

CAB201 Programming Principles

Assignment: Major Project - Mates_Rates_Rent_a_Car (MRRC)

Semester 1, 2018

Due Date: {27/05/2018 20:00}

Weighting: 40%

Assessment type: Individual, with optional pairs

Specification version: Version 0.2 (28th April 2018)

*In this assignment you have the option of working in pairs. If you work alone, you will not receive any special consideration. Programming pairs need to be registered with the CAB201 teaching team by email (cab201admin@qut.edu.au) **before 2/5/2018 23:59**.*

INTRODUCTION

This assignment aims to give you a real problem-solving experience, similar to what you might encounter in the workplace. You have been hired to complete a prototype for a project that has not been fully specified at this stage. There is some supporting documentation to the project in addition to this document. You are required to deliver a working prototype by the required date.

THE TASK

For this assignment your task is to develop a program for a car rental company called Mates Rates Rent a Car (MRRC) to help the operators of the company more efficiently manage their fleet of cars and their customers.

The operators of MRRC require the following functionality:

1. Enter/modify/delete customers
2. Enter/modify/delete cars
3. Search for suitable vehicles based on customers' requests
4. Rent out cars to customers
5. Return cars from customers
6. Display a rented cars report

It offers vehicles in 4 classes –

- Economy
- Family
- Luxury
- Commercial

Vehicles have various traits. A Vehicle class defines the shared properties and methods of all cars and support specialised properties and methods. For instance, luxury cars have GPS and opening roof, and commercial vehicles may have a diesel engine. Number of seats in cars may vary from 2 to 10 (fixed per car though.) Each car has a (possibly) unique rental cost per day.

Customers have various traits. A Customer class defines the shared properties and methods of all customers. Attributes required are Name (with suitably defined sub-fields), and Date of birth (with suitably defined sub-fields).

These classes are used by:

A Fleet class manages the rental of vehicles. The fleet class should load the vehicle fleet from file on start-up. This class should be based on an array of Vehicle objects. It should support operations on the fleet (e.g. add/modify/delete)

A CRM class manages the collection of customers. The CRM class should load the customers from file on start-up. This class should be based on an array of Customer objects. It should support operations on the Customer (e.g. add/modify/delete)

Both the Fleet and CRM classes should save their data to disk when the program is closed by the operator.

Upon start-up, the program has to load the Fleet (vehicles) and CRM (customers) from permanent storage on disk files. During operation the in-memory system is used to manage all operations. When the program is finished it must save the Fleet and CRM data to disk.

NOTE: There is no requirement to back up the data to disk in case the program is not terminated correctly. However, the program should work correctly if started and stopped any number of times, and preserve the state of the CRM and Fleet.

Graphical User Interface:

You are required to create a graphical user interface for this program using Windows Forms. The design of this GUI is largely up to you, however it should be functional and relatively simple, and should allow operators to complete all required functions.

The program should allow the operator to

1. Enter a new customer
2. Modify an existing customer
3. Delete a customer – but not if having a car rented at present

4. Enter a new vehicle
5. Modify an existing vehicle
6. Delete a vehicle – but not if the vehicle is rented at present

7. Search for a suitable car based on a customer's request (see the search capabilities section)
8. Rent a car to a customer
9. Return a car from a customer

10. Produce a report (to the screen) of all rented vehicles with option to:
 - a. order by customer name
 - b. order by vehicle registration

It is *required* that you use **Windows Forms**, rather than any other GUI technologies such as WPG, XNA, or web pages. This means that Visual Studio for Mac and MonoDevelop will likely be insufficient – you will need to ensure that you have access to Visual Studio in order to work on this assignment.

You must test your program in one of the CAB201 labs prior to submission. Markers will attempt to compile and run your code in that environment only. It is your responsibility to ensure that your code compiles and runs on PCs in the QUT computer labs of CAB201 as the target platform.

An example Windows Form will be released shortly, with some tips on how to use appropriate form components to build your application.

Search capabilities:

The search capabilities of the MRRC program offers various levels of functionality.

The following are just some examples.

Many more queries of the same types are possible.

There could be more/other attributes in queries.

Queries are in free text form, but only allow attribute values, and the operators AND,OR,().

(Note - queries should not be case sensitive).

