

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, 20. Oktober 2025

# Inhaltsverzeichnis

<b>1 Organisatorisches</b>	<b>6</b>
1.1 Team . . . . .	6
1.2 Aufteilung der Verantwortlichkeitsbereiche . . . . .	6
1.3 Aufwand . . . . .	7
<b>2 Anforderungsdefinition (Systemspezifikation)</b>	<b>8</b>
<b>3 Systementwurf</b>	<b>9</b>
3.1 Klassendiagramm . . . . .	9
3.2 Designentscheidungen . . . . .	10
<b>4 Dokumentation der Komponenten (Klassen)</b>	<b>11</b>
<b>5 Testprotokollierung</b>	<b>12</b>
<b>6 Quellcode</b>	<b>19</b>
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
6.17 main.cpp . . . . .	40

# 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.

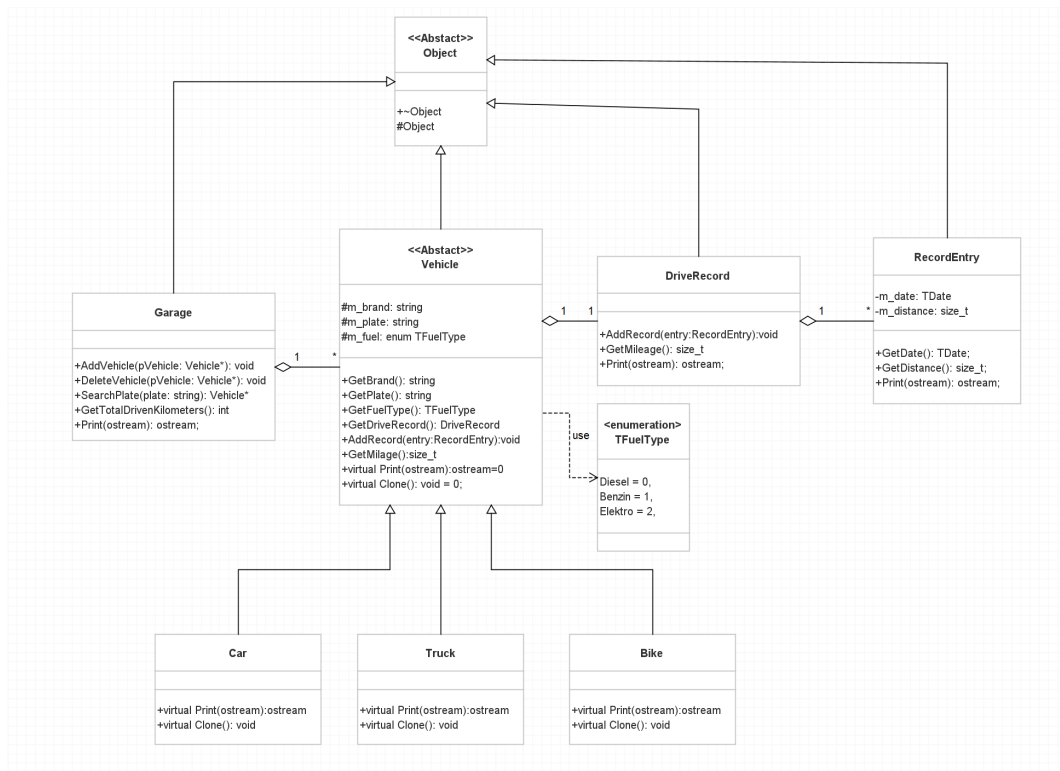
### Funktionen der Fahrzeuge

- 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 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](http://../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: 000001CABA4C2410 == Result: 000001CABA4C2410)
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 }
```

## 6.17 main.cpp

```
1  /*****  
2  * \file   main.cpp  
3  * \brief  Testdriver  
4  *  
5  * \author Simon / Simon  
6  * \date   October 2025  
7  *****/  
8  #include <iostream>  
9  #include <fstream>  
10 #include <sstream>  
11 #include <cassert>  
12 #include "Test.hpp"  
13 #include "RecordEntry.hpp"  
14 #include "DriveRecord.hpp"  
15 #include "Car.hpp"  
16 #include "Bike.hpp"  
17 #include "Truck.hpp"  
18 #include "Garage.hpp"  
19 #include "vld.h"  
20  
21 using namespace std;  
22 using namespace chrono;  
23  
24 static bool Test_RecordEntry(ostream & ost = cout);  
25 static bool Test_DriveRecord(ostream & ost = cout);  
26 static bool Test_Garage(ostream & ost = cout);  
27 static bool Test_Car(ostream & ost = cout);  
28 static bool Test_Bike(ostream & ost = cout);  
29 static bool Test_Truck(ostream & ost = cout);  
30  
31 #define WriteOutputFile true  
32  
33 int main(void){  
34     bool Test_OK = true;  
35  
36     Test_OK = Test_OK && Test_RecordEntry(cout);  
37     Test_OK = Test_OK && Test_DriveRecord(cout);  
38     Test_OK = Test_OK && Test_Garage(cout);  
39     Test_OK = Test_OK && Test_Car(cout);  
40     Test_OK = Test_OK && Test_Bike(cout);  
41     Test_OK = Test_OK && Test_Truck(cout);  
42  
43     if (Test_OK) TestCaseOK(cout);  
44     else TestCaseFail(cout);  
45  
46     if (WriteOutputFile) {  
47  
48         ofstream test_output;  
49         test_output.open("TestOutput.txt");  
50  
51         Test_OK = Test_OK && Test_RecordEntry(test_output);  
52         Test_OK = Test_OK && Test_DriveRecord(test_output);  
53         Test_OK = Test_OK && Test_Garage(test_output);  
54         Test_OK = Test_OK && Test_Car(test_output);  
55         Test_OK = Test_OK && Test_Bike(test_output);  
56         Test_OK = Test_OK && Test_Truck(test_output);  
57  
58         if (Test_OK) TestCaseOK(test_output);  
59         else TestCaseFail(test_output);  
60  
61         test_output.close();  
62     }  
63 }  
64  
65 bool Test_RecordEntry(ostream& ost)  
66 {  
67     assert(ost.good());  
68  
69     bool Test_OK = true;  
70     string error_msg;  
71  
72
```



