*REPORT FOR:*

# DESIGN AND DEVELOPMENT OF A

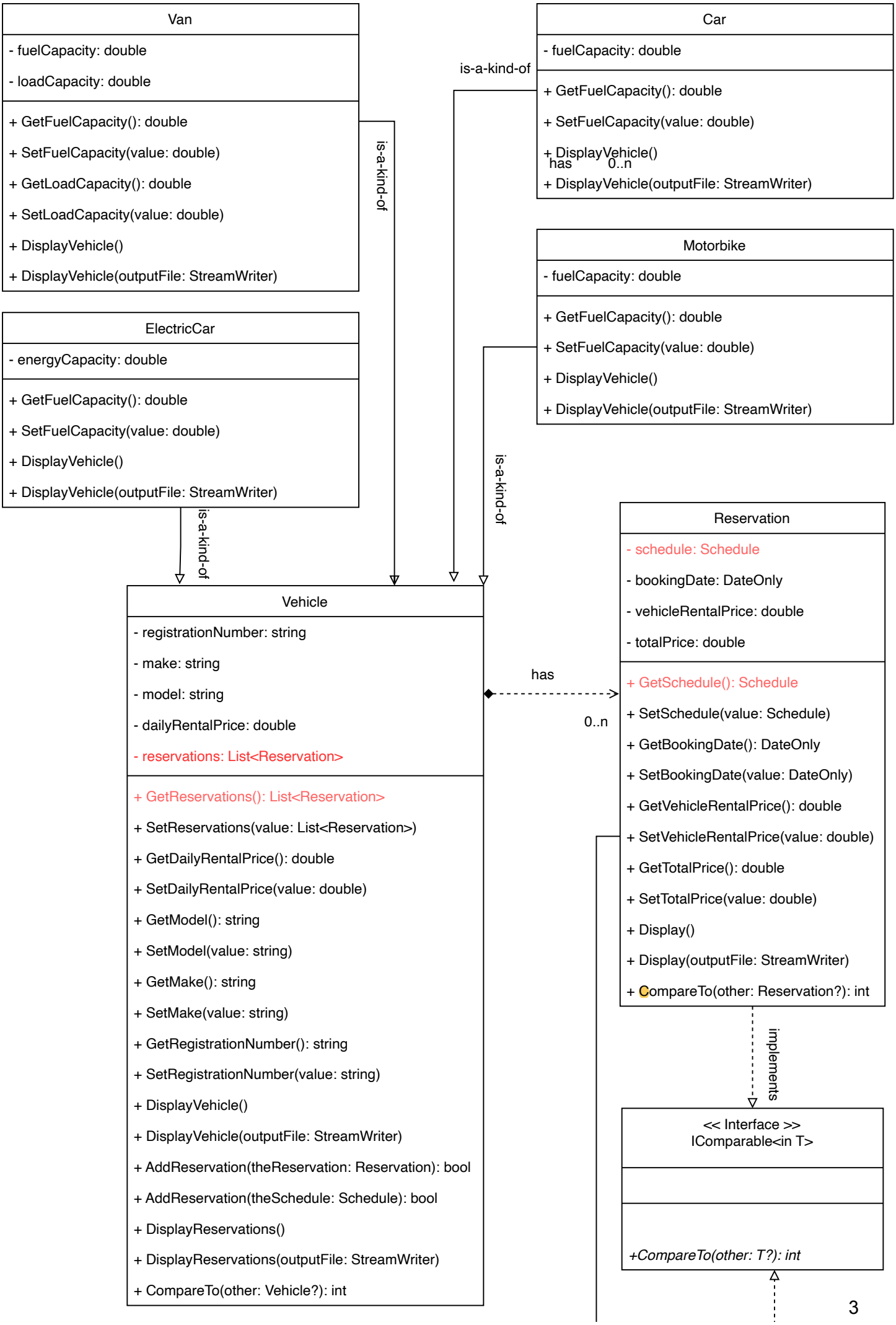
# VEHICLE RENTAL SOFTWARE