HSD
FH-HAGENBERG

Name(1):   Daniel Weyrer

Abgabetermin: 29.10.2019

Name(2):   Viktoria Streibl

Punkte:

Übungsgruppe: Gruppe 1

korrigiert:

Geschätzter Aufwand in Ph: 6 | 7

Effektiver Aufwand in Ph: 8 | 10

Beispiel1: Fuhrpark (24 Punkte)

Ein Fuhrpark soll verschiedene Fahrzeuge verwalten: PKWs, LKWs und Motorräder. Entwerfen Sie dazu ein geeignetes Klassendiagramm (Klassenhierarchie) und ordnen Sie folgende Eigenschaften den einzelnen Klassen zu: Automarke, Kennzeichen und die Kraftstoffart (Benzin, Diesel oder Gas). Weiters muss jedes Fahrzeug ein Fahrtenbuch führen. Ein Eintrag im Fahrtenbuch speichert das Datum und die Anzahl der gefahrenen Kilometer an diesem Tag.

Geben Sie Set- und Get-Methoden nur dann an, wenn sie sinnvoll sind!

Die Fahrzeuge stellen zur Ausgabe eine `Print`-Methode zur Verfügung!

Ein Fuhrpark soll folgende Aufgaben erledigen können:

1. Hinzufügen von neuen Fahrzeugen.

2. Entfernen von bestehenden Fahrzeugen.

3. Suchen eines Fahrzeuges nach seinem Kennzeichen.

4. Ausgeben aller Fahrzeuge samt ihrer Eigenschaften und dem Fahrtenbuch auf dem Ausgabestrom und in einer Datei.

5. Verwenden Sie im Fuhrpark zur Verwaltung aller Fahrzeuge einen entsprechenden Container!

6. Der Fuhrpark muss kopierbar und zweisbar sein!

Die Ausgabe soll folgendermaßen aussehen:

```
Fahrzeugart: Motorrad
Marke:       Honda CBR
```

```
Kennzeichen: FR-45AU
04.04.2018:  52 km
05.06.2018:   5 km

Fahrzeugart: PKW
Marke:       Opel Astra
Kennzeichen: LL-345UI
04.07.2018:  51 km
05.07.2018:  45 km

Fahrzeugart: LKW
Marke:       Scania 1100
Kennzeichen: PE-34MU
04.08.2018:  512 km
05.08.2018:   45 km
07.08.2018:  678 km
14.08.2018:  321 km
```

Die Fahrzeugart wird nicht als Attribut gespeichert, sondern bei der Ausgabe direkt ausgegeben! Für den Fuhrpark ist der Ausgabeoperator zu überschreiben.

Für jedes Fahrzeug soll die Summe der gefahrenen Kilometer ermittelt werden können und der Fuhrpark soll die Summe der gefahrenen Kilometer aller seiner Fahrzeuge liefern. Verwenden Sie dazu entsprechende Algorithmen.

Geben Sie wo nötig Exceptions und Fehlermeldungen aus!

Überlegen Sie sich die jeweils notwendigen Members und Methoden der einzelnen Klassen und implementieren Sie einen ausführlichen Testtreiber.

Verfassen Sie weiters eine Systemdokumentation (Funktionalität, Klassendiagramm, Schnittstellen der beteiligten Klassen, etc.)! In dieser soll enthalten sein, wie Sie die Aufgabenstellung auf die Teamteilnehmer verteilt haben. Geben Sie zusätzlich in den entsprechenden Header-Dateien den Verfasser an!

Führen Sie zusammen mit Ihrer Teamkollegin bzw. mit Ihrem Teamkollegen vor der Realisierung eine Aufwandsschätzung in (Ph) durch und notieren Sie die geschätzte Zeitdauer am Deckblatt. Dies gilt auch für alle nachfolgenden Übungen.

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!

SDP - Exercise 01

winter semester 2019/20

Viktoria Streibl - S1810306013

Daniel Weyrer - S1820306044

October 29, 2019

Contents

1 Organizational

1.1 Team

- Viktoria Streibl - S1810306013

- Daniel Weyrer - S1820306044

1.2 Roles and responsibilities

1.2.1 Jointly

- planning

- testing (Testdriver)

- Documentation

- Systemdocumentation

- Class Diagram

1.2.2 Viktoria Streibl

- Base Class for Vehicles

- Derived Classes

  Class Motorcycle

  Class Car

  Class Truck

- Class Logbook

  plausibility test of input data (current Mileage and Date)

