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

Name(1): Adam Kensy — Abgabetermin: 29.10.2019

Name(2): Philipp Holzer — Punkte:

Übungsgruppe: 2 — korrigiert:

Geschätzter Aufwand in Ph: 14 — Effektiver Aufwand in Ph: 16

## 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 - Uebung 1

**Wintersemester 2019/20**

Adam Kensy - S1810306018
Philipp Holzer - S1810306028

27. Oktober 2019

## Inhaltsverzeichnis

**1 Organisatorisches** **4**
1.1 Team . . . 4
1.2 Aufteilung und Verantwortlichkeitsbereiche . . . 4
1.3 Aufwand . . . 4

**2 Anforderungsdefinition (Systemspezifikation)** **5**

**3 Systementwurf** **6**
3.1 Klassendiagramm . . . 6
3.2 Klassendiagramm durch Visual Studio . . . 7
3.3 Komponentenübersicht . . . 7
3.4 Designentscheidungen . . . 7

**4 Komponentenentwurf** **9**
4.1 Klasse Object . . . 9
4.2 Klasse CarPool . . . 9
4.3 Klasse Vehicle . . . 9
4.4 Klasse Car, Truck und Motorcycle . . . 9
4.5 Klasse LogBook . . . 9
4.6 Klasse TEntry . . . 10

**5 Testprotokollierung** **11**
5.1 Testumgebung . . . 11
5.2 Testausgabe . . . 11

**6 Quellcode** **14**
6.1 Object . . . 14
6.2 CarPool . . . 14
6.3 Vehicles . . . 17
6.4 LogBook . . . 21
6.5 main . . . 24

# 1 Organisatorisches

## 1.1 Team

- Philipp Holzer, Matr.-Nr.: 1810306028
- Adam Kensy, Matr.-Nr.: 1810306018

## 1.2 Aufteilung und Verantwortlichkeitsbereiche

- Philipp Holzer
  - Planung
  - Klassendiagramm
  - Implementierung und Testen der Klassen
    - Object
    - Logbook
    - Vehicle
  - Dokumentation
- Adam Kensy
  - Planung
  - Klassendiagramm
  - Implementierung und Testen der Klassen
    - Carpool
    - Vehicle
    - Car, Truck, Motorcycle
  - Dokumentation

## 1.3 Aufwand

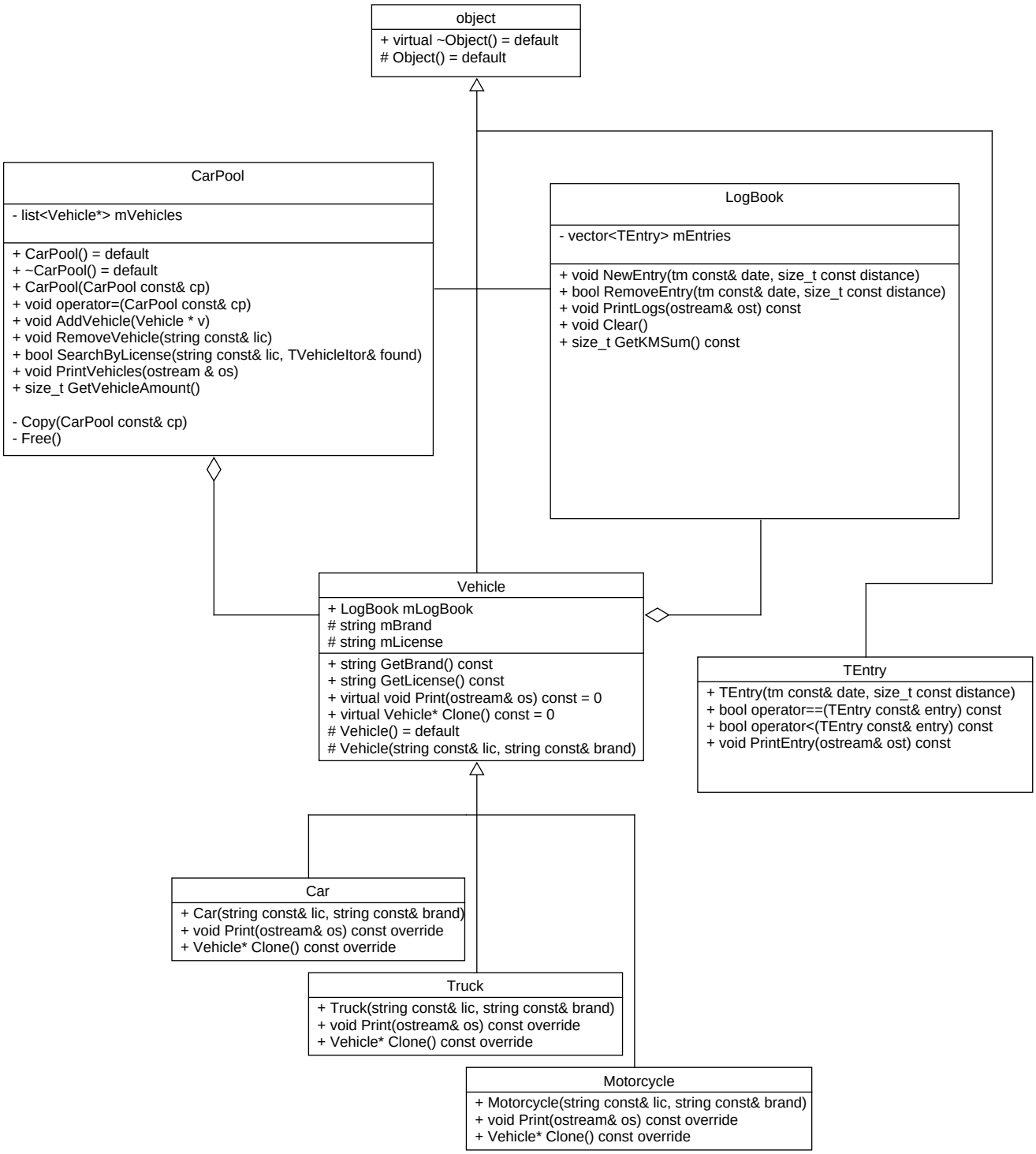
- Philipp Holzer geschätzt: 7 tatsächlich: 8
- Adam Kensy geschätzt: 7 tatsächlich: 8