```
73 | ost << TestStart;
74 | const TDate date{ 2025y,October,13d };
75 | const size_t distance = 150;
76 |
77 | RecordEntry entry1{ date, distance };
78 |
79 | Test_OK = Test_OK && check_dump(ost, "Test_RecordEntry_Get_Date", date,entry1.GetDate());
80 |
81 | Test_OK = Test_OK && check_dump(ost, "Test_RecordEntry_Get_Distance", distance,entry1.GetDistance());
82 |
83 | stringstream result;
84 | string expected = "13.10.2025:150km\n";
85 | entry1.Print(result);
86 |
87 | Test_OK = Test_OK && check_dump(ost, "Test_RecordEntry_Print", true, result.str() == expected);
88 |
89 | ofstream badstream;
90 | badstream.setstate(ios::badbit);
91 |
92 | try {
93 |     RecordEntry entry{ TDate(2025y,October,13d), 150 };
94 |     entry.Print(badstream);
95 | }
96 | catch (const string& err) {
97 |     error_msg = err;
98 | }
99 | catch (bad_alloc const& error) {
100 |     error_msg = error.what();
101 | }
102 | catch (const exception& err) {
103 |     error_msg = err.what();
104 | }
105 | catch (...) {
106 |     error_msg = "Unhandelt_Exception";
107 | }
108 |
109 | Test_OK = Test_OK && check_dump(ost, "Test_RecordEntry_Exception_Bad_Ostream", Object::ERROR_BAD_OSTREAM, error_msg);
110 |
111 | badstream.close();
112 |
113 |
114 | RecordEntry entrygreater{ {2025y,October,8d},10 };
115 | RecordEntry entryless{ {2025y,October,6d},6 };
116 |
117 | Test_OK = Test_OK && check_dump(ost, "Test_RecordEntry_less_than_operator", true, entryless<entrygreater);
118 |
119 | try {
120 |     RecordEntry entry{ TDate(2025y,October,13d), 0 };
121 | }
122 | catch (const string& err) {
123 |     error_msg = err;
124 | }
125 | catch (bad_alloc const& error) {
126 |     error_msg = error.what();
127 | }
128 | catch (const exception& err) {
129 |     error_msg = err.what();
130 | }
131 | catch (...) {
132 |     error_msg = "Unhandelt_Exception";
133 | }
134 |
135 | Test_OK = Test_OK && check_dump(ost, "Test_RecordEntry_Exception_Distance_0", RecordEntry::ERROR_DISTANCE_ZERO, error_msg);
136 |
137 |
138 | ost << TestEnd;
139 |
140 | return Test_OK;
141 | }
142 |
143 | bool Test_DriveRecord(ostream& ost)
144 | {
145 |     assert(ost.good());
146 |
147 |     bool Test_OK = true;
```

```
148     string error_msg;
149
150     ost << TestStart;
151     const TDate date{ 2025y,October,13d };
152     const TDate date1{ 2025y,October,10d };
153     const TDate date2{ 2025y,October,5d };
154     const size_t distance = 150;
155     const size_t distance1 = 150;
156     const size_t distance2 = 150;
157
158     RecordEntry entry{ date,distance };
159     RecordEntry entry1{ date1,distance1 };
160     RecordEntry entry2{ date2,distance2 };
161
162     DriveRecord dRecord;
163
164     dRecord.AddRecord(entry);
165     dRecord.AddRecord(entry1);
166     dRecord.AddRecord(entry2);
167
168     stringstream result;
169     stringstream expected;
170     dRecord.Print(result);
171
172     entry2.Print(expected);
173     entry1.Print(expected);
174     entry.Print(expected);
175
176     Test_OK = Test_OK && check_dump(ost, "Test_DriveRecord_Print_Sorted_and_Add_Record", true, result.str() == expected.str());
177
178     const size_t total_milage = 450;
179
180     Test_OK = Test_OK && check_dump(ost, "Test_DriveRecord_Get_Milage", total_milage, dRecord.GetMilage());
181
182     ofstream badstream;
183     badstream.setstate(ios::badbit);
184
185     try {
186         DriveRecord dEntry{};
187         dEntry.AddRecord(entry);
188         dEntry.Print(badstream);
189     }
190     catch (const string& err) {
191         error_msg = err;
192     }
193     catch (bad_alloc const& error) {
194         error_msg = error.what();
195     }
196     catch (const exception& err) {
197         error_msg = err.what();
198     }
199     catch (...) {
200         error_msg = "Unhandelt_Exception";
201     }
202
203     Test_OK = Test_OK && check_dump(ost, "Test_DriveRecord_Exception_Bad_Ostream", Object::ERROR_BAD_OSTREAM, error_msg);
204
205     const string NoExc = "No_Exception";
206
207     try {
208         DriveRecord dEntry{};
209         dEntry.Print(result);
210         error_msg = NoExc;
211     }
212     catch (const string& err) {
213         error_msg = err;
214     }
215     catch (bad_alloc const& error) {
216         error_msg = error.what();
217     }
218     catch (const exception& err) {
219         error_msg = err.what();
220     }
221     catch (...) {
222         error_msg = "Unhandelt_Exception";
```