1.2.3 Daniel Weyrer

- Main Class Carpool

1.3 Effort

1.3.1 Viktoria Streibl

- estimated: 7ph

- actually: 10 ph
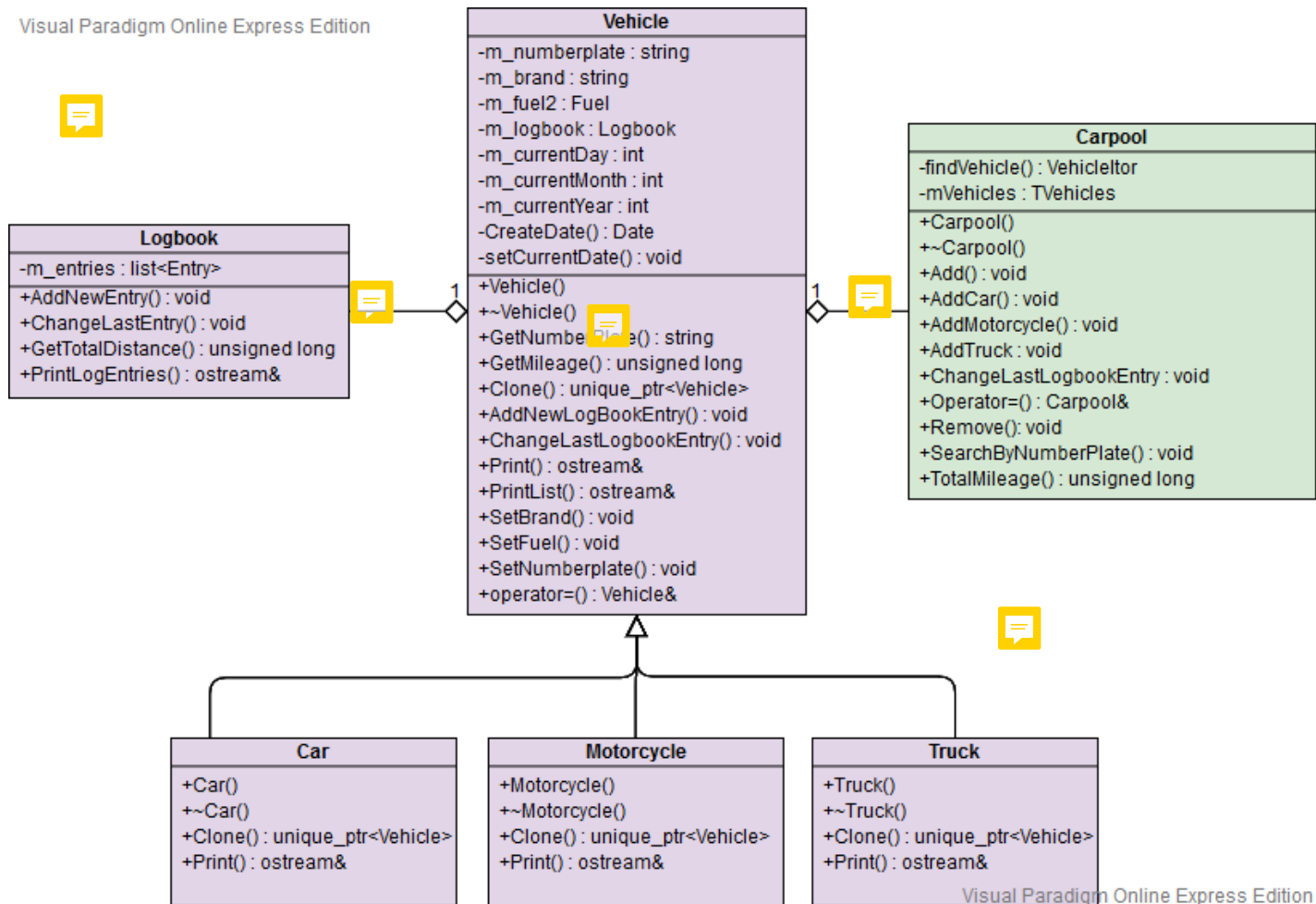
1.3.2 Daniel Weyrer

- estimated: 6 ph

- actually: 8 ph

2 Requirement Definition(System Specification)

It was a carpool desired the various types of vehicles includes, such as cars, trucks and motorcycles. Each vehicle type should also include and output some key data such as car make, license plate and fuel type. In addition, each vehicle must keep a logbook and enter the kilometers driven in one day. Any number of vehicles can be added and deleted in the program. You can search for the license plate and output all vehicles with the basic data.

3 System Design

3.1 Classdiagram

Visual Paradigm Online Express Edition

Vehicle

- -m_numberplate : string
- -m_brand : string
- -m_fuel2 : Fuel
- -m_logbook : Logbook
- -m_currentDay : int
- -m_currentMonth : int
- -m_currentYear : int
- -CreateDate() : Date
- -setCurrentDate() : void

- +Vehicle()
- +~Vehicle()
- +GetNumberplate() : string
- +GetMileage() : unsigned long
- +Clone() : unique_ptr<Vehicle>
- +AddNewLogBookEntry() : void
- +ChangeLastLogbookEntry() : void
- +Print() : ostream&
- +PrintList() : ostream&
- +SetBrand() : void
- +SetFuel() : void
- +SetNumberplate() : void
- +operator=() : Vehicle&

Logbook

- -m_entries : list<Entry>
- +AddNewEntry() : void
- +ChangeLastEntry() : void
- +GetTotalDistance() : unsigned long
- +PrintLogEntries() : ostream&

Carpool

- -findVehicle() : VehicleItor
- -mVehicles : TVehicles
- +Carpool()
- +~Carpool()
- +Add() : void
- +AddCar() : void
- +AddMotorcycle() : void
- +AddTruck : void
- +ChangeLastLogbookEntry : void
- +Operator=() : Carpool&
- +Remove(): void
- +SearchByNumberPlate() : void
- +TotalMileage() : unsigned long

Car

- +Car()
- +~Car()
- +Clone() : unique_ptr<Vehicle>
- +Print() : ostream&

Motorcycle

- +Motorcycle()
- +~Motorcycle()
- +Clone() : unique_ptr<Vehicle>
- +Print() : ostream&

Truck

- +Truck()
- +~Truck()
- +Clone() : unique_ptr<Vehicle>
- +Print() : ostream&

Visual Paradigm Online Express Edition

3.2 Design Decisions

3.2.1 Inheritance

The program should output the type of vehicle. To implement this as simple as possible, a base class has been implemented which contains all common functions and variables. A function "Print" is overloaded in the classes the heirs and there immediately the correct type is spent. Furthermore, this approach allows a quick expansion of other vehicle types.

3.2.2 Fuel

The Fuel was declared as enum. You can also expand this easily and it is uniform.

3.2.3 Search Vehicle

Vehicle is searched by vehicle number because it has to be unique. This is already regulated by the authority, which is why we do not check it it is unique.

3.2.4 Logbook

The logbook contains a date and a mileage of the respective day. This date is stored in a stuct to check if the day, month and year is available. If the date of a new entry is older than the date of the last entry, an error message will be displayed. Only current entries can be entered. If new entry is the same date as the old entry, then the mileage will be added up.

4 Component Design

4.1 Class Carpool

Manages all vehicles in the carpool. It contains the following functions:

- Add a new Vehicle

- Remove a Vehicle

- Add new logbook entry

- Change last logbook entry

- Search vehicle by numberplate

- Print all vehicles

- Get the sum of milages

The Carpool class manages all Verhicles. It uses unique pointers stored in a vector, to avoid shallow-copies. With the methods "AddCar", "AddTruck" and "AddMotorcycle" a new vehicle can be created. Should a car already exist with the same license plate. So this vehicle is not stored and an error message output. "Remove" deletes a vehicle with a specific flag. If none is stored with the license plate an exception get's thrown and caught in the same method. With "AddLogbookEntry" a day, month, year and the driven kilometers are given and saved as a new entry in the logbook. With "ChangeLastLogbookEntry" the last entry can be edited. With the method "SearchByNumberplate" it is possible to search for a vehicle with the specific flag and to output it. "PrintVehicles" outputs all stored vehicles, more exactly in the respective classes. "TotalMileage" provides the total number of kilometers driven, driven by all vehicles.

