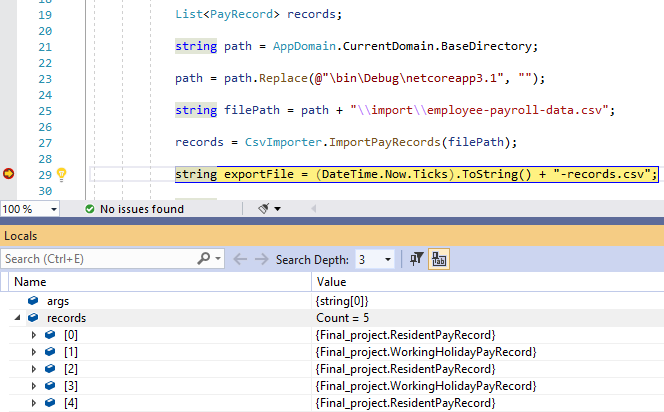
What is the licence?

Which platforms does it run on?

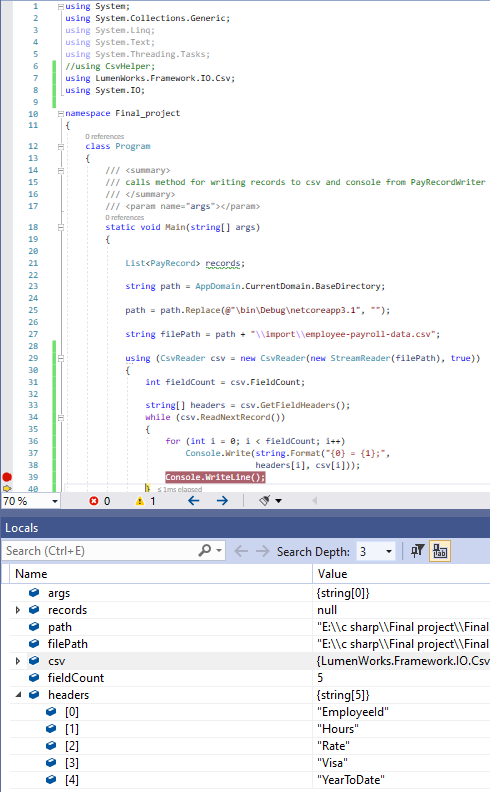
Is it compatible with our version of .NET?

Develop small console applications for testing the identified reusable components before implementing them into the project. Test the components and include screenshots of your testing.

**CsvHelper:**



**LumenWorksCsvReader:**



Evaluate the suitability of each reusable component with the project, using the identified criteria.

CsvHelper: Free to install, free licence, runs on .NET framework, compatible.

LumenWorksCsvReader: Free to install, free licence, runs on .NET framework, compatible.

Compare the functionality of each component with the project requirements and identify any gaps.

CsvHelper can read and write files.

LumenWorksCsvReader only reads.

Consider and outline what technical impact the reusable components would have on the project design.

CsvHelper v 15.0.0 available as NuGet package, compatible with our version of .NET

LumenWorksCsvReader v 1.0.0 available as NuGet package, compatible with our version of .NET

Determine any licensing requirements/issues for using each component for commercial use.

CsvHelper is free, no licensing required

LumenWorksCsvReader is free, no licensing required

Recommend appropriate reusable components for this project, outlining your reasons.

CsvHelper is a suitable recommendation. It performs the required tasks, it has

documentation and is free and compatible with our version of .NET.

LumenWorksCsvReader is not suitable. It does not perform write function. It is more difficult, requires more lines of codes.

## Part 3: Develop the application

### Task 1

Implement your application design to develop the application, ensuring that you follow the code and documentation conventions in the organisational requirements. You have been supplied with the payroll data file (*employee-payroll-data.csv)*.

**Note**: Each row in the file denotes a single shift however a pay record for a single employee can contain many shifts.