SYSTEM

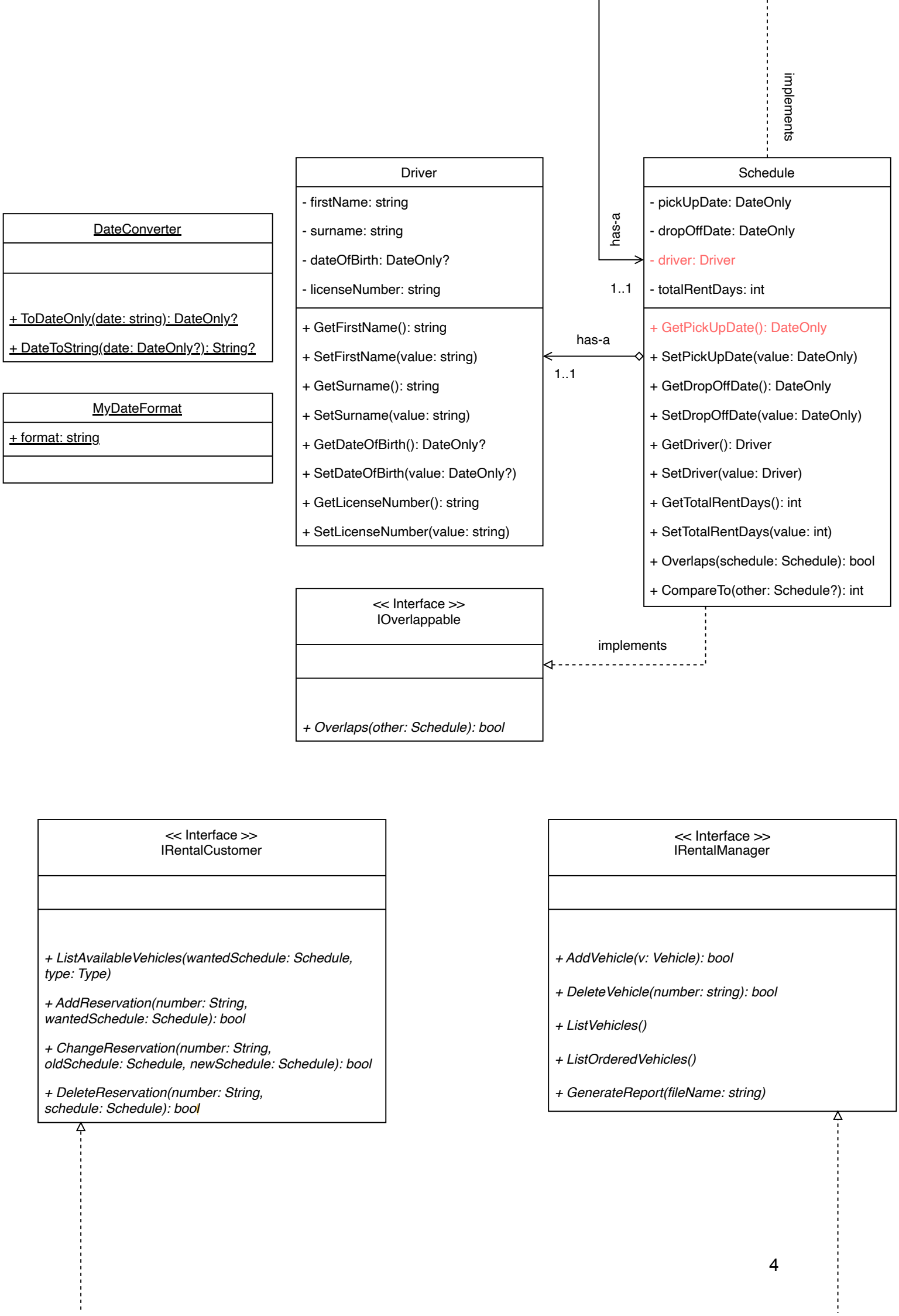
*ANTHONY GIBAH*
*20109235*
*UNIVERSITY OF WESTMINSTER*