# 2 Anforderungsdefinition (Systemspezifikation)

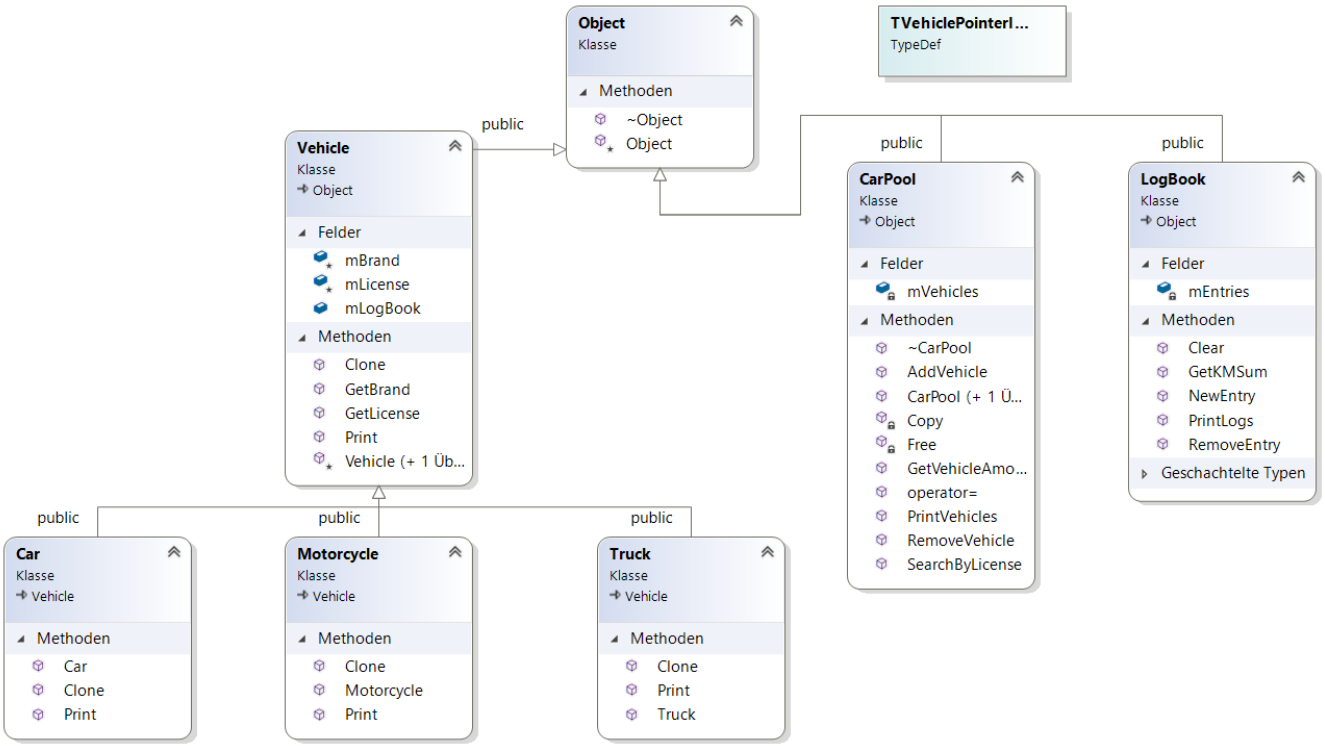
Gesucht ist ein Fuhrparkverwaltungssystem, welche verschieden Fahrzeuge und deren Fahrtenbücher verwaltet.
Dabei gibt es drei unterschiedliche Fahrzeugarten - PKW, LKW und Motorrad - wo jedes jeweils ein Kennzeichen, eine Marke und ein Fahrtenbuch hat.
Das Fahrtenbuch wird chronologisch abgespeichert - man kann auch Einträge entfernen.

# 3 Systementwurf

## 3.1 Klassendiagramm



## 3.2 Klassendiagramm durch Visual Studio




## 3.3 Komponentenübersicht

- **Klasse "Object"**
  Basis aller Klassen
- **Klasse "Carpool"**
  Verwaltet alle Fahrzeuge
  Besitzt eine Ausgabefunktion um alle enthaltenen Fahrzeuge auszugeben
- **Klasse "Logbook"**
  Das Fahrtenbuch der Fahrzeuge
  Einträge werden sortiert eingefügt
- **Klasse "Vehicle"**
  Abstrakte Basisklasse der Fahrzeuge
- **Klasse "Car", "Truck", "Motorcycle"**
  Konkrete Objekte für die Fahrzeuge

## 3.4 Designentscheidungen

- Es wurde keine Kennzeichen-Überprüfung implementiert, da von Staat zu Staat die Kennzeichen unterschiedlich variieren. Jedoch darf jedes Kennzeichen nur einmal vorkommen.
- Fehlermeldungen werden in der Konsole ausgegeben und es werden keine Exceptions nach außen geworfen.

- Um ein Fahrzeug hinzuzufügen muss ein Pointer auf ein dynamisch angelegtes Fahrzeugobjekt übergeben werden. Bereits am Stack erzeugte Fahrzeuge können mithilfe der Clone()-Funktion zum Fuhrpark hinzugefügt werden. Das Entfernen jedoch funktiort mittels Kennzeichenübergabe.
- Da wir davon ausgehen, dass im Fuhrpark Fahrzeuge öfter gesucht werden müssen, entschieden wir uns für eine Standard-list als Container.
- Wir benutzen die tm-Struktur aus der Library ¡ctime¿ für Datums-Angaben und setzen Werte gemäß der Dokumentation voraus.
- Es wird sortiert ins Fahrtenbuch eingetragen, dies macht am meisten Sinn, da man dies direkt nach einer Fahrt macht.
- Einträge werden nur innerhalb des LogBooks bearbeitet. Von außen können sie nur erstellt oder gelöscht werden.

# 4 Komponentenentwurf

## 4.1 Klasse Object

Diese Klasse stellt die Basis aller Klassen dar.