1. Your projects is a Visual Studio Console Application using the following naming convention: Project name: MyPayProject

Add a folder named *Import* to the root directory of the project and copy the provided file *employee-payroll-data.csv* into it **(see Figure 2 – Import folder)**

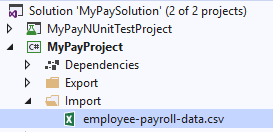


Figure – Import folder

Add the *TaxCalculator* class and required methods for calculating the resident and working holiday tax amounts.

Add the *PayRecord*, *ResidentPayRecord* and *WorkingHolidayPayRecord* classes, including properties and methods.

* 1. Implement **polymorphism** so the *ResidentPayRecord* and *WorkingHolidayPayRecord* classes behave differently by invoking the relevant methods in the *TaxCalculator* class.
  2. The *GetDetails* method must return the employee ID, gross, net and tax amounts.
  3. The working holiday GetDetails method must return the values for the *Visa* and *YearToDate* properties.

Add a class *CsvImporter* to the project and the *ImportPayRecords* method.

Implement the logic for the *ImportPayRecords* method so it:

* 1. opens the specified file by name using a StreamReader class from the reusable System.IO component
  2. iterates through each line (row) in the file
  3. determines when all the rows for an employee have been read
  4. instantiates a new instance of a resident or working holiday pay record, as appropriate.

If an employee row doesn’t contain either a value for *visa* or *year to date,* you can assume it is for a resident.

**Note**: It may be helpful to create a separate method for instantiating a new pay record object.

1. After the *Import* method has been completed, invoke the method from the *Main* method in the *Program* class to ensure it works correctly.

### Task 2

Incorporate the reusable component into your application.

1. Configure the development environment to install, reference and implement the selected reusable component from the relevant repository.

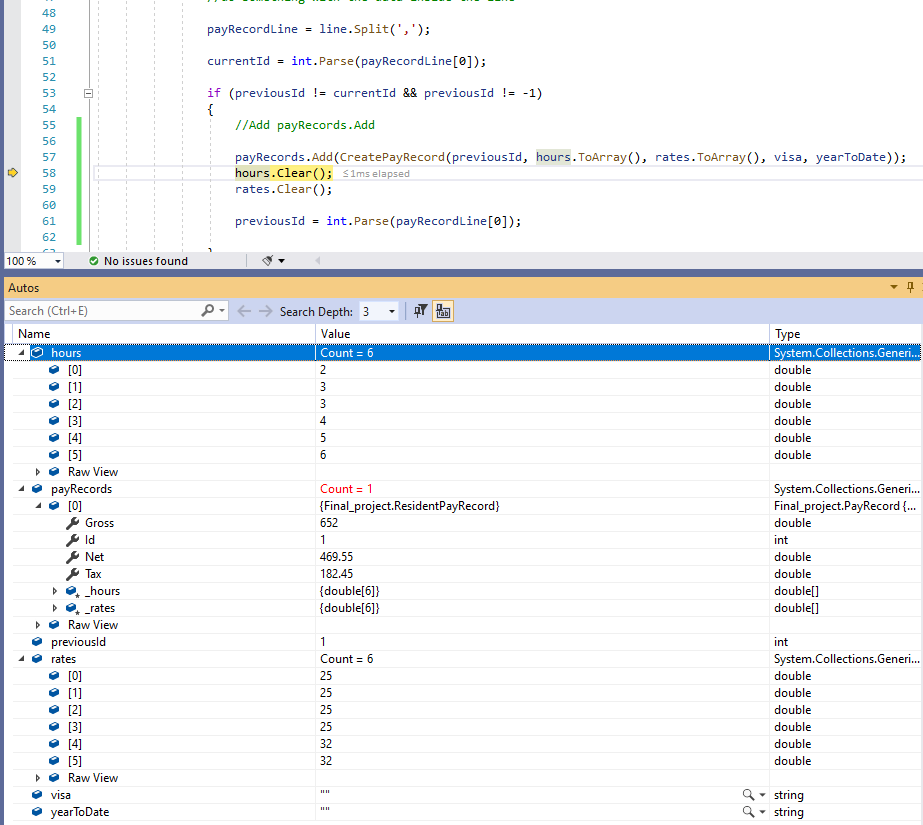
Add a new class to the project named *PayRecordWriter,* along with a method named *Write***.**

