Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

ONP

autor	Mykyta Shemechko
prowadzący	dr inż. Artur Pasierbek
rok akademicki	2021/2022
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium	piątek, 10:00 – 11:30
termin oddania sprawozdania	2022-07-04

1 Treść zadania

Program do zarządzania samochodami w firmie taksówkowej.

2 Analiza zadania

Celem projektu jest napisanie programu do zarządzania samochodami w firmie taksówkowej. Niezbędny jest interfejs posługiwania się takim programem. Także trzeba stworzyć klasy różnych typów samochodów dla ułatwienia zarządzania nimi.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Po uruchomieniu wymagane jest wprowadzenie jednej z trzech poleceń tekstowych.

```
list | add | remove
```

Po wykonaniu poleceń program zostanie zakończony.

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem obiektowym. W programie rozdzielono interfejs od logiki.

4.1 Ogólna struktura programu

Funkcja główna wywołuje funkcję menu głównego ”mainMenu”. Po wyświetleniu komunikatu, wywoływana jest funkcja ”chosenOption”, która w zależności od wybranej opcji wyświetla podalsze instrukcje. Przy wyświetlaniu listy, pobierane są dane obiektów z pliku z danymi. Przy dodawaniu samochodu, tworzony jest obiekt klasy ”Taxi”, ”Van”, ”Limo”albo ”SDTaxi”(self-driving taxi). Konstruktor każdej z tych klas zawiera też klasę kierowcy ”Driver”, która dziediczy po klasie ”Person”. Zadaniem klasy ”Person jest sprawdzanie numerów pesel.”Później dane tych objeków są zapisywane do pliku dla późniejszego użytkowania. Przy usuwaniu samochodu, dane odpowiedniego samochodu są ususwane z pliku z danymi.

4.2 Szczegółowy opis klas, metod i funkcji

Szczegółowy opis klas, metod i funkcji zawarty jest w załączniku.

5 Testowanie

Program został przetestowany na różnego rodzaju plikach. Jeżeli wpisywane przy dodawaniu samochodu dane są w jakiś spoób niepoprawne, program wyrzuci exception.

6 Wnioski

Programowanie obiektowe nie jest łatwe. Wskaźniki tworzą wiele problemów podczas programowania, które cały czas trzeba rozwiązywać. Czasami naprawienie jednego problemu w kodzie jakieś klasy powoduje stworzenie jeszcze dziesięciu w innych miejscach.

Dodatek
Szczegółowy opis klas, metod i funkcji

Projekt zaliczeniowy z PK-SSI

Wygenerowano przez Doxygen 1.9.3

Rozdział 1

FleetManager

A system for taxi companies to manage their car fleet.

Rozdział 2

Indeks hierarchiczny

2.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Rozdział 3

Indeks klas

3.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 4

Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

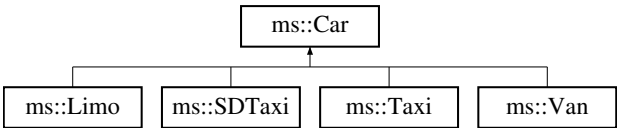
Rozdział 5

Dokumentacja klas

5.1 Dokumentacja klasy ms::Car

`#include <Car.h>`

Diagram dziedziczenia dla ms::Car



Metody publiczne

- **Car** (const std::string &licensePlate, const std::string licenseType, const std::string &brand, const std::string &model, const std::string &color, std::shared_ptr< ms::Driver > driver)
- std::string **getLicensePlate** () const
- virtual std::vector< std::string > **getCarData** () const

5.1.1 Opis szczegółowy

Klasa ogólna samochodów

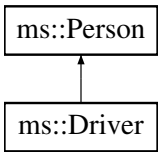
Dokumentacja dla tej klasy została wygenerowana z pliku:

- Car.h

5.2 Dokumentacja klasy ms::Driver

`#include <Driver.h>`

Diagram dziedziczenia dla ms::Driver



Metody publiczne

- **Driver** (const std::string &firstName, const std::string &lastName, const std::string &pesel, const std::string &licenseType)
- bool **checkLicenseType** (const std::string licenseType) const
- std::vector< std::string > **getDriverData** () const

5.2.1 Opis szczegółowy

Klasa Kierowcy

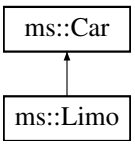
Dokumentacja dla tej klasy została wygenerowana z pliku:

- Driver.h

5.3 Dokumentacja klasy ms::Limo

`#include <Limo.h>`

Diagram dziedziczenia dla ms::Limo



Metody publiczne

- **Limo** (const std::string &licensePlate, const std::string licenseType, const std::string &brand, const std::string &model, const std::string &color, std::shared_ptr< ms::Driver > driver, const double &length)
- const double **getLength** ()
- virtual std::vector< std::string > getCarData () const

5.3.1 Opis szczegółowy

Klasa limuzyny

5.3.2 Dokumentacja funkcji składowych

5.3.2.1 getCarData()

```
virtual std::vector< std::string > ms::Limo::getCarData ( ) const  [inline], [virtual]
```

Reimplementowana z ms::Car.

