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
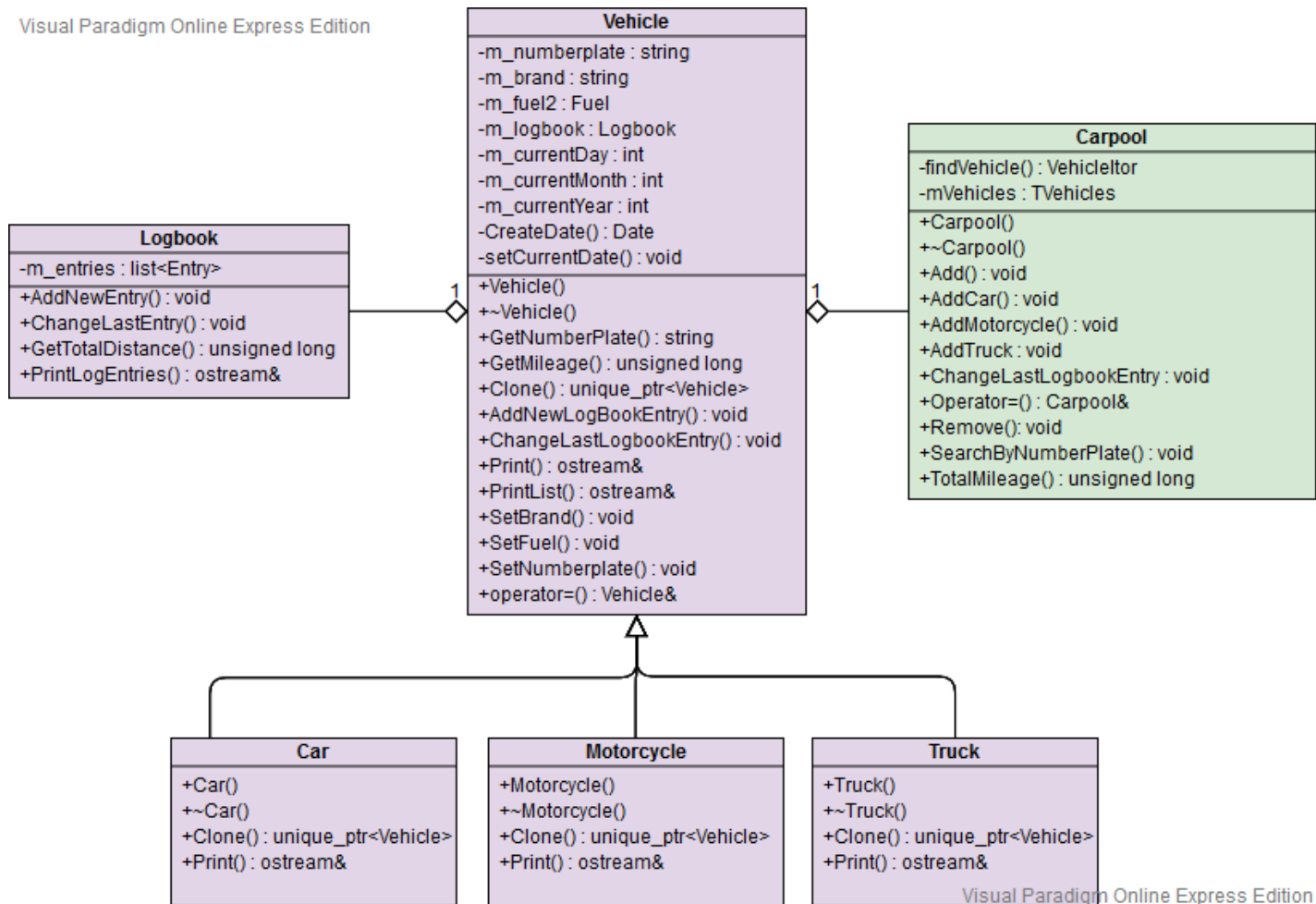
1.3.2 Daniel Weyrer

- estimated: 6ph

- 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

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&

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 searchs vehicles by numerplate and print all of them. It tests all logbook functions and the carpool.