* 1. The *Write* method must accept a list of *PayRecord* objects and write the ID, Gross, Net and Tax amounts to a comma delimited values (.csv) file.
  2. Use the naming convention ‘{DateTimeNow.Ticks}-records.csv’.
  3. The *Write* method must have an optional Boolean parameter named *writeToConsole,* with the default value *false*. If a *true* argument is passed in, the *Write* method must also write the values to the console as shown in **Figure 4 - Writing Pay values to Console**.



Figure - Writing Pay values to Console

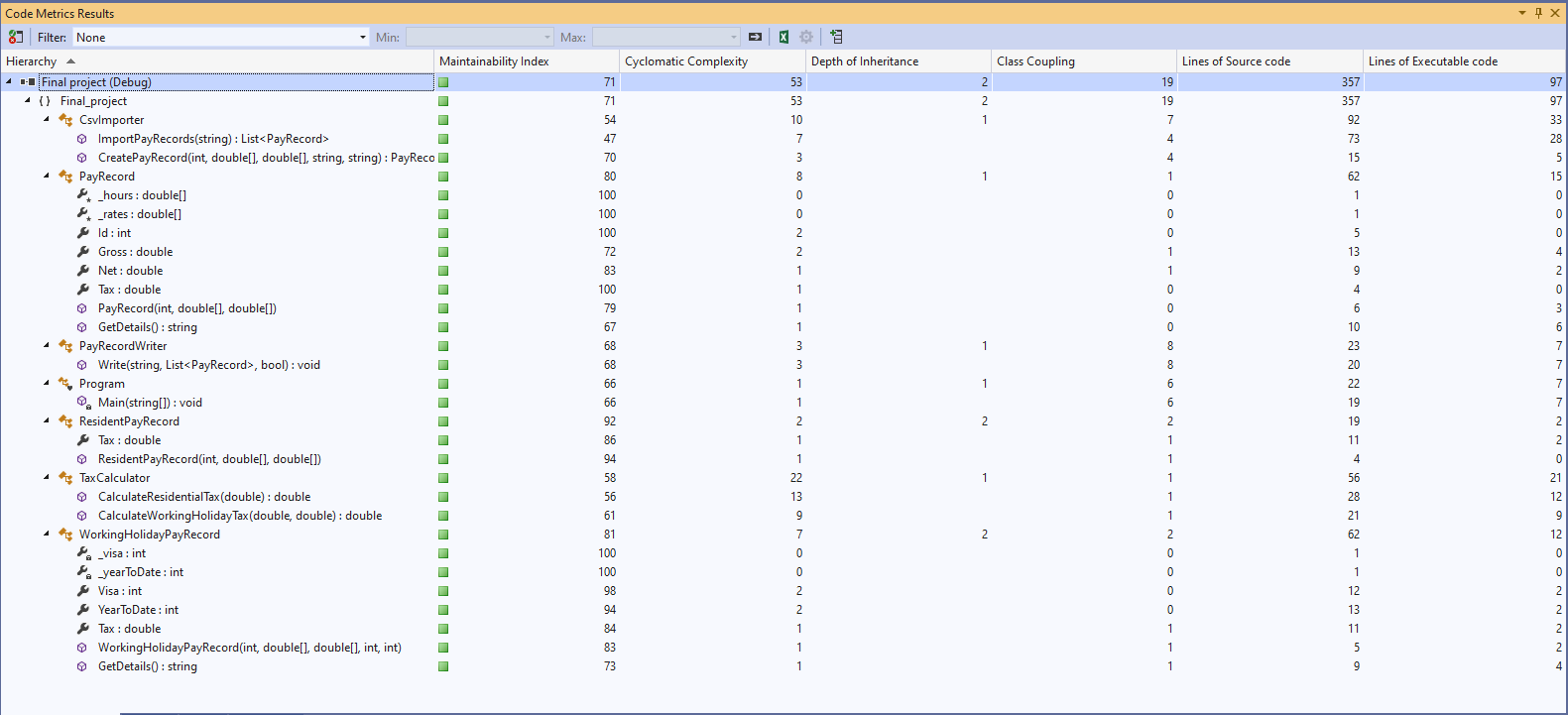
1. After completing the *Write* method, update the logic in *Main* method to invoke and test the Write method to ensure the reusable component is working as expected. Use a debugging tool and resolve any issues found. **(take a screenshot and paste it here)**



