Introducere

1. Contextul proiectului
   Aplicație ce trebuie realizată în echipă, care să soluționeze/automatizeze probleme din viața reală.

2. Problemele de rezolvat
   Închirierea unei mașini, nevoia de o mașină atunci când nu dispui de una.

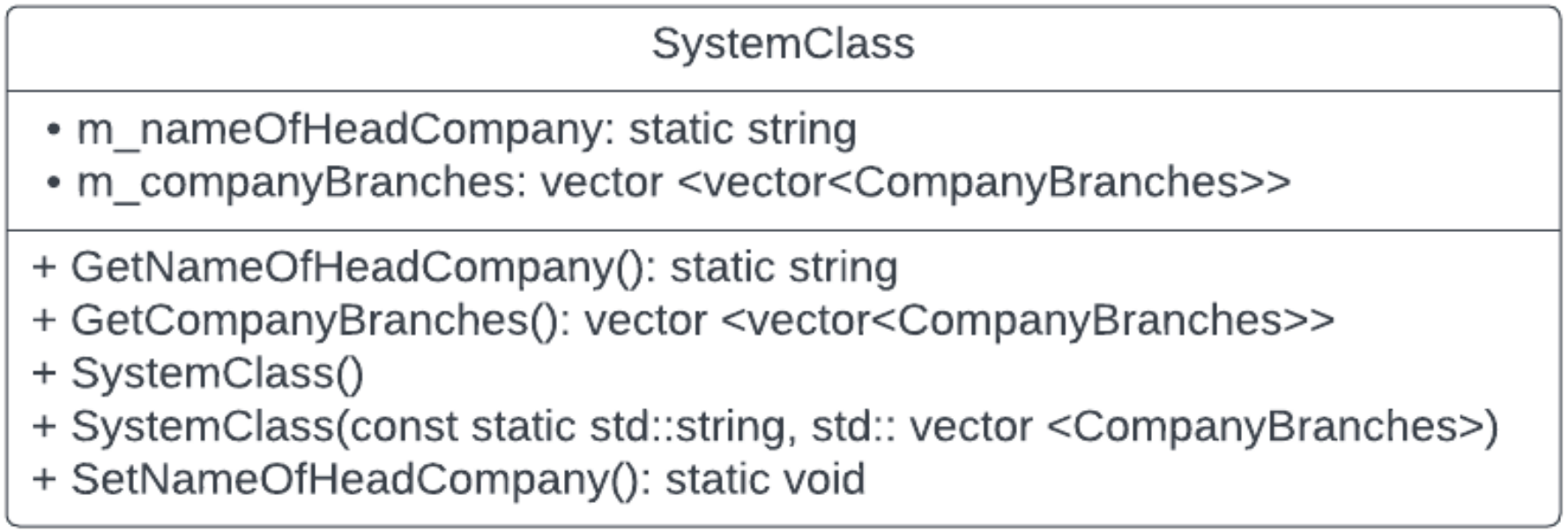
3. Obiective propuse
   Realizarea unei aplicatiii care să soluționeze problema găsirii unei mașini atunci când ai nevoie de una, dar să și aducă îmbunătățiri decât o soluție ce are capabilități limitate ce rezolvă strict o singură problema. Astfel dorim să rezolvăm problema principală, dar să și adăugăm noi opțiuni.

Descrierea soluţiei

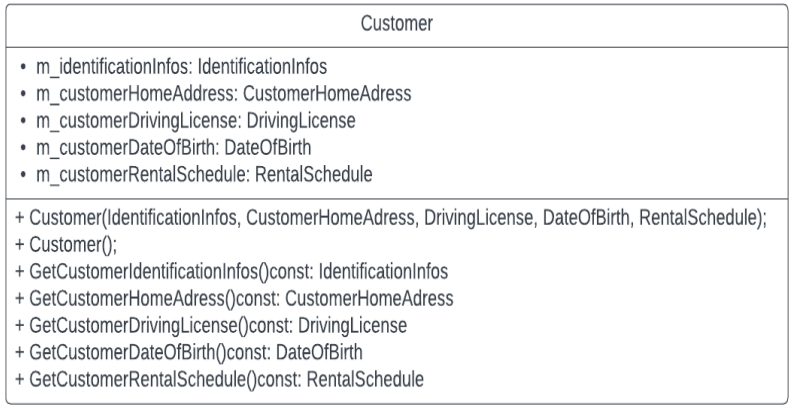
Putem realiza un sistem de gestiune al firmelor reprezentat prin tripletul (firmă, mașini, clienți). Firma se înregistrează și pune la dispoziție mașinile pe care le are, iar mai apoi clienții se vor înregistra în aplicație urmând să aleagă firma cu care vor să facă un contract.

