|  |
| --- |
| CS490 Project 1 |
| Car Rental System |
| Eric Lytle, Tamas Kapros |
|  |
| **Group Project Report** |
|  |

|  |
| --- |
| Github repository is available here: <https://github.com/ericlytle/490Project1> |

Assumptions

For the car rental system we had a set of assumptions that we had to make in order to complete the necessary requirements for the system. One of those assumptions was that there is not to be an add customer option for the system. For a customer to be added, it must be done manually. This could be an option to add to a future scope. This same assumption applies to cars as well. At the moment there is not a way for the system to add cars into the rental book. If there is a need for a new car, the car must be added manually for now. Again this option could be put on the future scope of the project to be completed.

Going along with that, there is also not an option to update the customer information in the system. We assumed that the customer information should remain static and if there is a need for an information change, this must be done manually. If it arises in the future that there is a growing need for customer updates, this could also be added to the future scope of the car rental system. This same assumption goes for cars, at the moment there is not a way to update a cars specs or Id. If there becomes a growing need to update the cars specs, ID’s should never be updated, then that feature can be added to the future scope as well.

Another assumption is that if more than one car is selected at the time the rent button is pressed, the system should prompt the user for dates for both cars. Just because they are both being entered at the same time does not imply that that was the date they were rented. It is possible that the company logs all their rentals at the end of the week in the system or some other bad practice.

The date entry assumption applies to car returns as well. The system will prompt for two separate date entries if more than one car is selected at a time. This will allow the individual who is using the system to enter any date they like for each individual car.

Going back to the Car ID, which should never be updated, this is because we assumed that each car ID is a unique identifier to the physical object that is the car. The car represents that physical object of car and therefore there can only be one of each. There can be multiple of the same specs (model, year, color), but there cannot be two of the exact same physical object.

Going along with the simple system aspect, we assumed that this system can only handle one transaction at a time. There is only one user to be logging rentals and returns at a time since the system will only handle one “selected customer” at a time. Future scope could add the capability to make the system handle multiple users if the need becomes present.

Decisions

During the design of the car rental system, our team had to come up with a set of decisions to make the car rental system run smoothly and efficiently. These decision were made to keep the system as a whole up the standards defined by the project manager. One of those such decisions was to make the controller of the system a singleton. This allowed us to use a list of available cars that would be the same no matter where the controller for the system was used. This also allowed us to use a simple selected customer object to store the customer we are processing transactions for. This decision made it easy to transition from transactions and store the data correctly.

Another decision we made goes along with the idea that the customer information cannot be modified. Since this was in the assumptions we decided to make the customer list be returned as unmodifiable. This will prevent any modifications to the returned list of customers.

One final business decision that we made was to hold all the available cars in a linked list inside the controller. This allowed us to add and removed cars as they were rented and returned. This decision made it easy to keep track of which cars are available to rent and would prevent any overbooking of a car.

From a programmer's perspective there were a several decisions that were made in terms of how the functions were designed or organized. Notably, we opted to use a processCar() function to specify a particular event on a car. The function accepts two parameters: a table object and a string consisting of "rent" or "return". By allowing the table to be specified, a particular table could be iterated through; it wouldn't have to be the same table each time. Secondly, by specifying "rent" or "return", a car could be processed in that respective fashion, such that duplicate code would not be necessary. Had both a function for processing rentals and returns existed, the code would be virtually identical and make the code more difficult to maintain. Ultimately this helped reduce the amount of code in the project.

In our approach to implement our solution, we opted to use NetBeans GUI Editor for creating forms. The unnecessary components were removed. This allowed us to rapidly develop our GUIs as opposed to a manually creating them.

References

stackoverflow.com

Oracle Java Documentation