### Task 3

Generate appropriate code metrics for your software to help with its maintenance.

**Take a screen shot and paste it here**



### Task 4

Ensure that your code follows the coding standards and is documented according to the *Organisational development principles and practices* listed in the scenario.

Generate the documentation to an XML file named *MyPayProject.xml*.

<?xml version="1.0"?>

<doc>

<assembly>

<name>Final project</name>

</assembly>

<members>

<member name="T:Final\_project.CsvImporter">

<summary>

class contains methods to read csv and store data in list based on number of columns

</summary>

</member>

<member name="M:Final\_project.CsvImporter.ImportPayRecords(System.String)">

<summary>

method to read csv file line by line, compiles data based on ID number, returns list

</summary>

<param name="file"></param>

<returns></returns>

</member>

<member name="M:Final\_project.CsvImporter.CreatePayRecord(System.Int32,System.Double[],System.Double[],System.String,System.String)">

<summary>

method to create object based on class of employee

</summary>

<param name="id"></param>

<param name="hours"></param>

<param name="rates"></param>

<param name="visa"></param>

<param name="yearToDate"></param>

<returns></returns>

</member>

<member name="T:Final\_project.PayRecord">

<summary>

parent class containing properties to store hours and rates, calculate gross, net and tax; constructor for each payRecord; method to write details to console

</summary>

</member>

<member name="F:Final\_project.PayRecord.\_hours">

<summary>

array to store hours for each line in csv file

</summary>

</member>

<member name="F:Final\_project.PayRecord.\_rates">

<summary>

array to store rates for each line in csv file

</summary>

</member>

<member name="P:Final\_project.PayRecord.Id">

<summary>

property Id read from csv file

</summary>

</member>

<member name="P:Final\_project.PayRecord.Gross">

<summary>

property of PayRecord parent class, calculates gross earnings from parallel arrays \_hours and \_rates

</summary>

</member>

<member name="P:Final\_project.PayRecord.Net">

<summary>

property of PayRecord parent class, calculates net earnings

</summary>

</member>

<member name="P:Final\_project.PayRecord.Tax">

<summary>

property tax, value is calculated from TaxCalculator class based on type of pay record (resident or whv)

</summary>

</member>

<member name="M:Final\_project.PayRecord.#ctor(System.Int32,System.Double[],System.Double[])">

<summary>

constructor for PayRecord accepting id, hours, rates

</summary>

<param name="id"></param>

<param name="hours"></param>

<param name="rates"></param>

</member>

<member name="M:Final\_project.PayRecord.GetDetails">

<summary>

method to write calculated amounts to console

</summary>

<returns></returns>

</member>

<member name="T:Final\_project.PayRecordWriter">

<summary>

class contains method for writing to csv and console

</summary>

</member>

<member name="M:Final\_project.PayRecordWriter.Write(System.String,System.Collections.Generic.List{Final\_project.PayRecord},System.Boolean)">

<summary>

static method for writing calculated pay records to csv file and optionally console