Example queries:

1. Any
offer any available car in the fleet
2. Any vehicle at given cost range
offer any available car in the fleet within a specified cost range
3. Single attribute query
Red
Family
8-Seater
6-cylinders
V8
GPS
(any single attribute – assume each attribute name is unique in the entire system)
4. A choice between two attributes (disjunction)
Family OR Luxury
Red OR Blue
GPS OR V8
5. A combination of two attributes (conjunction)
Red AND 4-cylinders
GPS AND V8
6. A choice between any number of attributes (disjunction)
Economy OR Family OR Luxury
Red OR Blue OR Green OR Purple
6-cylinders OR V8 OR Luxury
7. A combination of any number of attributes (conjunction)
Red AND 4-cylinders AND GPS
GPS AND V8 AND Red AND Commercial
8. A combination of any number of attributes, using both AND and OR with AND having the same priority (precedence) as OR
Economy OR Family AND 4-Cylinders
Red OR Blue OR Green OR Purple AND GPS
9. A combination of any number of attributes, using AND and OR with AND having a higher priority than OR, supporting parenthesis to resolve priority
((GPS AND V8) OR (Red OR Green)) AND Commercial OR Luxury

The system should allow the operator to select a vehicle from the search results and rent it to a customer. Calculate cost based on rate and duration of rental.

Rented vehicles cannot be rented again until returned.

The system should allow the customer to return a vehicle at any time.

CLASS SPECIFICATION

The following provides a partial specification of the classes required to create a prototype for the MRRC program. For all classes you may implement additional private methods and parameters. You may also implement additional public methods and parameters where appropriate and necessary to complete the functionality. For example, you may want to implement additional public methods in the Vehicle class which assist in the search functionality.

For each class, getters and setters may be replaced with appropriate Properties. In general getters and setters are not specified.

public class Vehicle

Vehicle
<ul style="list-style-type: none">- vehicleRego: string- make: string- model: string- year: int- vehicleClass: VehicleClass- numSeats: int- transmission: TransmissionType- fuel: FuelType- GPS: bool- sunRoof: bool- dailyRate: double- colour: string
<ul style="list-style-type: none">+ Vehicle(vehicleRego string, Class class, string make, string model, int year)+ Vehicle(vehicleRego string, Class class, string make, string model, int year, int numSeats, TransmissionType transmissionType, FuelType fuelType, bool GPS, bool sunRoof, double dailyRate, string colour)+ ToCSVString(): string+ ToString(): string+ GetAttributeList: List<string>

The vehicle class should have appropriate getters and setters for each attribute, or C# Properties can be used. You may want to add additional methods to enable searching.

VehicleClass, FuelType, and TransmissionType should be appropriately defined enums.

```
public Vehicle(string vehicleRego, VehicleClass vehicleClass, string make, string model, int year, int numSeats, TransmissionType transmissionType, FuelType fuelType, bool GPS, bool sunRoof, double dailyRate, string colour)
```

This constructor provides values for all parameters of the vehicle.

```
public Vehicle(vehicleRego string, Class class, string make, string model,  
int year)
```

This constructor provides only the mandatory parameters of the vehicle. Others are set based on the defaults of each class.

Overall defaults: Unless otherwise specified by the class, vehicles default to four seats, automatic transmission, petrol fuel, no GPS, no sun roof, and a black colour.

Economy vehicles: have automatic transmission only.

Default rental cost per day is \$50

Family vehicles: can have manual or automatic transmission. Default rental cost per day is \$80

Luxury vehicles: has GPS and a sunroof. Default rental cost per day is \$120

Commercial vehicle: has diesel engine by default. Default rental cost per day is \$130

```
public string ToCSVString()
```

This method should return a CSV representation of the vehicle that is consistent with the provided data files.

```
public override string ToString()
```

This method should return a string representation of the attributes of the vehicle.

```
public List<string> GetAttributeList()
```

This method should return a list of strings which represent each attribute. Values should be made to be unique, e.g. numSeats should not be written as '4' but as '4-Seater', sunroof should not be written as 'True' but as 'sunroof' or with no string added if there is no sunroof. Vehicle rego, class, make, model, year, transmission type, fuel type, daily rate, and colour can all be assumed to not overlap (i.e. if the make 'Mazda' exists, 'Mazda' will not exist in other attributes e.g. there is no model named 'Mazda'. Similarly, if the colour 'red' exists, there is no 'red' make. You do not need to maintain this restriction, only assume it is true.)

An alternative approach can be taken to the Vehicle class for additional marks. In this approach, the Vehicle class should be an abstract class, and each Class of car (economy, family, luxury, commercial) should extend the Vehicle class. Appropriate constructors should be set which make use of the default values of the different classes.

public class Customer

Customer
<ul style="list-style-type: none">- customerID: int- title: string- firstNames: string- lastName: string- gender: Gender- dateOfBirth: string
<ul style="list-style-type: none">+ Customer(int customerID, string title, string firstNames, string lastName, Gender gender, string dateOfBirth)+ ToCSVString(): string+ ToString(): string

The Customer class should have appropriate getters and setters for each attribute, or C# Properties can be used. You may want to add additional methods to enable your approach to searching.

