

Name: Simon Offenberger / Simon Vogelhuber

Aufwand in h:

Mat.Nr: S2410306027 / S2410306014

Punkte:

Übungsgruppe: 1

korrigiert:

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, elektrisch 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



HSD

FH-HAGENBERG

Systemdokumentation Projekt Fuhrpark

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 16. Oktober 2025

Inhaltsverzeichnis

| | |
|----------------------------------------------------------|-----------|
| 1 Organisatorisches | 4 |
| 1.1 Team | 4 |
| 1.2 Aufteilung der Verantwortlichkeitsbereiche | 4 |
| 1.3 Aufwand | 5 |
| 2 Anforderungsdefinition (Systemspezifikation) | 6 |
| 3 Systementwurf | 7 |
| 3.1 Klassendiagramm | 7 |
| 3.2 Designentscheidungen | 8 |
| 4 Dokumentation der Komponenten (Klassen) | 8 |
| 5 Testprotokollierung | 9 |
| 6 Quellcode | 13 |
| 6.1 Object.hpp | 13 |
| 6.2 RecordEntry.hpp | 13 |
| 6.3 RecordEntry.cpp | 14 |
| 6.4 DriveRecord.hpp | 15 |
| 6.5 DriveRecord.cpp | 16 |
| 6.6 Garage.hpp | 17 |
| 6.7 Garage.cpp | 18 |
| 6.8 TFuel.hpp | 19 |
| 6.9 Vehicle.hpp | 20 |
| 6.10 Vehicle.cpp | 21 |
| 6.11 Car.hpp | 22 |
| 6.12 Car.cpp | 23 |
| 6.13 Truck.hpp | 24 |
| 6.14 Truck.cpp | 24 |
| 6.15 Bike.hpp | 25 |
| 6.16 Bike.cpp | 26 |

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@fh-hagenberg.at
- Simon Vogelhuber, Matr.-Nr.: S2410306014, E-Mail: s2410306014@students.fh-hagenberg.at

1.2 Aufteilung der Verantwortlichkeitsbereiche

- Simon Offenberger
 - Design Klassendiagramm
 - Implementierung und Test der Klassen:
 - * Object,
 - * RecordEntry,
 - * DriveRecord,
 - * Vehicle,
 - Implementierung des Testtreibers
 - Dokumentation
- Simon Vogelhuber
 - Design Klassendiagramm
 - Implementierung und Komponententest der Klassen:
 - * Garage

- * Car
- * Bike
- * Truck
- Implementierung des Testtreibers
- Dokumentation

1.3 Aufwand

- Simon Offenberger: geschätzt 10 Ph / tatsächlich 8 Ph
- Simon Vogelhuber: geschätzt 10 Ph / tatsächlich 7,5 Ph

2 Anforderungsdefinition (Systemspezifikation)

In diesem System werden Fahrzeuge in einem Fuhrpark verwaltet. Zusätzlich soll auch noch ein Fahrtenbuch zu jedem Fahrzeug gespeichert werden.

Funktionen des Fahrtenbuches

- Berechnen des Kilometerstands der aufgezeichneten Fahrten.
- Speichere Datum und Distanz einer Fahrt.

Funktionen des Fuhrparks

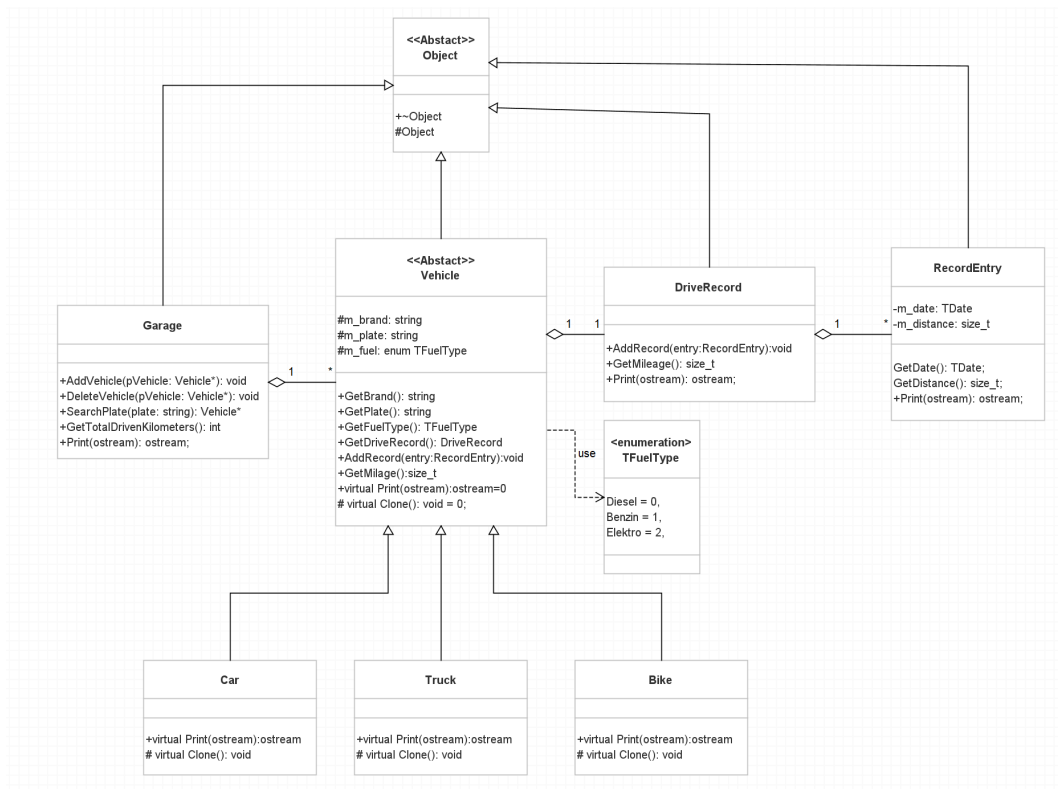
- Verwalten von verschiedenen Fahrzeugarten (Auto, LKW, Motorrad,...).
- Hinzufügen und löschen eines Fahrzeuges
- Ausgabe aller Fahrzeugdaten inklusive der Fahrtenbucheinträge.
- Suchen nach einem Fahrzeug mit dessen Kennzeichen.
- Berechnung der Gesamtkilometer aller Fahrzeuge im Fuhrpark.