## 4.2 Klasse CarPool

Diese Klasse verwaltet alle Fahrzeuge im Fuhrpark. Die Pointer auf alle Fahrzeuge werden in einer Liste gespeichert.
Hier wurde die Rule-Of-Three angewandt um Memory Leaks zu vermeiden in Fällen von Zuweisungen oder Kopiervorgängen.
Mit AddVehicle() können Fahrzeuge hinzugefügt werden. Dieser Funktion muss ein Pointer auf ein PKW, LKW oder Motorrad übergeben werden.
Mit RemoveVehicle() können Fahrzeuge entfernt werden und dieser Funktion muss ein Kennzeichen übergeben werden.
Mit SearchByLicense() kann nach Fahrzeugen gesucht werden. Dieser Funktion wird ebenfalls ein Kennzeichen übergeben und zusätzlich ein Iterator, als Übergangsparameter auf das Objekt. Dieser zeigt nach Ablauf der Funktion auf das gefundene Objekt, ansonsten auf das Ende Containers. Wurde das Kennzeichen im Fuhrpark gefunden wird true zurückgegeben ansonsten false, was und auch in der Konsole mitgeteilt wird.
Mit der PrintVehicles() Funktion wird der Fuhrpark komplett ausgegeben. Hierbei wird die Print Funktion jedes Fahrzeuges aufgerufen worin auch das LogBook ausgegeben wird.

## 4.3 Klasse Vehicle

Diese abstrakte Klasse, von der die einzelnen Fahrzeuge abgeleitet werden, besitzt alle Basis-Funktionen für ein Fahrzeug.
Mit den beiden Get-Funktionen (GetBrand() und GetLicense()) erfährt man die Marke und das Kennzeichen des Autos. Die Print()-Funktion ist eine abstrakte Methode welche dann in den einzelnen Fahrzeugen aufgerufen wird.
Die abstrake Clone()-Funktion wird benötigt um am Stack angelegte Objekte dem Fuhrpark hinzuzufügen da der Fuhrpark nur dynamisch erstellte Objekte verwaltet.

## 4.4 Klasse Car, Truck und Motorcycle

Die von der Klasse Vehicle abgeleiteten Klassen sehen beinahe identisch aus. Alle 3 haben ihren Konstruktur wo man ein Kennzeichen und eine Marke übergibt.
Alle 3 Klassen müssen eine Print() und eine Clone() Funktion implementiert haben.

## 4.5 Klasse LogBook

Diese Klasse stellt das Fahrtenbuch eines Fahrzeuges dar. Mit NewEntry() kann ein Eintrag erzeugt werden, wobei sortiert eingefügt wird und mit RemoveEntry() kann man einen Eintrag entfernen. Beide verlangen als Parameter ein Datum des Datentyp "tm"(aus der library ¡ctime¿).
Die PrintLogs()-Funktion wird in den einzelnen Print-Funktionen der abgeleiteten Fahrzeuge aufgerufen, sodass das eigene Fahrtenbuch formatiert ausgegeben wird.

Mit Clear() kann ein ganzes Fahrtenbuch gelöscht werden. Die GetKMSum() Funktion berechnet die gesamt gefahrenen Kilometer eines Fahrzeuges.

## 4.6 Klasse TEntry

Die Klasse TEntry bildet einen Eintrag im Fahrtenbuch ab. Sie ist nur für die Klasse LogBook sichtbar, da außerhalb des LogBooks keine Einträge einzeln bearbeitet werden. Sie werden nur hinzugefügt oder gelöscht.

# 5 Testprotokollierung

## 5.1 Testumgebung

Microsoft Visual Studio Enterprise 2019
Version 16.3.5
Microsoft Visual C++ 2019

Windows 10, 64Bit, Build 18362

Testdriver: main.cpp

## 5.2 Testausgabe

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Detector\vld.ini
Visual Leak Detector Version 2.5.1 installed.
**********************************
Testsection LogBook
**********************************
24.3.1993:       43 km
12.1.2008:       17 km
31.7.2019:       21 km
17.9.2019:       89 km
17.9.2019:       89 km
17.9.2019:      110 km
4.12.2019:       43 km

Entry got removed:
-----------------------------------
24.3.1993:       43 km
12.1.2008:       17 km
31.7.2019:       21 km
17.9.2019:       89 km
17.9.2019:      110 km
4.12.2019:       43 km

 Removing a nonexisting entry:
-----------------------------------
Entry does not exist! Couldn't delete entry.

Print km-sum:
-----------------------------------
323

Log cleared (for the next testcases it got filled again):
-----------------------------------

**********************************
Testsection CarPool
**********************************
All added vehicles:
-----------------------------------
Fahrzeugart:  Motorrad
Marke:        Kawazaki
Kennzeichen:  LL-HAGE1

Fahrzeugart:  LKW
Marke:        Mercedes
Kennzeichen:  LL-HARD3
24.3.1993:       98 km
12.1.2008:       67 km
31.7.2019:       45 km
17.9.2019:       21 km
4.12.2019:       34 km

Fahrzeugart:  PKW
```

```
Marke:         Opel Corsa (nagelneu)
Kennzeichen:   LL-ISSO5
24.3.1993:      927 km
12.1.2008:      231 km
31.7.2019:     7028 km
17.9.2019:     6837 km
4.12.2019:      211 km

Fahrzeugart:   Motorrad
Marke:         Yamaha
Kennzeichen:   LL-BERG2
24.3.1993:       23 km
12.1.2008:      432 km
31.7.2019:       26 km
17.9.2019:       45 km
4.12.2019:      117 km

Fahrzeugart:   LKW
Marke:         Koenigsegg
Kennzeichen:   LL-WARE4
24.3.1993:    82456 km
12.1.2008:     4567 km
31.7.2019:     4332 km
17.9.2019:     4321 km
4.12.2019:     6789 km

Fahrzeugart:   PKW
Marke:         Opel Corsa (verrostet und ohne Klima)
Kennzeichen:   LL-C000L
24.3.1993:     1093 km
12.1.2008:      265 km
31.7.2019:      483 km
17.9.2019:     7392 km
4.12.2019:       46 km

Adams Car (LL-ISSO5) got removed:
-----------------------------------
Fahrzeugart:   Motorrad
Marke:         Kawazaki
Kennzeichen:   LL-HAGE1