4.2 Class Vehicle

Is the base class of all vehicle types. It contains the following functions:

- Add Brand

- Add Numberplate

- Add Fuel

- Add new logbook entry

- Change last logbook entry

- Get numberplate

- Get driven kilometers

Vehicle is the base class of all vehicle types. It contains the brand, numberplate and fuel of a vehicle. Furthermore, it also manages the logbook entries of a single vehicle. There is a setter for brand, numberplate and fuel. For numerplate and the kilometers driven is also a getter. With "AddLogbookEntry" a day, month, year and the driven kilometers are given and saved as a new entry in the logbook. With "ChangeLastLogbookEntry" the last entry can be edited.

The private methode "CreateDate" converts the individual values of day, month and year to a struct named "Date". It also checks if the day is between 1 and 31. But not how many days in a month. That was not considered for this solution. Thus, a 30th February is possible as a date. The month also checks if it is between 1 and 12 and if the year is not negative. Another check is made if the date is not in the future. Which is why the private method "setCurrentDate" saves the current date, so it is able to compare later.

4.3 Class Car

This class represents a car.

- Print all car data and the associated log entries

"Print" overrides the function of the base clase and outputs the vehicle type: "PKW", as well as brand and numberplate. Further, all logbook entries will be read out and printed out.

4.4 Class Truck

This class represents a truck.

- Print all truck data and the associated log entries

"Print" overrides the function of the base clase and outputs the vehicle type: "LKW", as well as brand and numberplate. Further, all logbook entries will be read out and printed out.

4.5 Class Motorcycle

This class represents a motorcycle.

- Print all motorcycle data and the associated log entries

"Print" overrides the function of the base clase and outputs the vehicle type: "Motorrad", as well as brand and numberplate. Further, all logbook entries will be read out and printed out.

4.6 Class Logbook

Class for collection driver data. It saves the driven kilometer per day of a vehicle. It contains the following functions:

- Add new logbook entry

- Change last logbook entry

- Print all logbook entries

- Adds up all the kilometers together

This class has a struct "Date" that includes the current date with "Tag", "Monat" and year "Jahr". It saves the entries in a list. An entry consists of two parts, a date and the kilometers driven on this day. AddNewEntry adds a new entry to the list. However, if the new date is older than the last entry, so an error is advertised. If a new entry has the same date as that of the last one, the two mileage become added and written to the list. With "ChangeLastEntry" you can change the last saved entry. "GetTotalDistance" counts the kilometers that have been saved in Logbook and returns them.

4.7 TestDriver

The Testdriver test alle functions of the Carpool. It adds cars, trucks and motorcycles and deletes them. It searches vehicles by numberplate and print all of them. It tests all logbook functions and the carpool.

5 Test Protocol

It has been tested in the file "TestDriver", the following points have been tested:

- Add new vehicle

- Remove vehicle

- Find a vehicle

- Print vehicles

- Test Logbook

- Test Carpool

5.1 Console Output

```
 1 ##################################
 2 ## 1. Add Vehicles
 3 ##################################
 4 Cars added successfully...
 5 Trucks added successfully...
 6 Motorcycles added successfully...
 7
 8 ##################################
 9 ## 2. Remove Vehicle
10 ##################################
11 Car removed successfully...
12 Truck removed successfully...
13 Motorcycle removed successfully...
14 Delete failed: The entered numberplate is not registered in this carpool!
15
16 ##################################
17 ## 3. Find Vehicle
18 ##################################
19 Fahrzeugart: PKW
20 Marke: Fiat Multipla
21 Kennzeichen: RO-666H
22 The vehicle with the numberplate :RO-677H is not registered in this carpool!
23
24 ##################################
25 ## 4. Print vehicles
26 ##################################
27 Fahrzeugart: PKW
28 Marke: Audi
29 Kennzeichen: PE-21HR
30
31 Fahrzeugart: PKW
32 Marke: Daniels Sweetheart
33 Kennzeichen: LL-Caro1
34
35 Fahrzeugart: PKW
36 Marke: Fiat Multipla
37 Kennzeichen: RO-666H
38
39 Fahrzeugart: LKW
40 Marke: Man
41 Kennzeichen: LL-Bau1
42
43 Fahrzeugart: LKW
44 Marke: Volvo
45 Kennzeichen: FR-Erde1
46
47 Fahrzeugart: Motorrad
48 Marke: BMW
49 Kennzeichen: IL-24TW
```

```
50
51
52
53 #################################
54 ## 5. Logbook
55 #################################
56 Add 3 Entries
57 Add Entry with same date
58 Add old Entry
59 Wrong Date: Please input a date after the last entry
60 Add Entry for the future
61 Wrong Date: Back to the Future?
62
63 #################################
64 ## 6. Carpool copy & co
65 #################################
66 Original...
67 Fahrzeugart: PKW
68 Marke: Audi
69 Kennzeichen: PE-21HR
70
71 Fahrzeugart: PKW
72 Marke: Daniels Sweetheart
73 Kennzeichen: LL-Caro1
74 14.2.2017     45 km
75 14.2.2018     50 km
76
77 Fahrzeugart: PKW
78 Marke: Fiat Multipla
79 Kennzeichen: RO-666H
80
81 Fahrzeugart: LKW
82 Marke: Man
83 Kennzeichen: LL-Bau1
84
85 Fahrzeugart: LKW
86 Marke: Volvo
87 Kennzeichen: FR-Erde1
88
89 Fahrzeugart: Motorrad
90 Marke: BMW
91 Kennzeichen: IL-24TW
92
93
94 Copied via Copy-CTor...
95 Fahrzeugart: PKW
96 Marke: Audi
97 Kennzeichen: PE-21HR
98
99 Fahrzeugart: PKW
100 Marke: Daniels Sweetheart
101 Kennzeichen: LL-Caro1
102 14.2.2017     45 km
103 14.2.2018     50 km
104
105 Fahrzeugart: PKW
106 Marke: Fiat Multipla
107 Kennzeichen: RO-666H
108
109 Fahrzeugart: LKW
110 Marke: Man
111 Kennzeichen: LL-Bau1
112
113 Fahrzeugart: LKW
114 Marke: Volvo
115 Kennzeichen: FR-Erde1
116
117 Fahrzeugart: Motorrad
118 Marke: BMW
119 Kennzeichen: IL-24TW
120
121
122 Copied via Assignment...
```

```
123 Fahrzeugart: PKW
124 Marke: Audi
125 Kennzeichen: PE-21HR
126
127 Fahrzeugart: PKW
128 Marke: Daniels Sweetheart
129 Kennzeichen: LL-Caro1
130 14.2.2017     45 km
131 14.2.2018     50 km
132
133 Fahrzeugart: PKW
134 Marke: Fiat Multipla
135 Kennzeichen: RO-666H
136
137 Fahrzeugart: LKW
138 Marke: Man
139 Kennzeichen: LL-Bau1
140
141 Fahrzeugart: LKW
142 Marke: Volvo
143 Kennzeichen: FR-Erde1
144
145 Fahrzeugart: Motorrad
146 Marke: BMW
147 Kennzeichen: IL-24TW
```

