

Name: Simon Offenberger / Simon Vogelhuber

Aufwand in h: siehe Doku.

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

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.

Die folgenden Punkte gelten auch für alle nachfolgenden Übungen:

1. Werfen Sie wo nötig Exceptions und geben Sie Fehlermeldungen aus!
2. Implementieren Sie einen ausführlichen Testtreiber und geben sie entsprechende Meldungen für die Testprotokollierung aus.
3. Verfassen Sie weiters eine Systemdokumentation mit folgendem Inhalt:
 - Verteilung der Aufgaben auf die Teammitglieder.
 - Anforderungsdefinition mit eventuell zusätzlich getroffenen Annahmen. Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen und begründen Sie diese.
 - Systementwurf in Form eines Klassendiagrammes mit allen Klassen und deren Beziehungen, inklusive der wichtigsten Attribute und Methoden. Geben Sie zusätzlich in den entsprechenden Header-Dateien den Verfasser an! Das Klassendiagramm muss nicht vollständig dem implementierten Sourcecode entsprechen! Geben Sie weiters Ihre Designentscheidungen an und begründen sie diese!
 - Testausgaben: die Ausgaben sollen aussagekräftig sein, damit aus der Ausgabe erkennbar ist, was getestet wurde.

- Vollständig dokumentierter Sourcecode (Korrektur der Tutoren). Verwenden Sie Doxygen-Kommentare.
4. Die einzelnen Klassen (Komponenten) werden direkt im Quellcode dokumentiert und mit Hilfe von Doxygen eine HTML-Doku generiert.
 5. 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.

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!



HSD

FH-HAGENBERG

Systemdokumentation Projekt Fuhrpark

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 16. Oktober 2025

Inhaltsverzeichnis

1	Organisatorisches	6
1.1	Team	6
1.2	Aufteilung der Verantwortlichkeitsbereiche	6
1.3	Aufwand	7
2	Anforderungsdefinition (Systemspezifikation)	8
3	Systementwurf	9
3.1	Klassendiagramm	9
3.2	Designentscheidungen	10
4	Dokumentation der Komponenten (Klassen)	11
5	Testprotokollierung	12
6	Quellcode	19
6.1	Object.hpp	19
6.2	RecordEntry.hpp	20
6.3	RecordEntry.cpp	22
6.4	DriveRecord.hpp	23
6.5	DriveRecord.cpp	24
6.6	Garage.hpp	25
6.7	Garage.cpp	27
6.8	TFuel.hpp	29
6.9	Vehicle.hpp	30
6.10	Vehicle.cpp	32
6.11	Car.hpp	34
6.12	Car.cpp	35
6.13	Truck.hpp	36
6.14	Truck.cpp	37
6.15	Bike.hpp	38
6.16	Bike.cpp	39

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@students.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 9 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