```
223     }
224
225     Test_OK = Test_OK && check_dump(ost, "Test_DriveRecord_Empty_Print", error_msg, NoExc);
226
227     badstream.close();
228
229     ost << TestEnd;
230
231     return Test_OK;
232 }
233
234 static bool Test_Garage(ostream& ost)
235 {
236     assert(ost.good());
237
238     bool Test_OK = true;
239     string error_msg;
240
241     ost << TestStart;
242
243     // Testing search plate func
244     try
245     {
246
247         std::string testPlate = "SR770BA";
248         Car* testCar = new Car{ "UAZ", Diesel, testPlate };
249         testCar->AddRecord({ { 2025y, October, 13d }, 25 });
250
251         Garage testGarage;
252         testGarage.AddVehicle(testCar);
253         testGarage.AddVehicle(new Bike{"Kawasaki_Z650RS", Benzin, "SB13KK"});
254         testGarage.AddVehicle(new Truck{"Scania", Diesel, "SB132KK"});
255
256         Test_OK = Test_OK &&
257             check_dump(
258                 ost,
259                 "vehicle_plate_search",
260                 (const Vehicle*) testCar,
261                 testGarage.SearchPlate(testPlate)
262             );
263     }
264
265     catch (const string& err) {
266         error_msg = err;
267     }
268     catch (bad_alloc const& error) {
269         error_msg = error.what();
270     }
271     catch (const exception& err) {
272         error_msg = err.what();
273     }
274     catch (...) {
275         error_msg = "Unhandled_exception";
276     }
277
278     Test_OK = Test_OK && check_dump(ost, "Test_garage_plate_search_-_error_buffer", error_msg.empty(), true);
279     error_msg.clear();
280
281     // Searching invalid plate
282     try
283     {
284
285         std::string testPlate = "SR770BA";
286         Car* testCar = new Car{ "UAZ", Diesel, testPlate };
287         testCar->AddRecord({ { 2025y, October, 13d }, 25 });
288
289         Garage testGarage;
290         testGarage.AddVehicle(testCar);
291
292         Vehicle const* result = testGarage.SearchPlate("NOTREAL");
293         Test_OK = Test_OK && check_dump(ost, "Test_garage_plate_search_invalid_plate", result, (Vehicle const*)nullptr);
294     }
295
296     catch (const string& err) {
```

```
298     error_msg = err;
299 }
300 catch (bad_alloc const& error) {
301     error_msg = error.what();
302 }
303 catch (const exception& err) {
304     error_msg = err.what();
305 }
306 catch (...) {
307     error_msg = "Unhandled_exception";
308 }
309
310 Test_OK = Test_OK && check_dump(ost, "Test_garage_plate_search_invalid_plate_-_error_buffer", error_msg.empty(), true);
311 error_msg.clear();
312
313 try
314 {
315     std::string testPlate = "SR770BA";
316     Car* testCar = new Car( "UAZ", Diesel, testPlate );
317     testCar->AddRecord({ { 2025y,October,13d }, 25 });
318
319     Garage testGarage;
320     testGarage.AddVehicle(testCar);
321
322     // testing print
323     std::stringstream expectation;
324     std::stringstream result;
325
326     testCar->Print(expectation);
327     testGarage.Print(result);
328     Test_OK = Test_OK && check_dump(ost, "Test_Garage_Print", expectation.str(), result.str());
329 }
330
331 catch (const string& err) {
332     error_msg = err;
333 }
334
335 catch (bad_alloc const& error) {
336     error_msg = error.what();
337 }
338 catch (const exception& err) {
339     error_msg = err.what();
340 }
341 catch (...) {
342     error_msg = "Unhandled_exception";
343 }
344
345 Test_OK = Test_OK && check_dump(ost, "Test_garage_print_-_error_buffer", error_msg.empty(), true);
346 error_msg.clear();
347
348 // Empty Garage
349 try
350 {
351     Garage testGarage;
352     std::stringstream result;
353     testGarage.Print(result);
354     Test_OK = Test_OK && check_dump(ost, "Test_garage_print_empty_garage_", result.str().empty(), true);
355 }
356
357 catch (const string& err) {
358     error_msg = err;
359 }
360 catch (bad_alloc const& error) {
361     error_msg = error.what();
362 }
363 catch (const exception& err) {
364     error_msg = err.what();
365 }
366 catch (...) {
367     error_msg = "Unhandled_exception";
368 }
369
370 Test_OK = Test_OK && check_dump(ost, "Test_garage_print_empty_garage_-_error_buffer", error_msg.empty(), true);
371 error_msg.clear();
372
```