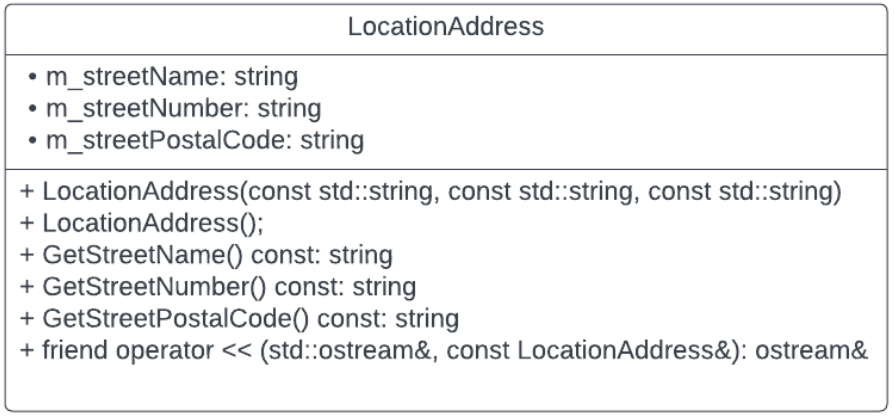
Proiectare şi implementare

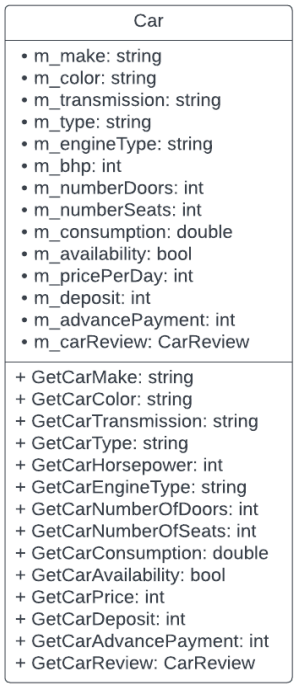
1. Reprezentarea grafică a claselor

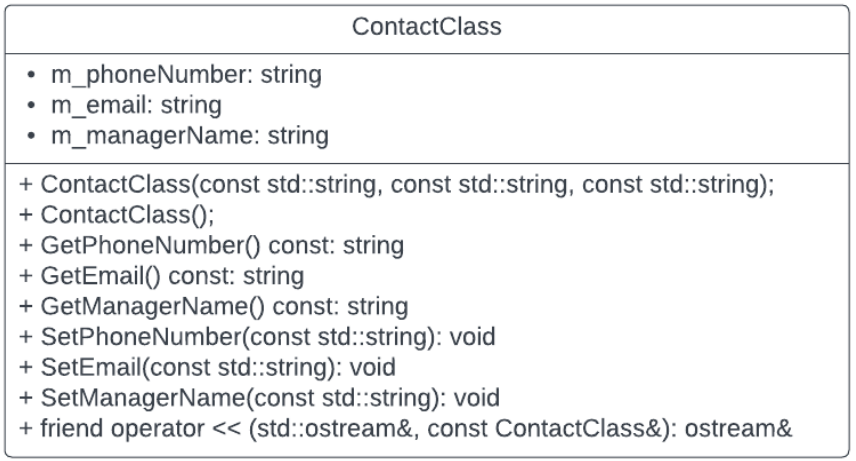

SystemClass
• m_nameOfHeadCompany: static string
• m_companyBranches: vector <vector<CompanyBranches>>
+ GetNameOfHeadCompany(): static string
+ GetCompanyBranches(): vector <vector<CompanyBranches>>
+ SystemClass()
+ SystemClass(const static std::string, std:: vector <CompanyBranches>)
+ SetNameOfHeadCompany(): static void


CompanyBranches
• m_branchName: string
• m_countryName: string
• m_cityName: string
• m_reviews: string
• m_administratorOBS: string
• m_numberOfCarsAvaible: unsigned long long
• m_branchMark: double
• m_branchCustomers: vector <Customer>
• m_locationAddress: LocationAddress
• m_branchCars: vector <Car>
• m_contactClass: ContactClass
+ CompanyBranches(const std::string, const std::string, const std::string, const std::string, const std::string, const unsigned long long, const double, const std::string, const std::string, const std::string, const std::string, const std::string, const std::string)
+ CompanyBranches(): string
+ GetBranchName(): string
+ GetCountryName(): string
+ GetCityName(): string
+ GetReviews(): string
+ GetAdministratorOBS(): string
+ GetNumberOfCarsAvaible(): unsigned long long
+ GetBranchMark(): double
+ GetBranchCustomers(): vector <Customer>
+ GetLocationAddress(): LocationAddress
+ GetBranchCars(): vector <Car>
+ GetContactClass(): ContactClass
+ friend operator << (std::ostream&, const CompanyBranches&): ostream&

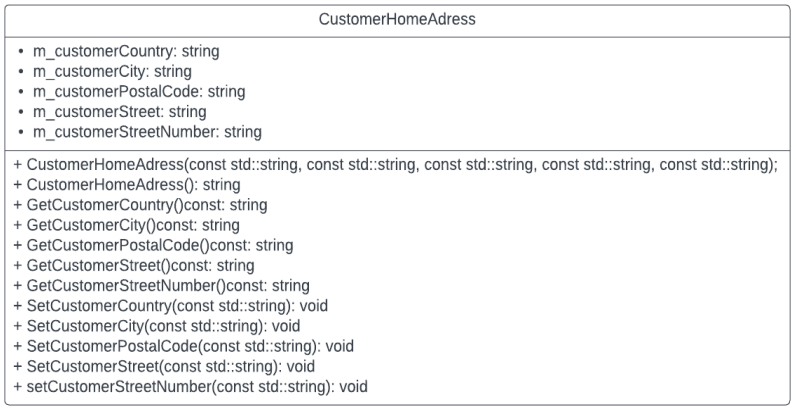

Customer
• m_identificationInfos: IdentificationInfos
• m_customerHomeAddress: CustomerHomeAdress
• m_customerDrivingLicense: DrivingLicense
• m_customerDateOfBirth: DateOfBirth
• m_customerRentalSchedule: RentalSchedule
+ Customer(IdentificationInfos, CustomerHomeAdress, DrivingLicense, DateOfBirth, RentalSchedule);
+ Customer();
+ GetCustomerIdentificationInfos()const: IdentificationInfos
+ GetCustomerHomeAdress()const: CustomerHomeAdress
+ GetCustomerDrivingLicense()const: DrivingLicense
+ GetCustomerDateOfBirth()const: DateOfBirth
+ GetCustomerRentalSchedule()const: RentalSchedule

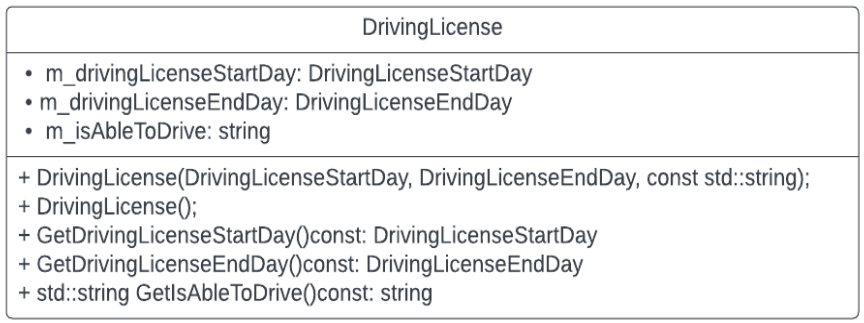

LocationAddress
• m_streetName: string
• m_streetNumber: string
• m_streetPostalCode: string
+ LocationAddress(const std::string, const std::string, const std::string)
+ LocationAddress();
+ GetStreetName() const: string
+ GetStreetNumber() const: string
+ GetStreetPostalCode() const: string
+ friend operator << (std::ostream&, const LocationAddress&): ostream&