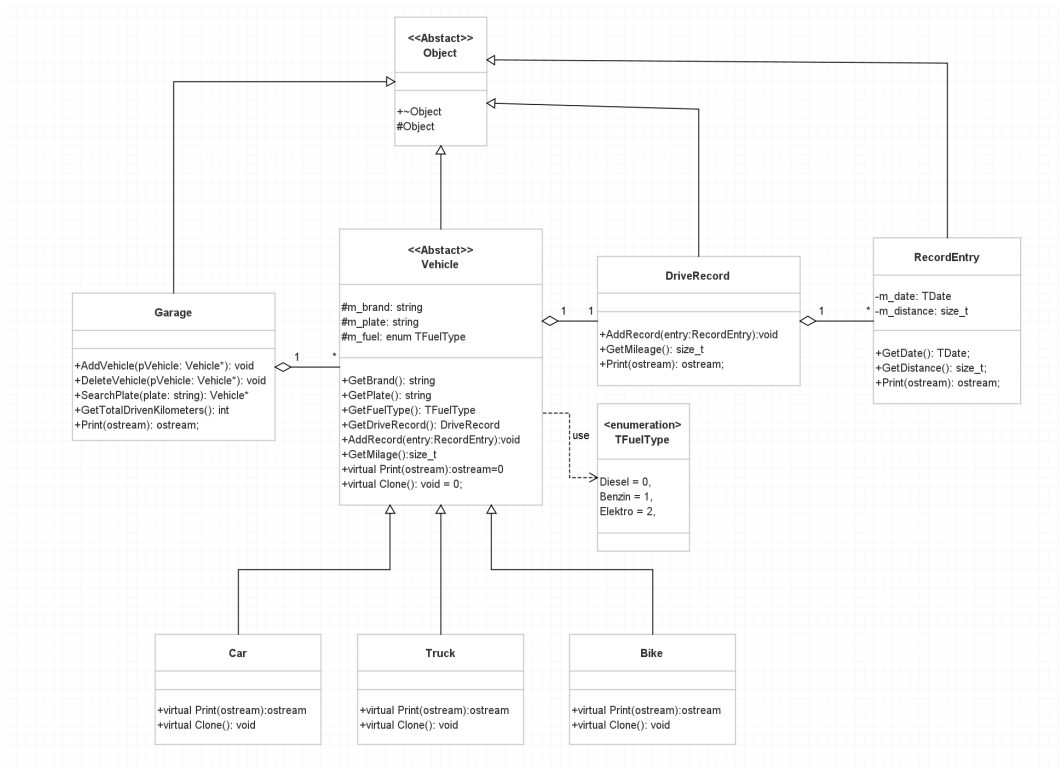
- Verwalten von verschiedenen Fahrzeugarten (Auto, LKW, Motorrad,...).
- Hinzufügen und löschen eines Fahrzeuges
- Ausgabe aller Fahrzeugdaten inklusive der Fahrtenbucheinträge.
- Suchen nach einem Fahrzeug mit dessen Kennzeichen.
- Berechnung der Gesamtkilometer aller Fahrzeuge im Fuhrpark.
- Der Fuhrpark muss kopierbar und zuweisbar sein.
- Nach hinzufügen der Fahrzeuge sind diese im Besitz des Fuhrparks, dieser ist dadurch auch für das Löschen verantwortlich.

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



CTOR und DTOR nur im Object eingetragen ,da dies implementierungsspezi-
fische Angaben sind und nicht im UML Standard definiert sind.

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 Basis-Klasse '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 größerwerdenden 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: 00000223107B1010 == Result: 00000223107B1010)
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 fueltype
137 [Test OK] Result: (Expected: 1 == Result: 1)
138
139 Test car fueltype - error buffer
140 [Test OK] Result: (Expected: true == Result: true)
141
142 Test car plate
143 [Test OK] Result: (Expected: SR770BA == Result: SR770BA)
144
145 Test car plate - error buffer
146 [Test OK] Result: (Expected: true == Result: true)
147
148 Test car brand
149 [Test OK] Result: (Expected: Steyr == Result: Steyr)
150
151 Test car brand - error buffer
152 [Test OK] Result: (Expected: true == Result: true)
153
154 Test car milage
155 [Test OK] Result: (Expected: 25 == Result: 25)
156
157 Test car milage - error buffer
158 [Test OK] Result: (Expected: true == Result: true)
159
160 Test car driveRecord
161 [Test OK] Result: (Expected: 13.10.2025:    25 km
162 == Result: 13.10.2025:    25 km
163 )
164
165 Test car driveRecord - error buffer
166 [Test OK] Result: (Expected: true == Result: true)
167
168 Test Car CTOR empty brand
```

```
169 [Test OK] Result: (Expected: ERROR: Passed in empty string! == Result:
    ↳ ERROR: Passed in empty string!)
170
171 Test Car CTOR empty plate
172 [Test OK] Result: (Expected: ERROR: Passed in empty string! == Result:
    ↳ ERROR: Passed in empty string!)
173
174
175 *****
176
177
178 *****
179 TESTCASE START
180 *****
181
182 Test Bike fueltype
183 [Test OK] Result: (Expected: 1 == Result: 1)
184
185 Test Bike fueltype - error buffer
186 [Test OK] Result: (Expected: true == Result: true)
187
188 Test Bike plate
189 [Test OK] Result: (Expected: SR770BA == Result: SR770BA)
190
191 Test Bike plate - error buffer
192 [Test OK] Result: (Expected: true == Result: true)
193
194 Test Bike brand
195 [Test OK] Result: (Expected: Steyr == Result: Steyr)
196
197 Test Bike brand - error buffer
198 [Test OK] Result: (Expected: true == Result: true)
199
200 Test Bike milage
201 [Test OK] Result: (Expected: 25 == Result: 25)
202
203 Test Bike milage - error buffer
204 [Test OK] Result: (Expected: true == Result: true)
205
206 Test Bike driveRecord
207 [Test OK] Result: (Expected: 13.10.2025:      25 km
208 == Result: 13.10.2025:      25 km
209 )
210
```