```
373     try
374     {
375         std::string testPlate = "SR770BA";
376         Car* testCar = new Car{ "UAZ", Diesel, testPlate };
377         Car* testCar2 = new Car{ "Mercedes", Benzin, "UU1234AB" };
378
379         testCar->AddRecord({ { 2025y,October,13d }, 25 });
380         testCar2->AddRecord({ { 2025y,October,13d }, 25 });
381
382         Garage testGarage;
383         testGarage.AddVehicle(testCar);
384         testGarage.AddVehicle(testCar2);
385
386         testGarage.DeleteVehicle(testGarage.SearchPlate(testPlate));
387         Vehicle const * const testPtr = testGarage.SearchPlate(testPlate);
388
389         Test_OK = Test_OK && check_dump(ost, "Test_Delete_Vehicle", testPtr, (Vehicle const* const) 0);
390     }
391
392     catch (const string& err) {
393         error_msg = err;
394     }
395     catch (bad_alloc const& error) {
396         error_msg = error.what();
397     }
398     catch (const exception& err) {
399         error_msg = err.what();
400     }
401     catch (...) {
402         error_msg = "Unhandled_exception";
403     }
404
405     Test_OK = Test_OK && check_dump(ost, "Test_garage_print_-_error_buffer", error_msg.empty(), true);
406     error_msg.clear();
407
408     //Test Copy and Swap
409     try
410     {
411         std::string testPlate = "SR770BA";
412         Car* testCar = new Car{ "UAZ", Diesel, testPlate };
413         Car* testCar2 = new Car{ "Mercedes", Benzin, "UU1234AB" };
414         Vehicle* const testPtr = nullptr;
415
416         testCar->AddRecord({ { 2025y,October,13d }, 25 });
417         testCar2->AddRecord({ { 2025y,October,13d }, 25 });
418
419         Garage testGarage;
420         testGarage.AddVehicle(testCar);
421         testGarage.AddVehicle(testCar2);
422
423         Garage testGarageCopy = testGarage;
424
425         Test_OK = Test_OK && check_dump(ost, "Test_Delete_Vehicle", testPtr, (Vehicle* const)0);
426     }
427
428     catch (const string& err) {
429         error_msg = err;
430     }
431     catch (bad_alloc const& error) {
432         error_msg = error.what();
433     }
434     catch (const exception& err) {
435         error_msg = err.what();
436     }
437     catch (...) {
438         error_msg = "Unhandled_exception";
439     }
440
441     Test_OK = Test_OK && check_dump(ost, "Test_Delete_Vehicle_-_error_buffer", error_msg.empty(), true);
442     error_msg.clear();
443
444     // Test GetTotalDrivenKilometers()
445
446     try
447     {
```

```
448     Car* const testCar1 = new Car{ "Madza", Elektro, "WD40AHAH" };
449     Car* const testCar2 = new Car{ "MG", Elektro, "DeiMama" };
450
451     testCar1->AddRecord({ { 2025y,October,13d }, 25 });
452     testCar1->AddRecord({ { 2025y,October,28d }, 34 });
453     testCar2->AddRecord({ { 2025y,September,13d }, 25 });
454     testCar2->AddRecord({ { 2025y,March,28d }, 34 });
455
456     size_t expect = testCar1->GetMilage() + testCar2->GetMilage();
457
458     Garage testGarage;
459     testGarage.AddVehicle(testCar1);
460     testGarage.AddVehicle(testCar2);
461
462     size_t result = testGarage.GetTotalDrivenKilometers();
463
464     Test_OK = Test_OK && check_dump(ost, "Test_GetTotalDrivenKilometers()", expect, result);
465 }
466
467 catch (const string& err) {
468     error_msg = err;
469 }
470 catch (bad_alloc const& error) {
471     error_msg = error.what();
472 }
473 catch (const exception& err) {
474     error_msg = err.what();
475 }
476 catch (...) {
477     error_msg = "Unhandled_exception";
478 }
479
480 Test_OK = Test_OK && check_dump(ost, "Test_GetTotalDrivenKilometers()_error_buffer", error_msg.empty(), true);
481 error_msg.clear();
482
483 //Test ostream operator
484 try
485 {
486     Car* const testCar1 = new Car{ "Madza", Elektro, "WD40AHAH" };
487
488     testCar1->AddRecord({ { 2025y,October,13d }, 25 });
489     testCar1->AddRecord({ { 2025y,October,28d }, 34 });
490     Garage testGarage;
491     testGarage.AddVehicle(testCar1);
492
493     std::stringstream expect;
494     std::stringstream result;
495
496     testGarage.Print(expect);
497     result << testGarage;
498
499     Test_OK = Test_OK && check_dump(ost, "Test_ostream_operator", expect.str(), result.str());
500 }
501
502 catch (const string& err) {
503     error_msg = err;
504 }
505 catch (bad_alloc const& error) {
506     error_msg = error.what();
507 }
508 catch (const exception& err) {
509     error_msg = err.what();
510 }
511 catch (...) {
512     error_msg = "Unhandled_exception";
513 }
514
515 Test_OK = Test_OK && check_dump(ost, "Test_ostream_operator_error_buffer", error_msg.empty(), true);
516 error_msg.clear();
517
518 // Adding Car as nullptr;
519 try
520 {
521     Car* const testCar1 = nullptr;
522     Garage testGarage;
```