5 Test Protocol

Im Testprotokoll werden die Testdaten und die Testergebnisse für alle Testfälle beschrieben. Weiters muss die Testumgebung angeführt sein (welches Testframework wurde verwendet, mit welchen Komponentenversionen wurde gestestet, welche Stubs wurden verwendet, etc). Wenn die Komponenten und Subsysteme getrennt getestet wurden, ist die Testprotokollierung für die Komponenten getrennt anzugeben. Weiters sind identifizierte Schwachstellen und Probleme festzuhalten.

```
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
```

```
#################################
## 4. Print vehicles
#################################
Fahrzeugart: PKW
Marke: Audi
Kennzeichen: PE-21HR

Fahrzeugart: PKW
Marke: Daniels Sweetheart
Kennzeichen: LL-Caro1

Fahrzeugart: PKW
Marke: Fiat Multipla
Kennzeichen: RO-666H

Fahrzeugart: LKW
Marke: Man
Kennzeichen: LL-Bau1

Fahrzeugart: LKW
Marke: Volvo
Kennzeichen: FR-Erde1

Fahrzeugart: Motorrad
Marke: BMW
Kennzeichen: IL-24TW


#################################
## 5. Logbook
#################################
Add 3 Entries
Add Entry with same date
Add old Entry
Wrong Date: Please input a date after the last entry
Add Entry for the future
Wrong Date: Back to the Future?

#################################
## 6. Carpool copy & co
#################################
Original...
Fahrzeugart: PKW
Marke: Audi
Kennzeichen: PE-21HR

Fahrzeugart: PKW
Marke: Daniels Sweetheart
Kennzeichen: LL-Caro1
14.2.2017.45 km
14.2.2018.50 km

Fahrzeugart: PKW
Marke: Fiat Multipla
Kennzeichen: RO-666H

Fahrzeugart: LKW
Marke: Man
Kennzeichen: LL-Bau1

Fahrzeugart: LKW
Marke: Volvo
Kennzeichen: FR-Erde1

Fahrzeugart: Motorrad
Marke: BMW
Kennzeichen: IL-24TW


Copied via Copy-CTor...
Fahrzeugart: PKW
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 Software Quality Metrics

7 Source Code

7.1 Class Carpool

7.1.1 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
```

```cpp
   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
```

7.1.2 Carpool.cpp

```cpp
/*-----------------------------------------------------------------------
|Workfile:    Carpool.cpp
|Description: Manages main functionality of the Carpool
|Name:        Weyrer Daniel          PKZ: S1820306044
|Date:        28.10.2019
|Remarks:     -
|Revision:    0
|_____*/
#include "Carpool.h"

using namespace std;

Carpool::Carpool() {
}

Carpool::~Carpool() {
}

//Deep copy already defined in the assignment operator
Carpool::Carpool(Carpool const& toCopy) {
  *this = toCopy;
}

//Adds car only if it isn't already contained in the Container; throws and catches an exception
//if the number is already contained
void Carpool::Add(TUptr v) {
  try {
    if (FindVehicle((*v).GetNumberplate()) != this->mVehicles.cend()) {
      throw exception("Add failed: Number already in the Database!");
    }

    //unique pointers can only be moved, not copied!
    mVehicles.emplace_back(move(v));
  }
  catch (exception const& ex) {
    cerr << ex.what() << endl;
  }
  catch (bad_alloc const& ex) {
    cerr << "memory allocation: " << ex.what() << endl;
  }
}

void Carpool::AddCar(std::string const& brand, std::string const& numberplate, Fuel fuel) {
  Car tmp{ brand, numberplate, fuel };
  Add(make_unique<Car>(tmp));
}

void Carpool::AddTruck(std::string const& brand, std::string const& numberplate, Fuel fuel) {
  Truck tmp{ brand, numberplate, fuel };
  Add(make_unique<Truck>(tmp));
}

void Carpool::AddMotorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel) {
  Motorcycle tmp{ brand, numberplate, fuel };
  Add(make_unique<Motorcycle>(tmp));
}

//uses private helper function "FindVehicle" to find the car which needs to be deleted
//throws and catches an exception if there is no such numberplate registered in the database
void Carpool::Remove(std::string const& numberplate) {
  try {
    VehicleCItor vehicleToDel = FindVehicle(numberplate);
    if (vehicleToDel == mVehicles.cend()) {
      throw exception("Delete failed: The entered numberplate is not registered in this carpool!");
    }
    else {
      mVehicles.erase(vehicleToDel);
    }
  }
  catch (exception const& ex) {
```

```cpp
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
              carpool! " << endl;
95    }
96    else {
97      (**foundVehicle).Print(cout);
98    }
99 }
100
101 //Private Helperfunction based on STL-Find_if and a UnaryPred
102 //Returns Iterator to given numberplate (if found)
103 //Returns Iterator == cont.end() if not found
104 VehicleItor Carpool::FindVehicle(std::string const& numberplate)  {
105    auto PredNumberP = [numberplate](unique_ptr<Vehicle> const&  v) {
106      return (numberplate == (*v).GetNumberplate());
107    };
108
109    return find_if(mVehicles.begin(), mVehicles.end(), PredNumberP);
110 }
111
112 unsigned long Carpool::TotalMileage() const {
113    unsigned long tmpMileage = 0;
114    for (VehicleCItor i = mVehicles.cbegin(); i != mVehicles.cend(); i++ ) {
115      tmpMileage += (**i).GetMileage();
116    }
117    return tmpMileage;
118 }
119
120 //Iterates through the container and performs a deep copy
121 Carpool& Carpool::operator=(Carpool const& toCopy) {
122    if (&toCopy != this)  {
123      for (VehicleCItor i = toCopy.mVehicles.cbegin(); i < toCopy.mVehicles.cend(); i++) {
124        Add((**i).Clone());
125      }
126    }
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 }
```