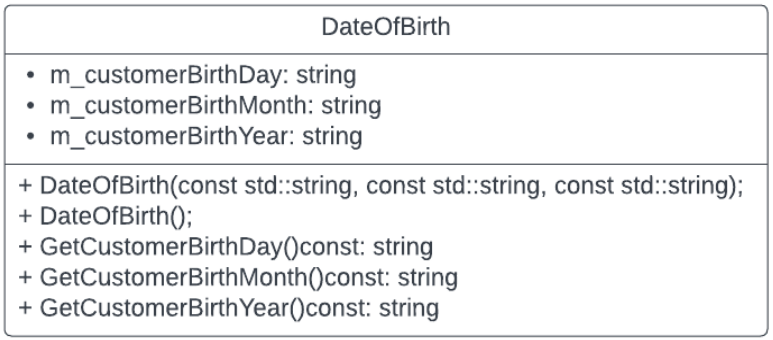

Car
• m_make: string
• m_color: string
• m_transmission: string
• m_type: string
• m_engineType: string
• m_bhp: int
• m_numberDoors: int
• m_numberSeats: int
• m_consumption: double
• m_availability: bool
• m_pricePerDay: int
• m_deposit: int
• m_advancePayment: int
• m_carReview: CarReview
+ GetCarMake: string
+ GetCarColor: string
+ GetCarTransmission: string
+ GetCarType: string
+ GetCarHorsepower: int
+ GetCarEngineType: string
+ GetCarNumberOfDoors: int
+ GetCarNumberOfSeats: int
+ GetCarConsumption: double
+ GetCarAvailability: bool
+ GetCarPrice: int
+ GetCarDeposit: int
+ GetCarAdvancePayment: int
+ GetCarReview: CarReview

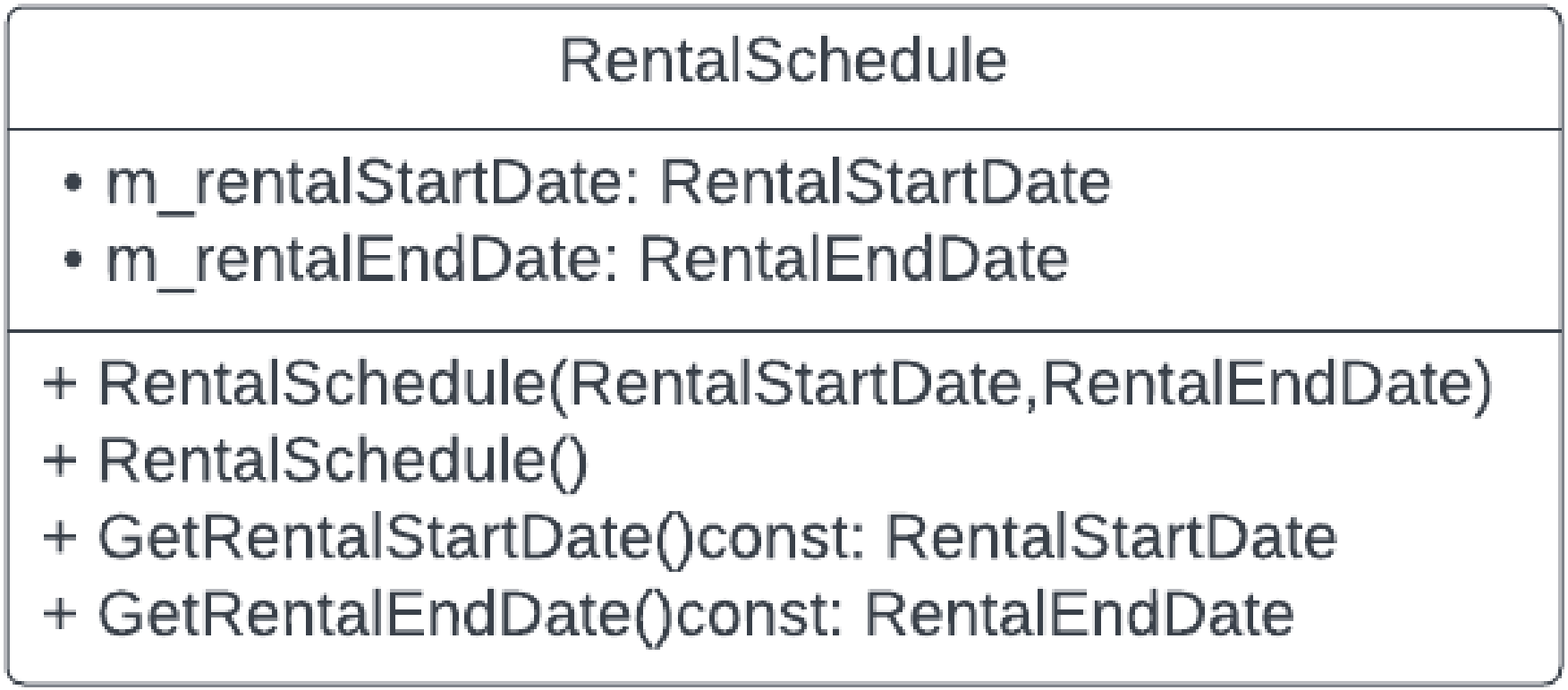

ContactClass
• m_phoneNumber: string
• m_email: string
• m_managerName: string
+ ContactClass(const std::string, const std::string, const std::string);
+ ContactClass();
+ GetPhoneNumber() const: string
+ GetEmail() const: string
+ GetManagerName() const: string
+ SetPhoneNumber(const std::string): void
+ SetEmail(const std::string): void
+ SetManagerName(const std::string): void
+ friend operator << (std::ostream&, const ContactClass&): ostream&