Fahrzeugart:   LKW
Marke:         Mercedes
Kennzeichen:   LL-HARD3
24.3.1993:       98 km
12.1.2008:       67 km
31.7.2019:       45 km
17.9.2019:       21 km
4.12.2019:       34 km

Fahrzeugart:   Motorrad
Marke:         Yamaha
Kennzeichen:   LL-BERG2
24.3.1993:       23 km
12.1.2008:      432 km
31.7.2019:       26 km
17.9.2019:       45 km
4.12.2019:      117 km

Fahrzeugart:   LKW
Marke:         Koenigsegg
Kennzeichen:   LL-WARE4
24.3.1993:    82456 km
12.1.2008:     4567 km
31.7.2019:     4332 km
17.9.2019:     4321 km
4.12.2019:     6789 km

Fahrzeugart:   PKW
Marke:         Opel Corsa (verrostet und ohne Klima)
Kennzeichen:   LL-C000L
24.3.1993:     1093 km
12.1.2008:      265 km
```

```
31.7.2019:     483 km
17.9.2019:    7392 km
4.12.2019:      46 km

Print LKW with license 'LL-HARD3' (SearchByLicense):
-----------------------------------
Fahrzeugart:  LKW
Marke:        Mercedes
Kennzeichen:  LL-HARD3
24.3.1993:      98 km
12.1.2008:      67 km
31.7.2019:      45 km
17.9.2019:      21 km
4.12.2019:      34 km

Number of vehicles in the car pool:
5

Print vehicles of copied object (Copy CTOR):
-----------------------------------
Fahrzeugart:  Motorrad
Marke:        Kawazaki
Kennzeichen:  LL-HAGE1

Fahrzeugart:  LKW
Marke:        Mercedes
Kennzeichen:  LL-HARD3
24.3.1993:      98 km
12.1.2008:      67 km
31.7.2019:      45 km
17.9.2019:      21 km
4.12.2019:      34 km

Fahrzeugart:  Motorrad
Marke:        Yamaha
Kennzeichen:  LL-BERG2
24.3.1993:      23 km
12.1.2008:     432 km
31.7.2019:      26 km
17.9.2019:      45 km
4.12.2019:     117 km

Fahrzeugart:  LKW
Marke:        Koenigsegg
Kennzeichen:  LL-WARE4
24.3.1993:   82456 km
12.1.2008:    4567 km
31.7.2019:    4332 km
17.9.2019:    4321 km
4.12.2019:    6789 km

Fahrzeugart:  PKW
Marke:        Opel Corsa (verrostet und ohne Klima)
Kennzeichen:  LL-C000L
24.3.1993:    1093 km
12.1.2008:     265 km
31.7.2019:     483 km
17.9.2019:    7392 km
4.12.2019:      46 km

Remove vehicle with license 'LL-HARD3' and assign PandA_Solutions to carpool1 (assignment
    operator):
-----------------------------------
Fahrzeugart:  Motorrad
Marke:        Kawazaki
Kennzeichen:  LL-HAGE1

Fahrzeugart:  Motorrad
Marke:        Yamaha
Kennzeichen:  LL-BERG2
24.3.1993:      23 km
12.1.2008:     432 km
31.7.2019:      26 km
```

```
17.9.2019:       45 km
4.12.2019:      117 km

Fahrzeugart:  LKW
Marke:        Koenigsegg
Kennzeichen:  LL-WARE4
24.3.1993:    82456 km
12.1.2008:     4567 km
31.7.2019:     4332 km
17.9.2019:     4321 km
4.12.2019:     6789 km

Fahrzeugart:  PKW
Marke:        Opel Corsa (verrostet und ohne Klima)
Kennzeichen:  LL-C000L
24.3.1993:     1093 km
12.1.2008:      265 km
31.7.2019:      483 km
17.9.2019:     7392 km
4.12.2019:       46 km

No memory leaks detected.
Visual Leak Detector is now exiting.

C:\Users\kensy\Google Drive\Hardware-Software-Design\3-Semester\SDP3\Uebung\Fuhrpark\Fuhrpark\
    CarPool\x64\Debug\CarPool.exe (Prozess "18932") wurde mit Code "0" beendet.
Um die Konsole beim Beenden des Debuggens automatisch zu schließen, aktivieren Sie "Extras" >
    "Optionen" > "Debuggen" > "Konsole␣beim␣Beenden␣des␣Debuggings␣automatisch␣schließen".
Drücken Sie eine beliebige Taste, um dieses Fenster zu schließen.
```

# 6 Quellcode

## 6.1 Object

Listing 1: CarPool/Object.h

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Object.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : common base class
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef OBJECT_H
11 #define OBJECT_H
12
13 class Object
14 {
15 protected:
16   Object() = default;
17
18 public:
19
20   virtual ~Object() = default;
21 };
22
23 #endif
```

Listing 2: CarPool/Object.cpp

## 6.2 CarPool

Listing 3: ./CarPool/CarPool/CarPool.h

```
1  ////////////////////////////////////////////////////////////////////////
```

```
2  // Workfile : LogBook.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : car pool system which includes different vehicle types
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef CARPOOL_H
11 #define CARPOOL_H
12
13 #include "../Object.h"
14 #include "../Vehicles/Vehicle.h"
15 #include "../Vehicles/Car.h"
16 #include "../Vehicles/Truck.h"
17 #include "../Vehicles/Motorcycle.h"
18 #include <list>
19
20 typedef std::list<Vehicle*>::iterator TVehiclePointerItor;
21
22 class CarPool : public Object
23 {
24 public:
25   //DTor for CarPool
26   ~CarPool() override;
27
28   //Default CTOR
29   CarPool() = default;
30
31   //Copy-CTOR
32   CarPool(CarPool const& cp);
33
34   //assignmentoperator
35   void operator=(CarPool const& cp);
36
37   //Adds a new car to the carpool
38   //param c: an existing car object
39   void AddVehicle(Vehicle * v);
40
41   //Removes an existing vehicle out of the carpool
42   //param veh: a vehicle that should be in the carpool
43   void RemoveVehicle(std::string const& license);
44
45   //searches through the CarPool for an existing vehicle
46   //param lic: license plate number of the vehicle
47   //param found: iterator which points on the found vehicle
48   //return: true if a vehicle was found else false
49   bool SearchByLicense(std::string const& lic, TVehiclePointerItor& found);
50
51   //prints the info of all vehicles in the carpool
52   void PrintVehicles(std::ostream & os) const;
53
54   //get function for amount of vehicles
55   size_t GetVehicleAmount();
56 private:
57
58   //container which includes all vehicles
59   std::list<Vehicle*> mVehicles;
60
61   //Helpfunction for DTOR and assignment operator
62   //param cp: the copied/assigned CarPool
63   void Copy(CarPool const& cp);
64
65   //Helpfunction for DTOR and assignment operator
66   //Frees all allocated memory
67   void Free();
68
69 };
70
71 #endif
```