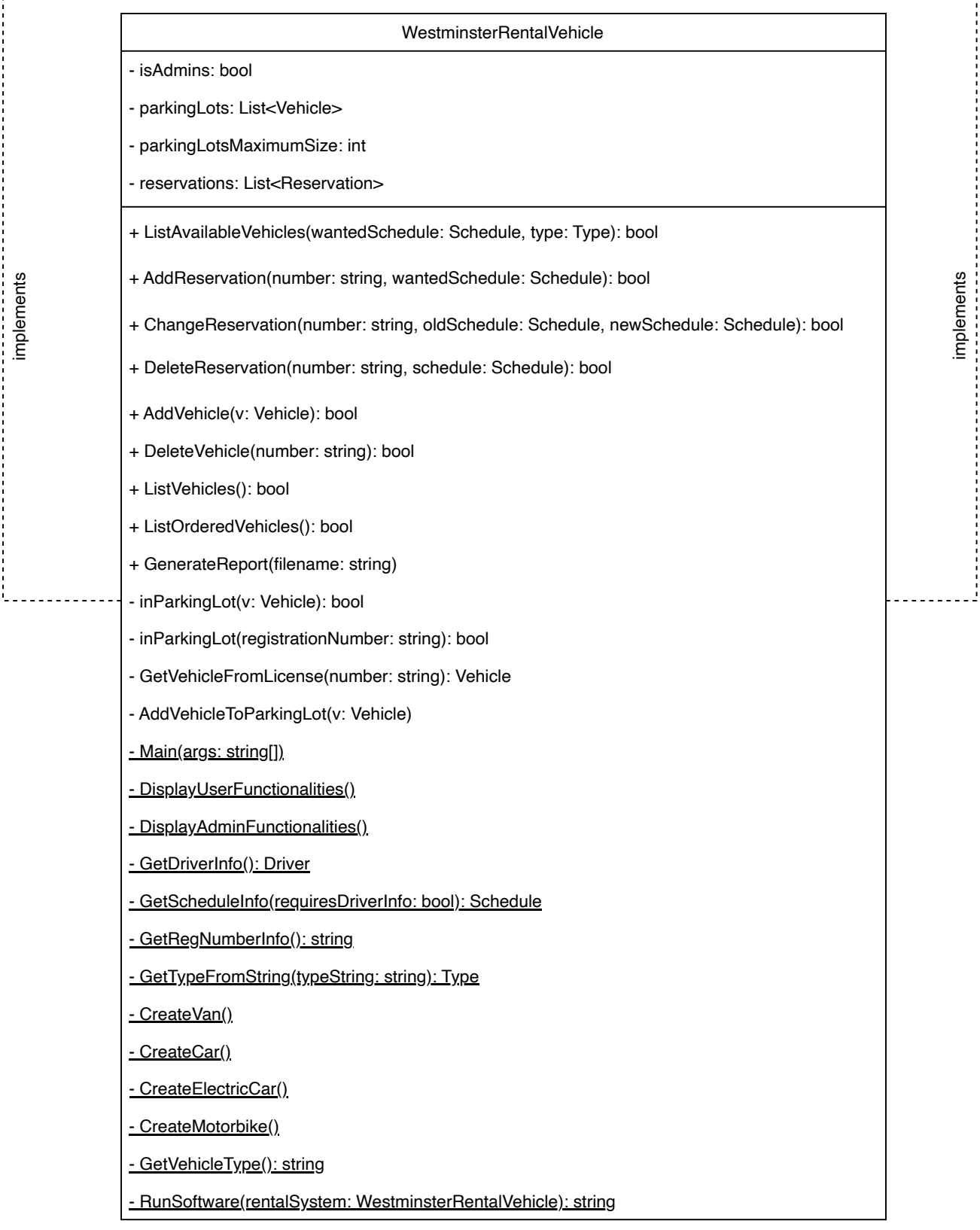
**Video link:** https://youtu.be/83HpLywTwWQ