```
211 Test Bike driveRecord - error buffer
212 [Test OK] Result: (Expected: true == Result: true)
213
214 Test Bike CTOR empty brand
215 [Test OK] Result: (Expected: ERROR: Passed in empty string! == Result:
    ↪ ERROR: Passed in empty string!)
216
217 Test Bike CTOR empty plate
218 [Test OK] Result: (Expected: ERROR: Passed in empty string! == Result:
    ↪ ERROR: Passed in empty string!)
219
220
221 *****
222
223
224 *****
225 TESTCASE START
226 *****
227
228 Test Truck fueltype
229 [Test OK] Result: (Expected: 1 == Result: 1)
230
231 Test Truck fueltype - error buffer
232 [Test OK] Result: (Expected: true == Result: true)
233
234 Test Truck plate
235 [Test OK] Result: (Expected: SR770BA == Result: SR770BA)
236
237 Test Truck plate - error buffer
238 [Test OK] Result: (Expected: true == Result: true)
239
240 Test Truck brand
241 [Test OK] Result: (Expected: Steyr == Result: Steyr)
242
243 Test car brand - error buffer
244 [Test OK] Result: (Expected: true == Result: true)
245
246 Test Truck milage
247 [Test OK] Result: (Expected: 50 == Result: 50)
248
249 Test Truck milage - error buffer
250 [Test OK] Result: (Expected: true == Result: true)
251
252 Test truck driveRecord
```

```
253 [Test OK] Result: (Expected: 13.10.2025:    25 km
254 == Result: 13.10.2025:    25 km
255 )
256
257 Test truck driveRecord - error buffer
258 [Test OK] Result: (Expected: true == Result: true)
259
260 Test truck CTOR empty brand
261 [Test OK] Result: (Expected: ERROR: Passed in empty string! == Result:
    ↪ ERROR: Passed in empty string!)
262
263 Test truck CTOR empty plate
264 [Test OK] Result: (Expected: ERROR: Passed in empty string! == Result:
    ↪ ERROR: Passed in empty string!)
265
266
267 *****
268
269 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     /**  
16      * Definitions of the Exceptionmessages  
17      */  
18     inline static const std::string ERROR_BAD_OSTREAM = "ERROR:_Provided_Ostream_is_bad";  
19     inline static const std::string ERROR_FAIL_WRITE = "ERROR:_Fail_to_write_on_provided_Ostream";  
20  
21     /**  
22      * Virtual DTOR, once virtual always virtual.  
23      */  
24     virtual ~Object() = default;  
25  
26 protected:  
27     /**  
28      * \brief protected CTOR -> abstract.  
29      */  
30     Object() = default;  
31 };  
32  
33 #endif // !1
```

6.2 RecordEntry.hpp

```

1  /*****
2  * \file   RecordEntry.hpp
3  * \brief  Class that defines an entry in a drive 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     * Definitions of the Exceptionmessages
25     */
26     inline static const std::string ERROR_DISTANCE_ZERO = "ERROR:_Distance_cannot_be_zero!";
27
28     /**
29     * \brief CTOR of a drive record.
30     *
31     * \param date : date when the drive happend
32     * \param distance : the distance of the drive in km
33     */
34     RecordEntry(const TDate& date, const size_t& distance);
35
36     /**
37     * \brief Getter of the distance member of the Record Entry Class.
38     *
39     * \return Distance of this Record Entry
40     */
41     size_t GetDistance() const;
42
43     /**
44     * \brief Getter of the data member of the Record Entry Class.
45     *
46     * \return Date of this Record Entry
47     */
48     TDate GetDate() const;
49
50     /**
51     * \brief Formatted output of this Record Entry on an ostream.
52     *
53     * \param ost : Refernce to an ostream where the Entry should be printed at.
54     * \return Referenced ostream
55     * \throw ERROR_BAD_OSTREAM
56     * \throw ERROR_WRITE_FAIL
57     */
58     virtual std::ostream& Print(std::ostream& ost = std::cout) const;
59
60     /**
61     * \brief less than operator, is used for storing the Entries in a multiset.
62     *
63     * \param rh : Righthandside of the less than operator
64     * \return true: left hand side is less than the right hand side.
65     * \return false: left hand side is greather or equal than the right hand side.
66     */
67     bool operator<(const RecordEntry& rh) const;
68
69 private:
70     TDate m_date;        // private date member
71     size_t m_distance;    // private distance member
72 };

```