Funktionen der Fahrzeuge

- Bereitstellen einer Print Funktion mit Info über das Fahrzeug und die Fahrtenbucheinträge.
- Hinzufügen von Fahrtenbucheinträgen.
- Ermittlung vom Kilometerstand eines Fahrzeugs.
- Speichern von Hersteller, Treibstoff und Kennzeichen des Fahrzeugs

3 Systementwurf

3.1 Klassendiagramm



3.2 Designentscheidungen

Im Klassendiagramm wurde der Polymorphismus angewendet, um unterschiedliche Fahrzeugarten mit der gemeinsamen Schnittstelle 'Vehicle' anzusprechen. Die Klasse 'Garage' speichert einen Container mit der abstrakte Basisklasse 'Vehicle' als Elementtyp und kann somit alle bestehenden und auch neuen Fahrzeugarten verwalten, die sich von der gemeinsamen Basisklasse 'Vehicle' ableiten. Für die Aufzeichnung eines Fahrtenbuches wurde die Klasse **DriveRecord** implementiert. Diese Klasse speichert mehrere Objekte der Klasse **RecordEntry**. Die Record Entries werden im Fahrtenbuch in einem **Multiset** gespeichert, damit sind die Einträge ins Fahrtenbuch immer nach dem Datum aufsteigend sortiert. Aus diesem Grund wurde der **operator<** für die Record Entries definiert. Dieser vergleicht das Datum der Einträge. Dadurch, dass die Einträge ins Fahrtenbuch als eigene Klasse implementiert wurde, lassen sich die einzelnen Einträge schnell und einfach erweitern.

Als Container für die Speicherung der Fahrzeuge in der Klasse **Garage** wurde der Vektor verwendet. Dieser erlaubt es schnell Fahrzeuge hinzuzufügen, und das Suchen geschieht relativ schnell in $O(n)$. Einzig und allein, das Löschen aus der Mitte des Vektors stellt bei größer werdenden Fuhrparks ein Problem dar. Wenn dies in der Verwendung des Fuhrparks öfters passiert sollte der verwendete Container ausgetauscht werden.

Die Klassen **Car**, **Truck** und **Bike** wurden für die Konkretisierung der Printfunktion verwendet. Diese Klassen lassen sich schnell und einfach erweitern, und können trotzdem weiter vom Fuhrpark verwaltet werden.

