# EOOP - final project

### Date: 08.06.2022 Semester: Spring 2022

### Author and Group:!!!!!!!!!!!!!!

### Subject (Keyword):Rent a Car

## **Description of the project**

### Overview of the project

### Rent a car service have cars which client can rent, Clients data and schedule for each car. Each people can come to service, fill out a form, choose the car and rant on dates which they want. Service making sure that one people can’t take more than one car in day and one car can’t rent more than one people in day.

### Class and data structures overview

### There are following classes: RentCarService, Car, Client, ScheduleForCar, ScheduleForClient and Schedule.

### RentCarService contain unlimited number of Cars and Clients which you access with passport number and methods for add, delete, update, rent, return and show cars, add new client, and cancel rent schedule.

### Car contain data about car and set of SchelduleForCar and methods to initializate new car, show this car, update mileage, price and deposit, add, delete and show schedule. These methods call method from RentCarService.

### Client contain data about client and set of SchelduleForClient and methods to initializate new Client, show this Client, add, delete and show schedule. These methods calls method from RentCarService.

### Schedule contain data from which days to which rent car. It has initialization and operator < for set.

### ScheduleForCar this is derived class from Schedule and contain pointer to client and method to show and initializate.

### ScheduleForClient this is derived class from Schedule and contain pointer to car and method to show and initializate.

### Restrictions, limits, assumptions

### R1. One people can’t take more than one car in day

### R2. One car can’t rent more than one people in day

### R3. Can’t be two people with same passport number

## **Case study (a memory map)**

**At the end of Preliminary Project.**

## **Declaration of the classes**

class *RentCarService*

{

private:

    vector<*Car* \*> cars;

// Vector of cars

    map<string, *Client* \*> clients;

// Map of clients with their passport number as key

*Car* \*chooseCar(int *index*);

// Choose car from the vector using function Car::showCar() and ask for the number of the car from the manager

*Client* \*chooseClient(*string* *passport\_number*);

// Choose client: ask for the passport number of the client and return the pointer to the client

public:

    void addCar(*string* *brand*, *string* *model*, *string* *color*, *string* *VIN*, int *year*, int *mileage*, float *price\_per\_day*, int *deposit*);

// Add a car to the vector with parameters

    void addClient(*string* *name*, *string* *surname*, *string* *date\_of\_birth*, *string* *passport\_number*, *string* *phone\_number*, *string* *email*, *string* *address*);

// Add client to the map with parameters

    void rentCar(int *index*, *string* *passport\_number*, *Schedule* *schedule*);

// This function call Car\* chooseCar() and Client\* chooseClient() and rent the car to the client with the given schedule

    void showCars();

// Show cars all cars in the vector using function Car::showCar()

    void showClient(*string* *passport\_number*);

// Show client with this passport number using function Client::showClient()

    void showCar(int *index*);

// Show car with this index using function Car::showCar()

    void deleteCar(int *index*);

// Delete car call Car\* chooseCar()

    void deleteClient(*string* *passport\_number*);

// Delete client call Client\* chooseClient()

    void deleteSchedule(int *index*, *string* *passport\_number*, *Schedule* *schedule*);

// Call Car\* chooseCar() and Client\* chooseClient() and delete the schedule with the given parameters

    void carUpdateMilage(int *index*, int *mileage*);

// Update car mileage call Car\* chooseCar()

    void carUpdatePrice(int *index*, float *price\_per\_day*);

// Update car price call Car\* chooseCar()

    void carUpdateDeposit(int *index*, int *deposit*);

// Update car deposit call Car\* chooseCar()

    RentCarService();

// Constructor

    ~RentCarService();

// Destructor

};

class *Car*

{

private:

    string brand;

// Brand

    string model;

// Model

    string color;

// Color

    string VIN;

// VIN

    int year;

// Year

    int mileage;

// Mileage

    float price\_per\_day;

// Price per day

    int deposit;

// Deposit

    set<*ScheduleForCar*> schedule;