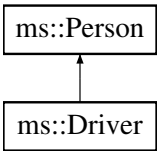
Dokumentacja dla tej klasy została wygenerowana z pliku:

- Limo.h

5.4 Dokumentacja klasy ms::Person

```
#include <Person.h>
```

Diagram dziedziczenia dla ms::Person



Metody publiczne

- Person (const std::string &firstName, const std::string &lastName, const std::string pesel)
- std::string **getFirstName** () const
- std::string **getLastName** () const
- std::string **getFullName** () const
- std::string **getPesel** () const
- std::vector< std::string > **getPersonData** () const

5.4.1 Opis szczegółowy

Klasa osoby

5.4.2 Dokumentacja konstruktora i destruktora

5.4.2.1 Person()

```
ms::Person::Person (
            const std::string & firstName,
            const std::string & lastName,
            const std::string pesel )  [inline]
```

Sprawdzane warunki pesel

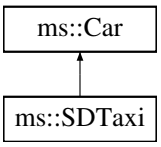
Dokumentacja dla tej klasy została wygenerowana z pliku:

- Person.h

5.5 Dokumentacja klasy ms::SDTaxi

`#include <SDTaxi.h>`

Diagram dziedziczenia dla ms::SDTaxi



Metody publiczne

- **SDTaxi** (const std::string &licensePlate, const std::string licenseType, const std::string &brand, const std::string &model, const std::string &color, std::shared_ptr< ms::Driver > driver, const std::string &speedSetting)
- const std::string **getSpeedSetting** ()
- virtual std::vector< std::string > getCarData () const

5.5.1 Opis szczegółowy

Klasa taxi prowadzących siebie

5.5.2 Dokumentacja funkcji składowych

5.5.2.1 getCarData()

`virtual std::vector< std::string > ms::SDTaxi::getCarData ( ) const [inline], [virtual]`

Reimplementowana z ms::Car.

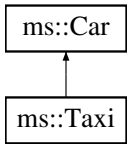
Dokumentacja dla tej klasy została wygenerowana z pliku:

- SDTaxi.h

5.6 Dokumentacja klasy ms::Taxi

`#include <Taxi.h>`

Diagram dziedziczenia dla ms::Taxi



Metody publiczne

- **Taxi** (const std::string &licensePlate, const std::string licenseType, const std::string &brand, const std::string &model, const std::string &color, std::shared_ptr< ms::Driver > driver, const std::string &luxClass)
- const std::string **getLuxClass** ()
- virtual std::vector< std::string > getCarData () const

5.6.1 Opis szczegółowy

Klasa taxi

5.6.2 Dokumentacja funkcji składowych

5.6.2.1 getCarData()

```
virtual std::vector< std::string > ms::Taxi::getCarData ( ) const  [inline], [virtual]
```

Reimplementowana z ms::Car.

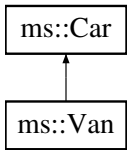
Dokumentacja dla tej klasy została wygenerowana z pliku:

- Taxi.h

5.7 Dokumentacja klasy ms::Van

```
#include <Van.h>
```

Diagram dziedziczenia dla ms::Van



Metody publiczne

- **Van** (const std::string &licensePlate, const std::string licenseType, const std::string &brand, const std::string &model, const std::string &color, std::shared_ptr< ms::Driver > driver, const int &capacity)
- const int **getCapacity** ()
- virtual std::vector< std::string > getCarData () const

5.7.1 Opis szczegółowy

Klasa furgonetki

5.7.2 Dokumentacja funkcji składowych

5.7.2.1 getCarData()

```
virtual std::vector< std::string > ms::Van::getCarData ( ) const  [inline], [virtual]
```

Reimplementowana z ms::Car.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Van.h

