Systemdokumentation
Projekt Fuhrpark

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 15. Oktober 2025

Inhaltsverzeichnis

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@fh-hagenberg.at
- Susi Sorglos, Matr.-Nr.: yyyy, E-Mail: Susi.Sorglos@fh-hagenberg.at

1.2 Aufteilung der Verantwortlichkeitsbereiche

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

    - Bike und
    - Truck
  - Implementierung des Testtreibers
  - Dokumentation

1.3 Aufwand

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

2 Anforderungsdefinition (Systemspezifikation)

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

Funktionen des Fahrtenbuches

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

Funktionen des Fuhrparks

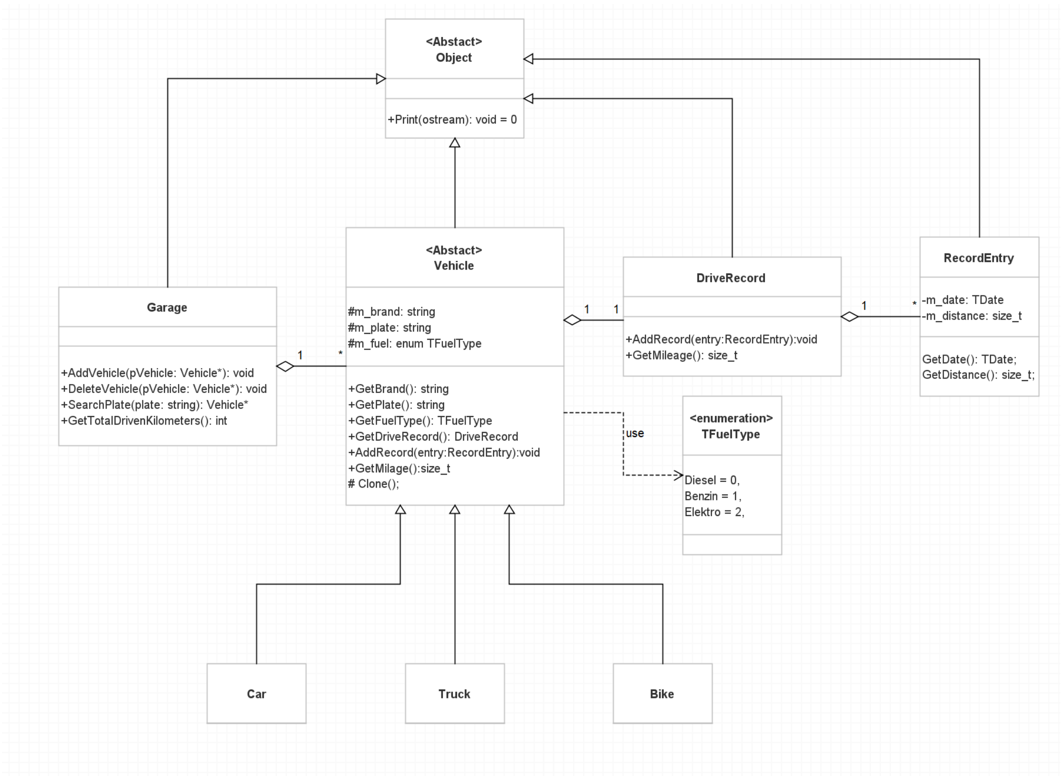
- Hinzufügen und löschen eines Fahrzeuges
- Ausgabe aller Fahrzeugdaten inklusive der Fahrtenbucheinträge.
- Suchen nach einem Fahrzeug mit dessen Kennzeichen.
- Berechnung der Gesamtkilometer aller Fahrzeuge im Fuhrpark.

Funktionen der Fahrzeuge

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

3 Systementwurf

3.1 Klassendiagramm



3.2 Designentscheidungen

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

Als Container für die Speicherung der Fahrzeuge in der Klasse **Garage** wurde der Vektor verwendet. Dieser erlaubt es schnell Fahrzeuge hinzuzufügen, und das Suchen geschieht relativ schnell in O(n). Einzig und allein, das Löschen aus der Mitte des Vektors stellt bei 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

```

*******************************************
             TESTCASE START
*******************************************

Test RecordEntry Get Date
[Test OK] Result: (Expected: 2025-10-13 == Result: 2025-10-13)

Test RecordEntry Get Distance
[Test OK] Result: (Expected: 150 == Result: 150)

Test RecordEntry Print
[Test OK] Result: (Expected: true == Result: true)

Test RecordEntry Exception Bad Ostream
[Test OK] Result: (Expected: ERROR: Provided Ostream is bad == Result:
    ↪ ERROR: Provided Ostream is bad)


*******************************************


*******************************************
             TESTCASE START
*******************************************

Test DriveRecord Print Sorted and Add Record
[Test OK] Result: (Expected: true == Result: true)

Test DriveRecord Get Milage
[Test OK] Result: (Expected: 450 == Result: 450)

Test DriveRecord Exception Bad Ostream
[Test OK] Result: (Expected: ERROR: Provided Ostream is bad == Result:
    ↪ ERROR: Provided Ostream is bad)

Test DriveRecord Empty Print
[Test OK] Result: (Expected: No Exception == Result: No Exception)


*******************************************

