7SENG011W Object Oriented Programming – Coursework (2023/24)	
Module Leader	FRANCESCO TUSA (F.TUSA@WESTMINSTER.AC.UK)
Unit	Coursework
Weighting	50%
Description	Design and Development of a *Vehicle Rental Software System*
Learning Outcomes Covered in this Assignment	This assignment contributes towards the following Learning Outcomes (LOs): *(LO1)* Demonstrate an understanding of a significant object-oriented programming language and development environment. *(LO2)* Critically evaluate the extensive classes provided by the language and demonstrate their use in the design and construction of an application. *(LO3)* Demonstrate a thorough understanding of a contemporary framework (such as the .NET framework), together with a critical understanding of the issues and problems associated with the use of third-party packages and frameworks. *(LO4)* Critically evaluate and communicate their work by both written and oral means.
Handed Out Date	Wednesday, 22nd of November 2023
Due Date	**Thursday, 4th of January 2024—submissions by 1 pm via Blackboard**
Expected deliverables	A zip file containing: • The system Class Diagram and Design Report; • A folder with the Visual Studio Project file (.csproj) and the C# source files (.cs only) of your software project; • Reference to a recorded and accessible video demonstration of the implemented system.
Submission Method	Electronic submission on Blackboard via the provided link. The file should have the following naming format: *7SENG011W _StudentNumber_firstName_lastName.zip*
Type of Feedback and Due Date	Electronic feedback and marks will be provided via the module's Blackboard board within *15 working days* from the submission deadline. **All marks will remain provisional until formally agreed by an Assessment Board**

Assessment regulations

Please refer to the *University Assessment Regulations* on the following website:

http://www.westminster.ac.uk/study/current-students/resources/academic-regulations

for a detailed description of *how you are assessed*, *penalties* and *late submissions*, what constitutes ***plagiarism***, etc.

Penalty for Late Submission

If you submit your coursework late but within 24 hours or one working day of the specified deadline, ten marks will be deducted from the final mark as a penalty for late submission, except for work which obtains a mark in the range of 50-59%, in which case the mark will be capped at 50%. Suppose you submit your coursework more than 24 hours or more than one working day after the specified deadline. In that case, you will be given a zero mark for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.

Mitigating Circumstances

It is recognised that, on occasion, illness or a personal crisis can mean that you fail to submit a work on time. In such cases, you must inform the Campus Office in writing on a mitigating circumstances form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the appropriate Assessment Board, which decides whether the zero mark will stand.

Plagiarism

By submitting the work through Blackboard, you acknowledge that this is solely your work. Any code not created by you MUST be commented as such. Any code discovered not to have been created by you will mean that the work will be submitted to academic standards for a potential assessment offence, which may result in a zero mark in the component or whole module.

Coursework Description

Problem Statement

This coursework assesses the knowledge and skills you acquired about Object Oriented Programming (OOP) throughout the module. The specific problem to be solved is related to the design and implementation of a *Vehicle Rental Software System* that allows:

- the management of different (types of) vehicles owned by the vehicle rental company;
- the creation, amendment, and cancellation of vehicle rental bookings.

Two types of users will interact with the system via a text menu that provides them with access to specific functionalities:

- an *admin* can add and delete vehicles, print vehicle information, etc;
- a *customer* can use the system to check vehicles available on particular dates and make bookings accordingly.

For simplicity, the implementation of registration and login functionalities for the above users is not requested.

System Design and Documentation (20 marks)

As part of the coursework, you are asked to analyse the problem, identify a set of required system functionalities, and provide the related design and implementation. **The submission of the following two items is mandatory**.

UML Class Diagram (mandatory—10 marks)

A UML Class Diagram that describes the OOP system design. The marking of the diagram will be done based on the following criteria:

- all the required entities are correctly represented with their relevant attributes and methods;
- object relationships (i.e., associations, aggregations, compositions) and their multiplicities, as well as the class relationships (i.e., generalisations), are modelled correctly;
- the design is consistent with the submitted software implementation.

Project Design Report (mandatory—10 marks)

A report (expected length no more than 1000 words) that highlights your reflection on the design process, i.e., the role of each class, interface, abstract class, of their members and relationships. **A template for the report is available on Blackboard**.

System Functionalities (implementation: 80 marks)

Modelling of Vehicles (10 marks)

The rental company owns different types of *Vehicle*s: *Van*s, *Car*s, *ElectricCar*s and *Motorbike*s. These vehicles have a registration *number*, *make, model*, and daily rental *price*. In addition to those items, each vehicle can also have other relevant features that you should identify and model accordingly. OOP principles, i.e., *abstraction*, *encapsulation*, *inheritance*, and *polymorphism*, should be used when modelling the vehicles.

