



HSD

FH-HAGENBERG

Systemdokumentation Projekt Fuhrpark

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 15. Oktober 2025

Inhaltsverzeichnis

1 Organisatorisches	3
1.1 Team	3
1.2 Aufteilung der Verantwortlichkeitsbereiche	3
1.3 Aufwand	4
2 Anforderungsdefinition (Systemspezifikation)	5
3 Systementwurf	6
3.1 Klassendiagramm	6
3.2 Designentscheidungen	7
4 Dokumentation der Komponenten (Klassen)	7
5 Testprotokollierung	8
6 Quellcode	11
6.1 Object.hpp	11
6.2 RecordEntry.hpp	11
6.3 RecordEntry.cpp	12
6.4 DriveRecord.hpp	13
6.5 DriveRecord.cpp	14
6.6 Garage.hpp	15
6.7 Garage.cpp	16
6.8 Vehicle.hpp	17
6.9 Vehicle.cpp	19
6.10 Car.hpp	20
6.11 Car.cpp	20
6.12 Truck.hpp	21
6.13 Truck.cpp	22
6.14 Bike.hpp	22
6.15 Bike.cpp	23

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@fh-hagenberg.at
- Susi Sorglos, Matr.-Nr.: yyyy, E-Mail: Susi.Sorglos@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 und
- * Truck
- Implementierung des Testtreibers
- Dokumentation

1.3 Aufwand

- Simon Offenberger: geschätzt 10 Ph / tatsächlich x Ph
- Simon Vogelhuber: geschätzt x Ph / tatsächlich x 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

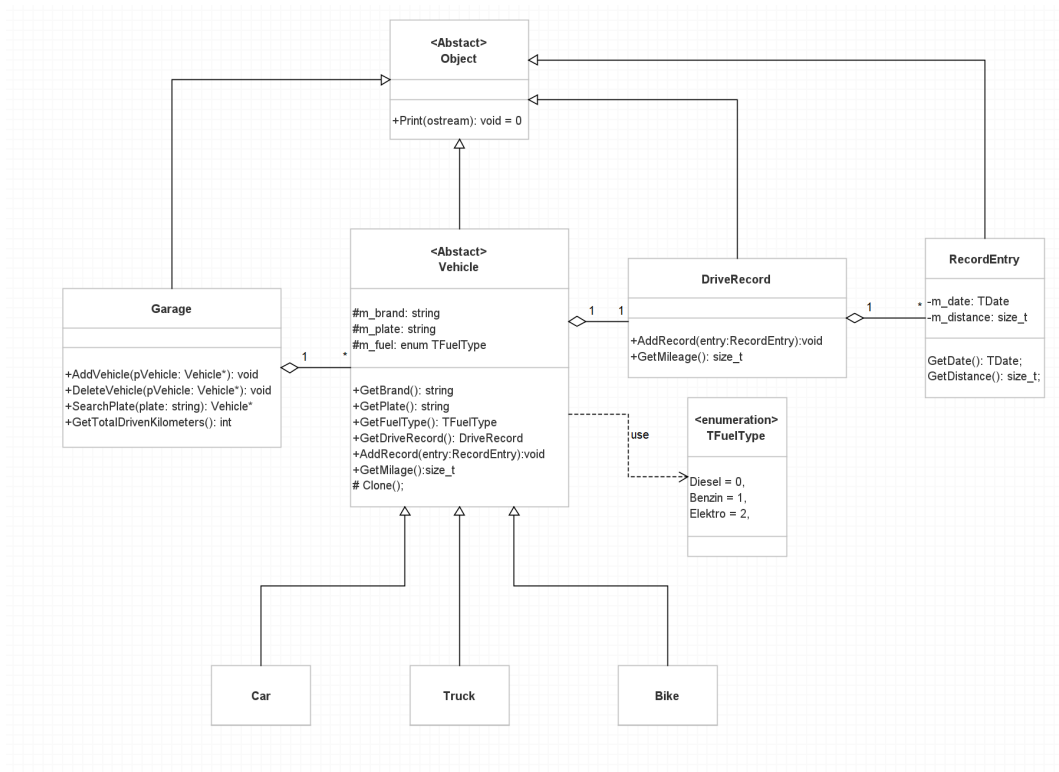
- Hinzufügen und löschen eines Fahrzeuges
- Ausgabe aller Fahrzeugdaten inklusive der Fahrtenbucheinträge.
- Suchen nach einem Fahrzeug mit dessen Kennzeichen.
- Berechnung der Gesamtkilometer aller Fahrzeuge im Fuhrpark.