7.2 Class Vehicle

7.2.1 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 enum class Fuel { Petrol, Diesel, Gas, Electricity };
19
20 class Vehicle
21 {
22 public:
23   virtual std::ostream& Print(std::ostream& ost) = 0;
24   virtual std::unique_ptr<Vehicle> Clone() = 0;
25
26   Vehicle();
27   virtual ~Vehicle();
28
29   std::string GetNumberplate();
30   unsigned long GetMileage() const;
31
32   void SetBrand(std::string brand);
33   void SetNumberplate(std::string numberplate);
34   void SetFuel(Fuel fuel);
35
36   void AddNewLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int const&
        distance);
37   void ChangeLastLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int const&
        distance);
38
39   Vehicle& operator=(Vehicle const& toCopy);
40
41 protected:
42   std::string m_brand;
43   std::string m_numberplate;
44   Fuel m_fuel;
45
46   std::ostream& PrintList(std::ostream& ost);
47
48 private:
49   Logbook m_logbook;
50   int const m_monthPerYear = 12;
51   int const m_daysPerMonth = 31;
52
53   int m_currentDay = 0;
54   int m_currentMonth = 0;
55   int m_currentYear = 0;
56
57   Logbook::Date CreateDate(size_t const& day, size_t const& month, size_t const& year);
58   void setCurrentDate();
59 };
60
61 #endif //VEHICLES
```

7.2.2 Vehicle.cpp

```
1  /*_____
2  |Workfile:      Vehicles.cpp
```

```cpp
 3  |Description: Base Class for different Vehicletypes
 4  |Name:        Viktoria Streibl          PKZ: S1810306013
 5  |Date:        28.10.2019
 6  |Remarks:    -
 7  |Revision:      0
 8  |_____*/
 9
10  #include "Vehicle.h"
11  using namespace std;
12
13  Vehicle::Vehicle() {
14    m_brand = "";
15    m_numberplate = "";
16    m_fuel = Fuel::Petrol;
17
18    setCurrentDate();
19  }
20
21  Vehicle::~Vehicle() {};
22
23
24  Vehicle& Vehicle::operator=(Vehicle const& toCopy) {
25    if (&toCopy != this)   {
26      m_logbook = toCopy.m_logbook;
27    }
28    return *this;
29  }
30
31  string Vehicle::GetNumberplate() {
32    return m_numberplate;
33  }
34
35  void Vehicle::SetBrand(string brand) {
36    m_brand = brand;
37  }
38
39  void Vehicle::SetNumberplate(string numberplate) {
40    m_numberplate = numberplate;
41  }
42
43  void Vehicle::SetFuel(Fuel fuel) {
44    m_fuel = fuel;
45  }
46
47  ostream& Vehicle::PrintList(ostream& ost) {
48    m_logbook.PrintLogEntries(ost);
49    return ost;
50  }
51
52  unsigned long Vehicle::GetMileage() const{
53    return m_logbook.GetTotalDistance();
54  }
55
56  void Vehicle::AddNewLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int
        const& distance) {
57    Logbook::Date date = CreateDate(day, month, year);
58    m_logbook.AddNewEntry(date, distance);
59  }
60
61  void Vehicle::ChangeLastLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int
        const& distance) {
62    Logbook::Date date = CreateDate(day, month, year);
63    m_logbook.ChangeLastEntry(date, distance);
64  }
65
66  Logbook::Date Vehicle::CreateDate(size_t const& day, size_t const& month, size_t const& year) {
67    Logbook::Date date;
68    date.day = 0;
69    date.month = 0;
70    date.year = 0;
71
72    try {
73      if (day <= 0 && day > m_daysPerMonth) {
```