Listing 4: ./CarPool/CarPool/CarPool.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : LogBook.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : car pool system which includes different vehicle types
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9  #include "../CarPool/CarPool.h"
10 #include <algorithm>
11 #include <iostream>
12
13 static const std::string cErrLicenseAlreadyExists = "License already exists! Couldn't add
       vehicle.";
14 static const std::string cErrVehicleDoesNotExist = "Vehicle does not exist! Couldn't
      remove vehicle.";
15 static const std::string cErrVehicleNullptr = "Given vehicle pointer is null!";
16
17 CarPool::~CarPool()
18 {
19   Free();
20 }
21
22 CarPool::CarPool(CarPool const& cp)
23 {
24   Copy(cp);
25 }
26
27 void CarPool::operator=(CarPool const& cp)
28 {
29   if (this != &cp)
30   {
31     Free();
32     Copy(cp);
33   }
34 }
35
36 void CarPool::AddVehicle(Vehicle * v)
37 {
38   if (v == nullptr)
39   {
40     std::cerr << cErrVehicleNullptr << std::endl;
41     return;
42   }
43   TVehiclePointerItor it;
44   if (!SearchByLicense(v->GetLicense(), it))
45   {
46     mVehicles.emplace_back(v);
47   }
48   else
49   {
50     std::cerr << cErrLicenseAlreadyExists << std::endl;
51   }
52 }
53
54 void CarPool::RemoveVehicle(std::string const& license)
55 {
56   TVehiclePointerItor it;
57   if (SearchByLicense(license, it))
58   {
59     delete* it;
60     *it = nullptr;
61     mVehicles.erase(it);
62   }
63   else
64   {
65     std::cerr << cErrVehicleDoesNotExist << std::endl;
66   }
67 }
68
69 bool CarPool::SearchByLicense(std::string const& lic, TVehiclePointerItor & found)
```

```
{
  auto compByLicense = [&](auto * v) { return v->GetLicense() == lic; };
  found = std::find_if(mVehicles.begin(), mVehicles.end(), compByLicense);
  if (found != mVehicles.cend())
  {
    return true;
  }
  else
  {
    return false;
  }
}

void CarPool::PrintVehicles(std::ostream & os) const
{
  if (!os.good())
  {
    std::cerr << "error write stream" << std::endl;
  }
  for (auto it = mVehicles.cbegin(); it != mVehicles.cend(); ++it)
  {
    (*it)->Print(os);
  }
}

size_t CarPool::GetVehicleAmount()
{
  return mVehicles.size();
}

void CarPool::Copy(CarPool const& cp)
{
  for (auto it = cp.mVehicles.cbegin(); it != cp.mVehicles.cend(); ++it)
  {
    AddVehicle((*it)->Clone());
  }
}

void CarPool::Free()
{
  for (auto it = mVehicles.begin(); it != mVehicles.end(); ++it)
  {
    delete* it;
    *it = nullptr;
  }

  mVehicles.clear();
}
```

## 6.3 Vehicles

Listing 5: ./CarPool/Vehicles/Vehicle.h

```
////////////////////////////////////////////////////////////////////////
// Workfile : Vehicle.h
// Author : Philipp Holzer / Adam Kensy
// Date : 15.10.2019
// Description : vehicle class
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////

#ifndef VEHICLE_H
#define VEHICLE_H
#include "../LogBook/LogBook.h"
#include "../Object.h"

class Vehicle : public Object
{
public:
```

```
18
19    LogBook mLogBook;
20
21    //Get-function for brand
22    std::string GetBrand() const;
23
24    //Get-function for license plate
25    std::string GetLicense() const;
26
27    //Prints the vehicle to the given ostream
28    virtual void Print(std::ostream& os) const = 0;
29
30    //creates a clone of itself on the heap
31    //return: pointer to the instance of itself on the heap
32    virtual Vehicle* Clone() const = 0;
33
34  protected:
35
36    Vehicle() = default;
37
38    Vehicle(std::string const& lic, std::string const& brand);
39
40    std::string mBrand;
41
42    std::string mLicense;
43  };
44
45  #endif
```

Listing 6: ./CarPool/Vehicles/Vehicle.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Vehicle.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : vehicle class
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9  #include "Vehicle.h"
10
11
12 std::string Vehicle::GetBrand() const
13 {
14   return mBrand;
15 }
16
17 std::string Vehicle::GetLicense() const
18 {
19   return mLicense;
20 }
21
22 Vehicle::Vehicle(std::string const& lic, std::string const& brand) : mLicense{lic},
       mBrand{brand}
23 {
24 }
```

Listing 7: ./CarPool/Vehicles/Car.h

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Car.h
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : vehicle class - car
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef CAR_H
11 #define CAR_H
```