Rozdział 6

Dokumentacja plików

6.1 Car.h

```
1 #ifndef FLEETMANAGER_CAR_H
2 #define FLEETMANAGER_CAR_H
3
4 #include <string>
5 #include <vector>
6 #include "Driver.h"
7 #include <memory>
8 namespace ms {
12      class Car {
13          std::string licensePlate;
14          std::string licenseType;
15          std::string brand;
16          std::string model;
17          std::string color;
18          std::shared_ptr<ms::Driver> driver;
19
20      public:
21          Car(const std::string& licensePlate, const std::string licenseType, const std::string& brand,
        const std::string& model, const std::string& color, std::shared_ptr<ms::Driver> driver) :
22                  licensePlate(licensePlate), licenseType(licenseType), brand(brand), model(model),
        color(color), driver(driver) {
23              if(!(driver == nullptr)){
24                  if (!driver->checkLicenseType(licenseType)) {
25                      throw ("\n" + licenseType + " license type is required for this vehicle!\n");
26                  }
27              }
28
29          }
30
31          std::string getLicensePlate() const {
32              return licensePlate;
33          }
34
35          virtual std::vector<std::string> getCarData() const {
36              std::vector<std::string> carData;
37              carData.push_back(licensePlate);
38              carData.push_back(licenseType);
39              carData.push_back(brand);
40              carData.push_back(model);
41              carData.push_back(color);
42              if(driver == nullptr){
43                  carData.push_back("no_driver");
44              } else {
45                  std::vector<std::string> driverData = driver->getDriverData();
46                  carData.insert(carData.end(), driverData.begin(), driverData.end());
47              }
48              //adding driver data to the car data
49              return carData;
50          }
51      };
52
53 }
54
55 #endif //FLEETMANAGER_CAR_H
```

6.2 Driver.h

```
1 #ifndef FLEETMANAGER_DRIVER_H
2 #define FLEETMANAGER_DRIVER_H
3
4
5 #include <string>
6 #include <vector>
7 #include "Person.h"
8
9 namespace ms {
13     class Driver : public ms::Person {
14         std::string licenseType;
15     public:
16         Driver(const std::string &firstName, const std::string &lastName, const std::string &pesel, const
       std::string& licenseType) :
17             Person(firstName, lastName, pesel), licenseType(licenseType){
18
19         }
20
21
22         bool checkLicenseType(const std::string licenseType) const {
23             if (Driver::licenseType == licenseType) return true;
24             return false;
25         }
26
27
28         std::vector<std::string> getDriverData() const {
29             std::vector<std::string> driverData;
30             driverData.push_back(Driver::getPesel());
31             driverData.push_back(Driver::getFirstName());
32             driverData.push_back(Driver::getLastName());
33             driverData.push_back(licenseType);
34             return driverData;
35         }
36     };
37 }
38
39 #endif //FLEETMANAGER_DRIVER_H
```

6.3 FleetOperations.h

```
1 #ifndef FLEETMANAGER_FLEETOPERATIONS_H
2 #define FLEETMANAGER_FLEETOPERATIONS_H
3 #include "Car.h"
4
5
6 typedef std::vector<std::shared_ptr<ms::Car» Fleet;
7 typedef std::vector<std::vector<std::string» FleetData;
8
14 std::shared_ptr<ms::Car> convertDataToCar(std::vector<std::string> input);
15
21 FleetData readFleetDataFromFile(std::string fileName);
22
23
29 void writeFleetDataToFile(std::string fileName, FleetData fleetData);
30
31
37 Fleet convertFleetDataToFleet(FleetData fleetData);
38
39
45 FleetData convertFleetToFleetData(Fleet fleet);
46
47 #endif //FLEETMANAGER_FLEETOPERATIONS_H
```

6.4 Interface.h

```
1 #ifndef FLEETMANAGER_INTERFACE_H
2 #define FLEETMANAGER_INTERFACE_H
3
4
9 void drawList(std::vector<std::vector<std::string» list);
10
11
16 void mainMenu(std::string saveFile);
17
18
24 void chosenOption(std::string option, std::string saveFile);
25
26
27 #endif //FLEETMANAGER_INTERFACE_H
```

6.5 Limo.h

```
1 #ifndef FLEETMANAGER_LIMO_H
2 #define FLEETMANAGER_LIMO_H
3
4 #include "Car.h"
5
6
7 namespace ms {
11      class Limo : public ms::Car {
12          double length;
13      public:
14          Limo(const std::string& licensePlate, const std::string licenseType, const std::string& brand,
        const std::string& model, const std::string& color, std::shared_ptr<ms::Driver> driver, const double
        &length) :
15                  Car(licensePlate, licenseType, brand, model, color, driver), length(length) {
16
17          }
18
19          const double getLength() {
20              return length;
21          }
22          virtual std::vector<std::string> getCarData() const {
23              std::vector<std::string> carData = Car::getCarData();
24              carData.insert(carData.begin(), std::to_string(length));
25              carData.insert(carData.begin(), "Limo");
26
27              return carData;
28          }
29
30      };
31 }
32
33 #endif //FLEETMANAGER_LIMO_H
```