6 Source Code

6.1 Class Carpool

6.1.1 Carpool.h

```
1  /*_____
2  |Workfile:    Carpool.h
3  |Description: [HEADER] Main Class managing Carpool-program
4  |Name:       Daniel Weyrer              PKZ: S1820306044
5  |Date:       28.10.2019
6  |Remarks:    -
7  |Revision:   0
8  |_____*/
9
10 #ifndef CARPOOL_H
11 #define CARPOOL_H
12
13 #include "Car.h"
14 #include "Motorcycle.h"
15 #include "Truck.h"
16
17 #include <iostream>
18 #include <iostream>
19 #include <algorithm>
20 #include <iterator>
21 #include <exception>
22 #include <string>
23 #include <vector>
24 #include <memory>
25 #include <ostream>
26
27 typedef std::unique_ptr<Vehicle> TUptr;
28 typedef std::vector<TUptr> TVehicles;
29 typedef TVehicles::iterator VehicleItor;
30 typedef TVehicles::const_iterator VehicleCItor;
31
32
33 class Carpool{
34 public:
35   //CTor & DTor
36   Carpool();
37   ~Carpool();
38   //Copy-CTor
39   Carpool(Carpool const& toCopy);
40
41   //Addfunctions to add derived classes to container
42   void AddCar(std::string const& brand, std::string const& numberplate, Fuel fuel);
43   void AddTruck(std::string const& brand, std::string const& numberplate, Fuel fuel);
44   void AddMotorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel);
45
46   //Removes Vehicle with the given numberplate
47   void Remove(std::string const& numberplate);
48
49   //Adds one entry in the logbook of the vehicle with the given numberplate
50   void AddLogbookEntry(std::string const& numberplate, int const& day, int const& month, int const&
        year, int const distance);
51
52   //Deletes current last entry in the logbook of the vehicle with the given numberplate and replaces
        it with the new val given
53   void ChangeLastLogbookEntry(std::string const& numberplate, int const& day, int const& month, int
        const& year, int const distance);
54
55   //Searches the vehicle with the given numberplate and Prints its data (if found)
56   //or throws an exception when there is no such numberplate in the database
57   void SearchByNumberplate(std::string const& numberplate);
58
59   //Adds tge total mileages of every single vehicle in the current pool
60   unsigned long TotalMileage() const;
61
62   //Overloaded assignment- and outputoperator
```

```
63    Carpool& operator =(Carpool const& toCopy);
64
65    friend std::ostream& operator<<(std::ostream& ost, Carpool const& c);
66
67  private:
68    //Stores all vehicles
69    TVehicles mVehicles;
70
71    //Helper functions
72    void Add(TUptr v);
73    VehicleItor FindVehicle(std::string const& numberplate);
74
75  };
76
77  #endif //CARPOOL_H
```