// Schedule for car

public:

    Car(*string* *brand*, *string* *model*, *string* *color*, *string* *VIN*, int *year*, int *mileage*, int *price\_per\_day*, int *deposit*);

// Constructor

    Car();

// Constructor

    ~Car();

// Destructor

    void addSchedule(*ScheduleForCar* \**schedule*);

// Add schedule to the set schedule

    void deleteSchedule(const *Schedule* \**schedule*);

// Delete schedule

    void showSchedule();

// Show schedule

    void showCar();

// Show car

    void updateMileage(int *mileage*);

// Check if the mileage is greater than the current mileage and update the mileage

    void updatePrice(int *price\_per\_day*);

// Update price per day

    void updateDeposit(int *deposit*);

// Update deposit

    bool checkSchedule(*ScheduleForCar* *schedule*);

// Check if the car is available for this schedule

};

class *Client*

{

private:

    string name;

// Name

    string surname;

// Surname

    string date\_of\_birth;

// Date of birth

    string passport\_number;

// Passport number

    string phone\_number;

// Phone number

    string email;

// Email

    string address;

// Address

    set<*ScheduleForClient*> schedule;

// Schedule for customer

public:

    Client(*string* *name*, *string* *surname*, *string* *date\_of\_birth*, *string* *passport\_number*, *string* *phone\_number*, *string* *email*, *string* *address*);

// Constructor

    Client();

// Constructor

    ~Client();

// Destructor

    void addSchedule(*ScheduleForClient* \**schedule*);

// Add schedule to the set schedule

    void deleteSchedule(const *Schedule* \**schedule*);

// Delete schedule

    void showSchedule();

// Show schedule

    void showClient();

// Show client

    bool checkSchedule(*ScheduleForClient* *schedule*);

// Check if the car is available for this schedule and return true if the car is available for this schedule

*string* getPassportNumber();

// Return passport number

};

class *Schedule*

{

private:

    int day\_from;

// Day from

    int month\_from;

// Month from

    int year\_from;

// Year from

    int day\_to;

// Day to

    int month\_to;

// Month to

    int year\_to;

// Year to

public:

    void printSchedule() const;

// Print schedule

    bool operator<(const *Schedule* &*other*) const;

// Operator <

    Schedule();

// Constructor

    ~Schedule() = default;

// Destructor

    void fillSchedule(int *day\_from*, int *month\_from*, int *year\_from*, int *day\_to*, int *month\_to*, int *year\_to*);

// Fill schedule

    friend class *ScheduleForClient*;

    friend class *ScheduleForCar*;

};

class *ScheduleForCar* : public *Schedule*

{

private:

*Client* \*client;

// Client

public:

    ScheduleForCar(const *Schedule* \**schedule*, *Client* \**client*);

// Constructor

    ScheduleForCar();

// Constructor

    ~ScheduleForCar() = default;

// Destructor

    void showScheduleForCar();

// Show schedule for car

    void deleteClientSchedule() const;

// Delete client schedule

};

class *ScheduleForClient* : public *Schedule*

{

private:

*Car* \*car;

// Car

public:

    ScheduleForClient(const *Schedule* \**schedule*, *Car* \**car*);

// Constructor

    ScheduleForClient();

// Constructor

    ~ScheduleForClient() = default;

// Destructor

    void showScheduleForClient();

// Show schedule for customer

    void deleteCarSchedule() const;

// Delete car schedule

};

## **Functional test cases**

*RentCarService* rcs;

    rcs.addCar("BMW", "X5", "Black", "VIN", 2019, 100, 100, 1000);

    rcs.addCar("Audi", "A4", "White", "VIN", 2020, 200, 200, 2000);

    rcs.addCar("Mercedes", "C class", "Red", "VIN", 2021, 300, 300, 3000);

    rcs.addCar("Toyota", "Corolla", "Blue", "VIN", 2023, 400, 400, 4000);

// add car with incorrect year

    rcs.addCar("Ford", "Focus", "Green", "VIN", 2020, -12, 500, 5000);