</summary>

<param name="file"></param>

<param name="records"></param>

<param name="writeToConsole"></param>

</member>

<member name="M:Final\_project.Program.Main(System.String[])">

<summary>

calls method for writing records to csv and console from PayRecordWriter

</summary>

<param name="args"></param>

</member>

<member name="T:Final\_project.ResidentPayRecord">

<summary>

child class from PayRecord, contains property Tax to calculate total tax paid

</summary>

</member>

<member name="P:Final\_project.ResidentPayRecord.Tax">

<summary>

inherits from PayRecord, overrides abstract property Tax an calls appropriate method from TaxCalculator

</summary>

</member>

<member name="M:Final\_project.ResidentPayRecord.#ctor(System.Int32,System.Double[],System.Double[])">

<summary>

constructor calls base constructor, id, hours, rates

</summary>

<param name="id"></param>

<param name="hours"></param>

<param name="rates"></param>

</member>

<member name="T:Final\_project.TaxCalculator">

<summary>

class contains methods forcalculating tax based on class of employee

</summary>

</member>

<member name="M:Final\_project.TaxCalculator.CalculateResidentialTax(System.Double)">

<summary>

method to calculate tax for residents, returns double

</summary>

<param name="gross"></param>

<returns></returns>

</member>

<member name="M:Final\_project.TaxCalculator.CalculateWorkingHolidayTax(System.Double,System.Double)">

<summary>

method to return tax for working holiday visa, returns double

</summary>

<param name="gross"></param>

<param name="yearToDate"></param>

<returns></returns>

</member>

<member name="T:Final\_project.WorkingHolidayPayRecord">

<summary>

child class of PayRecord, contains constructor and method to write details with additional parameters

</summary>

</member>

<member name="F:Final\_project.WorkingHolidayPayRecord.\_visa">

<summary>

child class requires 2 additional properties, read from csv file

</summary>

</member>

<member name="P:Final\_project.WorkingHolidayPayRecord.Visa">

<summary>

property to read visa from csv and store it to private variable

</summary>

</member>

<member name="P:Final\_project.WorkingHolidayPayRecord.YearToDate">

<summary>

property to read yearToDate from csv and store it to private variable

</summary>

</member>

<member name="P:Final\_project.WorkingHolidayPayRecord.Tax">

<summary>

inherits from PayRecord, overrides abstract property Tax an calls appropriate method from TaxCalculator

</summary>

</member>

<member name="M:Final\_project.WorkingHolidayPayRecord.#ctor(System.Int32,System.Double[],System.Double[],System.Int32,System.Int32)">

<summary>

constructor contains additional arguments \_visa and \_yearToDate

</summary>

<param name="id"></param>

<param name="hours"></param>

<param name="rates"></param>

<param name="visa"></param>

<param name="yearToDate"></param>

</member>

<member name="M:Final\_project.WorkingHolidayPayRecord.GetDetails">

<summary>

method to write calculated amounts to console with visa details

</summary>

<returns></returns>

</member>

</members>

</doc>

## Part 4: Test the application

Develop and perform unit test cases to ensure that the application logic and syntax meet the user requirements and the re-use components work correctly within the project.

### Task 1

Before creating a test project, you need to create test data to calculate the correct calculated expected values for each employee pay record. The correct values for the first two employees have already been provided. Complete the Total YTD gross, Total gross, Tax and Net amounts for the remaining employees in the *Test data document template* **(Use the document - Software testing documentation.docx).**