Gender should be an appropriately defined enum.

```
public Customer(int customerID, string title, string firstNames, string  
lastName, Gender gender, string dateOfBirth)
```

This constructor should construct a customer with the provided attributes.

```
public string ToCSVString()
```

This method should return a CSV representation of the customer that is consistent with the provided data files.

```
public override string ToString()
```

This method should return a string representation of the attributes of the customer.

public class Fleet

Fleet
- vehicles: List<Vehicle> - rentals: Dictionary<string, int> - fleetFile: string - rentalFile: string
+ Fleet() + AddVehicle(Vehicle vehicle): bool + RemoveVehicle(Vehicle vehicle): bool + RemoveVehicle(string vehicleRego): bool + GetFleet(): List<vehicle> + GetFleet(bool rented): List<vehicle> + IsRented(string vehicleRego): bool + IsRenting(int customerID): bool + RentedBy(string vehicleRego): int + RentCar(string vehicleRego, int customerID): bool + ReturnCar(string vehicleRego): bool + SaveToFile() + LoadFromFile() ---- Missing – search functions

A Fleet class manages the rental of vehicles. The fleet class should load the vehicle fleet from file on start-up. This class should be based on list of Vehicle objects. It should support operations on the fleet, such as adding, modifying and deleting vehicles. It should also handle the renting and returning of vehicles.

You may adjust some private variables or methods if they better suit your approach. For example, you may prefer to take a different approach to the Dictionary with the vehicle rego and customerID for maintaining which cars are being rented.

The methods needed to implement the search functionality have not been specified here. It is up to you to come up with your own approach.

The program must make sure that data entry makes sure that VehicleRego is unique (no duplicates allowed) and that it only has 6 alphanumeric characters (i.e. in [A-Z,0-9]) when entered from the keyboard. You can assume that data on disk (i.e. read from files) is always valid.

public Fleet()

If there is no fleet file at the specified location, this constructor constructs an empty fleet and empty rentals. Otherwise it loads the fleet and rentals from the appropriate files.

public bool AddVehicle(Vehicle vehicle)

This method adds the provided vehicle to the fleet assuming the vehicle registration does not already exist. It returns true if the add was successful (the vehicleRego wasn't already in the fleet), and false otherwise (a car with that rego was already in the fleet).

public bool RemoveVehicle(Vehicle vehicle)

This method removes the provided vehicle from the fleet if it is not currently rented. It returns true if the removal was successful and false otherwise.

public bool RemoveVehicle(string vehicleRego)

This method removes the vehicle with the provided rego from the fleet if it is not currently rented. It returns true if the removal was successful and false otherwise.

public List<Vehicle> GetFleet()

This method returns the fleet.

public List<Vehicle> GetFleet(bool rented)

This method returns a subset of vehicles in the fleet depending on whether they are currently rented. If rented is true, this method returns all rented vehicles. If it false, this method returns all not yet rented vehicles.

public bool IsRented(string vehicleRego)

This method returns whether the vehicle with the provided registration is currently being rented.

public bool IsRenting(int customerID)

This method returns whether the customer with the provided customer ID is currently renting a vehicle.

public int RentedBy(string vehicleRego)

This method returns the customer ID of the current renter of the vehicle. If it is rented by no one, it returns -1.

public bool RentCar(string vehicleRego, int customerID)

This method rents the vehicle with the provided registration to the customer with the provided ID, if the vehicle is not currently being rented. It returns true if the rental was possible, and false otherwise.

public int ReturnCar(string vehicleRego)

This method returns a vehicle. If the return is successful (it was currently being rented) it returns the customer ID of the customer who was renting it, otherwise it returns -1.

public void SaveToFile()

This method saves the current state of the fleet to file.

public void LoadFromFile()

This method loads the state of the fleet from file.

public class CRM

Customer
- customers: List<Customer> - crmFile: string
+ CRM() + AddCustomer(Customer customer): bool + RemoveCustomer(Customer customer, Fleet fleet): bool + RemoveCustomer(int customerID, Fleet fleet): bool + GetCustomers(): List<Customers> + SaveToFile() + LoadFromFile()

A CRM class manages the collection of customers. The CRM class should load the customers from file on start-up. This class should be based on a list of Customer objects. It should support operations on the Customer (e.g. add/modify/delete).