```
523     testGarage.AddVehicle(testCar1);
524 }
525 catch (const string& err) {
526     error_msg = err;
527 }
528 catch (...) {
529     error_msg = "Unhandled_exception";
530 }
531
532 Test_OK = Test_OK && check_dump(ost, "TestAdding_Car_as_nullptr;", error_msg, Garage::ERROR_NULLPTR);
533 error_msg.clear();
534
535 // Adding Deleting as nullptr;
536 try
537 {
538     Car* const testCar1 = nullptr;
539     Garage testGarage;
540     testGarage.DeleteVehicle(testCar1);
541 }
542 catch (const string& err) {
543     error_msg = err;
544 }
545 catch (...) {
546     error_msg = "Unhandled_exception";
547 }
548
549 Test_OK = Test_OK && check_dump(ost, "TestDeleting_Car_as_nullptr;", error_msg, Garage::ERROR_NULLPTR);
550 error_msg.clear();
551
552 // End of garage testing
553 ost << TestEnd;
554 return Test_OK;
555 }
556
557
558 static bool Test_Car(ostream& ost) {
559     assert(ost.good());
560
561     ost << TestStart;
562     bool Test_OK = true;
563     std::string error_msg;
564
565     // Test Fuel Type Getter
566     try
567     {
568         TFuel testType = Benzin;
569         Car testCar{ "Audi", testType, "SR770BA" };
570
571         Test_OK = Test_OK && check_dump(ost, "Test_car_fueltype", testCar.GetFuelType(), testType);
572     }
573     catch (const string& err) {
574         error_msg = err;
575     }
576     catch (bad_alloc const& error) {
577         error_msg = error.what();
578     }
579     catch (const exception& err) {
580         error_msg = err.what();
581     }
582     catch (...) {
583         error_msg = "Unhandled_exception";
584     }
585
586     Test_OK = Test_OK && check_dump(ost, "Test_car_fueltype_-_error_buffer", error_msg.empty(), true);
587     error_msg.clear();
588
589     // Test Plate Getter
590     try
591     {
592         TFuel testType = Benzin;
593         std::string testPlate = "SR770BA";
594         Car testCar{ "Audi", testType, testPlate };
595
596         Test_OK = Test_OK && check_dump(ost, "Test_car_plate", testCar.GetPlate(), testPlate);
597     }
```

```
598     catch (const string& err) {
599         error_msg = err;
600     }
601     catch (bad_alloc const& error) {
602         error_msg = error.what();
603     }
604     catch (const exception& err) {
605         error_msg = err.what();
606     }
607     catch (...) {
608         error_msg = "Unhandled_exception";
609     }
610
611     Test_OK = Test_OK && check_dump(ost, "Test_car_plate_-_error_buffer", error_msg.empty(), true);
612     error_msg.clear();
613
614     // Test Brand Getter
615     try
616     {
617         TFuel testType = Benzin;
618         std::string testPlate = "SR770BA";
619         std::string testBrand= "Steyr";
620         Car testCar{ testBrand, testType, testPlate };
621
622         Test_OK = Test_OK && check_dump(ost, "Test_car_brand", testCar.GetBrand(), testBrand);
623     }
624     catch (const string& err) {
625         error_msg = err;
626     }
627     catch (bad_alloc const& error) {
628         error_msg = error.what();
629     }
630     catch (const exception& err) {
631         error_msg = err.what();
632     }
633     catch (...) {
634         error_msg = "Unhandled_exception";
635     }
636
637     Test_OK = Test_OK && check_dump(ost, "Test_car_brand_-_error_buffer", error_msg.empty(), true);
638     error_msg.clear();
639
640     // Test Milage Getter
641     try
642     {
643         TFuel testType = Benzin;
644         std::string testPlate = "SR770BA";
645         std::string testBrand = "Steyr";
646         Car testCar{ testBrand, testType, testPlate };
647         size_t miles = 25;
648         testCar.AddRecord({ { 2025y,October,13d }, miles });
649
650         Test_OK = Test_OK && check_dump(ost, "Test_car_milage", testCar.GetMilage(), miles);
651     }
652     catch (const string& err) {
653         error_msg = err;
654     }
655     catch (bad_alloc const& error) {
656         error_msg = error.what();
657     }
658     catch (const exception& err) {
659         error_msg = err.what();
660     }
661     catch (...) {
662         error_msg = "Unhandled_exception";
663     }
664
665     Test_OK = Test_OK && check_dump(ost, "Test_car_milage_-_error_buffer", error_msg.empty(), true);
666     error_msg.clear();
667
668     // Test DriveRecord Getter
669     try
670     {
671         TFuel testType = Benzin;
672         std::string testPlate = "SR770BA";
```