4 Dokumentation der Komponenten (Klassen)

Die HTML-Startdatei befindet sich im Verzeichnis ../doxy/html/index.html

5 Testprotokollierung

```
1
2 *****
3 TESTCASE START
4 *****
5
6 Test RecordEntry Get Date
7 [Test OK] Result: (Expected: 2025-10-13 == Result: 2025-10-13)
8
9 Test RecordEntry Get Distance
10 [Test OK] Result: (Expected: 150 == Result: 150)
11
12 Test RecordEntry Print
13 [Test OK] Result: (Expected: true == Result: true)
14
15 Test RecordEntry Exception Bad Ostream
16 [Test OK] Result: (Expected: ERROR: Provided Ostream is bad == Result:
    ↪ ERROR: Provided Ostream is bad)
17
18 Test RecordEntry less than operator
19 [Test OK] Result: (Expected: true == Result: true)
20
21 Test RecordEntry Exceotion Distance = 0
22 [Test OK] Result: (Expected: ERROR: Distance cannot be zero! == Result:
    ↪ ERROR: Distance cannot be zero!)
23
24
25 *****
26
27
28 *****
29 TESTCASE START
30 *****
31
32 Test DriveRecord Print Sorted and Add Record
33 [Test OK] Result: (Expected: true == Result: true)
34
35 Test DriveRecord Get Milage
36 [Test OK] Result: (Expected: 450 == Result: 450)
37
38 Test DriveRecord Exception Bad Ostream
39 [Test OK] Result: (Expected: ERROR: Provided Ostream is bad == Result:
    ↪ ERROR: Provided Ostream is bad)
```

```
40
41 Test DriveRecord Empty Print
42 [Test OK] Result: (Expected: No Exception == Result: No Exception)
43
44
45 *****
46
47
48 *****
49 TESTCASE START
50 *****
51
52 vehicle plate search
53 [Test OK] Result: (Expected: 0000028418CD2560 == Result: 0000028418CD2560)
54
55 Test garage plate search - error buffer
56 [Test OK] Result: (Expected: true == Result: true)
57
58 Test garage plate search invalid plate
59 [Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)
60
61 Test garage plate search invalid plate - error buffer
62 [Test OK] Result: (Expected: true == Result: true)
63
64 Test Garage Print
65 [Test OK] Result: (Expected:
66 Fahrzeugart: PKW
67 Marke: UAZ
68 Kennzeichen: SR770BA
69 13.10.2025: 25 km
70 == Result:
71 Fahrzeugart: PKW
72 Marke: UAZ
73 Kennzeichen: SR770BA
74 13.10.2025: 25 km
75 )
76
77 Test garage print - error buffer
78 [Test OK] Result: (Expected: true == Result: true)
79
80 Test garage print empty garage
81 [Test OK] Result: (Expected: true == Result: true)
82
83 Test garage print empty garage - error buffer
```

```
84 [Test OK] Result: (Expected: true == Result: true)
85
86 Test Delete Vehicle
87 [Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)
88
89 Test garage print - error buffer
90 [Test OK] Result: (Expected: true == Result: true)
91
92 Test Delete Vehicle
93 [Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)
94
95 Test Delete Vehicle - error buffer
96 [Test OK] Result: (Expected: true == Result: true)
97
98 Test GetTotalDrivenKilometers()
99 [Test OK] Result: (Expected: 118 == Result: 118)
100
101 Test GetTotalDrivenKilometers() - error buffer
102 [Test OK] Result: (Expected: true == Result: true)
103
104 Test ostream operator
105 [Test OK] Result: (Expected:
106 Fahrzeugart: PKW
107 Marke: Madza
108 Kennzeichen: WD40AHAAH
109 13.10.2025: 25 km
110 28.10.2025: 34 km
111 == Result:
112 Fahrzeugart: PKW
113 Marke: Madza
114 Kennzeichen: WD40AHAAH
115 13.10.2025: 25 km
116 28.10.2025: 34 km
117 )
118
119 Test ostream operator - error buffer
120 [Test OK] Result: (Expected: true == Result: true)
121
122 TestAdding Car as nullptr;
123 [Test OK] Result: (Expected: ERROR: Passed in Nullptr! == Result: ERROR:
    ↪ Passed in Nullptr!)
124
125 TestDeleting Car as nullptr;
```

```
126 [Test OK] Result: (Expected: ERROR: Passed in Nullptr! == Result: ERROR:
    ↳ Passed in Nullptr!)
127
128
129 *****
130
131
132 *****
133         TESTCASE START
134 *****
135
136 Test Car Print without record
137 [Test OK] Result: (Expected: true == Result: true)
138
139 TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```
1  /*****  
2  * \file   Object.hpp  
3  * \brief  Root of all Objects  
4  *  
5  * \author Simon Offenberger  
6  * \date   October 2025  
7  *****/  
8  #ifndef OBJECT_HPP  
9  #define OBJECT_HPP  
10  
11 #include <iostream>  
12  
13 class Object {  
14 public:  
15     inline static const std::string ERROR_BAD_OSTREAM = "ERROR:_Provided_Ostream_is_bad";  
16     inline static const std::string ERROR_FAIL_WRITE = "ERROR:_Fail_to_write_on_provided_Ostream";  
17  
18     /**  
19      * Virtual DTOR, once virtual always virtual.  
20      */  
21     virtual ~Object() = default;  
22  
23 protected:  
24     /**  
25      * \brief protected CTOR -> abstract.  
26      */  
27     Object() = default;  
28 };  
29  
30 #endif // !1
```