The program must make sure that data entry makes sure that customerID is unique (no duplicates allowed). You can assume that data on disk (i.e. read from files) is always valid. You may want to implement this by automatically generating a unique customerID (for example, you might want to keep track of the current highest customer ID).

public CRM()

If there is no CRM file at the specified location, this constructor constructs an empty CRM with no customers. Otherwise it loads the customers from file.

public bool AddCustomer(Customer customer)

This method adds the provided customer to the customer list if the customer ID doesn't already exist in the CRM. It returns true if the addition was successful (the customer ID wasn't already in the CRM) and false otherwise (the customer ID was already in the CRM).

public bool RemoveCustomer(Customer customer, Fleet fleet)

This method removes the customer from the CRM if they are not currently renting a vehicle. It returns true if the removal was successful, otherwise it returns false.

public bool RemoveCustomer(int customerID, Fleet fleet)

This method removes the customer from the CRM if they are not currently renting a vehicle. It returns true if the removal was successful, otherwise it returns false.

public List<Customers> GetCustomers()

This method returns the list of current customers.

public void SaveToFile()

This method saves the current state of the CRM to file.

public void LoadFromFile()

This method loads the state of the CRM from file.

.

RECOMMENDED APPROACH

Tackling a large project such as this assignment may initially seem daunting, however breaking it down into smaller tasks that can be implemented and tested will make the task much more manageable and is an important skill for professional practice. You will develop this assignment as a **bottom-up implementation exercise**. This means that you will work based off a specification that has already been prepared and implement classes and methods, one by one, first implementing those classes and methods that don't rely on other classes to operate, then implementing the classes and methods that rely on those and so forth.

This assignment specification has been released as we cover the development of Windows Forms and event driven programming. There is a significant amount of work that can (and should) be done prior to beginning work on the GUI. It is strongly recommended that you attend the practicals for support. Begin with writing the simplest classes one by one, and creating a Console application in which you can test each class and its methods as you implement them. All functionality of the assignment can be tested via a console application – the GUI simply provides a wrapper to this functionality.

Note that the GUI is often developed separately from the detailed functionality, by different programmers, as a team effort. It is here where the value of OOP is demonstrated. A good place to start is the definition of the CRM and Vehicle classes, and load/save the data to disk, with a console interaction while you test it. The console interaction can later be implemented in a GUI.

A short guide on the best method to approach this is provided in the appendix.

Academic Integrity

Please read and follow the guidelines in QUT's Academic Integrity Kit, which is available from the Blackboard site on the Assessment page. Programs submitted for this assignment will be analysed by the MoSS (Measure of Software Similarity) plagiarism detection system (<http://theory.stanford.edu/~aiken/moss/>).

Final Comment

Though all care has been taken in the production of this specification and related documentation, there may be a need to notify by email any alterations/clarifications to this specification and related documentation.

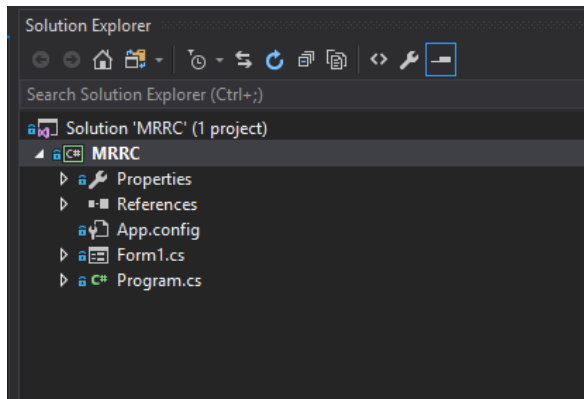
SO CHECK YOUR QUT EMAIL DAILY

APPENDIX – SETTING UP YOUR PROJECT IN VISUAL STUDIO

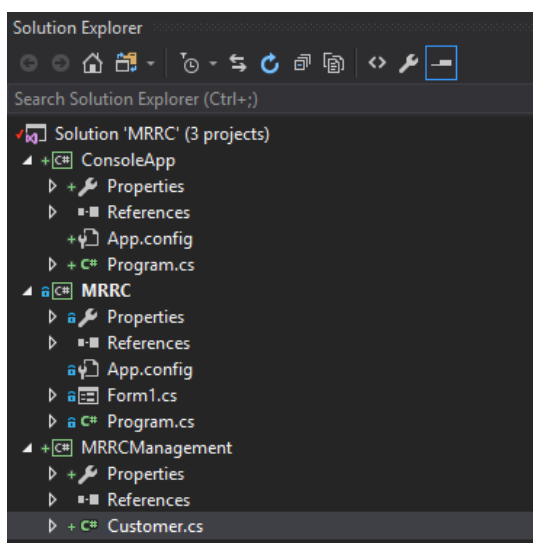
The following is a short guide to setting up your project in Visual Studio so that you can share the classes between a console application for testing and the eventual GUI.