```
673     std::string testBrand = "Steyr";
674     Car testCar{ testBrand, testType, testPlate };
675     size_t miles = 25;
676     DriveRecord driveRecord;
677     RecordEntry recordEntry = { { 2025y, October, 13d }, miles };
678     driveRecord.AddRecord(recordEntry);
679     testCar.AddRecord(recordEntry);
680
681     stringstream expect;
682     stringstream result;
683     driveRecord.Print(expect);
684     testCar.GetDriveRecord().Print(result);
685     Test_OK = Test_OK && check_dump(ost, "Test_car_driveRecord", expect.str(), result.str());
686 }
687 catch (const string& err) {
688     error_msg = err;
689 }
690 catch (bad_alloc const& error) {
691     error_msg = error.what();
692 }
693 catch (const exception& err) {
694     error_msg = err.what();
695 }
696 catch (...) {
697     error_msg = "Unhandled_exception";
698 }
699
700 Test_OK = Test_OK && check_dump(ost, "Test_car_driveRecord_error_buffer", error_msg.empty(), true);
701 error_msg.clear();
702
703 // Test Exception empty string
704 try
705 {
706     TFuel testType = Benzin;
707
708     Car testCar{ "", testType, "SB278FH" };
709 }
710 catch (const string& err) {
711     error_msg = err;
712 }
713 catch (bad_alloc const& error) {
714     error_msg = error.what();
715 }
716 catch (const exception& err) {
717     error_msg = err.what();
718 }
719 catch (...) {
720     error_msg = "Unhandled_exception";
721 }
722
723 Test_OK = Test_OK && check_dump(ost, "Test_Car_CTOR_empty_brand", Vehicle::ERROR_EMPTY_STRING, error_msg);
724 error_msg.clear();
725
726 try
727 {
728     TFuel testType = Benzin;
729
730     Car testCar{ "Audi", testType, "" };
731 }
732 catch (const string& err) {
733     error_msg = err;
734 }
735 catch (bad_alloc const& error) {
736     error_msg = error.what();
737 }
738 catch (const exception& err) {
739     error_msg = err.what();
740 }
741 catch (...) {
742     error_msg = "Unhandled_exception";
743 }
744
745 Test_OK = Test_OK && check_dump(ost, "Test_Car_CTOR_empty_plate", Vehicle::ERROR_EMPTY_STRING, error_msg);
746 error_msg.clear();
747
```

```
748     ost << TestEnd;
749     return Test_OK;
750 }
751 }
752
753
754 static bool Test_Bike(ostream& ost) {
755     assert(ost.good());
756
757     ost << TestStart;
758     bool Test_OK = true;
759     std::string error_msg;
760
761     // Test Fuel Type Getter
762     try
763     {
764         TFuel testType = Benzin;
765         Bike testCar{ "Audi", testType, "SR770BA" };
766
767         Test_OK = Test_OK && check_dump(ost, "Test_Bike_fueltype", testCar.GetFuelType(), testType);
768     }
769     catch (const string& err) {
770         error_msg = err;
771     }
772     catch (bad_alloc const& error) {
773         error_msg = error.what();
774     }
775     catch (const exception& err) {
776         error_msg = err.what();
777     }
778     catch (...) {
779         error_msg = "Unhandled_exception";
780     }
781
782     Test_OK = Test_OK && check_dump(ost, "Test_Bike_fueltype_{}_error_buffer", error_msg.empty(), true);
783     error_msg.clear();
784
785     // Test Plate Getter
786     try
787     {
788         TFuel testType = Benzin;
789         std::string testPlate = "SR770BA";
790         Bike testCar{ "Audi", testType, testPlate };
791
792         Test_OK = Test_OK && check_dump(ost, "Test_Bike_plate", testCar.GetPlate(), testPlate);
793     }
794     catch (const string& err) {
795         error_msg = err;
796     }
797     catch (bad_alloc const& error) {
798         error_msg = error.what();
799     }
800     catch (const exception& err) {
801         error_msg = err.what();
802     }
803     catch (...) {
804         error_msg = "Unhandled_exception";
805     }
806
807     Test_OK = Test_OK && check_dump(ost, "Test_Bike_plate_{}_error_buffer", error_msg.empty(), true);
808     error_msg.clear();
809
810     // Test Brand Getter
811     try
812     {
813         TFuel testType = Benzin;
814         std::string testPlate = "SR770BA";
815         std::string testBrand = "Steyr";
816         Bike testCar{ testBrand, testType, testPlate };
817
818         Test_OK = Test_OK && check_dump(ost, "Test_Bike_brand", testCar.GetBrand(), testBrand);
819     }
820     catch (const string& err) {
821         error_msg = err;
822     }
}
```