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
19 *****
20
21
22 *****
23 TESTCASE START
24 *****
25
26 Test DriveRecord Print Sorted and Add Record
27 [Test OK] Result: (Expected: true == Result: true)
28
29 Test DriveRecord Get Milage
30 [Test OK] Result: (Expected: 450 == Result: 450)
31
32 Test DriveRecord Exception Bad Ostream
33 [Test OK] Result: (Expected: ERROR: Provided Ostream is bad == Result:
    ↪ ERROR: Provided Ostream is bad)
34
35 Test DriveRecord Empty Print
36 [Test OK] Result: (Expected: No Exception == Result: No Exception)
37
38
39 *****
40
```



```
41
42 *****
43 TESTCASE START
44 *****
45
46 vehicle plate search
47 [Test OK] Result: (Expected: 000001FBA98DFCC0 == Result: 000001FBA98DFCC0)
48
49 Test garage plate search - error buffer
50 [Test OK] Result: (Expected: true == Result: true)
51
52 Test garage plate search invalid plate
53 [Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)
54
55 Test garage plate search invalid plate - error buffer
56 [Test OK] Result: (Expected: true == Result: true)
57
58 Test Garage Print
59 [Test OK] Result: (Expected:
60 Fahrzeugart: PKW
61 Marke: UAZ
62 Kennzeichen: SR770BA
63 13.10.2025: 25 km
64 == Result:
65 Fahrzeugart: PKW
66 Marke: UAZ
67 Kennzeichen: SR770BA
68 13.10.2025: 25 km
69 )
70
71 Test garage print - error buffer
72 [Test OK] Result: (Expected: true == Result: true)
73
74 Test garage print empty garage
75 [Test OK] Result: (Expected: true == Result: true)
76
77 Test garage print empty garage - error buffer
78 [Test OK] Result: (Expected: true == Result: true)
79
80 Test Delete Vehicle
81 [Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)
82
83 Test garage print - 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 Delete Vehicle - error buffer
90 [Test OK] Result: (Expected: true == Result: true)
91
92 TTest GetTotalDrivenKilometers()
93 [Test OK] Result: (Expected: 118 == Result: 118)
94
95 Test GetTotalDrivenKilometers() - error buffer
96 [Test OK] Result: (Expected: true == Result: true)
97
98 Test ostream operator
99 [Test OK] Result: (Expected:
100 Fahrzeugart: Auto
101 Marke: Madza
102 Kennzeichen: WD40AHAH
103 13.10.2025: 25 km
104 28.10.2025: 34 km
105 == Result:
106 Fahrzeugart: Auto
107 Marke: Madza
108 Kennzeichen: WD40AHAH
109 13.10.2025: 25 km
110 28.10.2025: 34 km
111 )
112
113 Test ostream operator - error buffer
114 [Test OK] Result: (Expected: true == Result: true)
115
116
117 *****
118
119
120 *****
121 TESTCASE START
122 *****
123
124 Test Car Print without record
125 [Test OK] Result: (Expected: true == Result: true)
126
127 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     /**
24      * .
25      */
26     virtual std::ostream& Print(std::ostream & ost = std::cout) const = 0;
27
28 protected:
29     Object() = default;
30 };
31
32
33 #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
23     /**
```

```
24     * \brief CTOR of a drive record.
25     *
26     * \param date : date when the drive happend
27     * \param distance : the distance of the drive in km
28     */
29     RecordEntry(const TDate & date, const size_t & distance) : m_date{ date }, m_distance{ distance } {}
30
31     /**
32     * \brief Getter of the distance member of the Record Entry Class.
33     *
34     * \return Distance of this Record Entry
35     */
36     size_t GetDistance() const;
37
38     /**
39     * \brief Getter of the data member of the Record Entry Class.
40     *
41     * \return Date of this Record Entry
42     */
43     TDate GetDate() const;
44
45     /**
46     * \brief Formatted output of this Record Entry on an ostream.
47     *
48     * \param ost : Reference to an ostream where the Entry should be printed at.
49     * \return Referenced ostream
50     */
51     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;
52
53     /**
54     * \brief less than operator, is used for storing the Entries in a multiset.
55     *
56     * \param rh : Righthandside of the less than operator
57     * \return true: left hand side is less than the right hand side.
58     * \return false: left hand side is greather or equal than the right hand side.
59     */
60     bool operator<(const RecordEntry& rh) const;
61
62 private:
63     TDate m_date; // private date member
64     size_t m_distance; // private distance member
65 };
66
67
68 #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 /**
13 * \brief Getter of the distance member of the Record Entry Class.
14 *
15 * \return Distance of this Record Entry
16 */
17 size_t RecordEntry::GetDistance() const
18 {
19     return m_distance;
20 }
```