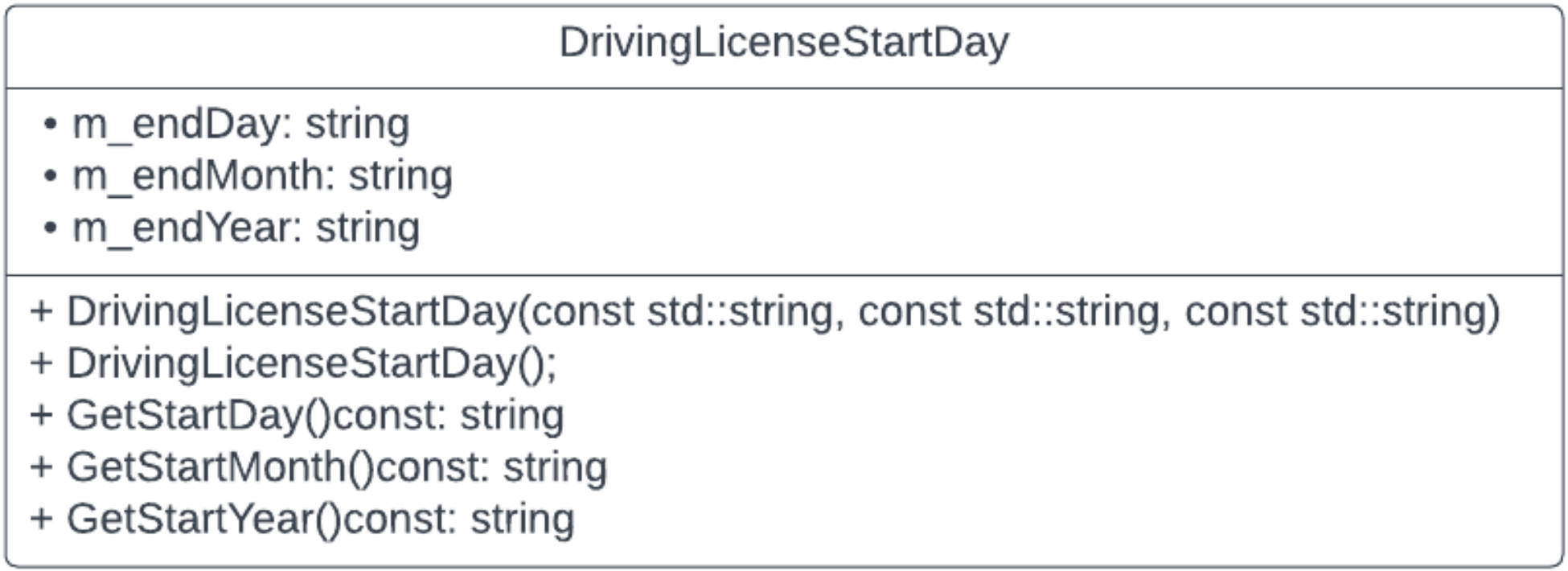

IdentificationInfos
• m_customerFirstName: string
• m_customerLastName: string
• m_customerEmail: string
• m_customerTelephone: string
• m_customerCNP: string
• m_customerIDSeries: string
• m_customerIDCardNumber: string
• m_customerPassword: string
+ IdentificationInfos(const std::string, const std::string, const std::string, const std::string, const std::string, const std::string, const std::string, const std::string)
+ IdentificationInfos();
+ GetCustomerFirstName()const: string
+ GetCustomerLastName()const: string
+ GetCustomerEmail()const: string
+ GetCustomerTelephone()const: string
+ GetCustomerCNP()const: string
+ GetCustomerIDSeries()const: string
+ GetCustomerIDCardNumber()const: string
+ GetCustomerPassword() const: string
+ SetCustomerFirstName(const std::string): void
+ SetCustomerLastName(const std::string): void
+ SetCustomerEmail(const std::string): void
+ SetCustomerTelephone(const std::string): void
+ SetCustomerCNP(const std::string): void
+ SetCustomerIDSeries(const std::string): void
+ SetCustomerIDCardNumber(const std::string): void
+ SetCustomerPassword(const std::string): void

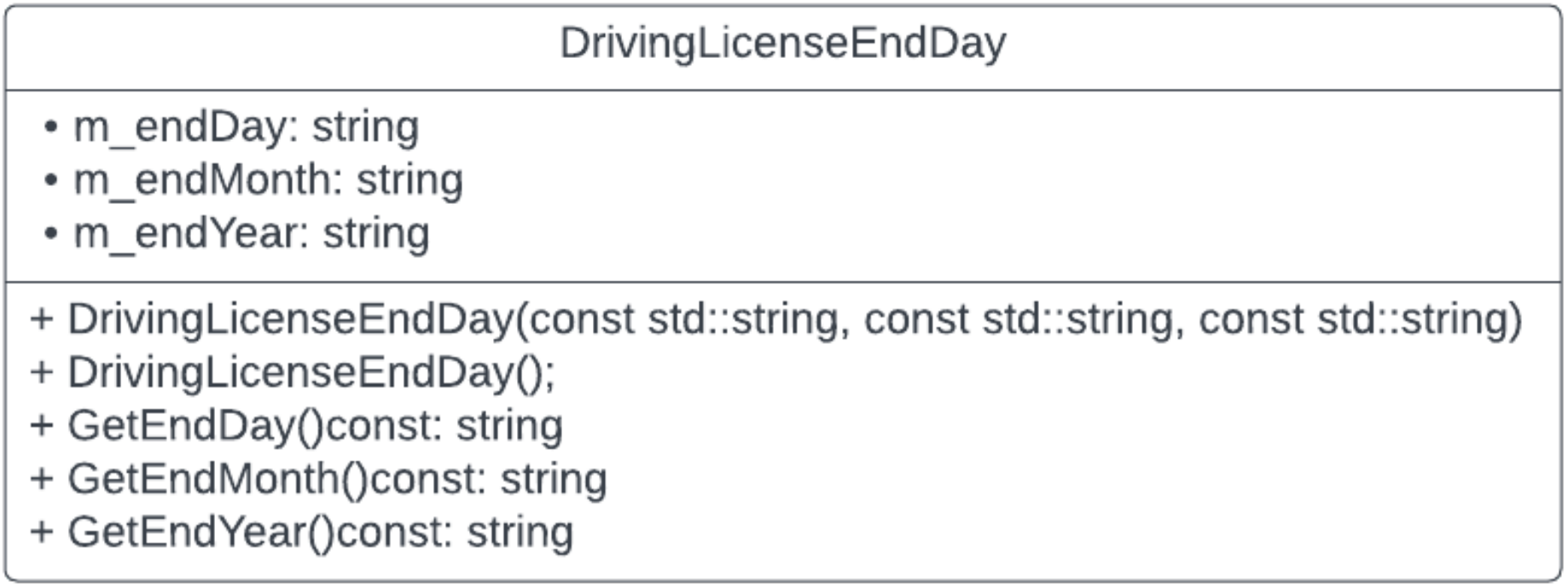

CustomerHomeAdress
• m_customerCountry: string
• m_customerCity: string
• m_customerPostalCode: string
• m_customerStreet: string
• m_customerStreetNumber: string
+ CustomerHomeAdress(const std::string, const std::string, const std::string, const std::string, const std::string);
+ CustomerHomeAdress(): string
+ GetCustomerCountry()const: string
+ GetCustomerCity()const: string
+ GetCustomerPostalCode()const: string
+ GetCustomerStreet()const: string
+ GetCustomerStreetNumber()const: string
+ SetCustomerCountry(const std::string): void
+ SetCustomerCity(const std::string): void
+ SetCustomerPostalCode(const std::string): void
+ SetCustomerStreet(const std::string): void
+ setCustomerStreetNumber(const std::string): void

