FH-OÖ Hagenberg/HSD
SDP3, WS 2019
Übung 1

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

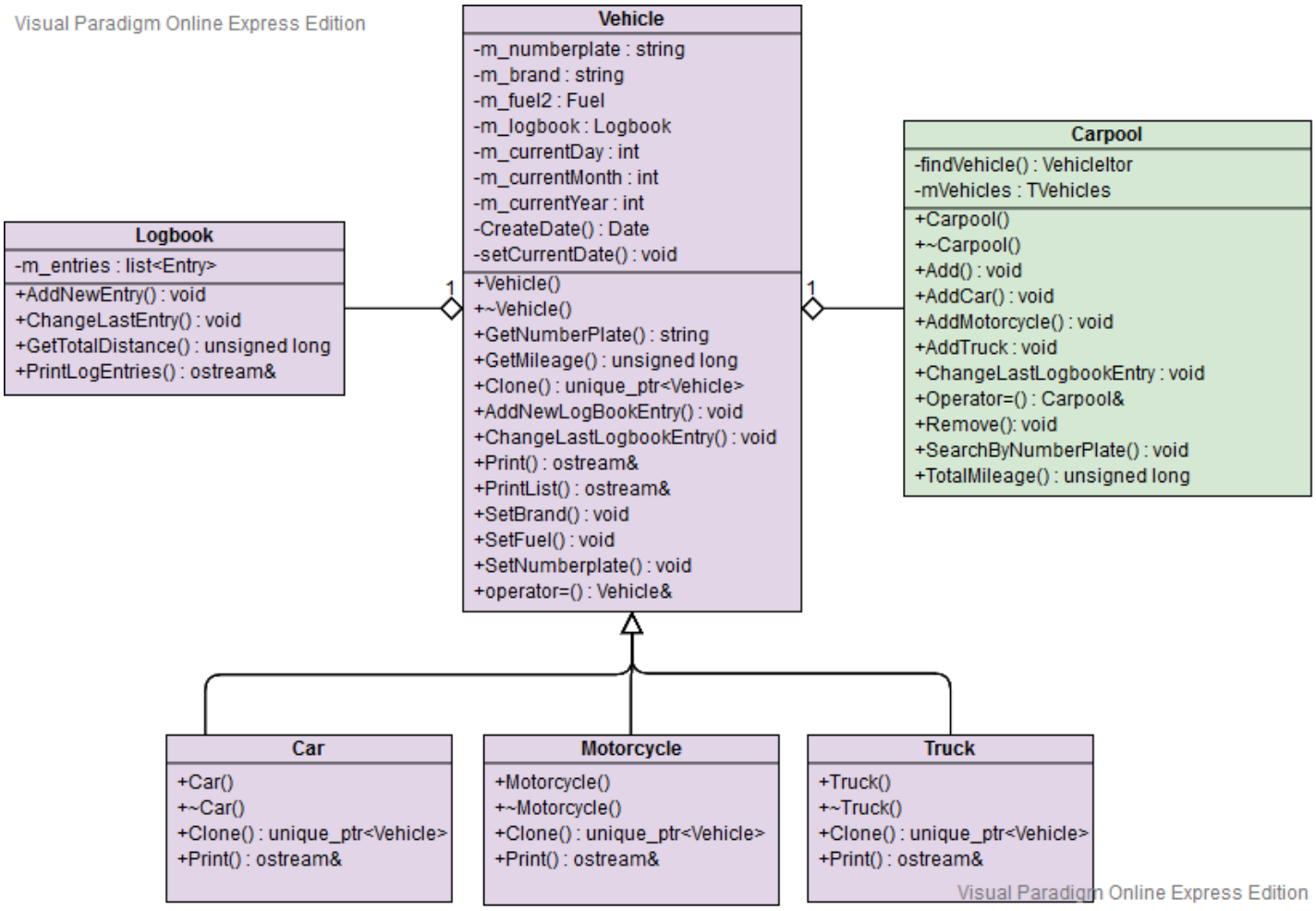
- estimated: 6ph
- actually: 8 ph

2 Requirenment 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



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

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 ##################################
```

```
Car removed successfully...
Truck removed successfully...
Motorcycle removed successfully...
Delete failed: The entered numberplate is not registered in this carpool!

###################################
## 3. Find Vehicle
###################################
Fahrzeugart: PKW
Marke: Fiat Multipla
Kennzeichen: RO-666H
The vehicle with the numberplate :RO-677H is not registered in this carpool!

