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

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.

6 Software Quality Metrics

7 Source Code

7.1 Class Carpool

7.1.1 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:
  Carpool();
  ~Carpool();
  Carpool(Carpool const& toCopy);

  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);

  void Remove(std::string const& numberplate);
  void AddLogbookEntry(std::string const& numberplate, int const& day, int const& month, int const&
      year, int const distance);
  void ChangeLastLogbookEntry(std::string const& numberplate, int const& day, int const& month, int
      const& year, int const distance);

  void SearchByNumberplate(std::string const& numberplate);

  unsigned long TotalMileage() const;

  Carpool& operator =(Carpool const& toCopy);

  friend std::ostream& operator<<(std::ostream& ost, Carpool const& c);

private:
  TVehicles mVehicles;

  void Add(TUptr v);
  VehicleItor FindVehicle(std::string const& numberplate);
```

```
60
61 };
62
63 #endif //CARPOOL_H
```

7.1.2 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 Carpool::Carpool(Carpool const& toCopy) {
20   *this = toCopy;
21 }
22
23 void Carpool::Add(TUptr v) {
24   try {
25     if (FindVehicle((*v).GetNumberplate()) != this->mVehicles.cend()) {
26       throw exception("Add failed: Number already in the Database!");
27     }
28
29     mVehicles.emplace_back(move(v));
30   }
31   catch (exception const& ex) {
32     cerr << ex.what() << endl;
33   }
34   catch (bad_alloc const& ex) {
35     cerr << "memory allocation: " << ex.what() << endl;
36   }
37 }
38
39 void Carpool::AddCar(std::string const& brand, std::string const& numberplate, Fuel fuel) {
40   Car tmp{ brand, numberplate, fuel };
41   Add(make_unique<Car>(tmp));
42 }
43
44 void Carpool::AddTruck(std::string const& brand, std::string const& numberplate, Fuel fuel) {
45   Truck tmp{ brand, numberplate, fuel };
46   Add(make_unique<Truck>(tmp));
47 }
48
49 void Carpool::AddMotorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel) {
50   Motorcycle tmp{ brand, numberplate, fuel };
51   Add(make_unique<Motorcycle>(tmp));
52 }
53
54 void Carpool::Remove(std::string const& numberplate) {
55   try {
56     VehicleCItor vehicleToDel = FindVehicle(numberplate);
57     if (vehicleToDel == mVehicles.cend()) {
58       throw exception("Delete failed: The entered numberplate is not registered in this carpool!");
59     }
60     else {
61       mVehicles.erase(vehicleToDel);
62     }
63   }
64   catch (exception const& ex) {
65     cerr << ex.what() << endl;
```

```
  }
  catch (bad_alloc const& ex) {
    cerr << "memory allocation: " << ex.what() << endl;
  }
}

void Carpool::AddLogbookEntry(std::string const& numberplate, int const& day,
              int const& month, int const& year, int const distance) {
  VehicleItor tmp = FindVehicle(numberplate);
  (**tmp).AddNewLogbookEntry(day, month, year, distance);
}

void Carpool::ChangeLastLogbookEntry(std::string const& numberplate, int const& day,
                  int const& month, int const& year, int const distance) {
  VehicleItor tmp = FindVehicle(numberplate);
  (**tmp).ChangeLastLogbookEntry(day, month, year, distance);
}

void Carpool::SearchByNumberplate(std::string const& numberplate) {
  VehicleCItor foundVehicle = FindVehicle(numberplate);
  if (foundVehicle == mVehicles.end()) {
    cerr << "The vehicle with the numberplate :" << numberplate << " is not registered in this
        carpool! " << endl;
  }
  else {
    (**foundVehicle).Print(cout);
  }
}

VehicleItor Carpool::FindVehicle(std::string const& numberplate)  {
  auto PredNumberP = [numberplate](unique_ptr<Vehicle> const&  v) {
    return (numberplate == (*v).GetNumberplate());
  };

  return find_if(mVehicles.begin(), mVehicles.end(), PredNumberP);
}

unsigned long Carpool::TotalMileage() const {
  unsigned long tmpMileage = 0;
  for (VehicleCItor i = mVehicles.cbegin(); i != mVehicles.cend(); i++ ) {
    tmpMileage += (**i).GetMileage();
  }
  return tmpMileage;
}

Carpool& Carpool::operator=(Carpool const& toCopy) {
  if (&toCopy != this)  {
    for (VehicleCItor i = toCopy.mVehicles.cbegin(); i < toCopy.mVehicles.cend(); i++) {
      Add((**i).Clone());
    }
  }
  return *this;
}


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;
}
```

7.2 Class Vehicle

7.2.1 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>

enum class Fuel { Petrol, Diesel, Gas, Electricity };

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

  Vehicle();
  virtual ~Vehicle();

  std::string GetNumberplate();
  unsigned long GetMileage() const;

  void SetBrand(std::string brand);
  void SetNumberplate(std::string numberplate);
  void SetFuel(Fuel fuel);

  void AddNewLogbookEntry(size_t const& day, size_t const& month, size_t const& year, int const&
      distance);
  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;

  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;