```
73 |  
74 |  
75 | #endif
```

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 size_t RecordEntry::GetDistance() const  
20 {  
21     return m_distance;  
22 }  
23  
24  
25 TDate RecordEntry::GetDate() const  
26 {  
27     return m_date;  
28 }  
29  
30  
31 std::ostream& RecordEntry::Print(std::ostream& ost) const  
32 {  
33     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;  
34  
35     ost << std::setfill('0') << right << std::setw(2) << m_date.day() << "."  
36         << std::setw(2) << static_cast<unsigned>(m_date.month()) << "."  
37         << std::setw(4) << m_date.year() << "; " << std::setfill('_')  
38         << std::setw(6) << m_distance << "_km\n";  
39  
40     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;  
41  
42     return ost;  
43 }  
44  
45  
46 bool RecordEntry::operator<(const RecordEntry& rh) const  
47 {  
48     return m_date < rh.m_date;  
49 }
```

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      * \throw ERROR_BAD_OSTREAM
44      * \throw ERROR_WRITE_FAIL
45      */
46     virtual std::ostream& Print(std::ostream& ost = std::cout) const;
47
48 private:
49     TCont m_driveRecords;
50 };
51
52
53
54 #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 void DriveRecord::AddRecord(const RecordEntry& entry)  
14 {  
15     m_driveRecords.insert(entry);  
16 }  
17  
18  
19 size_t DriveRecord::GetMilage() const  
20 {  
21     // use std::accumulate + lambda to calc the total Milage  
22     return std::accumulate(m_driveRecords.cbegin(), m_driveRecords.cend(), static_cast<size_t>(0),  
23         [](const size_t val, const RecordEntry& entry) {return val + entry.GetDistance();});  
24 }  
25  
26  
27 std::ostream& DriveRecord::Print(std::ostream& ost) const  
28 {  
29     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;  
30  
31     std::for_each(m_driveRecords.cbegin(), m_driveRecords.cend(), [&](const RecordEntry& entry) {entry.Print(ost);});  
32  
33     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;  
34  
35     return ost;  
36 }
```


6.6 Garage.hpp

```

1  /*****
2  * \file   Garage.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 Statement for the used Container to store the Vehicles
17 using TGarageCont = std::vector<Vehicle const*>;
18
19 class Garage : public Object {
20 public:
21
22     /**
23     * Definitions of the Exceptionmessages
24     */
25     inline static const std::string ERROR_NULLPTR= "ERROR:_Passed_in_Nullptr!";
26
27     /**
28     * \brief Default CTOR.
29     *
30     */
31     Garage() = default;
32
33     /**
34     * \brief Adds a vehicle to a vehicle collection.
35     * \brief A specific vehicle is passed in and casted to a vehicle Pointer.
36     * \brief This is allowed because Car,Truck and Bike are derived from Vehicle.
37     * \brief A car is a Vehicle.
38     * \brief This casted Pointer is copied into this methode and added to the collection
39     * \param newVehicle : Pointer to a Vehicle.
40     */
41     void AddVehicle(Vehicle const* const newVehicle);
42
43     /**
44     * \brief deletes Vehicle inside garage from provided pointer.
45     * \param pVehicle : Pointer to a Vehicle.
46     */
47     void DeleteVehicle(Vehicle const* const pVehicle);
48
49     /**
50     * \brief Functions searches for vehicle with matching plate.
51     *
52     * \return pointer to the vehicle inside the garage
53     */
54     Vehicle const* const SearchPlate(const std::string & plate) const;
55
56     /**
57     * \brief Formatted of every car and its drive record
58     * \param ost : Refernce to an ostream where the Entry should be printed at.
59     * \return Referenced ostream
60     * \throw ERROR_BAD_OSTREAM
61     * \throw ERROR_WRITE_FAIL
62     */
63     std::ostream& Print(std::ostream& ost = std::cout) const;
64
65     /**
66     * \brief Calculates sum of every kilometer every vehicle has driven
67     * \brief in total
68     * \return size_t total kilometers
69     */
70     size_t GetTotalDrivenKilometers() const;
71
72     /**

```