###################################
## 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
```

```
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

```
/*-------------------------------------------------------------------
|Workfile:    Carpool.h
|Description: [HEADER] Main Class managing Carpool-program
|Name:       Daniel Weyrer             PKZ: S1820306044
|Date:       28.10.2019
|Remarks:    -
|Revision:    0
|-------------------------------------------------------------------*/

#ifndef CARPOOL_H
#define CARPOOL_H

#include "Car.h"
#include "Motorcycle.h"
#include "Truck.h"

#include <iostream>
#include <iostream>
#include <algorithm>
#include <iterator>
#include <exception>
#include <string>
#include <vector>
#include <memory>
#include <ostream>

typedef std::unique_ptr<Vehicle> TUptr;
typedef std::vector<TUptr> TVehicles;
typedef TVehicles::iterator VehicleItor;
typedef TVehicles::const_iterator VehicleCItor;


class Carpool{
public:
  //CTor & DTor
  Carpool();
  ~Carpool();
  //Copy-CTor
  Carpool(Carpool const& toCopy);

  //Addfunctions to add derived classes to container
  void AddCar(std::string const& brand, std::string const& numberplate, Fuel fuel);
  void AddTruck(std::string const& brand, std::string const& numberplate, Fuel fuel);
  void AddMotorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel);

  //Removes Vehicle with the given numberplate
  void Remove(std::string const& numberplate);

  //Adds one entry in the logbook of the vehicle with the given numberplate
  void AddLogbookEntry(std::string const& numberplate, int const& day, int const& month, int const&
      year, int const distance);

  //Deletes current last entry in the logbook of the vehicle with the given numberplate and replaces
       it with the new val given
  void ChangeLastLogbookEntry(std::string const& numberplate, int const& day, int const& month, int
      const& year, int const distance);

  //Searches the vehicle with the given numberplate and Prints its data (if found)
  //or throws an exception when there is no such numberplate in the database
  void SearchByNumberplate(std::string const& numberplate);

  //Adds tge total mileages of every single vehicle in the current pool
  unsigned long TotalMileage() const;

  //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 };
```

```
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
           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   }
```

```
  return *this;
}

//Overloaded Output-Operator
ostream& operator<<(ostream& ost, Carpool const& c) {
  if (ost.good()) {
    auto LPrint = [&ost](TUptr const& x) { (*x).Print(ost); ost << endl; };
    for_each(c.mVehicles.cbegin(), c.mVehicles.cend(), LPrint);
  }
  return ost;
}
```

6.2 Class Vehicle

6.2.1 Vehicle.h

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

#ifndef VEHICLES
#define VEHICLES

#include "Logbook.h"
#include <string>
#include <list>
#include <time.h>

//fuel types of vehicle
enum class Fuel { Petrol, Diesel, Gas, Electricity };

class Vehicle
{
public:
  //general print method
  virtual std::ostream& Print(std::ostream& ost) = 0;
  //general clone method
  virtual std::unique_ptr<Vehicle> Clone() = 0;

  //constructor
  Vehicle();
  //deconstructor
  virtual ~Vehicle();

  //return numberplate of vehicle
  std::string GetNumberplate();
  //return driven kilometers of vehicle
  unsigned long GetMileage() const;

  //set brand of vehicle
  void SetBrand(std::string brand);
  //set numerplate of vehicle
  void SetNumberplate(std::string numberplate);
  //set fuel type of vehicle
  void SetFuel(Fuel fuel);

  //add a new logbook entry
  void AddNewLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int const&
      distance);

  //change the last logbook entry
  void ChangeLastLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int const&
       distance);

  Vehicle& operator=(Vehicle const& toCopy);

protected:
```

```
  std::string m_brand;
  std::string m_numberplate;
  Fuel m_fuel;

  //print all vehicles
  std::ostream& PrintList(std::ostream& ost);

private:
  Logbook m_logbook;
  int const m_monthPerYear = 12;
  int const m_daysPerMonth = 31;

  int m_currentDay = 0;
  int m_currentMonth = 0;
  int m_currentYear = 0;

  //create the struct Date by the given variables
  Logbook::Date CreateDate(size_t const& day, size_t const& month, size_t const& year);
  //get the current date
  void setCurrentDate();
};

#endif //VEHICLES
```

6.2.2 Vehicle.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;
}
```

```
47
48 ostream& Vehicle::PrintList(ostream& ost) {
49   //print all entries of logbook
50   m_logbook.PrintLogEntries(ost);
51   return ost;
52 }
53
54 unsigned long Vehicle::GetMileage() const{
55   //return total driven kilometers by vehicle
56   return m_logbook.GetTotalDistance();
57 }
58
59 void Vehicle::AddNewLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int
       const& distance) {
60   //create the struct Date by the given data
61   Logbook::Date date = CreateDate(day, month, year);
62   //add new entry, set date and distance
63   m_logbook.AddNewEntry(date, distance);
64 }
65
66 void Vehicle::ChangeLastLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int
       const& distance) {
67   //create the struct Date by the given data
68   Logbook::Date date = CreateDate(day, month, year);
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

