Car Rental Service Project Report

Rishabh Parwal

October 2024

1 Introduction

This is the report for the Car Rental Service Project. The aim of this project was to learn the basics of Object Oriented Programming in Python, and to create a working Car Rental Service running in a Command Line Interface.
Below is the explanation of the code, and what everything does.

2 Classes and Their Responsibilities

2.1 Vehicle Class

The `Vehicle` class models the basic structure of a vehicle in the rental service. It stores essential details of the vehicle, such as:

- **vehicle_id:** Unique ID for each vehicle.
- **make:** Manufacturer of the vehicle.
- **model:** Model name of the vehicle.
- **year:** The production year of the vehicle.
- **rental_rate:** The daily rental rate for the vehicle.
- **availability:** A boolean value indicating whether the vehicle is available for rent.

The class includes the `print_vehicle_details()` method to print out the details of the vehicle, including specific details for luxury vehicles if applicable.

2.2 LuxuryVehicle Class

The `LuxuryVehicle` class inherits from the `Vehicle` class and represents high-end vehicles. It adds two additional attributes:

- **sunroof:** A boolean to indicate the presence of a sunroof.
- **alcantara:** A boolean to indicate the availability of luxury Alcantara interiors.

The rental rate for luxury vehicles is automatically increased by 20% compared to standard vehicles. This class utilizes the constructor of the `Vehicle` class while overriding certain properties.

2.3 Customer Class

The `Customer` class models a customer in the rental system, storing:

- **customer_id:** A unique identifier for each customer.
- **name:** The name of the customer.
- **contact_info:** The customer's mobile number.
- **rental_history:** A list of rental records associated with the customer.

The class also includes the following methods:

- `input_customer_details()`: Collects customer details from input. It supports checking whether a customer is existing or new.
- `print_customer_details()`: Displays the customer's details.
- `print_rental_history()`: Prints the customer's past rental information.

2.4 rental_info Class (Nested in Customer)

The `rental_info` class records the details of a rental event, including:

- **vehicle_id:** ID of the rented vehicle.
- **rent_duration:** Duration of the rental (in days).
- **returned:** Boolean to track if the vehicle has been returned.

2.5 RentalManager Class

The `RentalManager` class serves as the main controller for the system. It provides an interface for customers and managers to interact with the system. It includes static methods for:

- `add_vehicles()`: Allows the manager to add new vehicles to the inventory.
- `remove_vehicles()`: Allows the manager to remove vehicles from the inventory.
- `generate_rental_report()`: Generates a report of all rentals, including customer details and rental history.

The class is designed with a user interaction loop, supporting operations such as renting, returning, listing, filtering, and sorting vehicles.

3 Functions

3.1 rent_vehicle() Function

The `rent_vehicle(Customer)` function allows a customer to rent a vehicle by inputting a vehicle ID and rental duration. If the vehicle is available, the rental cost is calculated and the vehicle is marked as rented.

3.2 return_vehicle() Function

The `return_vehicle(Customer)` function enables a customer to return a rented vehicle. The vehicle ID must be provided, and if the vehicle matches the customer's rental history, it is marked as returned.

3.3 print_vehicles() Function

The `print_vehicles()` function displays the details of all vehicles in the system. It leverages the `print_vehicle_details()` method from the `Vehicle` class.

3.4 filter_vehicles() Function

The `filter_vehicles(Vehicles, filters, sort_by)` function allows vehicles to be filtered and sorted based on multiple criteria such as vehicle make, model, year, rental rate, and availability. It supports dynamic filtering and sorting to meet customer preferences.

4 Key Features of the Car Rental Service

- **Comprehensive Vehicle Management:** The system supports adding, removing, and managing both regular and luxury vehicles.
- **Customer Interaction:** Customers can rent and return vehicles, view their rental history, and filter available vehicles according to their preferences.
- **Luxury Vehicle Handling:** The system differentiates between standard and luxury vehicles, offering higher rates and additional features for luxury models.
- **Rental Reports:** The system generates detailed rental reports for managerial use, displaying customer and rental information.
- **Dynamic Filtering:** Vehicles can be filtered and sorted based on criteria such as rental rates, availability, and other attributes.

5 Problems Faced

When trying to simulate timing functionality, faced a lot of problems in keeping track of time. After some exploration, I was able to figure out how to keep track of time, and how to use threading to do it simultaneously while the rest of the program is running. But, to remind a customer about a vehicle return due date or to arrange a pre booking, we need to continuously compare the current time to the time of booking or the time at which the rent duration ends. When trying to create a loop that compares the time, if we put the loop in a reminder function and run that function, then while that function is running the loop rest of the code will not be able to execute and will have to wait for the loop to end, which is not what was intended. I tried to create threading for this too, to create the reminder function in a new thread, but then a new problem arises of how to give parameters to the reminder function after a new car is rented to run the timer till the rent duration of the car, because if we initiate the timer function from inside the rent function, it again runs the loop inside the rent function and rest of the functionality stops, and if we initiate the loop at the start of the program, then we are not able to pass the required parameters to tell the function at what time it needs to send the reminder. I also tried multiprocessing but wasn't able to make it work with that either, and at the end I decided to give up on the timer functionality for now, and will make it sometime later.

6 Conclusion

This Car Rental Service system helped a lot in learning the basics of Object Oriented Programming in Python. Some things were not perfect, and I will try to make those work as intended in the future.