```
73     * \brief Copy CTOR of Garage. Is Needed because Garage
74     * \brief owns all the Vehicle Objects that are allocated on the heap.
75     *
76     * \param garage Garage that should be copied
77     */
78     Garage(const Garage& garage);
79
80     /**
81     * \brief Assign Operator for a Object of Garage.
82     *
83     * \param garage Garage of the right hand side of the assignment.
84     */
85     void operator=(Garage garage);
86
87     /**
88     * \brief DTOR of a Garage obj.
89     * \brief Frees all the allocated Memory
90     *
91     */
92     ~Garage();
93
94 private:
95     TGarageCont m_vehicles;
96 };
97
98 /**
99 * \brief Override for output operator
100 * \return ostream
101 */
102 std::ostream& operator <<(std::ostream& ost, Garage& garage);
103
104 #endif
```

6.7 Garage.cpp

```
1  /*****
2  * \file    Garage.cpp
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
45     return nullptr;
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& garage)
74 | {
75 |     for_each(
76 |         garage.m_vehicles.cbegin(), garage.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     * Defintions of the Exceptionmessages  
21     */  
22     inline static const std::string ERROR_EMPTY_STRING = "ERROR:_Passed_in_empty_string!";  
23  
24     /**  
25     * \brief Getter for the brand member.  
26     *  
27     * \return string with the brand name  
28     */  
29     std::string GetBrand() const;  
30  
31     /**  
32     * \brief Getter for the plate member.  
33     *  
34     * \return string with the plate name  
35     */  
36     std::string GetPlate() const;  
37  
38     /**  
39     * \brief Getter for the fuel member.  
40     *  
41     * \return TFuel with the specified fuel type  
42     */  
43     TFuel GetFuelType() const;  
44  
45     /**  
46     * \brief Getter for the drive record.  
47     *  
48     * \return const refernce to the drive record  
49     */  
50     const DriveRecord & GetDriveRecord() const;  
51  
52     /**  
53     * \brief Methode for adding a record entry to the drive record collection.  
54     *  
55     * \param entry : Entry which should be added to the drive record  
56     */  
57     void AddRecord(const RecordEntry& entry);  
58  
59     /**  
60     * \brief Getter for the total milage of a vehicle.  
61     *  
62     * \return Total milage of a vehicle  
63     */  
64     size_t GetMilage() const;  
65  
66     /**  
67     * @brief Creates a clone of the vehicle.  
68     *  
69     * \return a excat replicate of a vehicle  
70     */  
71     virtual Vehicle const* Clone() const = 0;  
72
```

```
73
74 /**
75  * \brief Print function that is implementet by dirved Classes.
76  *
77  * \param ost Reference to an ostream where the Result should be printed at
78  * \return referenced ostream
79  * \throw ERROR_BAD_OSTREAM
80  * \throw ERROR_WRITE_FAIL
81  */
82 virtual std::ostream& Print(std::ostream& ost = std::cout) const = 0;
83
84
85 protected:
86
87 /**
88  * \brief protected CTOR of a vehicle.
89  * \brief protected because it is a abstract class
90  *
91  * \param plate : string that represents the plate of the vehicle
92  * \param brand : string that represents the brand of the vehicle
93  * \param fuelType : Fuel type of the vehicle
94  */
95 Vehicle(const std::string& brand, const TFuel& fuelType, const std::string& plate);
96
97 private:
98     std::string m_brand;
99     std::string m_plate;
100     TFuel m_fuel;
101     DriveRecord m_record;
102 };
103
104
105 #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 }  
69  
70 Vehicle::Vehicle(const std::string& brand, const TFuel& fuelType, const std::string& plate) : m_fuel(fuelType)  
71 {  
72     if (brand.empty() || plate.empty()) throw ERROR_EMPTY_STRING;
```