6.2 RecordEntry.hpp

```
1  /*****  
2  * \file   RecordEntry.hpp  
3  * \brief  Class that defines an entry in a dirve record.  
4  * \brief  This record entry is used by the drive record class.  
5  * \brief  The drive record class stores multiple record entries.  
6  *  
7  * \author Simon Offenberger  
8  * \date   October 2025  
9  *****/  
10 #ifndef RECORD_ENTRY_HPP  
11 #define RECORD_ENTRY_HPP  
12  
13  
14 #include <chrono>  
15 #include "Object.hpp"  
16  
17 // Using Statement for date type  
18 using TDate = std::chrono::year_month_day;  
19  
20 class RecordEntry : public Object {  
21 public:  
22     inline static const std::string ERROR_DISTANCE_ZERO = "ERROR:_Distance_cannot_be_zero!";  
23  
24     /**  
25
```

```

26     * \brief CTOR of a drive record.
27     *
28     * \param date : date when the drive happend
29     * \param distance : the distance of the drive in km
30     */
31     RecordEntry(const TDate& date, const size_t& distance);
32
33     /**
34     * \brief Getter of the distance member of the Record Entry Class.
35     *
36     * \return Distance of this Record Entry
37     */
38     size_t GetDistance() const;
39
40     /**
41     * \brief Getter of the data member of the Record Entry Class.
42     *
43     * \return Date of this Record Entry
44     */
45     TDate GetDate() const;
46
47     /**
48     * \brief Formatted output of this Record Entry on an ostream.
49     *
50     * \param ost : Reference to an ostream where the Entry should be printed at.
51     * \return Referenced ostream
52     */
53     virtual std::ostream& Print(std::ostream& ost = std::cout) const;
54
55     /**
56     * \brief less than operator, is used for storing the Entries in a multiset.
57     *
58     * \param rh : Righthandside of the less than operator
59     * \return true: left hand side is less than the right hand side.
60     * \return false: left hand side is greather or equal than the right hand side.
61     */
62     bool operator<(const RecordEntry& rh) const;
63
64 private:
65     TDate m_date; // private date member
66     size_t m_distance; // private distance member
67 };
68
69
70 #endif // !1

```

6.3 RecordEntry.cpp

```

1  /*****
2  * \file   RecordEntry.cpp
3  * \brief  Implementation of a Record Entry
4  *
5  * \author Simon Offenberger
6  * \date   October 2025
7  *****/
8  #include "RecordEntry.hpp"
9
10 using namespace std;
11
12 RecordEntry::RecordEntry(const TDate& date, const size_t& distance) : m_date(date)
13 {
14     if (distance == 0) throw RecordEntry::ERROR_DISTANCE_ZERO;
15     m_distance = distance;
16 }
17
18 /**
19 * \brief Getter of the distance member of the Record Entry Class.
20 *

```

```

21 * \return Distance of this Record Entry
22 */
23 size_t RecordEntry::GetDistance() const
24 {
25     return m_distance;
26 }
27
28 /**
29 * \brief Getter of the data member of the Record Entry Class.
30 *
31 * \return Date of this Record Entry
32 */
33 TDate RecordEntry::GetDate() const
34 {
35     return m_date;
36 }
37
38 /**
39 * \brief Formatted output of this Record Entry on an ostream.
40 *
41 * \param ost : Refernce to an ostream where the Entry should be printed at.
42 * \return Referenced ostream
43 */
44 std::ostream& RecordEntry::Print(std::ostream& ost) const
45 {
46     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;
47
48     ost << std::setfill('0') << right << std::setw(2) << m_date.day() << "."
49         << std::setw(2) << static_cast<unsigned>(m_date.month()) << "."
50         << std::setw(4) << m_date.year() << "; " << std::setfill(' ')
51         << std::setw(6) << m_distance << "_km\n";
52
53     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;
54
55     return ost;
56 }
57
58 /**
59 * \brief less than operator, is used for storing the Entries in a multiset.
60 *
61 * \param rh : Righthandside of the less than operator
62 * \return true: left hand side is less than the right hand side.
63 * \return false: left hand side is greather or equal than the right hand side.
64 */
65 bool RecordEntry::operator<(const RecordEntry& rh) const
66 {
67     return m_date < rh.m_date;
68 }

```

6.4 DriveRecord.hpp

```

1  /*****
2  * \file   DriveRecord.hpp
3  * \brief  This Class implements a drive record book which holds multiple
4  * \brief  record entries in a TCont, which is defined as a multiset.
5  * \brief  The multiset is used because it stores the data sorted.
6  * \brief  This sorting mandatory because the entries should be date ascending.
7  *
8  * \author Simon Offenberger
9  * \date   October 2025
10 *****/
11 #ifndef DRIVE_RECORD_HPP
12 #define DRIVE_RECORD_HPP
13
14 #include <set>
15 #include "RecordEntry.hpp"
16 #include "Object.hpp"
17