6.6 Person.h

```
1 #ifndef FLEETMANAGER_PERSON_H
2 #define FLEETMANAGER_PERSON_H
3
4
5 #include <string>
6
7 namespace ms {
11      class Person {
12          std::string firstName;
13          std::string lastName;
14          std::string pesel;
15          constexpr const static uint8_t  w[] = {1, 3, 7, 9, 1, 3, 7, 9, 1, 3, 1};
16          const std::string wrong_pesel_exception = "\nInvalid pesel.\n";
17      public:
18          Person(const std::string& firstName, const std::string& lastName, const std::string pesel):
19          firstName(firstName), lastName(lastName), pesel(pesel) {
23              if (pesel.length() != 11) throw wrong_pesel_exception;
24              for (const auto c : pesel) {
25                  if (c < '0' || c > '9') throw wrong_pesel_exception;
26              }
27              uint16_t sum = 0;
28              for (size_t i = 0; i < pesel.size(); ++i) {
29                  sum += (pesel[i] - '0') * w[i];
30              }
31              if (sum % 10) throw wrong_pesel_exception;
32          }
33          std::string getFirstName() const {
34              return firstName;
35          }
36          std::string getLastName() const {
37              return lastName;
38          }
39
40          std::string getFullName() const {
41              return firstName + " " + lastName;
42          }
43          std::string getPesel() const {
44              return pesel;
45          }
46          std::vector<std::string> getPersonData() const {
47              std::vector<std::string> personData;
48              personData.push_back(firstName);
49              personData.push_back(lastName);
50              personData.push_back(pesel);
51              return personData;
52          }
```

```
53     };
54
55 }
56
57
58 #endif //FLEETMANAGER_PERSON_H
```

6.7 SDTaxi.h

```
1 #ifndef FLEETMANAGER_SDTAXI_H
2 #define FLEETMANAGER_SDTAXI_H
3
4
5 #include "Car.h"
6
7
8 namespace ms {
12     class SDTaxi : public ms::Car {
13         std::string speedSetting;
14     public:
15         SDTaxi(const std::string& licensePlate, const std::string licenseType, const std::string& brand,
       const std::string& model, const std::string& color, std::shared_ptr<ms::Driver> driver, const
       std::string &speedSetting) :
16                 Car(licensePlate, licenseType, brand, model, color, driver), speedSetting(speedSetting) {
17             if(driver != nullptr) {
18                 throw driver;
19             }
20             if(!(speedSetting== "quick" || speedSetting == "standard")) {
21                 throw "This speed setting is not allowed! Use either \"quick\" or \"standard\"";
22             }
23             if(licenseType != "None") throw "No driver's license is required for this vehicle!";
24         }
25
26         const std::string getSpeedSetting() {
27             return speedSetting;
28         }
29
30
31         virtual std::vector<std::string> getCarData() const {
32             std::vector<std::string> carData = Car::getCarData();
33             carData.insert(carData.begin(), speedSetting);
34             carData.insert(carData.begin(), "SDTaxi");
35             return carData;
36         }
37
38     };
39 }
40
41 #endif //FLEETMANAGER_SDTAXI_H
```

6.8 Taxi.h

```
1 #ifndef FLEETMANAGER_TAXI_H
2 #define FLEETMANAGER_TAXI_H
3
4
5 #include "Car.h"
6
7
8 namespace ms {
12     class Taxi : public ms::Car {
13         std::string luxClass;
14     public:
15         Taxi(const std::string& licensePlate, const std::string licenseType, const std::string& brand,
       const std::string& model, const std::string& color, std::shared_ptr<ms::Driver> driver, const
       std::string &luxClass) :
16                 Car(licensePlate, licenseType, brand, model, color, driver), luxClass(luxClass) {
17         }
18
19         const std::string getLuxClass() {
20             return luxClass;
21         }
22         virtual std::vector<std::string> getCarData() const {
23
24             std::vector<std::string> carData = Car::getCarData();
25             carData.insert(carData.begin(), luxClass);
26             carData.insert(carData.begin(), "Taxi");
27
28             return carData;
```

```
29         }
30
31     };
32 }
33
34 #endif //FLEETMANAGER_TAXI_H
```

6.9 Van.h

```
1 #ifndef FLEETMANAGER_VAN_H
2 #define FLEETMANAGER_VAN_H
3
4
5 #include "Car.h"
6
7
8 namespace ms {
12     class Van : public ms::Car {
13         int capacity;
14     public:
15         Van(const std::string& licensePlate, const std::string licenseType, const std::string& brand,
       const std::string& model, const std::string& color, std::shared_ptr<ms::Driver> driver, const int
       &capacity) :
16                 Car(licensePlate, licenseType, brand, model, color, driver), capacity(capacity) {
17
18         }
19
20         const int getCapacity() {
21             return capacity;
22         }
23         virtual std::vector<std::string> getCarData() const {
24             std::vector<std::string> carData = Car::getCarData();
25             carData.insert(carData.begin(), std::to_string(capacity));
26             carData.insert(carData.begin(), "Van");
27
28             return carData;
29         }
30
31     };
32 }
33
34 #endif //FLEETMANAGER_VAN_H
```

Skorowidz