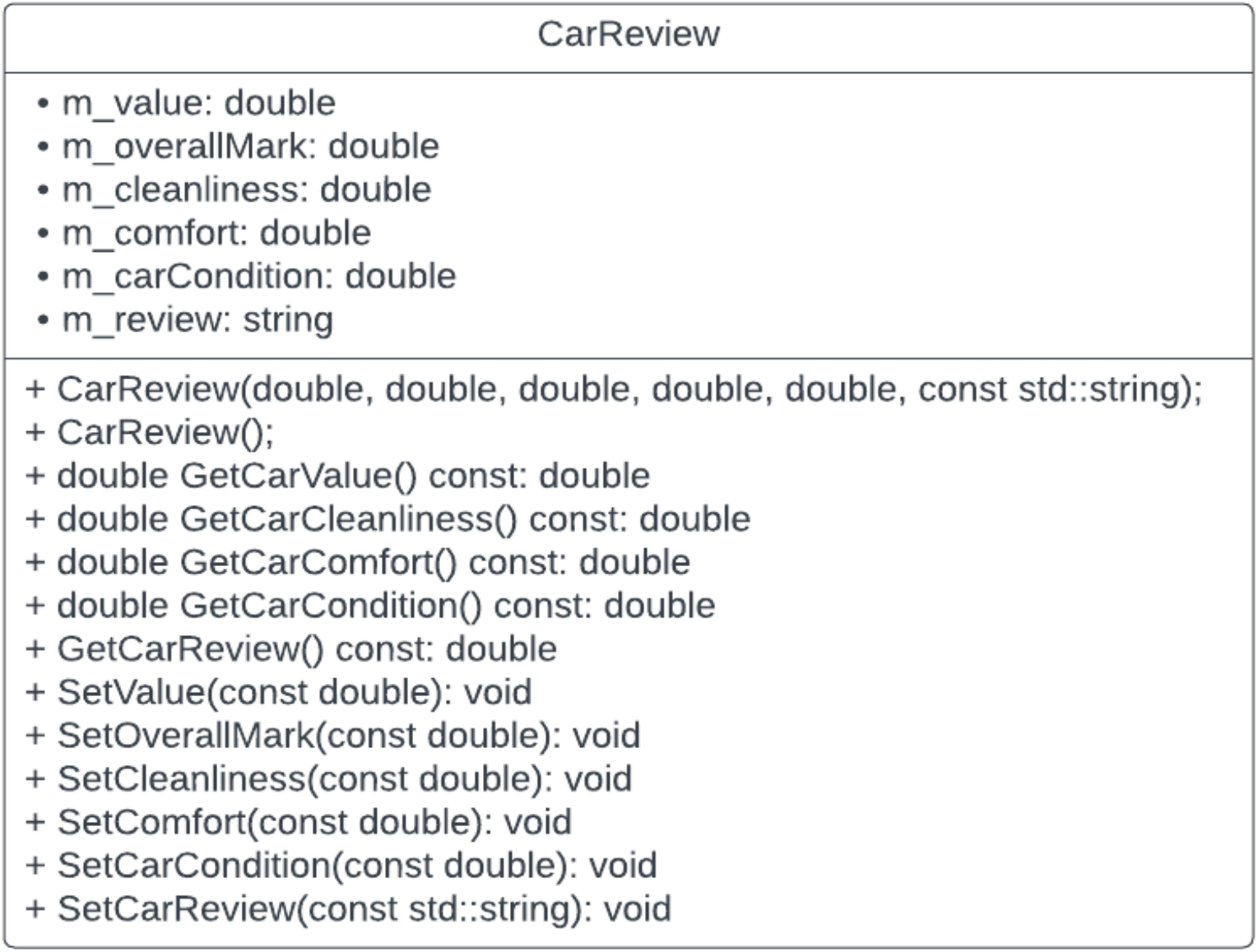

DrivingLicense
• m_drivingLicenseStartDay: DrivingLicenseStartDay
• m_drivingLicenseEndDay: DrivingLicenseEndDay
• m_isAbleToDrive: string
+ DrivingLicense(DrivingLicenseStartDay, DrivingLicenseEndDay, const std::string);
+ DrivingLicense();
+ GetDrivingLicenseStartDay()const: DrivingLicenseStartDay
+ GetDrivingLicenseEndDay()const: DrivingLicenseEndDay
+ std::string GetIsAbleToDrive()const: string