6.1.2 Carpool.cpp

```
1  /*_____
2  |Workfile:    Carpool.cpp
3  |Description: Manages main functionality of the Carpool
4  |Name:        Weyrer Daniel              PKZ: S1820306044
5  |Date:        28.10.2019
6  |Remarks:     -
7  |Revision:    0
8  |_____*/
9  #include "Carpool.h"
10
11  using namespace std;
12
13  Carpool::Carpool() {
14  }
15
16  Carpool::~Carpool() {
17  }
18
19  //Deep copy already defined in the assignment operator
20  Carpool::Carpool(Carpool const& toCopy) {
21    *this = toCopy;
22  }
23
24  //Adds car only if it isn't already contained in the Container; throws and catches an exception
25  //if the number is already contained
26  void Carpool::Add(TUptr v) {
27    try {
28      if (FindVehicle((*v).GetNumberplate()) != this->mVehicles.cend()) {
29        throw exception("Add failed: Number already in the Database!");
30      }
31
32      //unique pointers can only be moved, not copied!
33      mVehicles.emplace_back(move(v));
34    }
35    catch (exception const& ex) {
36      cerr << ex.what() << endl;
37    }
38    catch (bad_alloc const& ex) {
39      cerr << "memory allocation: " << ex.what() << endl;
40    }
41  }
42
43  void Carpool::AddCar(std::string const& brand, std::string const& numberplate, Fuel fuel) {
44    Car tmp{ brand, numberplate, fuel };
45    Add(make_unique<Car>(tmp));
46  }
47
48  void Carpool::AddTruck(std::string const& brand, std::string const& numberplate, Fuel fuel) {
49    Truck tmp{ brand, numberplate, fuel };
50    Add(make_unique<Truck>(tmp));
51  }
52
53  void Carpool::AddMotorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel) {
54    Motorcycle tmp{ brand, numberplate, fuel };
```

```cpp
55    Add(make_unique<Motorcycle>(tmp));
56  }
57
58  //uses private helper function "FindVehicle" to find the car which needs to be deleted
59  //throws and catches an exception if there is no such numberplate registered in the database
60  void Carpool::Remove(std::string const& numberplate) {
61    try {
62      VehicleCItor vehicleToDel = FindVehicle(numberplate);
63      if (vehicleToDel == mVehicles.cend()) {
64        throw exception("Delete failed: The entered numberplate is not registered in this carpool!");
65      }
66      else {
67        mVehicles.erase(vehicleToDel);
68      }
69    }
70    catch (exception const& ex) {
71      cerr << ex.what() << endl;
72    }
73    catch (bad_alloc const& ex) {
74      cerr << "memory allocation: " << ex.what() << endl;
75    }
76  }
77
78
79  void Carpool::AddLogbookEntry(std::string const& numberplate, int const& day,
80                  int const& month, int const& year, int const distance) {
81    VehicleItor tmp = FindVehicle(numberplate);
82    (**tmp).AddNewLogbookEntry(day, month, year, distance);
83  }
84
85  void Carpool::ChangeLastLogbookEntry(std::string const& numberplate, int const& day,
86                    int const& month, int const& year, int const distance) {
87    VehicleItor tmp = FindVehicle(numberplate);
88    (**tmp).ChangeLastLogbookEntry(day, month, year, distance);
89  }
90
91  void Carpool::SearchByNumberplate(std::string const& numberplate) {
92    VehicleCItor foundVehicle = FindVehicle(numberplate);
93    if (foundVehicle == mVehicles.end()) {
94      cerr << "The vehicle with the numberplate :" << numberplate << " is not registered in this
95          carpool! " << endl;
96    }
97    else {
98      (**foundVehicle).Print(cout);
99    }
100 }
101
102 //Private Helperfunction based on STL-Find_if and a UnaryPred
103 //Returns Iterator to given numberplate (if found)
104 //Returns Iterator == cont.end() if not found
105 VehicleItor Carpool::FindVehicle(std::string const& numberplate)  {
106   auto PredNumberP = [numberplate](unique_ptr<Vehicle> const&  v) {
107     return (numberplate == (*v).GetNumberplate());
108   };
109
110   return find_if(mVehicles.begin(), mVehicles.end(), PredNumberP);
111 }
112
113 unsigned long Carpool::TotalMileage() const {
114   unsigned long tmpMileage = 0;
115   for (VehicleCItor i = mVehicles.cbegin(); i != mVehicles.cend(); i++ ) {
116     tmpMileage += (**i).GetMileage();
117   }
118   return tmpMileage;
119 }
120
121 //Iterates through the container and performs a deep copy
122 Carpool& Carpool::operator=(Carpool const& toCopy) {
123   if (&toCopy != this)  {
124     for (VehicleCItor i = toCopy.mVehicles.cbegin(); i < toCopy.mVehicles.cend(); i++) {
125       Add((**i).Clone());
126     }
127   }
```

```
127    return *this;
128 }
129
130 //Overloaded Output-Operator
131 ostream& operator<<(ostream& ost, Carpool const& c) {
132    if (ost.good()) {
133       auto LPrint = [&ost](TUptr const& x) { (*x).Print(ost); ost << endl; };
134       for_each(c.mVehicles.cbegin(), c.mVehicles.cend(), LPrint);
135    }
136    return ost;
137 }
```

6.2 Class Vehicle

6.2.1 Vehicle.h

```
 1 /*----------------------------------------------------------------------
 2 |Workfile:    Vehicles.h
 3 |Description: [HEADER] Base Class for different Vehicletypes
 4 |Name:        Viktoria Streibl        PKZ: S1810306013
 5 |Date:        28.10.2019
 6 |Remarks:    -
 7 |Revision:    0
 8 |----------------------------------------------------------------------*/
 9
10 #ifndef VEHICLES
11 #define VEHICLES
12
13 #include "Logbook.h"
14 #include <string>
15 #include <list>
16 #include <time.h>
17
18 //fuel types of vehicle
19 enum class Fuel { Petrol, Diesel, Gas, Electricity };
20
21 class Vehicle
22 {
23 public:
24    //general print method
25    virtual std::ostream& Print(std::ostream& ost) = 0;
26    //general clone method
27    virtual std::unique_ptr<Vehicle> Clone() = 0;
28
29    //constructor
30    Vehicle();
31    //deconstructor
32    virtual ~Vehicle();
33
34    //return numberplate of vehicle
35    std::string GetNumberplate();
36    //return driven kilometers of vehicle
37    unsigned long GetMileage() const;
38
39    //set brand of vehicle
40    void SetBrand(std::string brand);
41    //set numerplate of vehicle
42    void SetNumberplate(std::string numberplate);
43    //set fuel type of vehicle
44    void SetFuel(Fuel fuel);
45
46    //add a new logbook entry
47    void AddNewLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int const&
          distance);
48
49    //change the last logbook entry
50    void ChangeLastLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int const&
          distance);
51
52    Vehicle& operator=(Vehicle const& toCopy);
53
54 protected:
```

```
55    std::string m_brand;
56    std::string m_numberplate;
57    Fuel m_fuel;
58
59    //print all vehicles
60    std::ostream& PrintList(std::ostream& ost);
61
62  private:
63    Logbook m_logbook;
64    int const m_monthPerYear = 12;
65    int const m_daysPerMonth = 31;
66
67    int m_currentDay = 0;
68    int m_currentMonth = 0;
69    int m_currentYear = 0;
70
71    //create the struct Date by the given variables
72    Logbook::Date CreateDate(size_t const& day, size_t const& month, size_t const& year);
73    //get the current date
74    void setCurrentDate();
75  };
76
77  #endif //VEHICLES
```