Table 6 - Test Data

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Hours** | **Rate** | **Visa** | **Ytd** | **Gross** | **Total YTD + gross** | **Total gross** | **Tax** | **Net** |
| 1 | 2 | 25 |  |  | 50 | N**/**A | **652.00** | **182.45** | **469.55** |
| 1 | 3 | 25 |  |  | 75 |
| 1 | 3 | 25 |  |  | 75 |
| 1 | 4 | 25 |  |  | 100 |
| 1 | 5 | 32 |  |  | 160 |
| 1 | 6 | 32 |  |  | 192 |
| 2 | 2 | 25 | 417 | 47520 | 50 | **47938** | **418.00** | **133.76** | **284.24** |
| 2 | 2 | 25 | 417 | 47520 | 50 |
| 2 | 2 | 25 | 417 | 47520 | 50 |
| 2 | 2 | 25 | 417 | 47520 | 50 |
| 2 | 2 | 25 | 417 | 47520 | 50 |
| 2 | 2 | 28 | 417 | 47520 | 56 |
| 2 | 2 | 28 | 417 | 47520 | 56 |
| 2 | 2 | 28 | 417 | 47520 | 56 |
| 3 | 8 | 36 |  |  | 288 | N/A | **2202.00** | **754.91** | **1447.09** |
| 3 | 8 | 36 |  |  | 288 |
| 3 | 8 | 36 |  |  | 288 |
| 3 | 8 | 36 |  |  | 288 |
| 3 | 8 | 37.5 |  |  | 300 |
| 3 | 8 | 37.5 |  |  | 300 |
| 3 | 6 | 37.5 |  |  | 225 |
| 3 | 6 | 37.5 |  |  | 225 |
| 4 | 5 | 34.5 | 462 | 23000 | 172.5 | 24104.00 | **1104.00** | **165.60** | **938.40** |
| 4 | 5 | 34.5 | 462 | 23000 | 172.5 |
| 4 | 5 | 34.5 | 462 | 23000 | 172.5 |
| 4 | 5 | 34.5 | 462 | 23000 | 172.5 |
| 4 | 5 | 34.5 | 462 | 23000 | 172.5 |
| 4 | 5 | 34.5 | 462 | 23000 | 172.5 |
| 4 | 2 | 34.5 | 462 | 23000 | 69 |
| 5 | 7 | 42.5 |  |  | 297.5 | N/A | **1797.45** | **597.14** | **1200.31** |
| 5 | 6.5 | 42.5 |  |  | 276.25 |
| 5 | 7 | 42.5 |  |  | 297.5 |
| 5 | 7 | 42.5 |  |  | 297.5 |
| 5 | 7 | 42.5 |  |  | 297.5 |
| 5 | 3 | 55.2 |  |  | 165.6 |
| 5 | 3 | 55.2 |  |  | 165.6 |

### Task 2

1. Before performing the unit tests, you will need to set up the tests as follows:
   1. Add a NUnit test project named MyPayNUnitTestProject to the solution.
   2. Add a folder named Import to the project and copy in the provided employee-payroll-data.csv file as shown in Figure 5: MyPayNUnitTestProject.

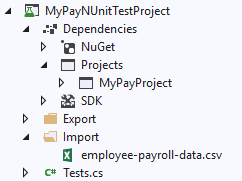


Figure : MyPayNUnitTestProject

* 1. Add a folder named Export to the test project.
  2. Add a reference to the project MyPayProject to the test project.
  3. Add a class named Tests.cs to the project
  4. Add a Setup method to the Test class that invokes the ImportPayRecords method and assigns the list of pay objects to a private field as shown in Figure 6: Test class.

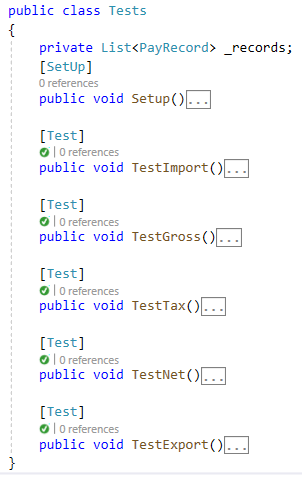


Figure : Test class

* 1. Add test methods to test the:
     + \_records field is not null and contains five objects
     + Gross amount calculated for each employee is correct
     + Tax amount calculated for each employee is correct
     + Net amount calculated for each employee is correct
     + writer successfully writes a file to the export folder by checking if the file exists.

1. Open *Test Explorer* and run all the tests to verify they pass**.** If any of the tests fail, debug the application and correct any errors until all the tests pass.