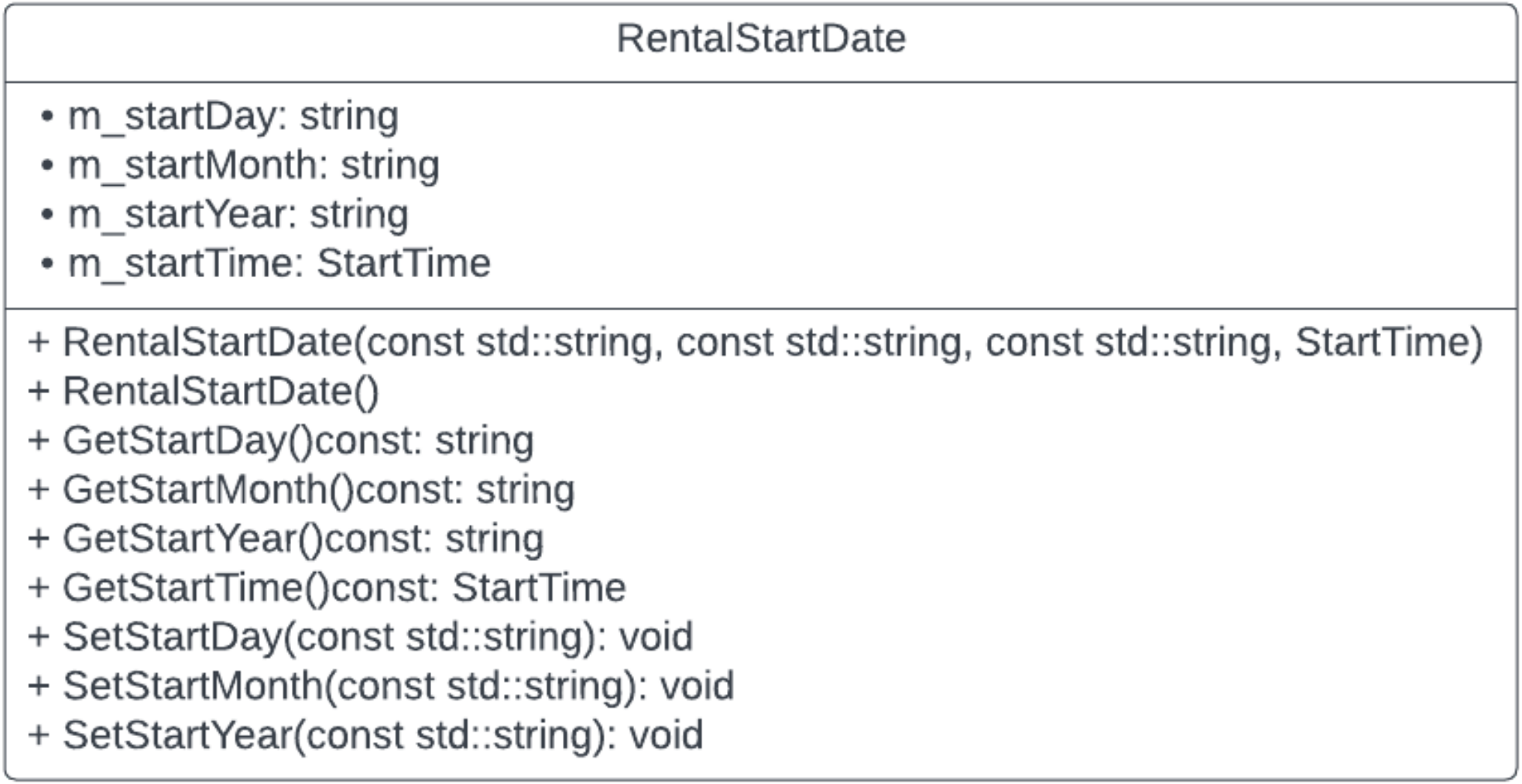

DateOfBirth
• m_customerBirthDay: string
• m_customerBirthMonth: string
• m_customerBirthYear: string
+ DateOfBirth(const std::string, const std::string, const std::string);
+ DateOfBirth();
+ GetCustomerBirthDay()const: string
+ GetCustomerBirthMonth()const: string
+ GetCustomerBirthYear()const: string

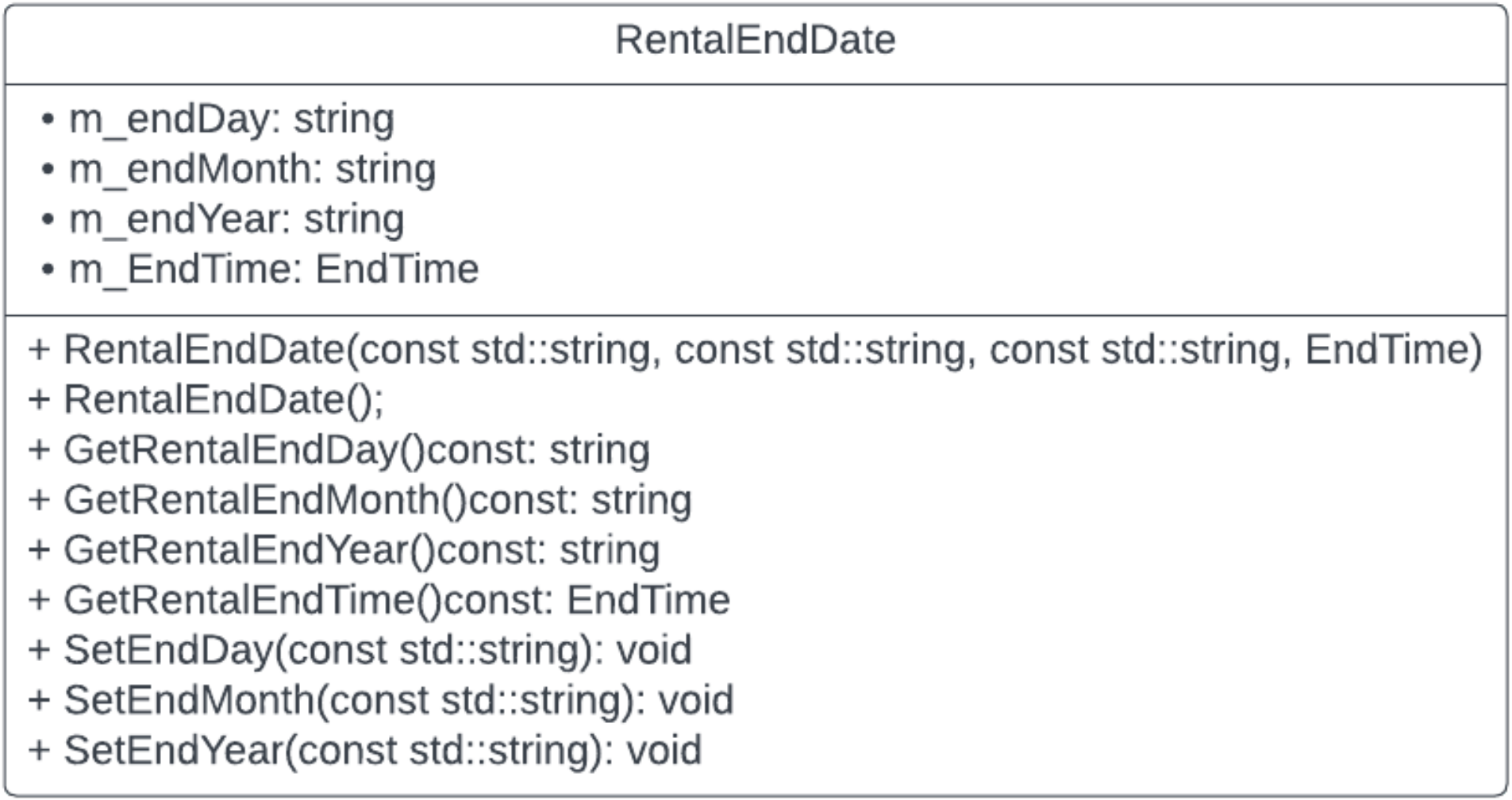

RentalSchedule
• m_rentalStartDate: RentalStartDate
• m_rentalEndDate: RentalEndDate
+ RentalSchedule(RentalStartDate,RentalEndDate)
+ RentalSchedule()
+ GetRentalStartDate()const: RentalStartDate
+ GetRentalEndDate()const: RentalEndDate

