Garage CRUD Application

Alexander King Perocho

Sheridan College

PROG10004: Programming Principles

Muhhammad Asif

December 1st 2022

Garage CRUD Application

Part I: Program Structure

This program is a CRUD application that manages a <Vehicle> resource inside a <Garage> storage and management object. The program utilizes a <garage.json> file as a database that can be loaded and updated based on user inputs. This data will persist so long as the file is not deleted. The resource manager <Garage> has a maximum capacity of 5 parking spaces, if the garage is full, the user will have to remove a vehicle in order to park another vehicle. Each vehicle created has attributes determined by the user: <make>, <model>, <year>, <color>. Vehicles can be searched for given the make and model, if there is more than one vehicle with the same make and model, they will be listed along with their attributes and what parking lot they are occupying. Each vehicle can be painted and their color may be edited correspondingly, the selection of which vehicle to be painted is represented through the parking lot number, which is unique to each vehicle, but the garage manages each lot number. If a vehicle is removed, it no longer occupies the parking space; although each lot can hold a unique vehicle, the lot number never changes or disappears, nor does the vehicle so long as it is occupying the space. There exists only 5 parking spaces / lots.

DataPersistence:

Handles Data Persistence, updating the json database with every change that occurs within the garage, and performs the initial check for whether or not there is a database to be loaded on program start.

Garage:

Acts as the storage & management for the database during runtime. Converts the information within the json database into a Garage so operations and editing can be done through OOP. The main logic and operation for managing the resources within this class are all done through the methods.

Vehicle:

This module contains a class that represents a resource to be managed within the <Garage>. It contains attributes associated with vehicles, of which the <color> may be edited.

Program:

This module handles user interactions, and updates a <garage> for data persistence related operations in a separate json file.

Main:

Main module begins the program and creates a garage with data requested from the DataPersistence module.

Constants:

This module is strictly for code cleanliness, it contains string values for constants.

garage.json:

This file represents the database that may be loaded and used within the program. So long as this file and its contents are not deleted, the program will run according to the information within it.

Part II: Program Interactivity and Logic

Create:

Adding a new vehicle to the garage and having it occupy a parking space.

Read (Search):

Checks the garage for any and all vehicles with matching make and model given by the user input.

Edit:

Paints a vehicle a specified color given by the user, the vehicle is chosen through which parking space they occupy via user input.

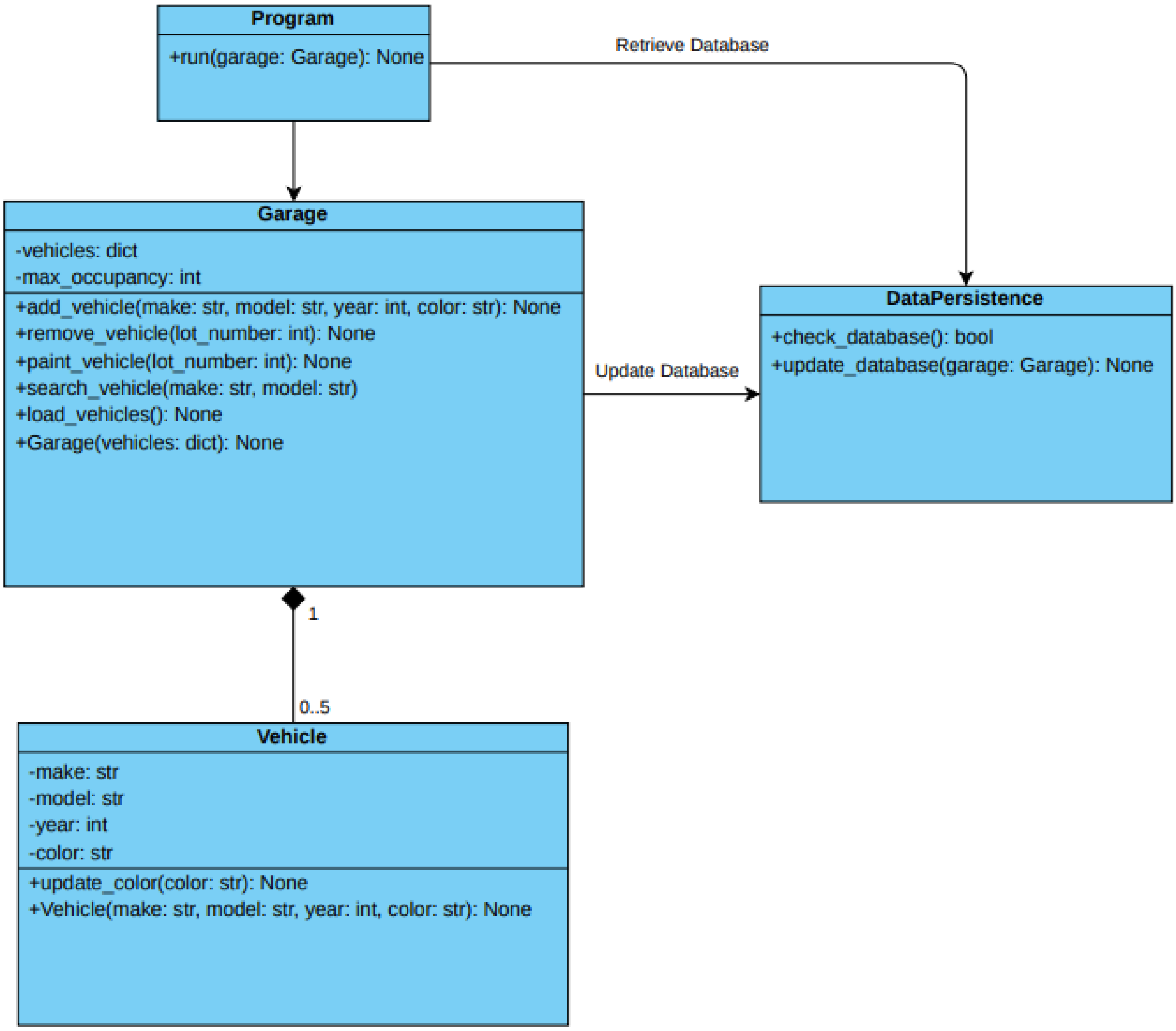
Delete:

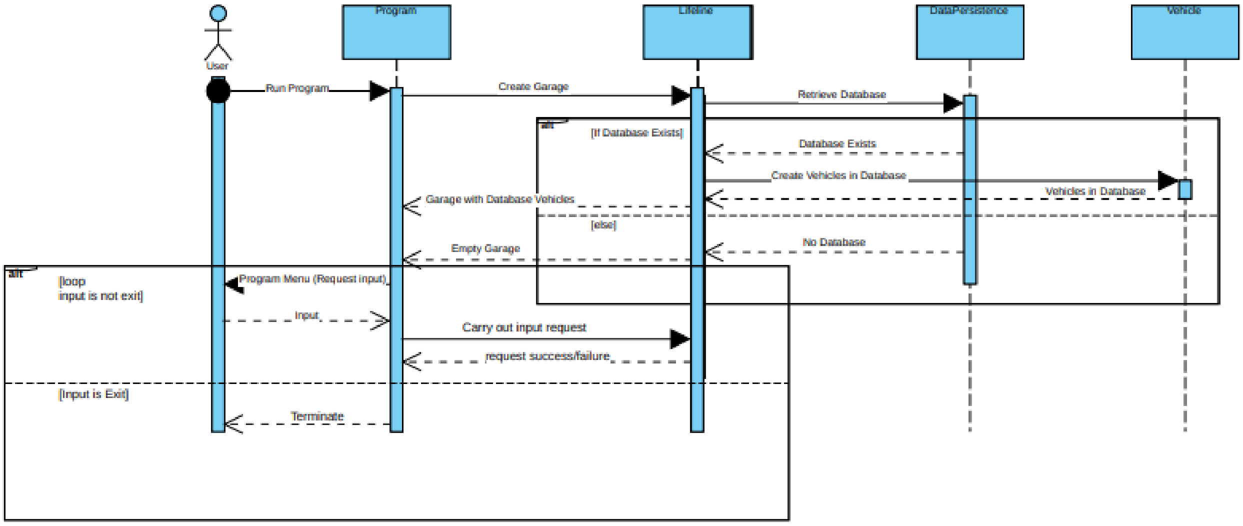
Remove a vehicle from the garage and free up the parking space occupied.

Part V: Program Development Process

I enjoyed this assignment, as it introduced me to real-world applications of file management, and databases in computer science. I tried to make the program as simple and efficient as possible again, which meant I had to compromise on some implementations that I would like to investigate further in future iterations of projects similar to this one. I would like to use an API call to some database online as options for the user to be able to choose from when managing their own data, instead of manually inputting all the attributes for whatever resource is being saved. I would also try to make the program more efficient by removing the redundancies of having objects represent the data, and having to convert the data back and forth in order for the program to run properly. Next time I would like to simply read, write, create and delete from the source file, while maintaining a backup file in case of an unexpected error, all without the need to implement objects to represent the data.

Diagrams


Program
+run(garage: Garage): None
Retrieve Database
Garage
-vehicles: dict
-max_occupancy: int
+add_vehicle(make: str, model: str, year: int, color: str): None
+remove_vehicle(lot_number: int): None
+paint_vehicle(lot_number: int): None
+search_vehicle(make: str, model: str)
+load_vehicles(): None
+Garage(vehicles: dict): None
Update Database
DataPersistence
+check_database(): bool
+update_database(garage: Garage): None
1
0..5
Vehicle
-make: str
-model: str
-year: int
-color: str
+update_color(color: str): None
+Vehicle(make: str, model: str, year: int, color: str): None


Program
Lifeline
DataPersistence
Vehicle
User
Run Program
Create Garage
Retrieve Database
alt
[If Database Exists]
Database Exists
Create Vehicles in Database
Vehicles in Database
Garage with Database Vehicles
[else]
No Database
Empty Garage
alt
[loop
input is not exit]
Program Menu (Request input)
Input
Carry out input request
request success/failure
[Input is Exit]
Terminate