Document your test results in the *Test data document* (**Software testing documentation.docx**) from Part 4 Task 1:

* 1. **List the results in the Test case tables, including one or more screenshots of** the test being run and the debugging tool, as well as any errors and corrections made.
  2. Explain how you decided on the action to take when debugging your code.

**Place answers in the document**

* 1. Outline the key principles and concepts of the debugging tools and explain how you applied these when troubleshooting.

**Place answers in the document**

## Part 5: Evaluation

1. Before handing over your application to the client, write an evaluation report for the IT company in a word-processed document, using a style appropriate to the audience and purpose. Include the following in your report:
   1. Evaluate how well you have met the user requirements and specifications. **This may be listed in a table, such as:**

|  |  |
| --- | --- |
| Specification | Evaluation |
| [Specification as listed in the software specifications] | [How well does your application meet this specification?] |
|  |  |
|  |  |

* 1. Using examples from your code, identify different data types, operators and expressions that you used. Explain why you used them compared with other options.

**Include explanation here**

## Part 6: Perform handover

Communicate with the client to hand over the application. Your assessor will observe the communication and complete **Observation Checklist 2**.

Ensure that you include the following in your communication:

1. Present and demonstration the application to the client.
2. Obtain the client’s acceptance of the application.

## Observation Checklist 1

The Observation Checklist will be used by your assessor to mark your performance in Part 1. Use this Checklist to understand what skills you need to demonstrate in Part 1. The Checklist lists the assessment criteria used to determine whether you have successfully completed this assessment event. All the criteria must be met. Your demonstration will be used as part of the overall evidence requirements of the unit. The assessor may ask questions while the demonstration is taking place or if appropriate directly after the task/activity has been completed.

Table 2 Observation Checklist

| Task # | Task/Activity Performed | S | U/S | Assessor Comments (Describe the student’s ability in demonstrating the required skills and knowledge) |
| --- | --- | --- | --- | --- |
| **1** | Reviews and clarifies the user requirements with the user |  |  | *Date of Observation:* |

## Assessment Checklist 1

The following checklist will be used by your assessor to mark your performance against the assessment criteria of Part 2. Use this checklist to understand what skills and/or knowledge you need to demonstrate in your submission/presentation. All the criteria described in the Assessment Checklist must be met. The assessor may ask questions while the submission/presentation is taking place or if appropriate directly after the task/activity has been submitted/completed.

Table : Assessment Checklist 1

| TASK /STEP # | Instructions | S | U/S | Assessor Comments |
| --- | --- | --- | --- | --- |
| **1** | Applies development methodologies |  |  | *Date of Observation:* |
| **2** | Develops a class diagram to plan application design (T1) |  |  |  |
| **3** | Identifies systems and devices for future programming needs (T2) |  |  |  |
| **4** | Writes a report using broad vocabulary, correct grammar and appropriate conventions and style for the audience and purpose. (T3) |  |  |  |
| **5** | Gathers and analyses relevant information, identifies options, establishes evaluation criteria and evaluates options as required (T3 Q1, 2, 3, 5) |  |  |  |
| **6** | Tests re-use components (T3 Q4) |  |  |  |
| **7** | Identifies gaps in functionality of re-use components (T3Q6) |  |  |  |
| **8** | Considers technical impact on project (T3 Q7) |  |  |  |
| **9** | Considers licensing issues (T3 Q8) |  |  |  |
| **10** | Recommends re-use components (T3 Q9) |  |  |  |

## Assessment Checklist 2

The following checklist will be used by your assessor to mark your performance against the assessment criteria of Part 3. Use this checklist to understand what skills and/or knowledge you need to demonstrate in your submission/presentation. All the criteria described in the Assessment Checklist must be met. The assessor may ask questions while the submission/presentation is taking place or if appropriate directly after the task/activity has been submitted/completed.