  Logbook::Date CreateDate(size_t const& day, size_t const& month, size_t const& year);
  void setCurrentDate();
};

#endif //VEHICLES
```

7.2.2 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;
```

```
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
```

```
    if (year > m_currentYear || month > m_currentMonth || day > m_currentDay) {
      throw exception("Wrong Date: Back to the Future?");
    }

    date.day = day;
    date.month = month;
    date.year = year;
  }
  catch(exception const& ex){
    cerr << ex.what() << endl;
  }

  return date;
}

void Vehicle::setCurrentDate() {
  time_t now = time(0);
  tm ltm;
  localtime_s(&ltm, &now);

  m_currentYear = ltm.tm_year + 1900;
  m_currentMonth = 1 + ltm.tm_mon;
  m_currentDay = ltm.tm_mday;
}
```

7.3 Class Logbook

7.3.1 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;
  };

  void AddNewEntry(Date const& date, int const& distance);
  std::ostream& PrintLogEntries(std::ostream& ost);
  unsigned long GetTotalDistance() const;
  void ChangeLastEntry(Date const& date, int const& distance);

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

#endif //LOGBOOK
```

7.3.2 Logbook.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) {
  Entry lastEntry = *m_entries.end();

  try {
    if (lastEntry.first.day == date.day &&
      lastEntry.first.month == date.month &&
      lastEntry.first.year == date.year)
    {
      m_entries.pop_back();
      m_entries.push_back(make_pair(date, distance + lastEntry.second));
    }
    else if (lastEntry.first.day == date.day &&
      lastEntry.first.month == date.month &&
      lastEntry.first.year == date.year)
    {
      throw exception("Wrong Date: Please input a date after "+
              lastEntry.first.day + '.' +
              lastEntry.first.month + '.' +
              lastEntry.first.year);
    }
    else {
      m_entries.push_back(make_pair(date, distance));
    }
  }
  catch (exception const& ex) {
    cerr << ex.what() << endl;
  }
}

ostream& Logbook::PrintLogEntries(ostream& ost) {
  if (ost.good()) {
    for (auto e : m_entries) {
      ost << e.first.day << ".";
      ost << e.first.month << ".";
      ost << e.first.year << ".";
      ost << e.second << " km";
      ost << endl;
    }
  }
  return ost;
}

unsigned long Logbook::GetTotalDistance() const {
  unsigned long sum = 0;
  for (auto e : m_entries) {
    sum += e.second;
  }
  return sum;
}

void Logbook::ChangeLastEntry(Date const& date, int const& distance) {
  m_entries.pop_back();
  m_entries.push_back(make_pair(date, distance));
}
```

7.4 Class Motorcycle

7.4.1 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:
  Motorcycle();
  Motorcycle(std::string const& brand, std::string const& numberplate, Fuel fuel);
  virtual ~Motorcycle();
  virtual std::ostream& Print(std::ostream& ost) override;
  virtual std::unique_ptr<Vehicle> Clone() override;
};

#endif //MOTORCYCLE
```

7.4.2 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){
  m_brand = brand;
  m_numberplate = numberplate;
  m_fuel = fuel;
}

Motorcycle::~Motorcycle() {};

ostream& Motorcycle::Print(ostream& ost) {
  if (ost.good()) {
    ost << "Fahrzeugart: Motorrad" << endl;
    ost << "Marke: " << m_brand << endl;
    ost << "Kennzeichen: " << m_numberplate << endl;
  }
  Vehicle::PrintList(ost);
  return ost;
}

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

7.5 Class Car

7.5.1 Car.h

```
/*-------------------------------------------------------------------------
|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:
  Car();
  Car(std::string const& brand, std::string const& numberplate, Fuel fuel);
  virtual ~Car();
  virtual std::ostream& Print(std::ostream& ost) override;
  virtual std::unique_ptr<Vehicle> Clone() override;
};
#endif // CAR
```

7.5.2 Car.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) {
  m_brand = brand;
  m_numberplate = numberplate;
  m_fuel = fuel;
}

Car::~Car() {};

ostream& Car::Print(ostream& ost) {
  if (ost.good()) {
    ost << "Fahrzeugart: PKW" << endl;
    ost << "Marke: " << m_brand << endl;
    ost << "Kennzeichen: " << m_numberplate << endl;
    Vehicle::PrintList(ost);
  }

  return ost;
}

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

7.6 Class Truck

7.6.1 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:
  Truck();
  Truck(std::string const& brand, std::string const& numberplate, Fuel fuel);
  virtual ~Truck();
  virtual std::ostream& Print(std::ostream& ost) override;
  virtual std::unique_ptr<Vehicle> Clone() override;
};

#endif // TRUCK
```

7.6.2 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
|-----------------------------------------------------------------------*/

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

using namespace std;

Truck::Truck() {
}

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

Truck::~Truck(){}

ostream& Truck::Print(ostream& ost) {
  if (ost.good()) {
    ost << "Fahrzeugart: LKW" << endl;
    ost << "Marke: " << m_brand << endl;
    ost << "Kennzeichen: " << m_numberplate << endl;
    Vehicle::PrintList(ost);

  }
  return ost;
}

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