```
74        throw exception("Wrong Date: Your day is not valid");
75      }
76      if (month <= 0 && month > m_monthPerYear) {
77        throw exception("Wrong Date: Your month is not valid");
78      }
79      if (year < 0) {
80        throw exception("Wrong Date: Your year is not valid");
81      }
82
83      if (year == m_currentYear) {
84        if (month > m_currentMonth) {
85          throw exception("Wrong Date: Back to the Future?");
86        }
87        else if (month == m_currentMonth) {
88          if (day > m_currentDay) {
89            throw exception("Wrong Date: Back to the Future?");
90          }
91        }
92      }
93      else if (year > m_currentYear) {
94        throw exception("Wrong Date: Back to the Future?");
95      }
96
97      date.day = day;
98      date.month = month;
99      date.year = year;
100   }
101   catch(exception const& ex){
102     cerr << ex.what() << endl;
103   }
104
105   return date;
106 }
107
108 void Vehicle::setCurrentDate() {
109   time_t now = time(0);
110   tm ltm;
111   localtime_s(&ltm, &now);
112
113   m_currentYear = ltm.tm_year + 1900;
114   m_currentMonth = 1 + ltm.tm_mon;
115   m_currentDay = ltm.tm_mday;
116 }
```

7.3 Class Logbook

7.3.1 Logbook.h

```
1  /*-------------------------------------------------------------------------
2  |Workfile:     Logbook.h
3  |Description: [HEADER] Class for collection driver data
4  |Name:        Viktoria Streibl          PKZ: S1810306013
5  |Date:        28.10.2019
6  |Remarks:     -
7  |Revision:    0
8  |-----------------------------------------------------------------------*/
9
10 #ifndef LOGBOOK
11 #define LOGBOOK
12
13 #include <string>
14 #include <stdio.h>
15 #include <list>
16 #include <iostream>
17
18 class Logbook
19 {
20 public:
21   struct Date {
22     size_t day;
23     size_t month;
24     size_t year;
```

```
25    };
26
27      void AddNewEntry(Date const& date, int const& distance);
28      std::ostream& PrintLogEntries(std::ostream& ost);
29      unsigned long GetTotalDistance() const;
30      void ChangeLastEntry(Date const& date, int const& distance);
31
32  private:
33      typedef std::pair<Date, int> Entry;
34      std::list<Entry> m_entries;
35    };
36
37  #endif //LOGBOOK
```

7.3.2 Logbook.cpp

```
 1  /*-----------------------------------------------------------------------
 2  |Workfile:    Logbook.cpp
 3  |Description: Class for collection driver data
 4  |Name:       Viktoria Streibl          PKZ: S1810306013
 5  |Date:        28.10.2019
 6  |Remarks:    -
 7  |Revision:    0
 8  |-----------------------------------------------------------------------*/
 9  #include "Logbook.h"
10  using namespace std;
11
12  void Logbook::AddNewEntry(Date const& date, int const& distance) {
13    if (date.day == 0 || date.month == 0 || date.year == 0) {
14      return;
15    }
16
17    if (m_entries.empty()) {
18      m_entries.push_back(make_pair(date, distance));
19    }else{
20      Entry lastEntry = m_entries.back();
21
22      try {
23        if (lastEntry.first.day == date.day &&
24          lastEntry.first.month == date.month &&
25          lastEntry.first.year == date.year)
26        {
27          m_entries.pop_back();
28          m_entries.push_back(make_pair(date, distance + lastEntry.second));
29        }
30
31        if (date.year == lastEntry.first.year) {
32          if (date.month < lastEntry.first.month) {
33            throw exception("Wrong Date: Please input a date after the last entry");
34          }
35          else if (date.month == lastEntry.first.month) {
36            if (date.day < lastEntry.first.day) {
37              throw exception("Wrong Date: Please input a date after the last entry");
38            }
39          }
40        }else if (date.year < lastEntry.first.year) {
41          throw exception("Wrong Date: Please input a date after the last entry");
42        }
43        else {
44          m_entries.push_back(make_pair(date, distance));
45        }
46      }
47      catch (exception const& ex) {
48        cerr << ex.what() << endl;
49      }
50    }
51  }
52
53  ostream& Logbook::PrintLogEntries(ostream& ost) {
54    if (ost.good()) {
55      for (auto e : m_entries) {
56        ost << e.first.day << ".";
```