```
12 #include "Vehicle.h"
13
14 class Car : public Vehicle
15 {
16 public:
17   Car(std::string const& lic, std::string const& brand);
18
19   void Print(std::ostream& os) const override;
20
21   Vehicle* Clone() const override;
22 };
23
24 #endif
```

Listing 8: ./CarPool/Vehicles/Car.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Car.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : vehicle class - car
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9  #include "Car.h"
10 #include "../PrintParameters.h"
11 #include <iomanip>
12
13 Car::Car(std::string const& lic, std::string const& brand) : Vehicle {lic, brand}
14 {
15 }
16
17 void Car::Print(std::ostream& os) const
18 {
19   if (!os.good())
20   {
21     std::cerr << "error write stream" << std::endl;
22   }
23   os << std::setw(14) << std::left << "Fahrzeugart:" << std::right << "PKW" << std::endl;
24   os << std::setw(14) << std::left << "Marke: " << std::right << mBrand << std::endl;
25   os << std::setw(14) << std::left << "Kennzeichen: "  << std::right << mLicense << std::
       endl;
26   mLogBook.PrintLogs(os);
27   os << std::endl;
28 }
29
30 Vehicle* Car::Clone() const
31 {
32   try
33   {
34     Car* pCar = new Car{ *this };
35     return pCar;
36   }
37   catch (std::bad_alloc const& ex)
38   {
39     std::cerr << ex.what() << std::endl;
40     std::cerr << cErrAllocation << std::endl;
41     return nullptr;
42   }
43 }
```

Listing 9: ./CarPool/Vehicles/Truck.h

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Truck.h
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : vehicle class - truck
6  // Remarks : -
7  // Revision : 0
```

```
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef TRUCK_H
11 #define TRUCK_H
12 #include "Vehicle.h"
13
14
15 class Truck : public Vehicle
16 {
17 public:
18   Truck(std::string const& lic, std::string const& brand);
19
20   void Print(std::ostream& os) const override;
21
22   Vehicle* Clone() const override;
23 };
24
25 #endif
```

Listing 10: ./CarPool/Vehicles/Truck.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Truck.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : vehicle class - truck
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9  #include "Truck.h"
10 #include "../PrintParameters.h"
11 #include <iomanip>
12
13 Truck::Truck(std::string const& lic, std::string const& brand) : Vehicle{lic, brand}
14 {
15 }
16
17 void Truck::Print(std::ostream& os) const
18 {
19   if (!os.good())
20   {
21     std::cerr << "error write stream" << std::endl;
22   }
23   os << std::setw(14) << std::left << "Fahrzeugart:" << std::right << "LKW" << std::endl;
24   os << std::setw(14) << std::left << "Marke: " << std::right << mBrand << std::endl;
25   os << std::setw(14) << std::left << "Kennzeichen: " << std::right << mLicense << std::
       endl;
26   mLogBook.PrintLogs(os);
27   os << std::endl;
28 }
29
30 Vehicle* Truck::Clone() const
31 {
32   try
33   {
34     Truck* pTruck = new Truck{ *this };
35     return pTruck;
36   }
37   catch (std::bad_alloc const& ex)
38   {
39     std::cerr << ex.what() << std::endl;
40     std::cerr << cErrAllocation << std::endl;
41     return nullptr;
42   }
43 }
```

Listing 11: ./CarPool/Vehicles/Motorcycle.h

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Motorcycle.h
```

```
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : vehicle class - motorcycle
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef MOTORCYCLE_H
11 #define MOTORCYCLE_H
12 #include "Vehicle.h"
13
14 class Motorcycle : public Vehicle
15 {
16 public:
17
18   Motorcycle(std::string& lic, std::string& brand);
19
20   void Print(std::ostream& os) const override;
21
22   Vehicle* Clone() const override;
23 };
24
25 #endif
```

Listing 12: ./CarPool/Vehicles/Motorcycle.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Motorcycle.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : vehicle class - motorcycle
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9  #include "Motorcycle.h"
10 #include "../PrintParameters.h"
11 #include <iomanip>
12
13 Motorcycle::Motorcycle(std::string& lic, std::string& brand) : Vehicle{lic, brand}
14 {
15 }
16
17 void Motorcycle::Print(std::ostream& os) const
18 {
19   if (!os.good())
20   {
21     std::cerr << "error write stream" << std::endl;
22   }
23   os << std::setw(14) << std::left << "Fahrzeugart:" << std::right << "Motorrad" << std::
       endl;
24   os << std::setw(14) << std::left << "Marke: " << std::right << mBrand << std::endl;
25   os << std::setw(14) << std::left << "Kennzeichen: " << std::right << mLicense << std::
       endl;
26   mLogBook.PrintLogs(os);
27   os << std::endl;
28 }
29
30 Vehicle* Motorcycle::Clone() const
31 {
32   try
33   {
34     Motorcycle* pMotorcycle = new Motorcycle{ *this };
35     return pMotorcycle;
36   }
37   catch (std::bad_alloc const& ex)
38   {
39     std::cerr << ex.what() << std::endl;
40     std::cerr << cErrAllocation << std::endl;
41     return nullptr;
42   }
43 }
```