Renting a Vehicle (10 marks)

The system should keep track of the dates when each *Vehicle* has been booked. It should be assumed that a reservation is described by a *Schedule* consisting of a *pick-up date* and a *drop-off date*. A vehicle pick-up is expected at the opening time of the rental branch. Similarly, the drop-off is intended just before the closing time of the branch. Therefore, for simplicity, only consider the date as a combination of day, month, and year and do not include hours and minutes.

A *Vehicle* can have multiple reservations if the related *Schedule*s do not overlap. For instance, *vehicle1* can have two bookings: *schedule1* with pick-up 10/11/2023, drop-off 15/11/2023 and *schedule2* with pick-up 16/11/2023, drop-off 18/11/2023. Define an interface *IOverlappable* with a method:

bool Overlaps(Schedule other)

for checking whether two *Schedule* objects overlap.

When a reservation is made based on a *Schedule*, information about the associated *Driver* must be added to the system, namely the driver's *name*, *surname*, *date of birth*, and *license number*.

Rental System Admin and User Functionalities (60 marks)

A class *WestminsterRentalVehicle* contains the main logic of the program and implements the interfaces *IRentalManager* and *IRentalCustomer*, whose functionalities are reported below. Proper methods must be implemented in the class to fulfil the design contract specified by those interfaces.

IRentalManager (25 marks)

Defines the operations that the admin can perform.

bool AddVehicle(Vehicle v)
To add a new vehicle into the rental system. Assuming 50 parking lots are available in the rental branch vehicle park, a message with the number of available parking lots should be displayed after a successful add operation. Please note that the system should not allow duplicate entries (i.e., a vehicle with a given registration number should be added once).

(3 marks)

bool DeleteVehicle(string number)
To delete a vehicle identified by *number* from the system. It displays a message with the deleted vehicle information and the number of parking spaces available in the garage.

(3 marks)

void ListVehicles()
To print the list of the vehicles in the system. For each vehicle, information on the registration *number*, the type of vehicle and the reservation *Schedule*s have to be displayed.

(2 marks)

void ListOrderedVehicles()
Like *ListVehicles* above, however, the output should be ordered alphabetically according to the vehicle *Make*. This must be implemented using *IComparable* or *IComparer*—no other solutions will be accepted.

(7 marks)

void GenerateReport(string fileName)
To save the current list of vehicles with their complete related information to a text file. Each entry should include the list of bookings for the vehicle with the driver's details. The bookings will be sorted in chronological order according to the start date.

(10 marks)

IRentalCustomer (25 marks)

Defines the operations that a customer can perform.

void ListAvailableVehicles(Schedule wantedSchedule, Type type)
To list the information relevant for the vehicles of a given *type* (understand and use the *System.Type* class for this) that are available on a specific *wantedSchedule.*

(7 marks)

bool AddReservation(string number, Schedule wantedSchedule)
To make a reservation for a *Vehicle*, identified by *number*, on a *Schedule wantedSchedule.* When a reservation is created for a vehicle, the associated total price should be calculated and added to the schedule based on the daily price of the selected vehicle.

(7 marks)

bool ChangeReservation(string number, Schedule oldSchedule, Schedule newSchedule)
To modify the start and/or end date of an existing reservation for the vehicle identified by *number*. The modification can be applied only if the new requested schedule does not overlap with existing ones for the vehicle. Otherwise, the modification must be rejected.

(8 marks)

bool DeleteReservation(string number, Schedule schedule)
To delete an existing reservation for a vehicle identified by *number* on a given *schedule*.

(3 marks)

Text Menu (10 marks)

Once started, the program should show on the screen a *text menu* for:

- the *customer*, with the functionalities defined by the *IRentalCustomer* interface, plus an additional option to access the *admin* menu;

  (4 marks)
- the *admin*, with the functionalities of the *IRentalManager* interface and the option to go back to the *user* menu.

  (6 marks)

Coding Standards and Robustness

The marking of the above system functionalities will not only consider the *correctness* of the code, i.e., whether the program produces the correct output for each set of input data.

The *code robustness*, i.e., *error handling, input validation,* and adherence to well-known *coding standards* (readability, proper use of comments and choice of identifiers), **will be integral parts of the marking process**.