```

```
18 // Using statement for the used container to store the record entries
19 using TCont = std::multiset<RecordEntry>;
20
21 class DriveRecord : public Object {
22 public:
23
24     /**
25      * \brief Methode for adding a record entry to a collection of drive records.
26      *
27      * \param entry : Record to be added to the collection
28      */
29     void AddRecord(const RecordEntry & entry);
30
31     /**
32      * \brief This methode adds up all the distance of all record entries.
33      *
34      * \return the sum of all distances in the collection
35      */
36     size_t GetMilage() const;
37
38     /**
39      * \brief Formatted output of all Record Entry on an ostream.
40      *
41      * \param ost : Reference to an ostream where the Entries should be printed at.
42      * \return Referenced ostream
43      */
44     virtual std::ostream& Print(std::ostream& ost = std::cout) const;
45
46 private:
47     TCont m_driveRecords;
48 };
49
50
51
52 #endif // !1
```

6.5 DriveRecord.cpp

```
1  /**
2   * \file   DriveRecord.cpp
3   * \brief  Implementation of a Drive Record
4   *
5   * \author Simon Offenberger
6   * \date   October 2025
7   */
8  #include <numeric>
9  #include <algorithm>
10 #include "DriveRecord.hpp"
11
12 /**
13 * \brief Methode for adding a record entry to a collection of drive records.
14 *
15 * \param entry : Record to be added to the collection
16 */
17 void DriveRecord::AddRecord(const RecordEntry& entry)
18 {
19     m_driveRecords.insert(entry);
20 }
21
22 /**
23 * \brief This methode adds up all the distance of all record entries.
24 *
25 * \return the sum of all distances in the collection
26 */
27 size_t DriveRecord::GetMilage() const
28 {
29     // use std::accumulate + lambda to calc the total Milage
30     return std::accumulate(m_driveRecords.cbegin(), m_driveRecords.cend(), static_cast<size_t>(0),
```



```

31         [(const size_t val, const RecordEntry& entry) {return val + entry.GetDistance();});
32     }
33
34     /**
35     * \brief Formatted output of all Record Entry on an ostream.
36     *
37     * \param ost : Refernce to an ostream where the Entries should be printed at.
38     * \return Referenced ostream
39     */
40     std::ostream& DriveRecord::Print(std::ostream& ost) const
41     {
42         if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;
43
44         std::for_each(m_driveRecords.cbegin(), m_driveRecords.cend(), [&](const RecordEntry& entry) {entry.Print(ost);});
45
46         if (ost.fail()) throw Object::ERROR_FAIL_WRITE;
47
48         return ost;
49     }

```

6.6 Garage.hpp

```

1  /*****
2  * \file   Vehicle.hpp
3  * \brief  This Class implements a polymorph container containing
4  * \brief  all derivatives of the 'Vehicle' Class.
5  * \author Simon Vogelhuber
6  * \date   October 2025
7  *****/
8  #ifndef GARAGE_HPP
9  #define GARAGE_HPP
10
11  #include <vector>
12  #include <string>
13  #include "Object.hpp"
14  #include "Vehicle.hpp"
15
16  using TGarageCont = std::vector<Vehicle const*>;
17
18  class Garage : public Object {
19  public:
20
21      inline static const std::string ERROR_NULLPTR= "ERROR:_Passed_in_Nullptr!";
22
23      Garage() = default;
24
25      /**
26       * \brief Adds a vehicle to a vehicle collection.
27       * \brief A specific vehicle is passed in and casted to a vehicle Pointer.
28       * \brief This is allowed because Car, Truck and Bike are derived from Vehicle.
29       * \brief A car is a Vehicle.
30       * \brief This casted Pointer is copied into this methode and added to the collection
31       * \param newVehicle : Pointer to a Vehicle.
32       */
33      void AddVehicle(Vehicle const* const newVehicle);
34
35      /**
36       * \brief deletes Vehicle inside garage from provided pointer.
37       * \param pVehicle : Pointer to a Vehicle.
38       */
39      void DeleteVehicle(Vehicle const* const pVehicle);
40
41      /**
42       * \brief Functions searches for vehicle with matching plate.
43       * \param pVehicle : Pointer to a Vehicle.
44       * \return pointer to the vehicle inside the garage
45       */
46      Vehicle const* const SearchPlate(const std::string & plate) const;

```

```
47
48  /**
49   * \brief Formatted of every car and its drive record
50   * \param ost : Refernce to an ostream where the Entry should be printed at.
51   * \return Referenced ostream
52   */
53  std::ostream& Print(std::ostream& ost = std::cout) const;
54
55  /**
56   * \brief Calculates sum of every kilometer every vehicle has driven
57   * \brief in total
58   * \return size_t total kilometers
59   */
60  size_t GetTotalDrivenKilometers() const;
61
62  Garage(const Garage&);
63  void operator=(Garage garage);
64
65  ~Garage();
66
67 private:
68  TGarageCont m_vehicles;
69 };
70
71 /**
72  * \brief Override for ostream operator
73  * \return ostream
74  */
75  std::ostream& operator <<(std::ostream& ost, Garage& garage);
76
77 #endif
```