## 6.4 LogBook

Listing 13: ./CarPool/LogBook/LogBook.h

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : LogBook.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : drivers log book for a vehicle
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////////
9
10 #ifndef LOGBOOK_H
11 #define LOGBOOK_H
12
13 #include "../Object.h"
14 #include <vector>
15 #include <ctime>
16 #include <iostream>
17
18 //Class which represents a log book for vehicles
19 class LogBook : public Object
20 {
21 public:
22
23   //Creates a new entry and adds it to the log book
24   //param date: Struct tm from ctime
25   //param distance: the driven distance in km
26   void NewEntry(tm const& date, size_t const distance);
27
28   //Removes one single entry and which contains exactly the given date and distance
29   //param date: Struct tm from ctime
30   //param distance: the driven distance in km
31   void RemoveEntry(tm const& date, size_t const distance);
32
33   //Prints the whole log book to the given ostream
34   //param ost: ostream to write
35   void PrintLogs(std::ostream& ost) const;
36
37   //Deletes all entries
38   void Clear();
39
40   //Calculates the total distance in km
41   //return: total distance in km
42   size_t GetKMSum() const;
43
44 private:
45
46   //This class represents an entry in the log book
47   class TEntry : public Object
48   {
49   public:
50     tm mDate;
51     size_t mDistance;
52
53     TEntry(tm const& date, size_t const distance);
54
55     bool operator==(TEntry const& entry) const;
56     bool operator<(TEntry const& entry) const;
57
58     //Prints a single entry to the given ostream
59     //param ost: ostream to print at
60     void PrintEntry(std::ostream& ost) const;
61   };
62
63   std::vector<TEntry> mEntries;
64 };
65
66 #endif
```

Listing 14: ./CarPool/LogBook/LogBook.cpp

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : LogBook.cpp
// Author : Philipp Holzer / Adam Kensy
// Date : 15.10.2019
// Description : drivers log book for a vehicle
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////////

#include "LogBook.h"
#include <algorithm>
#include <iomanip>
#include "../PrintParameters.h"

static const std::string cDistanceUnit = "km";

static const size_t cTmOffsetYears = 1900;
static const size_t cTmMonthOffset = 1;

static const std::string cErrEntryDoesNotExist = "Entry does not exist! Couldn't delete entry.";

void LogBook::NewEntry(tm const& date, size_t const distance)
{
  TEntry newEntry{ date, distance };
  //find the position where to insert the new entry
  auto it = std::find_if(mEntries.cbegin(), mEntries.cend(), [newEntry](TEntry const& e) { return newEntry < e; });
  mEntries.insert(it, newEntry);
}

void LogBook::RemoveEntry(tm const& date, size_t const distance)
{
  auto foundIt = std::find(mEntries.cbegin(), mEntries.cend(), TEntry{ date, distance });
  if (foundIt != mEntries.cend())
  {
    mEntries.erase(foundIt);
  }
  else
  {
    std::cerr << cErrEntryDoesNotExist << std::endl;
  }
}

void LogBook::PrintLogs(std::ostream& ost) const
{
  if (!ost.good())
  {
    std::cerr << "error write stream" << std::endl;
  }
  for (auto it = mEntries.cbegin(); it != mEntries.cend(); ++it)
  {
    it->PrintEntry(ost);
  }
}

void LogBook::Clear()
{
  mEntries.clear();
}

size_t LogBook::GetKMSum() const
{
  size_t sum = 0;
  for (auto it = mEntries.cbegin(); it != mEntries.cend(); ++it)
  {
    sum += it->mDistance;
  }

  return sum;
}
```

```
70
71 LogBook::TEntry::TEntry(tm const& date, size_t const distance): mDate{date}, mDistance{
       distance}
72 {
73 }
74
75 bool LogBook::TEntry::operator==(TEntry const& entry) const
76 {
77   return mDistance == entry.mDistance && mDate.tm_year == entry.mDate.tm_year && mDate.
         tm_mon == entry.mDate.tm_mon && mDate.tm_mday == entry.mDate.tm_mday;
78 }
79
80 bool LogBook::TEntry::operator<(TEntry const& entry) const
81 {
82   if (mDate.tm_year <= entry.mDate.tm_year)
83   {
84     if (mDate.tm_year < entry.mDate.tm_year)
85     {
86       return true;
87     }
88     else
89     {
90       if (mDate.tm_mon <= entry.mDate.tm_mon)
91       {
92         if (mDate.tm_mon < entry.mDate.tm_mon)
93         {
94           return true;
95         }
96         else
97         {
98           if (mDate.tm_mday < entry.mDate.tm_mday)
99           {
100            return true;
101          }
102          else
103          {
104            if (mDistance < entry.mDistance)
105            {
106              return true;
107            }
108            else
109            {
110              return false;
111            }
112          }
113        }
114      }
115      else
116      {
117        return false;
118      }
119    }
120  }
121  else
122  {
123    return false;
124  }
125 }
126
127 void LogBook::TEntry::PrintEntry(std::ostream& ost) const
128 {
129   if (!ost.good())
130   {
131     std::cerr << "error write stream" << std::endl;
132   }
133   ost << mDate.tm_mday << "." << mDate.tm_mon + cTmMonthOffset << "."
134     << mDate.tm_year + cTmOffsetYears << ":" << std::setw(8)<< std::right
135     << mDistance << " " << cDistanceUnit << std::endl;
136 }
```

## 6.5 main

Listing 15: "./CarPool/main.cpp"

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : main.cpp
3  // Author : Philipp Holzer / Adam Kensy
4  // Date : 15.10.2019
5  // Description : testdriver for the carpool
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////////
9
10 #include <iostream>
11 #include <string>
12 #include "CarPool/CarPool.h"
13 #include "Vehicles/Truck.h"
14 #include "Vehicles/Car.h"
15 #include "Vehicles/Motorcycle.h"
16 #include "Vehicles/Vehicle.h"
17 #include <vld.h>
18
19 using namespace std;
20
21
22 int main()
23 {
24   string lic_mc_1 = "LL-HAGE1";
25   string brand_mc_1 = "Kawazaki";
26
27   string lic_mc_2 = "LL-BERG2";
28   string brand_mc_2 = "Yamaha";
29
30   string lic_tr_1 = "LL-HARD3";
31   string brand_tr_1 = "Mercedes";
32
33   string lic_tr_2 = "LL-WARE4";
34   string brand_tr_2 = "Koenigsegg";
35
36   string lic_c_1 = "LL-ISSO5";
37   string brand_c_1 = "Opel Corsa (nagelneu)";
38
39   string lic_c_2 = "LL-C000L";
40   string brand_c_2 = "Opel Corsa (verrostet und ohne Klima)";
41
42
43   CarPool PandA_Solutions;
44
45   Motorcycle adams_bike{lic_mc_1, brand_mc_1};
46   Motorcycle phils_bike{lic_mc_2, brand_mc_2};
47
48   Truck adams_truck{lic_tr_1, brand_tr_1};
49   Truck phils_truck{lic_tr_2, brand_tr_2};
50
51   Car adams_car{lic_c_1, brand_c_1};
52   Car phils_car{lic_c_2, brand_c_2};
53
54   tm t1;
55   t1.tm_year = 108;
56   t1.tm_mon = 0;
57   t1.tm_mday = 12;
58
59   tm t2;
60   t2.tm_year = 93;
61   t2.tm_mon = 2;
62   t2.tm_mday = 24;
63
64   tm t3;
65   t3.tm_year = 119;
66   t3.tm_mon = 6;
67   t3.tm_mday = 31;
68
69   tm t4;
```