// add car with incorrect mileage

    rcs.addCar("BMW", "X5", "Black", "VIN", 2019, 100, -100, 1000);

// add car with incorrect price

    rcs.addCar("Audi", "A4", "White", "VIN", 2020, 200, 200, -2000);

// add car with incorrect deposit

    rcs.addClient("John", "Smith", "01.01.2000", "123", "+48578951235", "johnsmith@gmail.com", "Warsaw");

    rcs.addClient("Mike", "Smith", "01.01.2000", "124", "+48578951235", "mikesmith@gmail.com", "Warsaw");

    rcs.addClient("John", "Smith", "01.01.2000", "123", "+48578951235", "johnsmith@gmail.com", "Warsaw");

// add client with incorrect passport number

*Schedule* schedule;

    schedule.fillSchedule(10, 10, 2022, 19, 1, 2022);

// add schedule with incorrect data

    schedule.fillSchedule(36, 9, 2022, 20, 10, 2022);

// add schedule with incorrect data

    schedule.fillSchedule(10, 6, 2022, 10, 7, 2022);

    rcs.rentCar(0, "123", schedule);

    rcs.rentCar(1, "124", schedule);

    rcs.rentCar(1, "123", schedule);

// rent two cars with the same passport number in same time

    rcs.rentCar(0, "124", schedule);

// rent car which is already rented

    schedule.fillSchedule(10, 5, 2022, 15, 6, 2022);

    rcs.rentCar(0, "124", schedule);

// rent car which is already rented

    rcs.showCars();

    cout << "Schedule Clint with passport number 124:" << endl;

    rcs.showClient("124");

    cout << endl;

    rcs.deleteClient("123");

    rcs.deleteCar(1);

    rcs.showCars();

    cout << "Schedule Clint with passport number 124:" << endl;

    rcs.showClient("124");

    cout << endl;

    rcs.rentCar(0, "124", schedule);

    rcs.showCars();

    rcs.deleteSchedule(0, "124", schedule);

    cout << "Schedule Clint with passport number 124:" << endl;

    rcs.showClient("124");

    cout << endl;

    rcs.showCars();

    rcs.carUpdateMilage(0, 1000);

    rcs.carUpdatePrice(0, 1000);

    rcs.carUpdateDeposit(0, 2000);

    rcs.carUpdateMilage(0, 999);

// update car with incorrect mileage

    rcs.carUpdatePrice(0, 0);

// update car with incorrect price

    rcs.carUpdateDeposit(0, -1000);

// update car with incorrect deposit

    rcs.showCars();

*RentCarService* \*rcs2 = **new** *RentCarService*();

    rcs2->addCar("BMW", "X5", "Black", "VIN", 2019, 100, 100, 1000);

    rcs2->addCar("Audi", "A4", "White", "VIN", 2020, 200, 200, 2000);

    rcs2->addCar("Mercedes", "C class", "Red", "VIN", 2021, 300, 300, 3000);

    rcs2->addCar("Toyota", "Corolla", "Blue", "VIN", 2023, 400, 400, 4000);

// add car with incorrect year

    rcs2->addCar("Ford", "Focus", "Green", "VIN", 2020, -12, 500, 5000);

// add car with incorrect mileage

    rcs2->addCar("BMW", "X5", "Black", "VIN", 2019, 100, -100, 1000);

// add car with incorrect price

    rcs2->addCar("Audi", "A4", "White", "VIN", 2020, 200, 200, -2000);

// add car with incorrect deposit

    rcs2->addClient("John", "Smith", "01.01.2000", "123", "+48578951235", "johnsmith@gmail.com", "Warsaw");

    rcs2->addClient("Mike", "Smith", "01.01.2000", "124", "+48578951235", "mikesmith@gmail.com", "Warsaw");

    rcs2->addClient("John", "Smith", "01.01.2000", "123", "+48578951235", "johnsmith@gmail.com", "Warsaw");