6.7 Garage.cpp

```
1  /**
2   * \file   Vehicle.c
3   * \brief  Implementation of Garage.h
4   * \author Simon Vogelhuber
5   * \date   October 2025
6   *****/
7  #include "Garage.hpp"
8  #include <algorithm>
9  #include <numeric>
10
11 void Garage::AddVehicle(Vehicle const * const newVehicle)
12 {
13     if (newVehicle == nullptr) throw ERROR_NULLPTR;
14     // Add the new vehicle to the collection.
15     m_vehicles.push_back(newVehicle);
16 }
17
18 /**
19  * \brief deletes Vehicle inside garage from provided pointer.
20  * \param pVehicle : Pointer to a Vehicle.
21  */
22 void Garage::DeleteVehicle(Vehicle const * const pVehicle)
23 {
24     if (pVehicle == nullptr) throw ERROR_NULLPTR;
25
26     // if pVehicle is inside m_Vehicles -> erase and free
27     auto itr = std::find(m_vehicles.begin(), m_vehicles.end(), pVehicle);
28     if (itr != m_vehicles.end())
29     {
30         m_vehicles.erase(itr);
31         delete pVehicle;
32     }
33 }
34
```

```
35 const Vehicle* const Garage::SearchPlate(const std::string & plate) const
36 {
37     for (const auto &elem : m_vehicles)
38     {
39         if (elem->GetPlate() == plate)
40         {
41             return elem;
42         }
43     }
44     return nullptr;
45 }
46
47
48 std::ostream& Garage::Print(std::ostream& ost) const
49 {
50     if (!ost.good())
51         throw Object::ERROR_BAD_OSTREAM;
52
53     for (auto& elem : m_vehicles)
54     {
55         elem->Print(ost);
56     }
57
58     if (ost.fail())
59         throw Object::ERROR_FAIL_WRITE;
60
61     return ost;
62 }
63
64 size_t Garage::GetTotalDrivenKilometers() const
65 {
66     size_t sum = std::accumulate(m_vehicles.cbegin(), m_vehicles.cend(), static_cast<size_t>(0),
67     [](auto last_val, auto vehicle) {
68         return last_val + vehicle->GetMilage();
69     });
70     return sum;
71 }
72
73 Garage::Garage(const Garage&)
74 {
75     for_each(
76         m_vehicles.cbegin(), m_vehicles.cend(),
77         [&](auto v) {AddVehicle(v->Clone());});
78 }
79
80
81 void Garage::operator=(Garage garage)
82 {
83     std::swap(m_vehicles, garage.m_vehicles);
84 }
85
86 Garage::~Garage()
87 {
88     for (auto elem : m_vehicles)
89     {
90         delete elem;
91     }
92     m_vehicles.clear();
93 }
94
95
96 std::ostream& operator<<(std::ostream& ost, Garage& garage)
97 {
98     garage.Print(ost);
99     return ost;
100 }
```

6.8 TFuel.hpp

```
1  /*****  
2  * \file    TFuel.hpp  
3  * \brief   This Enum provides a specifier for the fuel type  
4  *  
5  * \author  Simon Offenberger  
6  * \date    October 2025  
7  *****/  
8  #ifndef TFUEL_HPP  
9  #define TFUEL_HPP  
10  
11 // Enumeration for a fuel type  
12 enum TFuel {  
13     Diesel = 0,  
14     Benzin = 1,  
15     Elektro = 2,  
16 };  
17  
18 #endif // !1
```

6.9 Vehicle.hpp

```
1  /*****  
2  * \file    Vehicle.hpp  
3  * \brief   This class implements an abstract vehicle which is used in the  
4  * \brief   Garage class. It implements all the core features of a vehicle  
5  *  
6  * \author  Simon Offenberger  
7  * \date    October 2025  
8  *****/  
9  #ifndef VEHICLE_HPP  
10 #define VEHICLE_HPP  
11  
12 #include "Object.hpp"  
13 #include "DriveRecord.hpp"  
14 #include "TFuel.hpp"  
15  
16 class Vehicle: public Object {  
17 public:  
18  
19     /**  
20      * \brief Getter for the brand member.  
21      *  
22      * \return string with the brand name  
23      */  
24     std::string GetBrand() const;  
25  
26     /**  
27      * \brief Getter for the plate member.  
28      *  
29      * \return string with the plate name  
30      */  
31     std::string GetPlate() const;  
32  
33     /**  
34      * \brief Getter for the fuel member.  
35      *  
36      * \return TFuel with the specified fuel type  
37      */  
38     TFuel GetFuelType() const;  
39  
40     /**  
41      * \brief Getter for the drive record.  
42      *  
43      * \return const reference to the drive record  
44      */  
45     const DriveRecord & GetDriveRecord() const;  
46  
47     /**
```

```

48     * \brief Methode for adding a record entry to the drive record collection.
49     *
50     * \param entry : Entry which should be added to the drive record
51     */
52     void AddRecord(const RecordEntry& entry);
53
54     /**
55     * \brief Getter for the total milage of a vehicle.
56     *
57     * \return Total milage of a vehicle
58     */
59     size_t GetMilage() const;
60
61     /**
62     * @brief Creates a clone of the vehicle.
63     *
64     * \return a exact replicate of a vehicle
65     */
66     virtual Vehicle const* Clone() const = 0;
67
68
69     /**
70     * \brief Print function that is implementet by dirved Classes.
71     *
72     * \param ost Reference to an ostream where the Result should be printed at
73     * \return referenced ostream
74     */
75     virtual std::ostream& Print(std::ostream& ost = std::cout) const = 0;
76
77
78 protected:
79
80     /**
81     * \brief protected CTOR of a vehicle.
82     * \brief protected because it is a abstract class
83     *
84     * \param brand : string that represents the brand of the vehicle
85     * \param fuelType : Fuel type of the vehicle
86     */
87     Vehicle(const std::string& brand, const TFuel& fuelType, const std::string& plate) : m_brand{ brand }, m_fuel{ fuelType }, m_plate{ plate } {}
88
89 private:
90     std::string m_brand;
91     std::string m_plate;
92     TFuel m_fuel;
93     DriveRecord m_record;
94 };
95
96
97 #endif // !1