Table : Assessment Checklist 2

| TASK  /STEP # | Instructions | S | U/S | Assessor Comments |
| --- | --- | --- | --- | --- |
| **1** | Applies development methodologies |  |  | *Date of Observation:* |
| **2** | Develops code using correct syntax, data types and mathematical language, as required |  |  |  |
| **3** | Develops application following coding conventions and object-oriented methodologies |  |  |  |
| **4** | Configures development environment to incorporate re-use component (T1 Q1-4) |  |  |  |
| **5** | Adds re-use component to project (T1 Q5) |  |  |  |
| **6** | Configures development environment to incorporate re-use components from repository (T2 Q1, 2) |  |  |  |
| **7** | Tests re-use component functionality using a debugging tool and resolves re-use dependencies (T2 Q3) |  |  |  |
| **9** | Applies techniques for collecting metrics (T3) |  |  |  |
| **10** | Documents application following documentation conventions (T4) |  |  |  |

## Assessment Checklist 3

The following checklist will be used by your assessor to mark your performance against the assessment criteria of Part 4. Use this checklist to understand what skills and/or knowledge you need to demonstrate in your submission/presentation. All the criteria described in the Assessment Checklist must be met. The assessor may ask questions while the submission/presentation is taking place or if appropriate directly after the task/activity has been submitted/completed.

Table : Assessment Checklist 3

| TASK /STEP # | Instructions | S | U/S | Assessor Comments |
| --- | --- | --- | --- | --- |
| **1** | Applies development methodologies |  |  | *Date of Observation:* |
| **2** | Develops tests to meet requirements and specifications, including functionality of re-use components |  |  |  |
| **3** | Runs two unit test cases and modifies application |  |  |  |
| **4** | Documents tests following documentation conventions |  |  |  |
| **5** | Uses analytical processes when debugging |  |  |  |
| **6** | Understands key principles and concepts of debugging tools (T2) |  |  |  |

## Assessment Checklist 4

The following checklist will be used by your assessor to mark your performance against the assessment criteria of Part 5. Use this checklist to understand what skills and/or knowledge you need to demonstrate in your submission/presentation. All the criteria described in the Assessment Checklist must be met. The assessor may ask questions while the submission/presentation is taking place or if appropriate directly after the task/activity has been submitted/completed.

Table : Assessment Checklist 4

| TASK  /STEP # | Instructions | S | U/S | Assessor Comments |
| --- | --- | --- | --- | --- |
| **1** | Reviews and ensures application meets user requirements and specifications, evaluating effectiveness of decisions |  |  | *Date of Observation:* |
| **2** | Determines and evaluates different data types, operators and expressions |  |  |  |

## Observation Checklist 2

The Observation Checklist will be used by your assessor to mark your performance in any of the previous three event types. Use this Checklist to understand what skills you need to demonstrate in Part 6. The Checklist lists the assessment criteria used to determine whether you have successfully completed this assessment event. All the criteria must be met. Your demonstration will be used as part of the overall evidence requirements of the unit. The assessor may ask questions while the demonstration is taking place or if appropriate directly after the task/activity has been completed.

Table 2 Observation Checklist

| Task # | Task/Activity Performed | S | U/S | Assessor Comments (Describe the student’s ability in demonstrating the required skills and knowledge) |
| --- | --- | --- | --- | --- |
| **1** | Presents and demonstrates the application to the user |  |  | *Date of Observation:* |
| **2** | Obtains user acceptance |  |  |  |

## Assessment Feedback

### Assessment outcome

Satisfactory

Unsatisfactory

**Assessor feedback**

☐ Has the Assessment Declaration been signed and dated by the student?

☐ Are you assured that the evidence presented for assessment is the student’s own work?

Was the assessment event successfully completed?

If no, was the resubmission/re-assessment successfully completed?

Was reasonable adjustment in place for this assessment event?  
*If yes, ensure it is detailed on the assessment document.*

Comments:

### Assessor name, signature and date:

### Student acknowledgement of assessment outcome

Would you like to make any comments about this assessment?

### Student name, signature and date

***NOTE: Make sure you have written your name at the bottom of each page of your submission before attaching the cover sheet and submitting to your assessor for marking.***