```
823     catch (bad_alloc const& error) {
824         error_msg = error.what();
825     }
826     catch (const exception& err) {
827         error_msg = err.what();
828     }
829     catch (...) {
830         error_msg = "Unhandled_exception";
831     }
832
833     Test_OK = Test_OK && check_dump(ost, "Test_Bike_brand_-_error_buffer", error_msg.empty(), true);
834     error_msg.clear();
835
836     // Test Milage Getter
837     try
838     {
839         TFuel testType = Benzin;
840         std::string testPlate = "SR770BA";
841         std::string testBrand = "Steyr";
842         Car testCar{ testBrand, testType, testPlate };
843         size_t miles = 25;
844         testCar.AddRecord({ { 2025y,October,13d }, miles });
845
846         Test_OK = Test_OK && check_dump(ost, "Test_Bike_milage", testCar.GetMilage(), miles);
847     }
848     catch (const string& err) {
849         error_msg = err;
850     }
851     catch (bad_alloc const& error) {
852         error_msg = error.what();
853     }
854     catch (const exception& err) {
855         error_msg = err.what();
856     }
857     catch (...) {
858         error_msg = "Unhandled_exception";
859     }
860
861     Test_OK = Test_OK && check_dump(ost, "Test_Bike_milage_-_error_buffer", error_msg.empty(), true);
862     error_msg.clear();
863
864     // Test DriveRecord Getter
865     try
866     {
867         TFuel testType = Benzin;
868         std::string testPlate = "SR770BA";
869         std::string testBrand = "Steyr";
870         Bike testCar{ testBrand, testType, testPlate };
871         size_t miles = 25;
872         DriveRecord driveRecord;
873         RecordEntry recordEntry = { { 2025y,October,13d }, miles };
874         driveRecord.AddRecord(recordEntry);
875         testCar.AddRecord(recordEntry);
876
877         stringstream expect;
878         stringstream result;
879         driveRecord.Print(expect);
880         testCar.GetDriveRecord().Print(result);
881         Test_OK = Test_OK && check_dump(ost, "Test_Bike_driveRecord", expect.str(), result.str());
882     }
883     catch (const string& err) {
884         error_msg = err;
885     }
886     catch (bad_alloc const& error) {
887         error_msg = error.what();
888     }
889     catch (const exception& err) {
890         error_msg = err.what();
891     }
892     catch (...) {
893         error_msg = "Unhandled_exception";
894     }
895
896     Test_OK = Test_OK && check_dump(ost, "Test_Bike_driveRecord_-_error_buffer", error_msg.empty(), true);
897     error_msg.clear();
```

```
898
899 // Test Exception empty string
900 try
901 {
902     TFuel testType = Benzin;
903
904     Bike testCar{ "", testType, "SB278FH" };
905 }
906 catch (const string& err) {
907     error_msg = err;
908 }
909 catch (bad_alloc const& error) {
910     error_msg = error.what();
911 }
912 catch (const exception& err) {
913     error_msg = err.what();
914 }
915 catch (...) {
916     error_msg = "Unhandled_exception";
917 }
918
919 Test_OK = Test_OK && check_dump(ost, "Test_Bike_CTOR_empty_brand", Vehicle::ERROR_EMPTY_STRING, error_msg);
920 error_msg.clear();
921
922 try
923 {
924     TFuel testType = Benzin;
925
926     Bike testCar{ "Audi", testType, "" };
927 }
928 catch (const string& err) {
929     error_msg = err;
930 }
931 catch (bad_alloc const& error) {
932     error_msg = error.what();
933 }
934 catch (const exception& err) {
935     error_msg = err.what();
936 }
937 catch (...) {
938     error_msg = "Unhandled_exception";
939 }
940
941 Test_OK = Test_OK && check_dump(ost, "Test_Bike_CTOR_empty_plate", Vehicle::ERROR_EMPTY_STRING, error_msg);
942 error_msg.clear();
943
944 ost << TestEnd;
945 return Test_OK;
946 }
947
948 static bool Test_Truck(ostream& ost){
949     assert(ost.good());
950
951     ost << TestStart;
952     bool Test_OK = true;
953     std::string error_msg;
954
955     // Test Fuel Type Getter
956     try
957     {
958         TFuel testType = Benzin;
959         Truck testCar{ "Audi", testType, "SR770BA" };
960
961         Test_OK = Test_OK && check_dump(ost, "Test_Truck_fueltype", testCar.GetFuelType(), testType);
962     }
963     catch (const string& err) {
964         error_msg = err;
965     }
966     catch (bad_alloc const& error) {
967         error_msg = error.what();
968     }
969     catch (const exception& err) {
970         error_msg = err.what();
971     }
972     catch (...) {
```