```

6.10 Vehicle.cpp

```

1  /*****
2  * \file   Vehicle.cpp
3  * \brief  Implementation of the abstract vehicle class
4  *
5  * \author  Simon Offenberger
6  * \date   October 2025
7  *****/
8  #include "Vehicle.hpp"
9
10 /**
11 * \brief Getter for the brand member.
12 *
13 * \return string with the brand name
14 */
15 std::string Vehicle::GetBrand() const

```

```
16 {
17     return m_brand;
18 }
19
20 /**
21  * \brief Getter for the plate member.
22  *
23  * \return string with the plate name
24  */
25 std::string Vehicle::GetPlate() const
26 {
27     return m_plate;
28 }
29
30 /**
31  * \brief Getter for the fuel member.
32  *
33  * \return TFuel with the specified fuel type
34  */
35 TFuel Vehicle::GetFuelType() const
36 {
37     return m_fuel;
38 }
39
40 /**
41  * \brief Getter for the drive record.
42  *
43  * \return const reference to the drive record
44  */
45 const DriveRecord & Vehicle::GetDriveRecord() const
46 {
47     return m_record;
48 }
49
50 /**
51  * \brief Methode for adding a record entry to the drive record collection.
52  *
53  * \param entry : Entry which should be added to the drive record
54  */
55 void Vehicle::AddRecord(const RecordEntry& entry)
56 {
57     m_record.AddRecord(entry);
58 }
59
60 /**
61  * \brief Getter for the total milage of a vehicle.
62  *
63  * \return Total milage of a vehicle
64  */
65 size_t Vehicle::GetMilage() const
66 {
67     return m_record.GetMilage();
68 }
```

6.11 Car.hpp

```
1  /**
2   * \file    Car.hpp
3   * \brief   Header fo the specific Class Car
4   *
5   * \author  Simon
6   * \date   October 2025
7   */
8  #ifndef CAR_HPP
9  #define CAR_HPP
10
11  #include "Vehicle.hpp"
12
```

```
13 class Car : public Vehicle {
14 public:
15
16     /**
17      * \brief CTOR of a CAR -> calles the Base Class vehicle CTOR.
18      *
19      * \param brand string that identifies the brand.
20      * \param fuelType Fueltype of the Car
21      * \param plate string that identifies the plate.
22      */
23     Car(const std::string & brand, const TFuel & fuelType, const std::string & plate) : Vehicle(brand, fuelType, plate) {}
24
25     /**
26      * \brief Function that print all the vehicle specific info with the drive record.
27      *
28      * \param ost where the data should be printed at
29      * \return referenced ostream
30      */
31     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;
32
33     /**
34      * @brief Creates a clone of the vehicle.
35      *
36      * \return a excat replicate of a vehicle
37      */
38     virtual Vehicle const* Clone() const;
39
40 private:
41 };
42
43
44 #endif // !1
```

6.12 Car.cpp

```
1  /**
2   * \file Car.cpp
3   * \brief Implemetation of a Car
4   *
5   * \author Simon
6   * \date October 2025
7   */
8  #include "Car.hpp"
9
10 using namespace std;
11
12 /**
13  * \brief Function that print all the vehicle specific info with the drive record.
14  *
15  * \param ost where the data should be printed at
16  * \return referenced ostream
17  */
18  std::ostream& Car::Print(std::ostream& ost) const
19  {
20      if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;
21
22      ost << endl << left << setw(14) << "Fahrzeugart:" << "PKW" << endl;
23      ost << left << setw(14) << "Marke:" << GetBrand() << endl;
24      ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;
25      GetDriveRecord().Print(ost);
26
27      if (ost.fail()) throw Object::ERROR_FAIL_WRITE;
28
29      return ost;
30  }
31
32 /**
33  * @brief Creates a clone of the vehicle.
```

```
34 *  
35 * \return a excat replicate of a vehicle  
36 */  
37 Vehicle const* Car::Clone() const  
38 {  
39     return new Car(*this);  
40 }
```