Video with System Explanation and Demonstration (mandatory)

A pre-recorded **video with voiceover—not exceeding 15 minutes**—must be submitted. If the video's file size exceeds the file upload size limit of Blackboard, it is strongly advised to share your video using a cloud storage service, e.g., OneDrive or Google Drive, or a streaming service such as YouTube. **In this case, the video access link must be included in the submission**.

To receive the marks corresponding to a functionality you implemented, a demonstration (test) of that functionality and an explanation of the related code implementation must be included in the video.

The video should show a **window with the code** and a **window with the console terminal**. When a system functionality is demonstrated (tested), you must explain the related code. Your understanding of OOP design and implementation principles and your ability to explain the code will be assessed and used for marking each implemented functionality.

Therefore, if you fail to provide a video recording of your system according to the above guidelines, the mark shall be 1%-20% (based on the marking of the submitted design artefacts).

Viva

Your final mark will be determined based on your ability to showcase your understanding of the code implementation during a viva that will take place after the submission. Additionally, the evaluator is authorised to initiate a misconduct case if there are any doubts about the originality of your work. **Please carefully review your submission and ensure it is your original—and individual—work.**

Marking Schema

7SENG011W Object Oriented Programming Coursework 2023/24

Student Name:		**Student Number:**		**Marker:**		
Coursework Item		**Assessment Criteria and Mark Range**			**Mark**	**Feedback**
		System Design (Mandatory Submission—20 marks)				
UML Class Diagram (10)	Not Done (0) **Overall Mark capped to 40**	Model lacks significant detail or it is not aligned with the coursework specification *(1-3)*	Core classes correctly identified with relevant attributes and methods; some issues with the modelling of the relationships *(4-7)*	All classes correctly identified with relevant attributes and methods; correct modelling of relationships and multiplicities. The diagram is consistent with the implementation *(8-10)*		
Design Report (10)	Not Done (0) **Overall Mark capped to 40**	The report lacks significant detail or it is not aligned with the provided Class Diagram *(1-3)*	The report discusses the core entities of the system but there is no clear justification of why they have been included in the design *(4-7)*	The report is well written and properly describes the system design, highlighting excellent knowledge and application of OOP principles *(8-10)*		
		System Implementation (80 marks)				
Modelling of Vehicles (10)	Not Done or not presented in the video (0)	The implementation does not follow proper OOP principles or it is not aligned with the class diagram; or not properly presented in the video (1-3)	A working implementation with abstraction and encapsulation; some minor issues, e.g., some missing attributes, methods or usage of not efficient data structures (4-7)	A working implementation that follows all the relevant OOP principles, including inheritance and polymorphism; makes proper usage of data structures ad object relationships (8-10)		
Renting a Vehicle: Schedule, IOverlappable and Driver (10)	Not Done or not described in the report (0)	The implementation does not follow proper OOP principles or it is not aligned with the class diagram; or not properly described in the report (1-3)	A working implementation with abstraction and encapsulation; some minor issues, e.g., some missing attributes, methods or usage of not efficient data structures (4-7)	A working implementation that follows all the relevant OOP principles, makes proper usage of data structures and object relationships (8-10)		
	Not Done or not presented in the video	**Not** working or **partially** working due to **major issues**; or **not properly presented** in the video	Working with **minor issues**	**Fully** working, well implemented with proper error/exception handling		
IRentalManager (25)						
Add Vehicle (3)	0	1	2	3		
Delete Vehicle (3)	0	1	2	3		
List Vehicles (2)	0	1		2		
List Vehicles sorted alphabetically by make (based on IComparer or IComparator) (7)	0	(1-2)	(3-5)	(6-7)		
Generate report on a text file (10)	0	(1-3)	(4-7)	(8-10)		
IRentalCustomer (25)						
List Vehicles of a given type available on a wantedSchedule (7)	0	(1-2)	(3-5)	(6-7)		
Add reservation for a Vehicle identified by number on a wantedSchedule (7)	0	(1-2)	(3-5)	(6-7)		
Change a reservation for a Vehicle (8)	0	(1-2)	(3-5)	(6-8)		
Delete a reservation for a Vehicle (3)	0	1	2	3		
Text Menu (10)						
User Menu (4)	0	1	(2-3)	4		
Admin Menu (6)	0	(1-2)	(3-4)	(5-6)		
		Video (Mandatory Submission—Contributes to the marking of each of the above items)				
Video with System Explaination and Demonstration	Not Submitted **Overall Mark capped to 20**	Submitted				
				Final Mark	0	
				General Feedback		