```
57        ost << e.first.month << ".";
58        ost << e.first.year << ".";
59        ost << e.second << " km";
60        ost << endl;
61    }
62  }
63  return ost;
64 }
65
66 unsigned long Logbook::GetTotalDistance() const {
67   unsigned long sum = 0;
68   for (auto e : m_entries) {
69     sum += e.second;
70   }
71   return sum;
72 }
73
74 void Logbook::ChangeLastEntry(Date const& date, int const& distance) {
75   m_entries.pop_back();
76   m_entries.push_back(make_pair(date, distance));
77 }
```

7.4 Class Motorcycle

7.4.1 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   Motorcycle();
18   Motorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel);
19   virtual ~Motorcycle();
20   virtual std::ostream& Print(std::ostream& ost) override;
21   virtual std::unique_ptr<Vehicle> Clone() override;
22 };
23
24 #endif //MOTORCYCLE
```

7.4.2 Motorcycle.cpp

```
1 /*_____
2 |Workfile:    Motorcycle.cpp
3 |Description: 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 #include "Motorcycle.h"
11 #include <iostream>
12
13 using namespace std;
14
15 Motorcycle::Motorcycle() {
16 }
17 Motorcycle::Motorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel){
18   m_brand = brand;
```

```
19    m_numberplate = numberplate;
20    m_fuel = fuel;
21 }
22
23 Motorcycle::~Motorcycle() {};
24
25 ostream& Motorcycle::Print(ostream& ost) {
26    if (ost.good()) {
27      ost << "Fahrzeugart: Motorrad" << endl;
28      ost << "Marke: " << m_brand << endl;
29      ost << "Kennzeichen: " << m_numberplate << endl;
30    }
31    Vehicle::PrintList(ost);
32    return ost;
33 }
34
35 std::unique_ptr<Vehicle> Motorcycle::Clone() {
36    return make_unique<Motorcycle>(*this);
37 }
```

7.5 Class Car

7.5.1 Car.h

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

7.5.2 Car.cpp

```
 1 /*-------------------------------------------------------------------
 2 |Workfile:    Car.cpp
 3 |Description: Class for a Vehicle of type Car
 4 |Name:        Viktoria Streibl          PKZ: S1810306013
 5 |Date:        28.10.2019
 6 |Remarks:    -
 7 |Revision:    0
 8 |_____*/
 9
10 #include "Car.h"
11 #include <iostream>
12
13 using namespace std;
14
15 Car::Car() {
16 }
17
18 Car::Car(std::string const& brand, std::string const& numberplate, Fuel fuel) {
19    m_brand = brand;
20    m_numberplate = numberplate;
```

```
21    m_fuel = fuel;
22 }
23
24 Car::~Car() {};
25
26 ostream& Car::Print(ostream& ost) {
27    if (ost.good()) {
28      ost << "Fahrzeugart: PKW" << endl;
29      ost << "Marke: " << m_brand << endl;
30      ost << "Kennzeichen: " << m_numberplate << endl;
31      Vehicle::PrintList(ost);
32    }
33
34    return ost;
35 }
36
37 std::unique_ptr<Vehicle> Car::Clone() {
38    return make_unique<Car>(*this);
39 }
```

7.6 Class Truck

7.6.1 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    Truck();
18    Truck(std::string const& brand, std::string const& numberplate, Fuel fuel);
19    virtual ~Truck();
20    virtual std::ostream& Print(std::ostream& ost) override;
21    virtual std::unique_ptr<Vehicle> Clone() override;
22 };
23
24 #endif // TRUCK
```

7.6.2 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    m_brand = brand;
20    m_numberplate = numberplate;
```

```cpp
21    m_fuel = fuel;
22 }
23
24 Truck::~Truck(){}
25
26 ostream& Truck::Print(ostream& ost) {
27   if (ost.good()) {
28     ost << "Fahrzeugart: LKW" << endl;
29     ost << "Marke: " << m_brand << endl;
30     ost << "Kennzeichen: " << m_numberplate << endl;
31     Vehicle::PrintList(ost);
32
33   }
34   return ost;
35 }
36
37 std::unique_ptr<Vehicle> Truck::Clone() {
38     return make_unique<Truck>(*this);
39 }
```