6.13 Truck.hpp

```
1  /*****  
2  * \file   Truck.hpp  
3  * \brief  Header fo the specific Class Truck  
4  *  
5  * \author Simon  
6  * \date   October 2025  
7  *****/  
8  #ifndef TRUCK_HPP  
9  #define TRUCK_HPP  
10  
11 #include "Vehicle.hpp"  
12  
13 class Truck : public Vehicle {  
14 public:  
15  
16     /**  
17      * \brief CTOR of a Truck -> calles the Base Class vehicle CTOR.  
18      *  
19      * \param brand string that identifies the brand.  
20      * \param fuelType Fueltype of the Truck  
21      * \param plate string that identifies the plate.  
22      */  
23     Truck(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}  
24  
25     /**  
26      * \brief Function that print all the vehicle specific info with the drive record.  
27      *  
28      * \param ost where the data should be printed at  
29      * \return referenced ostream  
30      */  
31     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;  
32  
33     /**  
34      * @brief Creates a clone of the vehicle.  
35      *  
36      * \return a excat replicate of a vehicle  
37      */  
38     virtual Vehicle const* Clone() const;  
39  
40 private:  
41 };  
42  
43  
44 #endif
```

6.14 Truck.cpp

```
1  /*****  
2  * \file   Truck.cpp  
3  * \brief  Implementation of a Truck  
4  *  
5  *****/
```



```

5  * \author Simon
6  * \date   October 2025
7  *****/
8  #include "Truck.hpp"
9
10 using namespace std;
11
12 /**
13  * \brief Function that print all the vehicle specific info with the drive record.
14  *
15  * \param ost where the data should be printed at
16  * \return referenced ostream
17  */
18 std::ostream& Truck::Print(std::ostream& ost) const
19 {
20     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;
21
22     ost << endl << left << setw(14) << "Fahrzeugart:" << "LKW" << endl;
23     ost << left << setw(14) << "Marke:" << GetBrand() << endl;
24     ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;
25     GetDriveRecord().Print(ost);
26
27     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;
28
29     return ost;
30 }
31
32 /**
33  * @brief Creates a clone of the vehicle.
34  *
35  * \return a exact replicate of a vehicle
36  */
37 Vehicle const* Truck::Clone() const
38 {
39     return new Truck(*this);
40 }

```

6.15 Bike.hpp

```

1  /**
2  * \file   Bike.hpp
3  * \brief  Header fo the specific Class Bike
4  *
5  * \author Simon
6  * \date   October 2025
7  *****/
8  #ifndef BIKE_HPP
9  #define BIKE_HPP
10
11 #include "Vehicle.hpp"
12
13 class Bike : public Vehicle {
14 public:
15
16     /**
17     * \brief CTOR of a Bike -> calles the Base Class vehicle CTOR.
18     *
19     * \param brand string that identifies the brand.
20     * \param fuelType Fueltype of the Bike
21     * \param plate string that identifies the plate.
22     */
23     Bike(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}
24
25     /**
26     * \brief Function that print all the vehicle specific info with the drive record.
27     *
28     * \param ost where the data should be printed at
29     * \return referenced ostream

```

```
30     */
31     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;
32
33     /**
34     * @brief Creates a clone of the vehicle.
35     *
36     * \return a exact replicate of a vehicle
37     */
38     virtual Vehicle const* Clone() const;
39
40 private:
41 };
42
43
44 #endif
```

6.16 Bike.cpp

```
1  /*****
2  * \file   Bike.cpp
3  * \brief  Implementation of the Bike Class
4  *
5  * \author Simon
6  * \date   October 2025
7  *****/
8  #include "Bike.hpp"
9
10 using namespace std;
11
12 /**
13 * \brief Function that print all the vehicle specific info with the drive record.
14 *
15 * \param ost where the data should be printed at
16 * \return referenced ostream
17 */
18 std::ostream& Bike::Print(std::ostream& ost) const
19 {
20     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;
21
22     ost << endl << left << setw(14) << "Fahrzeugart:" << "Motorrad" << endl;
23     ost << left << setw(14) << "Marke:" << GetBrand() << endl;
24     ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;
25     GetDriveRecord().Print(ost);
26
27     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;
28
29     return ost;
30 }
31
32 /**
33 * @brief Creates a clone of the vehicle.
34 *
35 * \return a exact replicate of a vehicle
36 */
37 Vehicle const* Bike::Clone() const
38 {
39     return new Bike(*this);
40 }
```