6.2.2 Vehicle.cpp

```cpp
/*----------------------------------------------------------------------
|Workfile:    Vehicles.cpp
|Description: Base Class for different Vehicletypes
|Name:        Viktoria Streibl            PKZ: S1810306013
|Date:        28.10.2019
|Remarks:    -
|Revision:    0
|----------------------------------------------------------------------*/

#include "Vehicle.h"
using namespace std;

Vehicle::Vehicle() {
  //set default values
  m_brand = "";
  m_numberplate = "";
  m_fuel = Fuel::Petrol;

  setCurrentDate();
}

Vehicle::~Vehicle() {};

//copy
Vehicle& Vehicle::operator=(Vehicle const& toCopy) {
  if (&toCopy != this)  {
    m_logbook = toCopy.m_logbook;
  }
  return *this;
}

string Vehicle::GetNumberplate() {
  return m_numberplate;
}

void Vehicle::SetBrand(string brand) {
  m_brand = brand;
}

void Vehicle::SetNumberplate(string numberplate) {
  m_numberplate = numberplate;
}

void Vehicle::SetFuel(Fuel fuel) {
  m_fuel = fuel;
}

ostream& Vehicle::PrintList(ostream& ost) {
  //print all entries of logbook
  m_logbook.PrintLogEntries(ost);
  return ost;
}

unsigned long Vehicle::GetMileage() const{
  //return total driven kilometers by vehicle
  return m_logbook.GetTotalDistance();
}

void Vehicle::AddNewLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int
    const& distance) {
  //create the struct Date by the given data
  Logbook::Date date = CreateDate(day, month, year);
  //add new entry, set date and distance
  m_logbook.AddNewEntry(date, distance);
}

void Vehicle::ChangeLastLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int
    const& distance) {
  //create the struct Date by the given data
  Logbook::Date date = CreateDate(day, month, year);
```

```cpp
69    //renew entry, set new date and new distance
70    m_logbook.ChangeLastEntry(date, distance);
71 }
72
73 Logbook::Date Vehicle::CreateDate(size_t const& day, size_t const& month, size_t const& year) {
74    Logbook::Date date;
75    date.day = 0;
76    date.month = 0;
77    date.year = 0;
78
79    try {
80      //check if day is between 1 and 31
81      if (day <= 0 && day > m_daysPerMonth) {
82        throw exception("Wrong Date: Your day is not valid");
83      }
84      //check if month is between 1 and 12
85      if (month <= 0 && month > m_monthPerYear) {
86        throw exception("Wrong Date: Your month is not valid");
87      }
88      //check if year is not negativ
89      if (year < 0) {
90        throw exception("Wrong Date: Your year is not valid");
91      }
92
93      //check if the new date is not in the future
94      if (year == m_currentYear) {
95        if (month > m_currentMonth) {
96          throw exception("Wrong Date: Back to the Future?");
97        }
98        else if (month == m_currentMonth) {
99          if (day > m_currentDay) {
100            throw exception("Wrong Date: Back to the Future?");
101          }
102        }
103      }
104      else if (year > m_currentYear) {
105        throw exception("Wrong Date: Back to the Future?");
106      }
107
108      date.day = day;
109      date.month = month;
110      date.year = year;
111    }
112    catch(exception const& ex){
113      cerr << ex.what() << endl;
114    }
115
116    return date;
117 }
118
119 void Vehicle::setCurrentDate() {
120    time_t now = time(0);
121    tm ltm;
122    localtime_s(&ltm, &now);
123
124    //calculate the current year
125    m_currentYear = ltm.tm_year + 1900;
126    //calculate the current month
127    m_currentMonth = 1 + ltm.tm_mon;
128    //gets the current day
129    m_currentDay = ltm.tm_mday;
130 }
```

6.3 Class Logbook

6.3.1 Logbook.h

```
/*_____
|Workfile:    Logbook.h
|Description: [HEADER] Class for collection driver data
|Name:        Viktoria Streibl          PKZ: S1810306013
|Date:        28.10.2019
|Remarks:     -
|Revision:    0
|_____*/

#ifndef LOGBOOK
#define LOGBOOK

#include <string>
#include <stdio.h>
#include <list>
#include <iostream>

class Logbook
{
public:
  struct Date {
    size_t day;
    size_t month;
    size_t year;
  };

  //add new logbook entry to list
  void AddNewEntry(Date const& date, int const& distance);

  //print all logbook entries
  std::ostream& PrintLogEntries(std::ostream& ost);

  //calculate all driven kilometers
  unsigned long GetTotalDistance() const;

  //change the last logbook entry in the list
  void ChangeLastEntry(Date const& date, int const& distance);

private:
  typedef std::pair<Date, int> Entry;
  std::list<Entry> m_entries;
};

#endif //LOGBOOK
```

6.3.2 Logbook.cpp

```cpp
/*-----------------------------------------------------------------------
|Workfile:    Logbook.cpp
|Description: Class for collection driver data
|Name:        Viktoria Streibl          PKZ: S1810306013
|Date:        28.10.2019
|Remarks:     -
|Revision:    0
|_____*/
#include "Logbook.h"
using namespace std;

void Logbook::AddNewEntry(Date const& date, int const& distance) {
  //check if date is valid
  if (date.day == 0 || date.month == 0 || date.year == 0) {
    return;
  }

  //check if there is already an entry in the list
  if (m_entries.empty()) {
    //add new entry to list
    m_entries.push_back(make_pair(date, distance));
  }else{
    //get last entry from the list
    Entry lastEntry = m_entries.back();

    try {
      //check if the new date is equal to the last entry's date
      if (lastEntry.first.day == date.day &&
        lastEntry.first.month == date.month &&
        lastEntry.first.year == date.year)
      {
        //delete last entry
        m_entries.pop_back();
        //add new entry + old entry's kilometer
        m_entries.push_back(make_pair(date, distance + lastEntry.second));
      }

      //check if the new entry is older than the last one
      if (date.year == lastEntry.first.year) {
        if (date.month < lastEntry.first.month) {
          throw exception("Wrong Date: Please input a date after the last entry");
        }
        else if (date.month == lastEntry.first.month) {
          if (date.day < lastEntry.first.day) {
            throw exception("Wrong Date: Please input a date after the last entry");
          }
        }
      }else if (date.year < lastEntry.first.year) {
        throw exception("Wrong Date: Please input a date after the last entry");
      }
      else {
        //add new entry to list
        m_entries.push_back(make_pair(date, distance));
      }
    }
    catch (exception const& ex) {
      cerr << ex.what() << endl;
    }
  }
}

ostream& Logbook::PrintLogEntries(ostream& ost) {
  //check if ostream is valid
  if (ost.good()) {
    //print all log entries
    for (auto e : m_entries) {
      ost << e.first.day << ".";
      ost << e.first.month << ".";
      ost << e.first.year << "     ";
      ost << e.second << " km";
```

```
71        ost << endl;
72      }
73    }
74    return ost;
75 }
76
77 unsigned long Logbook::GetTotalDistance() const {
78    unsigned long sum = 0;
79    //adds up all driven kilometers
80    for (auto e : m_entries) {
81      sum += e.second;
82    }
83    return sum;
84 }
85
86 void Logbook::ChangeLastEntry(Date const& date, int const& distance) {
87    //remove last entry
88    m_entries.pop_back();
89    //add new entry
90    m_entries.push_back(make_pair(date, distance));
91 }
```