**TABLE OF CONTENTS**

# Overview and UML Class Diagrams




implements
Driver
- firstName: string
- surname: string
- dateOfBirth: DateOnly?
- licenseNumber: string
+ GetFirstName(): string
+ SetFirstName(value: string)
+ GetSurname(): string
+ SetSurname(value: string)
+ GetDateOfBirth(): DateOnly?
+ SetDateOfBirth(value: DateOnly?)
+ GetLicenseNumber(): string
+ SetLicenseNumber(value: string)
Schedule
- pickUpDate: DateOnly
- dropOffDate: DateOnly
- driver: Driver
- totalRentDays: int
+ GetPickUpDate(): DateOnly
+ SetPickUpDate(value: DateOnly)
+ GetDropOffDate(): DateOnly
+ SetDropOffDate(value: DateOnly)
+ GetDriver(): Driver
+ SetDriver(value: Driver)
+ GetTotalRentDays(): int
+ SetTotalRentDays(value: int)
+ Overlaps(schedule: Schedule): bool
+ CompareTo(other: Schedule?): int
has-a
1..1
has-a
1..1
DateConverter
+ ToDateOnly(date: string): DateOnly?
+ DateToString(date: DateOnly?): String?
MyDateFormat
+ format: string
<< Interface >>
IOverlappable
+ Overlaps(other: Schedule): bool
implements
<< Interface >>
IRentalCustomer
+ ListAvailableVehicles(wantedSchedule: Schedule, type: Type)
+ AddReservation(number: String, wantedSchedule: Schedule): bool
+ ChangeReservation(number: String, oldSchedule: Schedule, newSchedule: Schedule): bool
+ DeleteReservation(number: String, schedule: Schedule): bool
<< Interface >>
IRentalManager
+ AddVehicle(v: Vehicle): bool
+ DeleteVehicle(number: string): bool
+ ListVehicles()
+ ListOrderedVehicles()
+ GenerateReport(fileName: string)


WestminsterRentalVehicle
- isAdmins: bool
- parkingLots: List<Vehicle>
- parkingLotsMaximumSize: int
- reservations: List<Reservation>
+ ListAvailableVehicles(wantedSchedule: Schedule, type: Type): bool
+ AddReservation(number: string, wantedSchedule: Schedule): bool
+ ChangeReservation(number: string, oldSchedule: Schedule, newSchedule: Schedule): bool
+ DeleteReservation(number: string, schedule: Schedule): bool
+ AddVehicle(v: Vehicle): bool
+ DeleteVehicle(number: string): bool
+ ListVehicles(): bool
+ ListOrderedVehicles(): bool
+ GenerateReport(filename: string)
- inParkingLot(v: Vehicle): bool
- inParkingLot(registrationNumber: string): bool
- GetVehicleFromLicense(number: string): Vehicle
- AddVehicleToParkingLot(v: Vehicle)
- Main(args: string[])
- DisplayUserFunctionalities()
- DisplayAdminFunctionalities()
- GetDriverInfo(): Driver
- GetScheduleInfo(requiresDriverInfo: bool): Schedule
- GetRegNumberInfo(): string
- GetTypeFromString(typeString: string): Type
- CreateVan()
- CreateCar()
- CreateElectricCar()
- CreateMotorbike()
- GetVehicleType(): string
- RunSoftware(rentalSystem: WestminsterRentalVehicle): string
implements
implements

**Video link:** https://youtu.be/83HpLywTwWQ

Given above is the UML Diagram for the software system which outlines the relationships between the classes and interfaces. The diagram also shows the attributes and methods in each class and their access modifiers like private and public. The report gives details of the design of a Vehicle Rental Software System for Westminster Rental. The company makes use of this software to rent out vehicles. The characteristics of this software include the functionalities of a user(customer) and the functionalities of an admin(manager).

The program demonstrates the application of the core concepts of the object-oriented programming language, C# , which include Abstraction, Polymorphism, Encapsulation and Inheritance.