1. Create a new project when starting up Visual Studio, selecting 'Windows Form App (.NET Framework)' and calling it 'MRRC'. This will set up the GUI application, which we'll come back to later.

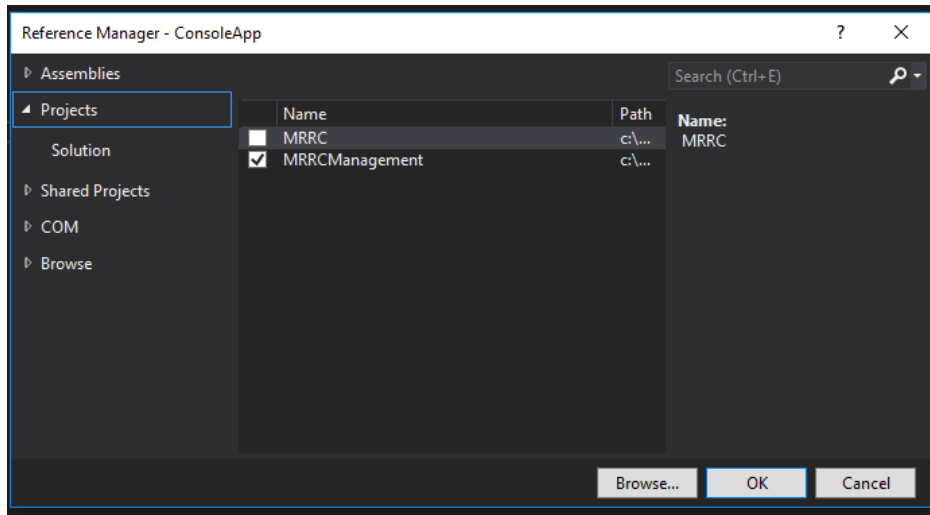
Your solution explorer should now look like this:



2. Right click on 'Solution 'MRRC'' and click 'Add -> New Project...'. From this screen, select 'Class Library (.NET Framework)' (Not .NET Standard!). Call this class library 'MRRCManagement'. You should add your Vehicle, Customer, CRM, and Fleet classes into here. These classes can then be shared between different projects so they are separated from the interface. Make sure these classes are declared public.
3. Right click on 'Solution 'MRRC'' and click 'Add -> New Project...'. From this screen, select 'Console App (.NET Framework)' and name it appropriately (e.g. ConsoleApp). You can now use this Console project to test your classes in a console application before moving onto the GUI.



4. To use the classes in the console project or in the GUI project, you will need to add a reference to the library. Right click on the project you want to add this to, e.g. the ConsoleApp, and click 'Add Reference'. This will open the reference manager. Click 'Projects' from the side bar, then tick the library you want to add (in this case MRRCManagement), then click 'OK'.



You can now use this library in your code by adding 'using MRRCManagement' to the other directive statements at the top of your code. You should now be able to use any classes from within that library if they were in the same namespace.





If you have issues, check that your classes in the library are public, that you have included the reference to the library, and that it is listed in the build dependencies (Right click on Project -> 'Build Dependencies' -> 'Project Dependencies' and tick the box if it hasn't already been ticked)

You can right-click on a project (either the GUI or the Console App) and select 'Set as StartUp Project' to change which project runs when you click 'Start' or debug.





APPENDIX – ADDING DATA AND LINKING TO FILES

To be able to load the data on the fleet and the vehicles, you need to be able to link to the file. The following is instructions on where to include the data files and how to reference them in your code.

1. Once you have created your project as explained above, open the folder on your computer that contains the visual studio project. This should look something like this:

Name	Date modified	Type
 MRRManagement	28/04/2018 6:19 AM	File folder
 MRRC	28/04/2018 6:20 AM	File folder
 ConsoleTester	26/04/2018 10:20 ...	File folder
 MRRC.sln	26/04/2018 10:23 ...	Visual Studio Solu.

2. Create a new folder named 'Data'

 MRRManagement	28/04/2018 6:19 AM	File folder
 MRRC	28/04/2018 6:20 AM	File folder
 Data	26/04/2018 10:41 ...	File folder
 ConsoleTester	26/04/2018 10:20 ...	File folder
 MRRC.sln	26/04/2018 10:23 ...	Visual Studio Solu...

3. Place the provided csv files inside this Data folder.
4. In your Fleet or Customer class, you can now link to this file like so –

```
private string fleetFile = @"..\..\Data\fleet.csv";
```

Files are loaded relative to where the program is executed, so this method will work with executables from both the Debug and Release folders.