6.4 Class Motorcycle

6.4.1 Motorcycle.h

```
 1 /*_____
 2 |Workfile:    Motorcycle.h
 3 |Description: [HEADER] Class for a Vehicle of type Motorcycle
 4 |Name:        Viktoria Streibl         PKZ: S1810306013
 5 |Date:        28.10.2019
 6 |Remarks:     -
 7 |Revision:    0
 8 |_____*/
 9
10 #ifndef MOTORCYCLE
11 #define MOTORCYCLE
12
13 #include "Vehicle.h"
14 class Motorcycle : public Vehicle
15 {
16 public:
17    //constructor
18    Motorcycle();
19    //create a new motorcycle
20    Motorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel);
21    //deconstrcutor
22    virtual ~Motorcycle();
23    //Prints the brand, type, numberplate and logbook dat
24    virtual std::ostream& Print(std::ostream& ost) override;
25    //clone motorcycle
26    virtual std::unique_ptr<Vehicle> Clone() override;
27 };
28
29 #endif //MOTORCYCLE
```

6.4.2 Motorcycle.cpp

```cpp
/*-----------------------------------------------------------------------
|Workfile:    Motorcycle.cpp
|Description: Class for a Vehicle of type Motorcycle
|Name:        Viktoria Streibl          PKZ: S1810306013
|Date:        28.10.2019
|Remarks:     -
|Revision:    0
|-----------------------------------------------------------------------*/

#include "Motorcycle.h"
#include <iostream>

using namespace std;

Motorcycle::Motorcycle() {
}
Motorcycle::Motorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel){
  //set brand, numberplate and fuel
  m_brand = brand;
  m_numberplate = numberplate;
  m_fuel = fuel;
}

Motorcycle::~Motorcycle() {};

ostream& Motorcycle::Print(ostream& ost) {
  //chek if ostream is valid
  if (ost.good()) {
    ost << "Fahrzeugart: Motorrad" << endl;
    ost << "Marke: " << m_brand << endl;
    ost << "Kennzeichen: " << m_numberplate << endl;
    //print all logbook entries
    Vehicle::PrintList(ost);
  }
  return ost;
}

std::unique_ptr<Vehicle> Motorcycle::Clone() {
  return make_unique<Motorcycle>(*this);
}
```

6.5 Class Car

6.5.1 Car.h

```cpp
/*_____
|Workfile:    Car.h
|Description: [HEADER] Class for a Vehicle of type Car
|Name:        Viktoria Streibl          PKZ: S1810306013
|Date:        28.10.2019
|Remarks:     -
|Revision:    0
|_____*/

#ifndef CAR
#define CAR

#include "Vehicle.h"

class Car : public Vehicle
{
public:
  //constructor
  Car();
  //create a new car
  Car(std::string const& brand, std::string const& numberplate, Fuel fuel);
  //deconstrcutor
  virtual ~Car();
  //Prints the brand, type, numberplate and logbook data
  virtual std::ostream& Print(std::ostream& ost) override;
  //clone car
  virtual std::unique_ptr<Vehicle> Clone() override;
};
#endif // CAR
```

6.5.2 Car.cpp

```cpp
/*_____
|Workfile:    Car.cpp
|Description: Class for a Vehicle of type Car
|Name:        Viktoria Streibl          PKZ: S1810306013
|Date:        28.10.2019
|Remarks:     -
|Revision:    0
|_____*/

#include "Car.h"
#include <iostream>

using namespace std;

Car::Car() {
}

Car::Car(std::string const& brand, std::string const& numberplate, Fuel fuel) {
  //set brand, numberplate and fuel
  m_brand = brand;
  m_numberplate = numberplate;
  m_fuel = fuel;
}

Car::~Car() {};

ostream& Car::Print(ostream& ost) {
  //chek if ostream is valid
  if (ost.good()) {
    ost << "Fahrzeugart: PKW" << endl;
    ost << "Marke: " << m_brand << endl;
    ost << "Kennzeichen: " << m_numberplate << endl;
    //print all logbook entries
    Vehicle::PrintList(ost);
  }

```

```
37    return ost;
38 }
39
40 std::unique_ptr<Vehicle> Car::Clone() {
41    return make_unique<Car>(*this);
42 }
```

6.6 Class Truck

6.6.1 Truck.h

```
 1 /*_____
 2 |Workfile:    Truck.h
 3 |Description: [HEADER] Class for a Vehicle of type Truck
 4 |Name:        Viktoria Streibl          PKZ: S1810306013
 5 |Date:        28.10.2019
 6 |Remarks:     -
 7 |Revision:    0
 8 |_____*/
 9
10 #ifndef TRUCK
11 #define TRUCK
12 #include "Vehicle.h"
13
14 class Truck : public Vehicle
15 {
16 public:
17   //constructor
18   Truck();
19   //create a new truck
20   Truck(std::string const& brand, std::string const& numberplate, Fuel fuel);
21   //deconstrcutor
22   virtual ~Truck();
23   //Prints the brand, type, numberplate and logbook dat
24   virtual std::ostream& Print(std::ostream& ost) override;
25   //clone truck
26   virtual std::unique_ptr<Vehicle> Clone() override;
27 };
28
29 #endif // TRUCK
```