```
973     error_msg = "Unhandled_exception";
974 }
975
976 Test_OK = Test_OK && check_dump(ost, "Test_Truck_fueltype_-_error_buffer", error_msg.empty(), true);
977 error_msg.clear();
978
979 // Test Plate Getter
980 try
981 {
982     TFuel testType = Benzin;
983     std::string testPlate = "SR770BA";
984     Truck testCar{ "Audi", testType, testPlate };
985
986     Test_OK = Test_OK && check_dump(ost, "Test_Truck_plate", testCar.GetPlate(), testPlate);
987 }
988 catch (const string& err) {
989     error_msg = err;
990 }
991 catch (bad_alloc const& error) {
992     error_msg = error.what();
993 }
994 catch (const exception& err) {
995     error_msg = err.what();
996 }
997 catch (...) {
998     error_msg = "Unhandled_exception";
999 }
1000
1001 Test_OK = Test_OK && check_dump(ost, "Test_Truck_plate_-_error_buffer", error_msg.empty(), true);
1002 error_msg.clear();
1003
1004 // Test Brand Getter
1005 try
1006 {
1007     TFuel testType = Benzin;
1008     std::string testPlate = "SR770BA";
1009     std::string testBrand = "Steyr";
1010     Truck testCar{ testBrand, testType, testPlate };
1011
1012     Test_OK = Test_OK && check_dump(ost, "Test_Truck_brand", testCar.GetBrand(), testBrand);
1013 }
1014 catch (const string& err) {
1015     error_msg = err;
1016 }
1017 catch (bad_alloc const& error) {
1018     error_msg = error.what();
1019 }
1020 catch (const exception& err) {
1021     error_msg = err.what();
1022 }
1023 catch (...) {
1024     error_msg = "Unhandled_exception";
1025 }
1026
1027 Test_OK = Test_OK && check_dump(ost, "Test_car_brand_-_error_buffer", error_msg.empty(), true);
1028 error_msg.clear();
1029
1030 // Test Milage Getter
1031 try
1032 {
1033     TFuel testType = Benzin;
1034     std::string testPlate = "SR770BA";
1035     std::string testBrand = "Steyr";
1036     Truck testCar{ testBrand, testType, testPlate };
1037     size_t miles = 25;
1038     testCar.AddRecord({ { 2025y,October,13d }, miles });
1039     testCar.AddRecord({ { 2025y,October,13d }, miles });
1040
1041     Test_OK = Test_OK && check_dump(ost, "Test_Truck_milage", testCar.GetMilage(), 2*miles);
1042 }
1043 catch (const string& err) {
1044     error_msg = err;
1045 }
1046 catch (bad_alloc const& error) {
1047     error_msg = error.what();
```

```
1048     }
1049     catch (const exception& err) {
1050         error_msg = err.what();
1051     }
1052     catch (...) {
1053         error_msg = "Unhandled_exception";
1054     }
1055
1056     Test_OK = Test_OK && check_dump(ost, "Test_Truck_milage_-_error_buffer", error_msg.empty(), true);
1057     error_msg.clear();
1058
1059     // Test DriveRecord Getter
1060     try
1061     {
1062         TFuel testType = Benzin;
1063         std::string testPlate = "SR770BA";
1064         std::string testBrand = "Steyr";
1065         Truck testCar{ testBrand, testType, testPlate };
1066         size_t miles = 25;
1067         DriveRecord driveRecord;
1068         RecordEntry recordEntry = { { 2025y,October,13d }, miles };
1069         driveRecord.AddRecord(recordEntry);
1070         testCar.AddRecord(recordEntry);
1071
1072         stringstream expect;
1073         stringstream result;
1074         driveRecord.Print(expect);
1075         testCar.GetDriveRecord().Print(result);
1076         Test_OK = Test_OK && check_dump(ost, "Test_truck_driveRecord", expect.str(), result.str());
1077     }
1078     catch (const string& err) {
1079         error_msg = err;
1080     }
1081     catch (bad_alloc const& error) {
1082         error_msg = error.what();
1083     }
1084     catch (const exception& err) {
1085         error_msg = err.what();
1086     }
1087     catch (...) {
1088         error_msg = "Unhandled_exception";
1089     }
1090
1091     Test_OK = Test_OK && check_dump(ost, "Test_truck_driveRecord_-_error_buffer", error_msg.empty(), true);
1092     error_msg.clear();
1093
1094     // Test Exception empty string
1095     try
1096     {
1097         TFuel testType = Benzin;
1098
1099         Truck testCar{ "", testType, "SB278FH" };
1100     }
1101     catch (const string& err) {
1102         error_msg = err;
1103     }
1104     catch (bad_alloc const& error) {
1105         error_msg = error.what();
1106     }
1107     catch (const exception& err) {
1108         error_msg = err.what();
1109     }
1110     catch (...) {
1111         error_msg = "Unhandled_exception";
1112     }
1113
1114     Test_OK = Test_OK && check_dump(ost, "Test_truck_CTOR_empty_brand", Vehicle::ERROR_EMPTY_STRING,error_msg);
1115     error_msg.clear();
1116
1117     try
1118     {
1119         TFuel testType = Benzin;
1120
1121         Truck testCar{ "Audi", testType, "" };
1122     }
```

```
1123     catch (const string& err) {
1124         error_msg = err;
1125     }
1126     catch (bad_alloc const& error) {
1127         error_msg = error.what();
1128     }
1129     catch (const exception& err) {
1130         error_msg = err.what();
1131     }
1132     catch (...) {
1133         error_msg = "Unhandled_exception";
1134     }
1135
1136     Test_OK = Test_OK && check_dump(ost, "Test_truck_CTOR_empty_plate", Vehicle::ERROR_EMPTY_STRING, error_msg);
1137     error_msg.clear();
1138
1139
1140     ost << TestEnd;
1141     return Test_OK;
1142 }
```