```
73 |  
74 |     m_brand = brand;  
75 |     m_plate = plate;  
76 |  
77 | }
```

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     * \throw ERROR_EMPTY_STRING  
23     */  
24     Car(const std::string & brand, const TFuel & fuelType, const std::string & plate) : Vehicle(brand, fuelType, plate) {}  
25  
26     /**  
27     * \brief Function that print all the vehicle specific info with the drive record.  
28     *  
29     * \param ost where the data should be printed at  
30     * \return referenced ostream  
31     * \throw ERROR_BAD_OSTREAM  
32     * \throw ERROR_WRITE_FAIL  
33     */  
34     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;  
35  
36     /**  
37     * @brief Creates a clone of the vehicle.  
38     *  
39     * \return a excat replicate of a vehicle  
40     */  
41     virtual Vehicle const* Clone() const;  
42  
43 private:  
44 };  
45  
46 #endif // !1
```

6.12 Car.cpp

```
1  /*****  
2  * \file   Car.cpp  
3  * \brief  Implementation of a Car  
4  *  
5  * \author Simon  
6  * \date   October 2025  
7  *****/  
8  #include "Car.hpp"  
9  
10 using namespace std;  
11  
12  
13 std::ostream& Car::Print(std::ostream& ost) const  
14 {  
15     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;  
16  
17     ost << endl << left << setw(14) << "Fahrzeugart:" << "PKW" << endl;  
18     ost << left << setw(14) << "Marke:" << GetBrand() << endl;  
19     ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;  
20     GetDriveRecord().Print(ost);  
21  
22     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;  
23  
24     return ost;  
25 }  
26  
27  
28 Vehicle const* Car::Clone() const  
29 {  
30     return new Car(*this);  
31 }
```

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     * \throw ERROR_EMPTY_STRING  
23     */  
24     Truck(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}  
25  
26     /**  
27     * \brief Function that print all the vehicle specific info with the drive record.  
28     *  
29     * \param ost where the data should be printed at  
30     * \return referenced ostream  
31     * \throw ERROR_BAD_OSTREAM  
32     * \throw ERROR_WRITE_FAIL  
33     */  
34     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;  
35  
36     /**  
37     * @brief Creates a clone of the vehicle.  
38     *  
39     * \return a excat replicate of a vehicle  
40     */  
41     virtual Vehicle const* Clone() const;  
42  
43 private:  
44 };  
45  
46  
47 #endif
```

6.14 Truck.cpp

```
1  /*****  
2  * \file   Truck.cpp  
3  * \brief  Implementation of a Truck  
4  *  
5  * \author Simon  
6  * \date   October 2025  
7  *****/  
8  #include "Truck.hpp"  
9  
10 using namespace std;  
11  
12  
13 std::ostream& Truck::Print(std::ostream& ost) const  
14 {  
15     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;  
16  
17     ost << endl << left << setw(14) << "Fahrzeugart:" << "LKW" << endl;  
18     ost << left << setw(14) << "Marke:" << GetBrand() << endl;  
19     ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;  
20     GetDriveRecord().Print(ost);  
21  
22     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;  
23  
24     return ost;  
25 }  
26  
27  
28 Vehicle const* Truck::Clone() const  
29 {  
30     return new Truck(*this);  
31 }
```

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     * \throw ERROR_EMPTY_STRING  
23     */  
24     Bike(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}  
25  
26     /**  
27     * \brief Function that print all the vehicle specific info with the drive record.  
28     *  
29     * \param ost where the data should be printed at  
30     * \return referenced ostream  
31     * \throw ERROR_BAD_OSTREAM  
32     * \throw ERROR_WRITE_FAIL  
33     */  
34     virtual std::ostream& Print(std::ostream& ost = std::cout) const override;  
35  
36     /**  
37     * @brief Creates a clone of the vehicle.  
38     *  
39     * \return a excat replicate of a vehicle  
40     */  
41     virtual Vehicle const* Clone() const;  
42  
43 private:  
44 };  
45  
46  
47 #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 std::ostream& Bike::Print(std::ostream& ost) const  
14 {  
15     if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;  
16  
17     ost << endl << left << setw(14) << "Fahrzeugart:" << "Motorrad" << endl;  
18     ost << left << setw(14) << "Marke:" << GetBrand() << endl;  
19     ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;  
20     GetDriveRecord().Print(ost);  
21  
22     if (ost.fail()) throw Object::ERROR_FAIL_WRITE;  
23  
24     return ost;  
25 }  
26  
27  
28 Vehicle const* Bike::Clone() const  
29 {  
30     return new Bike(*this);  
31 }
```