```
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   //add new logbook entry to list
28   void AddNewEntry(Date const& date, int const& distance);
29
30   //print all logbook entries
31   std::ostream& PrintLogEntries(std::ostream& ost);
32
33   //calculate all driven kilometers
34   unsigned long GetTotalDistance() const;
35
36   //change the last logbook entry in the list
37   void ChangeLastEntry(Date const& date, int const& distance);
38
39 private:
40   typedef std::pair<Date, int> Entry;
41   std::list<Entry> m_entries;
42 };
43
44 #endif //LOGBOOK
```

6.3.2 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   //check if date is valid
14   if (date.day == 0 || date.month == 0 || date.year == 0) {
15     return;
16   }
17
18   //check if there is already an entry in the list
19   if (m_entries.empty()) {
20     //add new entry to list
21     m_entries.push_back(make_pair(date, distance));
22   }else{
23     //get last entry from the list
24     Entry lastEntry = m_entries.back();
25
26     try {
27       //check if the new date is equal to the last entry's date
28       if (lastEntry.first.day == date.day &&
29         lastEntry.first.month == date.month &&
30         lastEntry.first.year == date.year)
31       {
32         //delete last entry
33         m_entries.pop_back();
34         //add new entry + old entry's kilometer
35         m_entries.push_back(make_pair(date, distance + lastEntry.second));
36       }
37
38       //check if the new entry is older than the last one
39       if (date.year == lastEntry.first.year) {
40         if (date.month < lastEntry.first.month) {
41           throw exception("Wrong Date: Please input a date after the last entry");
42         }
43         else if (date.month == lastEntry.first.month) {
44           if (date.day < lastEntry.first.day) {
45             throw exception("Wrong Date: Please input a date after the last entry");
46           }
47         }
48       }else if (date.year < lastEntry.first.year) {
49         throw exception("Wrong Date: Please input a date after the last entry");
50       }
51       else {
52         //add new entry to list
53         m_entries.push_back(make_pair(date, distance));
54       }
55     }
56     catch (exception const& ex) {
57       cerr << ex.what() << endl;
58     }
59   }
60 }
61
62 ostream& Logbook::PrintLogEntries(ostream& ost) {
63   //check if ostream is valid
64   if (ost.good()) {
65     //print all log entries
66     for (auto e : m_entries) {
67       ost << e.first.day << ".";
68       ost << e.first.month << ".";
69       ost << e.first.year << "    ";
70       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
```

```
  for (auto e : m_entries) {
    sum += e.second;
  }
  return sum;
}

void Logbook::ChangeLastEntry(Date const& date, int const& distance) {
  //remove last entry
  m_entries.pop_back();
  //add new entry
  m_entries.push_back(make_pair(date, distance));
}
```

6.4 Class Motorcycle

6.4.1 Motorcycle.h

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

#ifndef MOTORCYCLE
#define MOTORCYCLE

#include "Vehicle.h"
class Motorcycle : public Vehicle
{
public:
  //constructor
  Motorcycle();
  //create a new motorcycle
  Motorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel);
  //deconstrcutor
  virtual ~Motorcycle();
  //Prints the brand, type, numberplate and logbook dat
  virtual std::ostream& Print(std::ostream& ost) override;
  //clone motorcycle
  virtual std::unique_ptr<Vehicle> Clone() override;
};

#endif //MOTORCYCLE
```

6.4.2 Motorcycle.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;
}
```

```
23
24 Motorcycle::~Motorcycle() {};
25
26 ostream& Motorcycle::Print(ostream& ost) {
27   //chek if ostream is valid
28   if (ost.good()) {
29     ost << "Fahrzeugart: Motorrad" << endl;
30     ost << "Marke: " << m_brand << endl;
31     ost << "Kennzeichen: " << m_numberplate << endl;
32     //print all logbook entries
33     Vehicle::PrintList(ost);
34   }
35   return ost;
36 }
37
38 std::unique_ptr<Vehicle> Motorcycle::Clone() {
39   return make_unique<Motorcycle>(*this);
40 }
```

6.5 Class Car

6.5.1 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   //constructor
19   Car();
20   //create a new car
21   Car(std::string const& brand, std::string const& numberplate, Fuel fuel);
22   //deconstrcutor
23   virtual ~Car();
24   //Prints the brand, type, numberplate and logbook data
25   virtual std::ostream& Print(std::ostream& ost) override;
26   //clone car
27   virtual std::unique_ptr<Vehicle> Clone() override;
28 };
29 #endif // CAR
```

6.5.2 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 }
```

```

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);
  }

  return ost;
}

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

6.6 Class Truck

6.6.1 Truck.h

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

#ifndef TRUCK
#define TRUCK
#include "Vehicle.h"

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

#endif // TRUCK
```

6.6.2 Truck.cpp

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

```cpp

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

using namespace std;

Truck::Truck() {
}

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

Truck::~Truck(){}

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

  }
  return ost;
}

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