```
70  t4.tm_year = 119;
71  t4.tm_mon = 11;
72  t4.tm_mday = 4;
73
74  tm t5;
75  t5.tm_year = 119;
76  t5.tm_mon = 8;
77  t5.tm_mday = 17;
78
79  adams_bike.mLogBook.NewEntry(t1, 17);
80  adams_bike.mLogBook.NewEntry(t2, 43);
81  adams_bike.mLogBook.NewEntry(t3, 21);
82  adams_bike.mLogBook.NewEntry(t4, 43);
83  adams_bike.mLogBook.NewEntry(t5, 89);
84  adams_bike.mLogBook.NewEntry(t5, 110);
85  adams_bike.mLogBook.NewEntry(t5, 89);
86
87  cout << "**********************************" << endl;
88  cout << "Testsection LogBook " << endl;
89  cout << "**********************************" << endl;
90
91  adams_bike.mLogBook.PrintLogs(cout);
92  cout << endl;
93
94  cout << "Entry got removed: " << endl;
95  cout << "-----------------------------------" << endl;
96
97  adams_bike.mLogBook.RemoveEntry(t5, 89);
98  adams_bike.mLogBook.PrintLogs(cout);
99  cout << endl;
100
101 cout << " Removing a nonexisting entry: " << endl;
102 cout << "-----------------------------------" << endl;
103
104 adams_bike.mLogBook.RemoveEntry(t5, 1717);
105
106 cout << endl;
107
108 cout << "Print km-sum: " << endl;
109 cout << "-----------------------------------" << endl;
110
111 cout << adams_bike.mLogBook.GetKMSum() << endl;
112 cout << endl;
113
114 cout << "Log cleared (for the next testcases it got filled again): " << endl;
115 cout << "-----------------------------------" << endl;
116 cout << endl;
117
118 adams_bike.mLogBook.NewEntry(t1, 17);
119 adams_bike.mLogBook.NewEntry(t2, 43);
120 adams_bike.mLogBook.NewEntry(t3, 21);
121 adams_bike.mLogBook.NewEntry(t4, 43);
122 adams_bike.mLogBook.NewEntry(t5, 89);
123 adams_bike.mLogBook.NewEntry(t5, 110);
124 adams_bike.mLogBook.NewEntry(t5, 89);
125
126 adams_bike.mLogBook.Clear();
127 adams_bike.mLogBook.PrintLogs(cout);
128
129 phils_bike.mLogBook.NewEntry(t1, 432);
130 phils_bike.mLogBook.NewEntry(t2, 23);
131 phils_bike.mLogBook.NewEntry(t3, 26);
132 phils_bike.mLogBook.NewEntry(t4, 117);
133 phils_bike.mLogBook.NewEntry(t5, 45);
134
135 adams_truck.mLogBook.NewEntry(t1, 67);
136 adams_truck.mLogBook.NewEntry(t2, 98);
137 adams_truck.mLogBook.NewEntry(t3, 45);
138 adams_truck.mLogBook.NewEntry(t4, 34);
139 adams_truck.mLogBook.NewEntry(t5, 21);
140
141 phils_truck.mLogBook.NewEntry(t1, 4567);
142 phils_truck.mLogBook.NewEntry(t2, 82456);
```

```
143   phils_truck.mLogBook.NewEntry(t3, 4332);
144   phils_truck.mLogBook.NewEntry(t4, 6789);
145   phils_truck.mLogBook.NewEntry(t5, 4321);
146
147   adams_car.mLogBook.NewEntry(t1, 231);
148   adams_car.mLogBook.NewEntry(t2, 927);
149   adams_car.mLogBook.NewEntry(t3, 7028);
150   adams_car.mLogBook.NewEntry(t4, 211);
151   adams_car.mLogBook.NewEntry(t5, 6837);
152
153   phils_car.mLogBook.NewEntry(t1, 265);
154   phils_car.mLogBook.NewEntry(t2, 1093);
155   phils_car.mLogBook.NewEntry(t3, 483);
156   phils_car.mLogBook.NewEntry(t4, 46);
157   phils_car.mLogBook.NewEntry(t5, 7392);
158
159   PandA_Solutions.AddVehicle(adams_bike.Clone());
160   PandA_Solutions.AddVehicle(adams_truck.Clone());
161   PandA_Solutions.AddVehicle(adams_car.Clone());
162
163   PandA_Solutions.AddVehicle(phils_bike.Clone());
164   PandA_Solutions.AddVehicle(phils_truck.Clone());
165   PandA_Solutions.AddVehicle(phils_car.Clone());
166
167   cout << "**********************************" << endl;
168   cout << "Testsection CarPool " << endl;
169   cout << "**********************************" << endl;
170
171   cout << "All added vehicles: " << endl;
172   cout << "----------------------------------" << endl;
173   PandA_Solutions.PrintVehicles(cout);
174
175   PandA_Solutions.RemoveVehicle(adams_car.GetLicense());
176
177   cout << "All nullptr to Vehicles: " << endl;
178   cout << "----------------------------------" << endl;
179   PandA_Solutions.AddVehicle(nullptr);
180   cout << endl;
181
182   cout << "Adams Car (LL-ISSO5) got removed: " << endl;
183   cout << "----------------------------------" << endl;
184
185   PandA_Solutions.PrintVehicles(cout);
186
187   cout << "Print LKW with license 'LL-HARD3' (SearchByLicense): " << endl;
188   cout << "----------------------------------" << endl;
189
190   TVehiclePointerItor it;
191   if (PandA_Solutions.SearchByLicense("LL-HARD3", it))
192   {
193     (*it)->Print(cout);
194   }
195   else
196   {
197     cerr << "LKW with the license 'LL-HARD3' does not exist";
198   }
199
200   cout << "Number of vehicles in the car pool: " << endl;
201   cout << PandA_Solutions.GetVehicleAmount() << endl << endl;
202
203   cout << "Print vehicles of copied object (Copy CTOR): " << endl;
204   cout << "----------------------------------" << endl;
205
206   CarPool carpool1{ PandA_Solutions };
207   carpool1.PrintVehicles(cout);
208
209   cout << "Remove vehicle with license 'LL-HARD3' and assign PandA_Solutions to carpool1
        (assignment operator): " << endl;
210   cout << "----------------------------------" << endl;
211   PandA_Solutions.RemoveVehicle("LL-HARD3");
212   carpool1 = PandA_Solutions;
213   carpool1.PrintVehicles(cout);
214
```

```
215   //test self assignment
216   carpool1 = carpool1;
217
218   return 0;
219 }
```