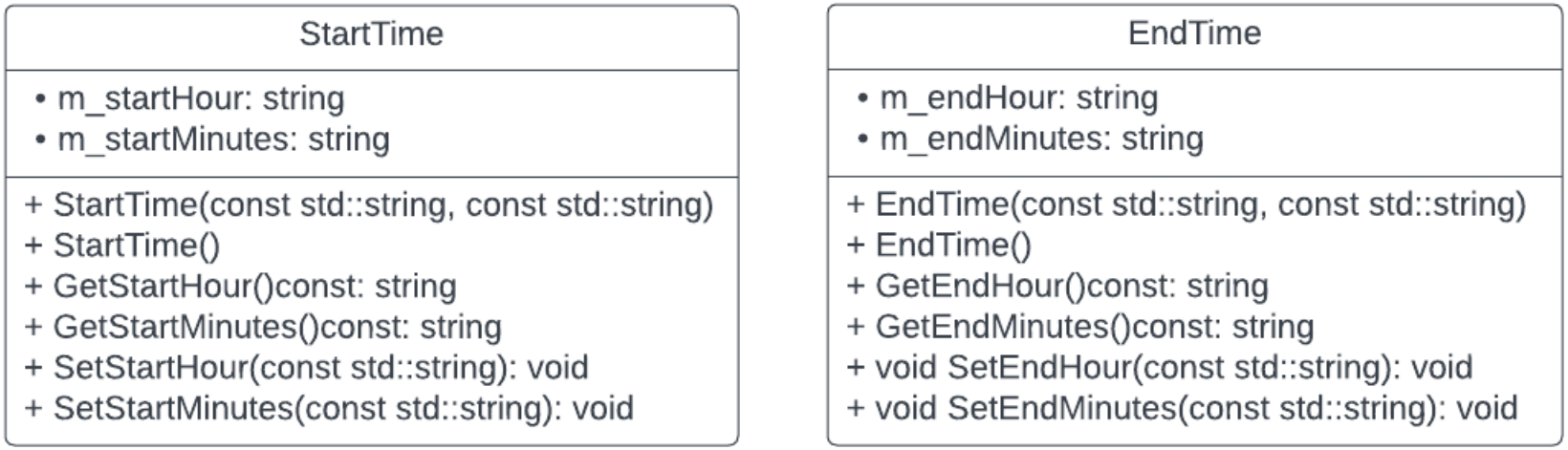

DrivingLicenseStartDay
• m_endDay: string
• m_endMonth: string
• m_endYear: string
+ DrivingLicenseStartDay(const std::string, const std::string, const std::string)
+ DrivingLicenseStartDay();
+ GetStartDay()const: string
+ GetStartMonth()const: string
+ GetStartYear()const: string


DrivingLicenseEndDay
• m_endDay: string
• m_endMonth: string
• m_endYear: string
+ DrivingLicenseEndDay(const std::string, const std::string, const std::string)
+ DrivingLicenseEndDay();
+ GetEndDay()const: string
+ GetEndMonth()const: string
+ GetEndYear()const: string


CarReview
• m_value: double
• m_overallMark: double
• m_cleanliness: double
• m_comfort: double
• m_carCondition: double
• m_review: string
+ CarReview(double, double, double, double, double, const std::string);
+ CarReview();
+ double GetCarValue() const: double
+ double GetCarCleanliness() const: double
+ double GetCarComfort() const: double
+ double GetCarCondition() const: double
+ GetCarReview() const: double
+ SetValue(const double): void
+ SetOverallMark(const double): void
+ SetCleanliness(const double): void
+ SetComfort(const double): void
+ SetCarCondition(const double): void
+ SetCarReview(const std::string): void


RentalStartDate
• m_startDay: string
• m_startMonth: string
• m_startYear: string
• m_startTime: StartTime
+ RentalStartDate(const std::string, const std::string, const std::string, StartTime)
+ RentalStartDate()
+ GetStartDay()const: string
+ GetStartMonth()const: string
+ GetStartYear()const: string
+ GetStartTime()const: StartTime
+ SetStartDay(const std::string): void
+ SetStartMonth(const std::string): void
+ SetStartYear(const std::string): void


RentalEndDate
• m_endDay: string
• m_endMonth: string
• m_endYear: string
• m_EndTime: EndTime
+ RentalEndDate(const std::string, const std::string, const std::string, EndTime)
+ RentalEndDate();
+ GetRentalEndDay()const: string
+ GetRentalEndMonth()const: string
+ GetRentalEndYear()const: string
+ GetRentalEndTime()const: EndTime
+ SetEndDay(const std::string): void
+ SetEndMonth(const std::string): void
+ SetEndYear(const std::string): void


StartTime
• m_startHour: string
• m_startMinutes: string
+ StartTime(const std::string, const std::string)
+ StartTime()
+ GetStartHour()const: string
+ GetStartMinutes()const: string
+ SetStartHour(const std::string): void
+ SetStartMinutes(const std::string): void
EndTime
• m_endHour: string
• m_endMinutes: string
+ EndTime(const std::string, const std::string)
+ EndTime()
+ GetEndHour()const: string
+ GetEndMinutes()const: string
+ void SetEndHour(const std::string): void
+ void SetEndMinutes(const std::string): void