# SYSTEM ENTITIES

### Class: Vehicle

The vehicle class represents vehicles owned and managed by the rental company. Its state and behavior are characterized by the attributes and methods respectively. Some of them include:

Attributes:

- registrationNumber: string
- make: string
- model: string
- dailyRentalPrice: string
- reservations: List<Reservation>

Methods:

- DisplayVehicle(): void – Displays the vehicle details.
- DisplayReservations(): void – Displays the vehicle's reservation details.
- CompareTo(other: Vehicle?): int – Used for sorting vehicles by make.

### Class: Car, Van, Motorbike, ElectricCar

They inherit attributes and methods from the Vehicle class. Some of their unique attributes include:

Attributes:

- fuelCapacity: double – (Car, Van, Motorbike) This is the maximum amount of fuel they can have in their tank.
- energyCapacity: double – (ElectricCar) This is the maximum amount of fuel the Car can have in its tank.

Methods:

- GetFuelCapacity(): double – This returns the value of the fuel capacity.
- SetFuelCapacity(value: double): double – This is used to assign a value to the fuel capacity.

### Interface: IRentalCustomer

The IRentalCustomer is an interface that is used to ensure that the program follows a specific design contract for customer. These design contracts are specified by abstract methods. Some of them include:

Methods:

- ListAvailableVehicles(wantedSchedule: Schedule, type: Type) – To list all vehicles in a schedule.
- AddReservation(number: string, wantedSchedule: Schedule): bool – To add reservations to a vehicle using the vehicle registration number.

- ChangeReservation(number: string, oldSchedule: Schedule, newSchedule: Schedule): bool – To change the reservation of a vehicle.
- DeleteReservation(number: string, schedule: Schedule): bool – This is used to delete a reservation of a vehicle with a specific registration number.

### Interface: IRentalManager

The IRentalManager is an interface that is used to ensure that the program follows a specific design contract for the admin. These design contracts are specified by abstract methods. Some of them include:

Methods:

- AddVehicle(v: Vehicle): bool – This method allows an admin to add a vehicle to the system.
- DeleteVehicle(number: string): bool – This method allows the admin to delete a vehicle using its registration number.
- ListVehicles() – This method is used to list all the vehicles in the system.
- ListOrderedVehicles() – This method is used to list all the vehicles in order of their make.
- GenerateReport(filename: string) – This method is used to generate the report as a file.

### Interface: IOverlappable

The IOverlappable is an interface that is used to ensure or check that two schedules overlap. This design contract is specified by the abstract method which includes:

Methods:

- Overlaps(other: Schedule): bool – Checks if the current schedule objects overlaps with other schedule object.

### Class: Driver

The Driver class is an entity for the driver. It describes the attributes a driver should have. Some of its attributes and methods include:

Attributes:

- firstName: string
- surname: string
- dateOfBirth: DateOnly?
- licenseNumber: string

Methods:

- GetFirstName() – Returns the first name of the driver.
- GetSurname() – Returns the surname of the driver.
- GetLicenseNumber() – Returns the license of the driver.

### Class: Schedule

The Schedule class is an entity for the pick-up and drop-off dates for the renting of a vehicle. Some of its attributes include:

Attributes:

- pickUpDate: DateOnly
- dropOffDate: DateOnly
- Driver: Driver
- totalRentDays: int

Methods:

- Overlaps(schedule: Schedule): bool – This checks if this Schedule object overlaps with another Schedule object.
- CompareTo(other: Schedule?): int – This will be used to sort the Schedules based on pick-up date.

### Class: WestminsterRentalVehicle

The WestminsterRentalVehicle class contains all the logic of the program. The program runs from this class, therefore, it has the main method in it. It also implements the **IRentalCustomer** and **IRentalManager** interfaces.

Attributes:

- parkingLotsMaximumSize: int
- parkingLots: List<Vehicle>

Methods:

- Methods implemented from the **IRentalCustomer** and **IRentalManager** interfaces.

# OOP PRINCIPLES APPLIED

### Encapsulation

Encapsulation which involves implementation hiding was applied in this program by making all the attributes in the non-static classes private and providing interfaces for accessing these private attributes. This ensures that the access to these attributes is restricted, and methods are provided to limit the kind of access that other classes and objects have with them.