// add client with incorrect passport number

    schedule.fillSchedule(10, 10, 2022, 19, 1, 2022);

// add schedule with incorrect data

    schedule.fillSchedule(36, 9, 2022, 20, 10, 2022);

// add schedule with incorrect data

    schedule.fillSchedule(10, 6, 2022, 10, 7, 2022);

    rcs2->rentCar(0, "123", schedule);

    rcs2->rentCar(1, "124", schedule);

    rcs2->rentCar(1, "123", schedule);

// rent two cars with the same passport number in same time

    rcs2->rentCar(0, "124", schedule);

// rent car which is already rented

    schedule.fillSchedule(10, 5, 2022, 15, 6, 2022);

    rcs2->rentCar(0, "124", schedule);

// rent car which is already rented

    rcs2->showCars();

    cout << "Schedule Clint with passport number 124:" << endl;

    rcs2->showClient("124");

    cout << endl;

    rcs2->deleteClient("123");

    rcs2->deleteCar(1);

    rcs2->showCars();

    cout << "Schedule Clint with passport number 124:" << endl;

    rcs2->showClient("124");

    cout << endl;

    rcs2->rentCar(0, "124", schedule);

    rcs2->showCars();

    rcs2->deleteSchedule(0, "124", schedule);

    cout << "Schedule Clint with passport number 124:" << endl;

    rcs2->showClient("124");

    cout << endl;

    rcs2->showCars();

    rcs2->carUpdateMilage(0, 1000);

    rcs2->carUpdatePrice(0, 1000);

    rcs2->carUpdateDeposit(0, 2000);

    rcs2->carUpdateMilage(0, 999);

// update car with incorrect mileage

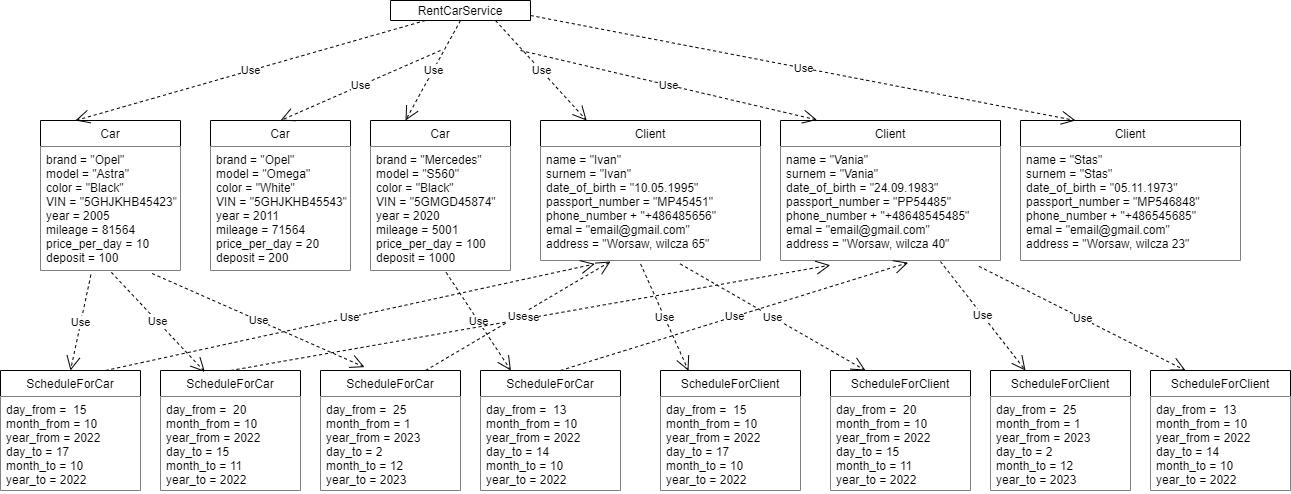
    rcs2->carUpdatePrice(0, 0);

// update car with incorrect price

    rcs2->carUpdateDeposit(0, -1000);

// update car with incorrect deposit

    rcs2->showCars();

****