2. Structura proiect:

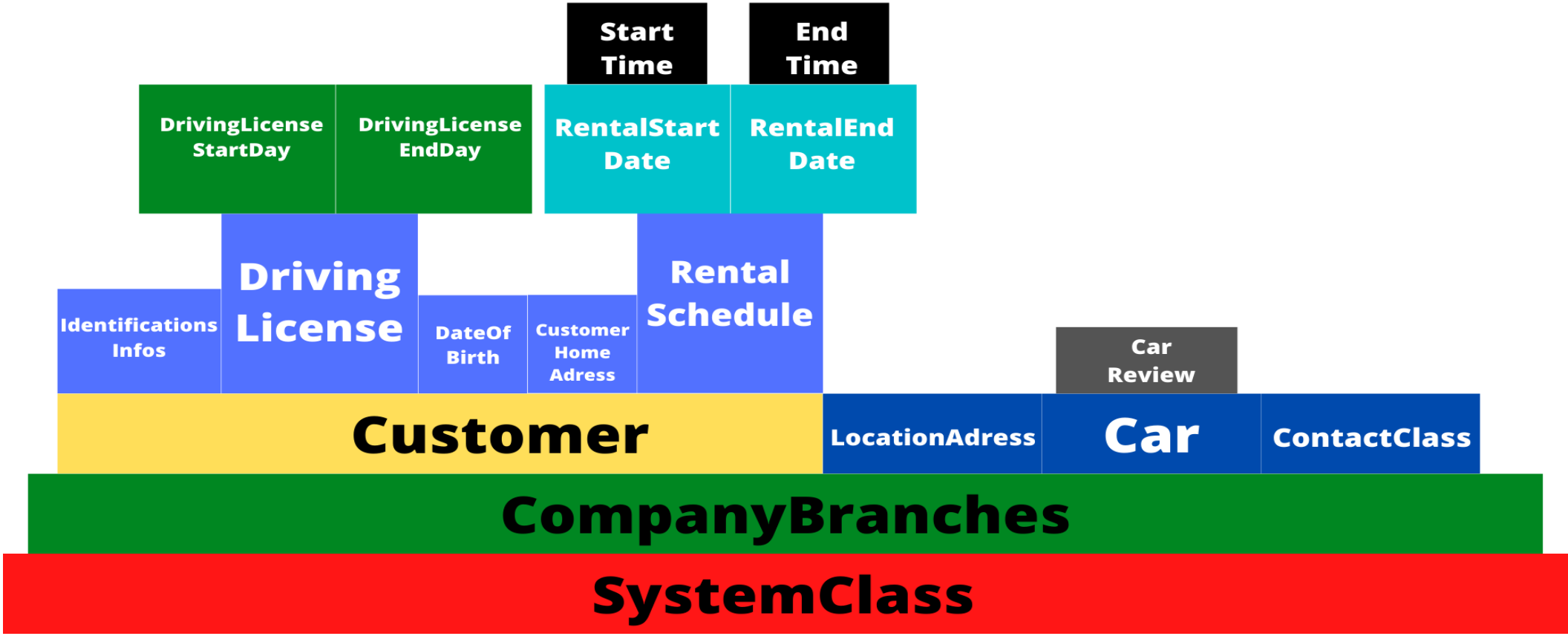
System Class

- company's name – static member
- Company's locations - array
    - Branch Name
    - Country
    - City
    - Number of cars available
    - Mark (extracted from the reviews)
    - Review
    - Customers - array
        - First name
        - Last name
        - Email
        - Telephone
        - CNP
        - ID card series
        - ID card number
        - Mark for the company branch
        - Rental Process Class
            - RentalStartDate
                - Start day
                - Start month
                - Start year

        - Start Time
            - Hour
            - Minutes
    - RentalEndtDate
        - End day
        - End month
        - End year
        - EndTime
            - Hour
            - Minutes
- Driving license Class
    - StartDate
        - Start day
        - Start month
        - Start year
    - EndDate
        - End day
        - End month
        - End year
    - Is the person able to drive? (yes - true/no - false)
- Customers's home address
    - Country
    - City
    - Postal code
    - Street
    - Number
- Date of birth Class
    - Day
    - Month
    - Year

- Car Class
  - Make
  - Color
  - Brake horsepower
  - Engine
  - Doors
  - Seats
  - Transmission (automatic/manual)
  - Consumption
  - Availability (if during a certain period, the car is available for rental)
  - Price per day
  - Deposit
  - Payment in advance
  - Type (economy/medium/premium)
  - Car's review Class
    - Value considering the price
    - Car cleanliness
    - Comfort
    - Car condition
    - Overall mark
    - Review (very good/ good/ awful)
- Location's address Class
  - Street
  - Number
  - Postal code
- Contact Class
  - Telephone
  - Email
  - Manager name

3. Schema bloc



4. Descrierea modului de implementare
   Am folosit OOP-ul pentru realizarea aplicației. Am împărțit clasele și metodele în fișiere separate. Am utilizat fișiere text și CSV pentru preluarea și scrierea datelor. Ne-am bazat foarte mult pe algoritmii și structurile de date din STL.

5. Descrierea algoritmilor utilizati
   Am folosit algoritmi din STL în principal ca cei de aflare a dimensiunii unui vector, de aflare a referinței de început și sfârșit, de ștergere a unui element(erease(), size(), begin(), end()). Am implementat algoritmi de validare a input-ului, de căutare secvențială, de preluare și scriere în fișiere, dar și funcții complexe ce reprezintă "features" în proiectul nostru ca: funcție pentru înregistrarea unei firme, pentru logare și înregistrare.

6. Modul de utilizare a aplicaţiei implementate
   Utilizarea aplicației se face din consolă, fiind nevoit un mediu specializat de rulare, adică un IDE. Recomandarea este VisualStudioCommunity 2022.

7. Descrierea funcţionalităţilor modulelor implementate
   Modulele implementate au ca scop structurarea și organizarea aplicației. Unele module conțin definiții de clase și prototipuri de metode, alte module conțin implementările metodelor, alte module conțin prototipuri de funcții, alte module conțin implementări la funcții create de noi, iar alte module conțin bucăți de cod separate pentru a reduce dimensiunea unui fișier.

Rezultate

O soluție care poati fi utilizată de cei care au nevoie de o mașină. Rezolvarea unei probleme din viața reală. Centralizarea firmelor într-un singur lor, făcând procesul mult mai ușor și accesibil. Un proiect complex, robust și funcțional cu o scalabilitate foarte mare. Abstractizarea cu success într-un limbaj de programare a unei probleme din viața reală. Folosim sisteme de logare, de preluare date dintr-o bază de date(CSV) și scriere in fișiere.

Alte informatii:

Sunt 18 clase:

1. SystemClass este clasa de bază: este alcătuită dintr-un atribut simplu și un vector cu elemente de tip clasă.
2. CompanyBranches este alcătuită din 7 atribute simple și 4 atribute de timp vector cu elemente de tip clasă.
3. Customer este alcătuită din 5 clasă.
4. LocationAdress este alcătuită din 4 atribute simple.
5. Car este alcătuită din 1 clasa și 13 atribute simple.
6. CarReview este alcătuită din 6 atribute simple
7. ContactClass este alcătuită din 3 atribute simple.
8. IdentificationInfos este alcătuită din 8 atribute simple.
9. DrivingLicense este alcătuită din 2 clase și un atribut simplu.
10. DateOfBirth este alcătuită din 3 atribute simple.
11. CustomerHomeAdress este alcătuită din 5 atribute simple.
12. RentalSchedule este alcătuită din 2 clase.
13. DrivingLicenseStartDay este alcătuită din 3 atribute simple.
14. DrivingLicenseEndtDay este alcătuită din 3 atribute simple.
15. RentalStartDate este alcătuită din 3 atribute simple și o clasă.
16. RentalEndDate este alcătuită din 3 atribute simple și o clasa.
17. StartTime este alcătuită din 2 atribute simple.
18. EndtTime este alcătuită din 2 atribute simple

.

Structuri de date folosite: vector

Paradigma de programare folosita: OOP

IDE folosit: VisualStudioCommunity 2022

Librarii importante folosite:

- STL: vector, string
- Windows.h
- iterator
- fstream
- ostream

Extra:

- lucrul cu fisiere
- stocarea datelor in fisiere de tip CSV
- structurare proiect in fisiere separate(nume_fisier.h)
- Implementarea claselor si a metodelor s-a facut in fisiere separate