### Inheritance

The Van, Car, ElectricCar and Motorbike classes inherited from the Vehicle class. This is because they are all a-kind-of the Vehicle class. This means that each of those classes have inherited the attributes and methods of the Vehicle class. The classes however have attributes that are unique to them like in the case of the ElectricCar class, there is an attribute energyCapacity: double that is unique to it.

### Polymorphism

The Van, Car, ElectricCar and Motorbike classes make use of dynamic polymorphism which is also runtime polymorphism through method overriding. The DisplayVehicle() method is overridden in these classes to allow the classes display specific attributes that the base class do not have. They also make use of static polymorphism or method overloading to display vehicle to console, or to file in DisplayVehicle() and DisplayVehicle(outputfile: StreamWriter).

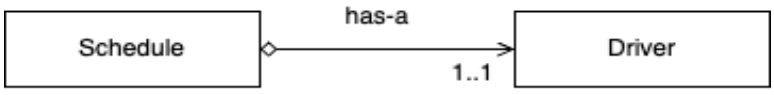
### Abstraction

IRentalManager and IRentalCustomer were used to create a design contract implemented by the WestminsterRentalSystem class. This is an example of abstraction.

# RELATIONSHIPS

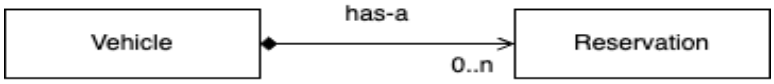
The relationships that exist between the classes already described in the UML diagrams are furthers described below.

### Aggregation



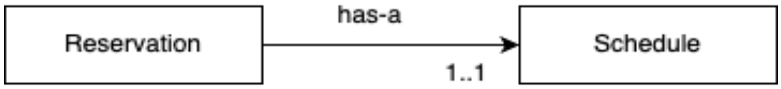

The relationship between the classes above is known as aggregation. This is because the schedule comprises of the information of the driver that is creating this schedule. It is not a composition relationship because the Schedule object does not create the Driver object, so the relationship is not strong enough to be a composition.

### Composition




The relationship between the classes above is known as composition. The Vehicle object is created within the reservation object. This means that the Reservation object only exists within a Vehicle object. If a Vehicle is deleted, the respective Reservation objects will be deleted. The composition relationship is a form of aggregation.

### Association




The relationship between the classes above is known as association. A Reservation object has a Schedule which comprises of the pick-up date and the drop-off date. The relationship is not as strong as the composition relationship because the Schedule object is not created within the Reservation object.

# REFLECTION ON DESIGN AND IMPLEMENTATION

The design of the system meets the specified requirements by making use of the required classes, such as the Vehicle class and classes that inherit it, which include Van, ElectricCar, Car and Motorbike. It makes use of the required Interfaces for the system which include IRentalManager, IRentalCustomer, IOverlapable and IComparable.

The code implementation is well aligned with design through its implementation of the design contracts provided by the design. These design contracts include the User functionalities and Admin functionalities. These functionalities were implemented according to the design and produce the expected results like adding a vehicle to the parking lot.

The system can be improved by making use of a Graphical User Interface (GUI). The current software makes use of a console, which does not provide the best User Experience (UX) for the software. The system can also be improved by adding more functionalities to the system.

## References


1. Microsoft. (2023, 12 01). *How to use the DateOnly and TimeOnly structures.* Retrieved from microsoft.com: https://learn.microsoft.com/en-us/dotnet/standard/datetime/how-to-use-dateonly-timeonly
2. Microsoft. (2023, 08 29). *String interpolation using $.* Retrieved from microsoft.com: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated
3. Microsoft. (2023, 12 12). *IComparable<T> Interface.* Retrieved from microsoft.com: https://learn.microsoft.com/en-us/dotnet/api/system.icomparable-1?view=net-8.0
4. Tusa, D. F. (2023). *UML class diagrams, object relationships, generalisation and inheritance.* London: University of Westminster.
5. Tusa, D. F. (2023). *Collections and sorting; overview of streams and files.* London: University of Westminster.