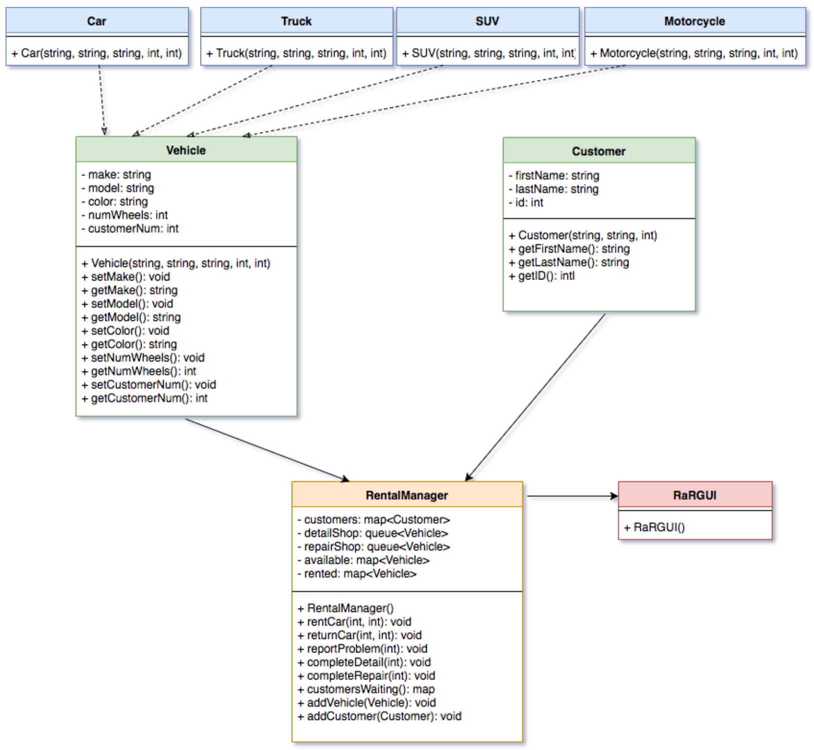
Mohamed Salad
John Poulos
Charles Valdez

Rent-A-Ride Project Report

Introduction:

Our team is name Rent-A-Ride and our project is based on a vehicle rental system. We wanted to simulate a user owning, renting and returning vehicles. Customers and Vehicles are initially loaded through a CSV file containing information. The data is loaded into vectors that can store Customer and Vehicle objects. If a user selects a vehicle and clicks the rent button, our vector will be modified to assign the selected car to the proper user, and our view is updated.

Design:

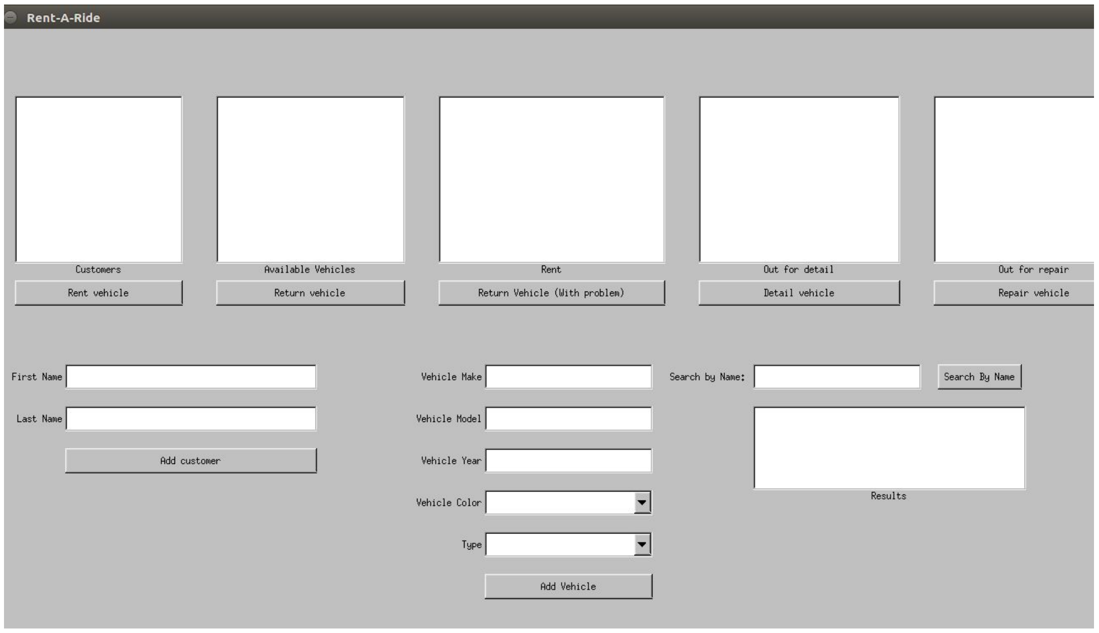

Car
+ Car(string, string, string, int, int)
Truck
+ Truck(string, string, string, int, int)
SUV
+ SUV(string, string, string, int, int
Motorcycle
+ Motorcycle(string, string, string, int, int)
Vehicle
- make: string
- model: string
- color: string
- numWheels: int
- customerNum: int
+ Vehicle(string, string, string, int, int)
+ setMake(): void
+ getMake(): string
+ setModel(): void
+ getModel(): string
+ setColor(): void
+ getColor(): string
+ setNumWheels(): void
+ getNumWheels(): int
+ setCustomerNum(): void
+ getCustomerNum(): int
Customer
- firstName: string
- lastName: string
- id: int
+ Customer(string, string, int)
+ getFirstName(): string
+ getLastName(): string
+ getID(): intl
RentalManager
- customers: map<Customer>
- detailShop: queue<Vehicle>
- repairShop: queue<Vehicle>
- available: map<Vehicle>
- rented: map<Vehicle>
+ RentalManager()
+ rentCar(int, int): void
+ returnCar(int, int): void
+ reportProblem(int): void
+ completeDetail(int): void
+ completeRepair(int): void
+ customersWaiting(): map
+ addVehicle(Vehicle): void
+ addCustomer(Customer): void
RaRGUI
+ RaRGUI()

Our app is using the model view controller design pattern.

Most of our models are all extended by the base class Vehicle, which represents different types of automobiles. All vehicles that are extended have make, model, color, numWheels and a customerId corresponding the the user who owns the car. Our final model Customer describes the important attribute that corresponds to a single user, first name, last name and id.

The brains of our program is handled in RentalManager. This class will handle binding values to our models as well as wiring up the logic to the corresponding views in RaRGUI. Each column is a separate vector. When a vehicle changes status it is moved into the appropriate vector and the view is updated to reflect the change.

Interface



Our user interface follows the Crud style. Create, which is the addition of both new customers and new vehicles. Read, which happens when the rental manager connects the GUI. Update which happens when a user interacts with the buttons, and Delete which happens when a car is rented out. Our GUI has 5 separate browsers (list) that represent Customers. Available vehicles, Rented vehicles, Vehicles out for detail and vehicles out for repair. Underneath the Browser the user can see all the possible operations they can perform on each car. If a user wants to be added, we made two input boxes one for the first name and another for the last name. The user simply has to enter that information and then click add customer to add a user. The same principle for adding a vehicle, enter the make model year color and tag and clicking add vehicle will add that vehicle. Finally in order to search all a user needs to enter is a name and in the results box the output will show.

Testing:

Testing was done using a dataset that we created. The dataset was generated using a helper program we wrote called make Dataset. This program accepts different “seed” files that contain lists of makes and models for each type of vehicle. The program then generates a random amount of each vehicle in the seed file and assigns them a random color. For reference the seed files each contained about 10 makes and models and the final dataset consisted of close to 7000 vehicles. The dataset also randomly set the status of each vehicle to available, rented, needs detail, or needs repair. This was done so that when loading the program the vehicles were dispersed among the different columns. All testing was done as if we were users interacting with the program. We used the gui buttons to perform the different operations and tracked the vehicles as they moved from column to column.

Issues/Status:

One issue we had was updating the GUI as customers were added and shifted around. Because we stored items in various vectors whenever we removed an item to another group we needed to update the gui to match those results. Another issue is whenever we wanted to select items in the gui we had to match that item with the object in the vector. Initially moving a vehicle from the rent group into the repair group would move the wrong vehicle. Our overall status is good, we've accomplished everything we wanted to. The documentation for FLTK is stable and allowed us to progress very fast. Our model view control pattern was something the entire team was familiar with so that little to no time to implement.

Conclusion:

In conclusion we feel very good about our project. In our class we learned allot about the core basics in C++ that was useful to apply in the project. We were skeptical about the GUI capabilities C++ has because we didn't think C++ and GUI went hand in hand. But as we found out there does exist a platform (FLTK) and the documentation was excellent. Presenting every week really helped pull our ideas together and helped us come with a sound conclusion on what we wanted to include.