6.6.2 Truck.cpp

```
 1 /*_____
 2 |Workfile:    Truck.cpp
 3 |Description: Class for a Vehicle of type Truck
 4 |Name:        Viktoria Streibl          PKZ: S1810306013
 5 |Date:        28.10.2019
 6 |Remarks:     -
 7 |Revision:    0
 8 |_____*/
 9
10 #include "Truck.h"
11 #include <iostream>
12
13 using namespace std;
14
15 Truck::Truck() {
16 }
17
18 Truck::Truck(std::string const& brand, std::string const& numberplate, Fuel fuel) {
19   //set brand, numberplate and fuel
20   m_brand = brand;
21   m_numberplate = numberplate;
22   m_fuel = fuel;
23 }
24
25 Truck::~Truck(){}
26
27 ostream& Truck::Print(ostream& ost) {
28   //chek if ostream is valid
```

```
29    if (ost.good()) {
30      ost << "Fahrzeugart: LKW" << endl;
31      ost << "Marke: " << m_brand << endl;
32      ost << "Kennzeichen: " << m_numberplate << endl;
33      //print all logbook entries
34      Vehicle::PrintList(ost);
35
36    }
37    return ost;
38 }
39
40 std::unique_ptr<Vehicle> Truck::Clone() {
41     return make_unique<Truck>(*this);
42 }
```

6.6.3 TestDriver.cpp

```
1 /*_____
2 |Workfile:    TestDriver.cpp
3 |Description: File for testing the code
4 |Name:        Viktoria Streibl         PKZ: S1810306013
5        Daniel Weyrer            PKZ: S1820306044
6 |Date:        28.10.2019
7 |Remarks:     -
8 |Revision:    0
9 |_____*/
10 #include "Carpool.h"
11
12 void SectionHeader(std::string headerTitle);
13
14 int main() {
15
16   /*
17     1. Hinzufügen von neuen Fahrzeugen.
18     2. Entfernen von bestehenden Fahrzeugen.
19     3. Suchen eines Fahrzeuges nach seinem Kennzeichen.
20     4. Ausgeben aller Fahrzeuge samt ihrer Eigenschaften und dem Fahrtenbuch auf dem Ausgabestrom
21     und in einer Datei.
22     5. Verwenden Sie im Fuhrpark zur Verwaltung aller Fahrzeuge einen entsprechenden Container!
23     6. Der Fuhrpark muss kopierbar und zweisbar sein!
24   */
25
26   //1. Add Vehicles
27   SectionHeader("1. Add Vehicles");
28   Carpool carpool;
29   //Cars
30   carpool.AddCar("Audi", "PE-21HR", Fuel::Diesel);
31   carpool.AddCar("Daniels Sweetheart", "LL-Caro1", Fuel::Gas);
32   carpool.AddCar("Fiat Multipla", "RO-666H", Fuel::Petrol);
33   carpool.AddCar("VW Golf", "PE-35CK", Fuel::Electricity);
34   std::cout << "Cars added successfully..." << std::endl;
35   //Trucks
36   carpool.AddTruck("Man", "LL-Bau1", Fuel::Diesel);
37   carpool.AddTruck("Mercedes", "LE-Bau2", Fuel::Diesel);
38   carpool.AddTruck("Volvo", "FR-Erde1", Fuel::Petrol);
39   std::cout << "Trucks added successfully..." << std::endl;
40   //Motorcycles
41   carpool.AddMotorcycle("BMW", "IL-24TW", Fuel::Petrol);
42   carpool.AddMotorcycle("Honda", "PE-VS13", Fuel::Petrol);
43   std::cout << "Motorcycles added successfully..." << std::endl;
44   std::cout << std::endl;
45
46   //2. Remove Vehicle
47   SectionHeader("2. Remove Vehicle");
48   carpool.Remove("PE-35CK");  //should be able to delete
49   std::cout << "Car removed successfully..." << std::endl;
50   carpool.Remove("LE-Bau2");  //should be able to delete
51   std::cout << "Truck removed successfully..." << std::endl;
52   carpool.Remove("PE-VS13");  //should be able to delete
53   std::cout << "Motorcycle removed successfully..." << std::endl;
54
55   carpool.Remove("PE-35CK");  //no car should be found
```

```cpp
56    std::cout << std::endl;
57
58
59    //3. Find Vehicle
60    SectionHeader("3. Find Vehicle");
61    carpool.SearchByNumberplate("RO-666H"); //should find car
62    carpool.SearchByNumberplate("RO-677H"); //no car should be found
63    std::cout << std::endl;
64
65    //4. Print vehicles
66    SectionHeader("4. Print vehicles");
67    std::cout << carpool << std::endl;
68    std::cout << std::endl;
69
70    //5. Logbook
71    SectionHeader("5. Logbook");
72    std::cout << "Add 3 Entries" << std::endl;
73    carpool.AddLogbookEntry("LL-Caro1", 14, 2, 2017, 45);   //works...
74    carpool.AddLogbookEntry("LL-Caro1", 25, 2, 2017, 46);   //works...
75    carpool.AddLogbookEntry("LL-Caro1", 14, 2, 2018, 5);    //works...
76
77    std::cout << "Add Entry with same date" << std::endl;
78    carpool.AddLogbookEntry("LL-Caro1", 14, 2, 2018, 45);   //should be combined with the last entry
79    carpool.AddLogbookEntry("LL-Caro1", 23, 12, 2018, 45);   //works...
80
81    std::cout << "Add old Entry" << std::endl;
82    carpool.AddLogbookEntry("LL-Caro1", 14, 12, 2017, 45);    //dont work - to old
83
84    std::cout << "Add Entry for the future" << std::endl;
85    carpool.AddLogbookEntry("LL-Caro1", 14, 12, 2019, 45);    //dont work - to old
86    std::cout << std::endl;
87
88    //6. Carpool copy & co
89    SectionHeader("6. Carpool copy & co");
90    Carpool copied{ carpool };
91    Carpool copied2;
92    copied2 = copied;
93
94    std::cout << "Original..." << std::endl;
95    std::cout << carpool << std::endl;
96    std::cout << "Copied via Copy-CTor..." << std::endl;
97    std::cout << copied << std::endl;
98    std::cout << "Copied via Assignment..." << std::endl;
99    std::cout << copied2 << std::endl;
100
101    return 0;
102 }
103
104 void SectionHeader(std::string headerTitle) {
105    std::cout << "################################" << std::endl;
106    std::cout << "## " << headerTitle << std::endl;
107    std::cout << "################################" << std::endl;
108 }
```