```

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

```

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 override;
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  #include <vector>
9  #include <string>
10 #include "Object.hpp"
11 #include "Vehicle.hpp"
12
13 using TGarageCont = std::vector<Vehicle const *>;
14
15 class Garage : public Object {
16 public:
17
18     inline static const std::string ERROR_NULLPTR= "ERROR:_Passed_in_Nullptr!";
19
20     Garage() = default;
21
22     /**
23      * \brief Adds a vehicle to a vehicle collection.
24      * \brief A specific vehicle is passed in and casted to a vehicle Pointer.
25      * \brief This is allowed because Car,Truck and Bike are derived from Vehicle.
26      * \brief A car is a Vehicle.
27      * \brief This casted Pointer is copied ito this methode and added to the collection
28      * \param newVehicle : Pointer to a Vehicle.
29      */
30     void AddVehicle(Vehicle const * const newVehicle);
31
32     /**
33      * \brief deletes Vehicle inside garage from provided pointer.
34      * \param pVehicle : Pointer to a Vehicle.
35      */
36     void DeleteVehicle(Vehicle const * const pVehicle);
37
38     /**
39      * \brief Functions searches for vehicle with matching plate.
40      * \param pVehicle : Pointer to a Vehicle.
41      * \return pointer to the vehicle inside the garage
42      */
43     Vehicle const * const SearchPlate(const std::string & plate) const;
44
45     /**
46      * \brief Formatted of every car and its drive record
47      * \param ost : Refernce to an ostream where the Entry should be printed at.
48      * \return Referenced ostream
49      */
50     std::ostream& Print(std::ostream& ost = std::cout) const override;
51
52     /**

```

```

53     * \brief Calculates sum of every kilometer every vehicle has driven
54     * \brief in total
55     * \return size_t total kilometers
56     */
57     size_t GetTotalDrivenKilometers() const;
58
59
60     // TODO: Copy / assignement implementation
61     // TODO: overload for output operator <<
62
63     Garage(const Garage&);
64     void operator=(Garage garage);
65
66     ~Garage();
67
68 private:
69     TGarageCont m_vehicles;
70 };
71
72 /**
73  * \brief Override for ostream operator
74  * \return ostream
75  */
76 std::ostream& operator <<(std::ostream& ost, Garage& garage);

```

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 is inside m_Vehicles -> erase and free
25     auto itr = std::find(m_vehicles.begin(), m_vehicles.end(), pVehicle);
26     if (itr != m_vehicles.end())
27     {
28         m_vehicles.erase(itr);
29         delete pVehicle;
30     }
31 }
32
33 const Vehicle* const Garage::SearchPlate(const std::string & plate) const
34 {
35     for (const auto &elem : m_vehicles)
36     {
37         if (elem->GetPlate() == plate)
38         {
39             return elem;
40         }
41     }

```



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

6.8 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
15 // Enumeration for a fuel type
16 enum TFuel {
17     Diesel = 0,
18     Benzin = 1,
19     Elektro = 2,
20 };
21
22 class Vehicle: public Object {
23 public:
24
25     /**
26      * \brief Getter for the brand member.
27      *
28      * \return string with the brand name
29      */
30     std::string GetBrand() const;
31
32     /**
33      * \brief Getter for the plate member.
34      *
35      * \return string with the plate name
36      */
37     std::string GetPlate() const;
38
39     /**
40      * \brief Getter for the fuel member.
41      *
42      * \return TFuel with the specified fuel type
43      */
44     TFuel GetFuelType() const;
45
46     /**
47      * \brief Getter for the drive record.
48      *
49      * \return const reference to the drive record
50      */
51     const DriveRecord & GetDriveRecord() const;
52
53     /**
54      * \brief Methode for adding a record entry to the drive record collection.
55      *
56      * \param entry : Entry which should be added to the drive record
57      */
58     void AddRecord(const RecordEntry& entry);
59
60     /**
61      * \brief Getter for the total milage of a vehicle.
62      *
63      * \return Total milage of a vehicle
64      */
65     size_t GetMilage() const;
66
67     /**
68      * @brief Creates a clone of the vehicle.
69      *
70      * \return a exact replicate of a vehicle
71      */
72     virtual Vehicle const* Clone() const = 0;
73
74 protected:
75
76     /**
77      * \brief protected CTOR of a vehicle.
78      * \brief protected because it is a abstract class
79      *
80      * \param brand : string that represents the brand of the vehicle
81      * \param fuelType : Fuel type of the vehicle
82      */
83     Vehicle(const std::string& brand, const TFuel& fuelType, const std::string& plate) : m_brand{ brand }, m_fuel{ fuelType }, m_plate{ plate }

