# Statement of Completeness (Template): Mates Rates Rent-a-Car

This statement of completeness will need to *accurately* state the functionality which has been implemented. Use this checklist of program functionality as you complete your project.

**Student Name: Lewis Watson**

**Student ID: n10221131**

## Functionality Checklist

**In the following section, you are required to mark which functionality you have implemented. In the column on the right please mark ‘Y’ where you have completed this functionality, and ‘N’ where you have not. Please fill in any additional text boxes requested, and please note any limitations or bugs in the box at the end of each section. You may expand the table if you need more room for comments.**

|  |  |  |
| --- | --- | --- |
| **File I/O** | | |
| **File location** | *If your solution does not load files from the specified location in the specification, please note here where the files are stored.*  **Customers:** Data/customer.csv  **Fleet:** Data/fleet.csv  **Rentals:** Data/rentals.csv | |
| **Read** | CRM | Y/N |
| Fleet | Y/N |
| Rentals (must meet above criteria first) | Y/N |
| **Write** | CRM (must be able to read CRM first) | Y/N |
| Fleet (must be able to read CRM first) | Y/N |
| Rentals (must meet above criteria and read Rentals first) | Y/N |
| **Comments** | Not a limitation, but the way the code works is that the entity side of things is completely agnostic. The business logic is contained elsewhere so you could potentially load in duplicates if you edited the .csv files manually. Doing it through the program itself will not allow duplicates. | |

*Please note that your assignment must be able to load the files in the same format as provided in the example files.*

|  |  |  |
| --- | --- | --- |
| **CRM And Fleet Functionality** | | |
| **CRM** | View customers | Y/N |
| Add customer   * With validation – all fields are valid options, no repeated customer ID | Y/N  Y/N |
| Remove customer   * With validation – only if not renting vehicle | Y/N  Y/N |
| Edit customer   * With validation – same as add | Y/N  Y/N |
| **Fleet** | View vehicles | Y/N |
| Add vehicle   * With validation – all fields are valid options, and there is no repeated registrations | Y/N  Y/N |
| Remove vehicle   * With validation – only if not already being rented | Y/N  Y/N |
| Edit vehicle   * With validation – same as add | Y/N  Y/N |
| *Extra marks – implemented vehicle classes with inheritance*  Please include a brief explanation of the inheritance here if implemented:  I have used a solution akin to single table inheritance. Each vehicle grade has its own class and extends the abstract *Vehicle* class. The *Vehicle* class itself houses the *VehicleGrade* enum field which is used as a discriminator property for creating any type of vehicle. For example, a *CommericalVehicle* is an instance of *Vehicle* *where vehicleGrade = VehicleGrade.Commercial*. This default grade behaviour is implemented within the constructors of each respective vehicle type.  The whole code base uses inheritance to abstract common functionality. Lots of generic interfaces are used in the Repositories and RepositoryFactories too. | Y/N |
| **Renting** | View rented vehicles report. This should show the vehicle, customer, and the daily cost of the rental. | Y/N |
| Rent vehicle. This should show the total cost of the rental.   * With validation – only if vehicle and customer exist, vehicle is not already being rented, and customer is not already renting | Y/N  Y/N |
| Return vehicle.   * With validation – only if vehicle was being rented by the customer | Y/N  Y/N |
| **Comments** | *No known bugs, logical errors or runtime errors present* | |
| **Changes from spec.** | File logic was separated into its many layers. The class diagrams provided became the domain layer, and the surrounding parts formed infrastructure and application layers  For example, the file name properties were removed from the classes. Thiswas done to separate concerns within the code base, and make each part perform one specific task only.  The fleet contains a list of rentals and a list of vehicles. This makes the fleet act as an aggregate and the vehicles + rentals act as entities that form the aggregate. This has allowed me to separate concerns effectively. | |

|  |  |  |
| --- | --- | --- |
| **Search Functionality** | | |
| **Possible to Query - Simple** | Any   * Be able to see any unrented vehicle in the fleet | Y/N |
| Single attribute query   * Example:   + Red | Y/N |
| A choice between two attributes (disjunction)   * Example:   + Family OR Luxury | Y/N |
| A combination of two attributes (conjunction)   * Example:   + Family AND Luxury | Y/N |
| **Inter-mediate** | A choice of any number of attributes (disjunction)   * Example:   + Family OR Luxury OR Red | Y/N |
| A combination of any number of attributes (conjunction)   * E.g.:   + Luxury AND Red | Y/N |
| **Advanced** | A combination of any number of attributes, using both AND and OR, where the operators AND and OR should have the same priority (precedence)   * E.g.:   + Economy OR Family AND 4-Cylinders | Y/N |
| ***Ninja*** | A combination of any number of attributes, using AND and OR, with AND having a higher priority than OR, supporting parenthesis to resolve priority   * E.g.:   + ((GPS AND Sunroof) OR (Red OR Green)) AND Commercial OR Luxury | Y/N |
| **Comments** | *No known bugs, logical errors or runtime errors present* | |

## Program Transcript

***You are also required to attach a full transcript demonstrating all of the features of your program (you can use the checklist above as a basis). You will practice this in a future workshop. You must include this transcript as part of your submission. Please state the name of the transcript file.***

**Transcript File Name: transcript-part-a.txt & transcript-part-b.txt**