```

```
84
85 private:
86     std::string m_brand;
87     std::string m_plate;
88     TFuel m_fuel;
89     DriveRecord m_record;
90 };
91
92
93 #endif // !1
```

6.9 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 }
```

6.10 Car.hpp

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

6.11 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 exact replicate of a vehicle
36  */
37 Vehicle const* Car::Clone() const
38 {
39     return new Car(*this);
40 }

```

6.12 Truck.hpp

```

1  #ifndef TRUCK_HPP
2  #define TRUCK_HPP
3
4  #include "Vehicle.hpp"
5
6  class Truck : public Vehicle {
7  public:
8
9      /**
10     * \brief CTOR of a Truck -> calles the Base Class vehicle CTOR.
11     *
12     * \param brand string that identifies the brand.
13     * \param fuelType Fueltype of the Truck
14     * \param plate string that identifies the plate.
15     */
16     Truck(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}
17
18     /**
19     * \brief Function that print all the vehicle specific info with the drive record.
20     *
21     * \param ost where the data should be printed at
22     * \return referenced ostream
23     */
24     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;
25
26     /**
27     * @brief Creates a clone of the vehicle.
28     *
29     * \return a exact replicate of a vehicle
30     */

```

```
31     virtual Vehicle const* Clone() const;
32
33 private:
34 };
35
36
37 #endif
```

6.13 Truck.cpp

```
1  /**
2   * \file   Truck.cpp
3   * \brief  Implementation of a Truck
4   *
5   * \author Simon
6   * \date   October 2025
7   *
8   *****/
9  #include "Truck.hpp"
10
11 using namespace std;
12
13 /**
14 * \brief Function that print all the vehicle specific info with the drive record.
15 *
16 * \param ost where the data should be printed at
17 * \return referenced ostream
18 */
19 std::ostream& Truck::Print(std::ostream& ost) const
20 {
21     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;
22
23     ost << endl << left << setw(14) << "Fahrzeugart:" << "LKW" << endl;
24     ost << left << setw(14) << "Marke:" << GetBrand() << endl;
25     ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;
26     GetDriveRecord().Print(ost);
27
28     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;
29
30     return ost;
31 }
32
33 /**
34 * @brief Creates a clone of the vehicle.
35 *
36 * \return a exact replicate of a vehicle
37 */
38 Vehicle const* Truck::Clone() const
39 {
40     return new Truck(*this);
41 }
```

6.14 Bike.hpp

```
1  #ifndef BIKE_HPP
2  #define BIKE_HPP
3
4  #include "Vehicle.hpp"
5
6  class Bike : public Vehicle {
7  public:
8
```

```
9  /**
10 * \brief CTOR of a Bike -> calles the Base Class vehicle CTOR.
11 *
12 * \param brand string that identifies the brand.
13 * \param fuelType Fueltype of the Bike
14 * \param plate string that identifies the plate.
15 */
16 Bike(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}
17
18 /**
19 * \brief Function that print all the vehicle specific info with the drive record.
20 *
21 * \param ost where the data should be printed at
22 * \return referenced ostream
23 */
24 virtual std::ostream& Print(std::ostream& ost = std::cout) const override;
25
26 /**
27 * @brief Creates a clone of the vehicle.
28 *
29 * \return a excat replicate of a vehicle
30 */
31 virtual Vehicle const* Clone() const;
32
33 private:
34 };
35
36
37 #endif
```

6.15 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 excat replicate of a vehicle
36 */
```

```
37 Vehicle const* Bike::Clone() const
38 {
39     return new Bike(*this);
40 }
```