```

```

*******************************************
             TESTCASE START
*******************************************

vehicle plate search
[Test OK] Result: (Expected: 000001FBA98DFCC0 == Result: 000001FBA98DFCC0)

Test garage plate search - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test garage plate search invalid plate
[Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)

Test garage plate search invalid plate - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test Garage Print
[Test OK] Result: (Expected:
Fahrzeugart:  PKW
Marke:        UAZ
Kennzeichen:  SR770BA
13.10.2025:    25 km
 == Result:
Fahrzeugart:  PKW
Marke:        UAZ
Kennzeichen:  SR770BA
13.10.2025:    25 km
)

Test garage print - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test garage print empty garage
[Test OK] Result: (Expected: true == Result: true)

Test garage print empty garage - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test Delete Vehicle
[Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)

Test garage print - error buffer
[Test OK] Result: (Expected: true == Result: true)
```

```

Test Delete Vehicle
[Test OK] Result: (Expected: 0000000000000000 == Result: 0000000000000000)

Test Delete Vehicle - error buffer
[Test OK] Result: (Expected: true == Result: true)

TTest GetTotalDrivenKilometers()
[Test OK] Result: (Expected: 118 == Result: 118)

Test GetTotalDrivenKilometers() - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test ostream operator
[Test OK] Result: (Expected:
Fahrzeugart:  Auto
Marke:        Madza
Kennzeichen:  WD40AHAH
13.10.2025:    25 km
28.10.2025:    34 km
 == Result:
Fahrzeugart:  Auto
Marke:        Madza
Kennzeichen:  WD40AHAH
13.10.2025:    25 km
28.10.2025:    34 km
)

Test ostream operator - error buffer
[Test OK] Result: (Expected: true == Result: true)


*******************************************


*******************************************
             TESTCASE START
*******************************************

Test Car Print without record
[Test OK] Result: (Expected: true == Result: true)

TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```
/*******************************************************************//**
 * \file   Object.hpp
 * \brief  Root of all Objects
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#ifndef OBJECT_HPP
#define OBJECT_HPP

#include <iostream>

class Object {
public:

   inline static const std::string ERROR_BAD_OSTREAM = "ERROR:␣Provided␣Ostream␣is␣bad";
   inline static const std::string ERROR_FAIL_WRITE = "ERROR:␣Fail␣to␣write␣on␣provided␣Ostream";

   /**
    * Virtual DTOR, once virtual always virtual.
    */
   virtual ~Object() = default;

   /**
    * .
    */
   virtual std::ostream& Print(std::ostream & ost = std::cout) const = 0;

protected:
   Object() = default;
};

#endif // !1
```

6.2 RecordEntry.hpp

```
/*******************************************************************//**
 * \file   RecordEntry.hpp
 * \brief  Class that defines an entry in a dirve record.
 * \brief  This record entry is used by the drive record class.
 * \brief  The drive record class stores multiple record entries.
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#ifndef RECORD_ENTRY_HPP
#define RECORD_ENTRY_HPP


#include <chrono>
#include "Object.hpp"

// Using Statement for date type
using TDate = std::chrono::year_month_day;

class RecordEntry : public Object {
public:

   /**
```

```
     * \brief CTOR of a drive record.
     *
     * \param date : date when the drive happend
     * \param distance : the distance of the drive in km
     */
    RecordEntry(const TDate & date,const size_t & distance) : m_date{ date }, m_distance{ distance } {}

    /**
     * \brief Getter of the distance member of the Record Entry Class.
     *
     * \return Distance of this Record Entry
     */
    size_t GetDistance() const;

    /**
     * \brief Getter of the data member of the Record Entry Class.
     *
     * \return Date of this Record Entry
     */
    TDate GetDate() const;

    /**
     * \brief Formatted output of this Record Entry on an ostream.
     *
     * \param ost : Refernce to an ostream where the Entry should be printed at.
     * \return Referenced ostream
     */
    virtual std::ostream& Print(std::ostream& ost = std::cout) const override;

    /**
     * \brief less than operater, is used for storing the Entries in a multiset.
     *
     * \param rh : Righthandside of the less than operator
     * \return true:  left hand side is less than the right hand side.
     * \return false: left hand side is greather or equal than the right hand side.
     */
    bool operator<(const RecordEntry& rh) const;

private:
    TDate m_date;      // private date member
    size_t m_distance;   // private distance member
};


#endif // !1
```

6.3 RecordEntry.cpp

```
/*******************************************************************//**
 * \file   RecordEntry.cpp
 * \brief  Implementation of a Record Entry
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#include "RecordEntry.hpp"

using namespace std;

/**
* \brief Getter of the distance member of the Record Entry Class.
*
* \return Distance of this Record Entry
*/
size_t RecordEntry::GetDistance() const
{
    return m_distance;
}
```

```cpp

/**
* \brief Getter of the data member of the Record Entry Class.
*
* \return Date of this Record Entry
*/
TDate RecordEntry::GetDate() const
{
   return m_date;
}

/**
* \brief Formatted output of this Record Entry on an ostream.
*
* \param ost : Refernce to an ostream where the Entry should be printed at.
* \return Referenced ostream
*/
std::ostream& RecordEntry::Print(std::ostream& ost) const
{
   if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;

   ost << std::setfill('0')<< right << std::setw(2) << m_date.day() << "."
      << std::setw(2) << static_cast<unsigned>(m_date.month()) << "."
      << std::setw(4) << m_date.year() << ":" << std::setfill(' ')
      << std::setw(6) << m_distance << " km\n";

   if (ost.fail()) throw Object::ERROR_FAIL_WRITE;

   return ost;
}

/**
* \brief less than operater, is used for storing the Entries in a multiset.
*
* \param rh : Righthandside of the less than operator
* \return true:  left hand side is less than the right hand side.
* \return false: left hand side is greather or equal than the right hand side.
*/
bool RecordEntry::operator<(const RecordEntry& rh) const
{
   return m_date < rh.m_date;
}
```

6.4 DriveRecord.hpp

```cpp
/*****************************************************************//**
 * \file   DriveRecord.hpp
 * \brief  This Class implements a drive record book which holds multiple
 * \brief  record entries in a TCont, which is defined as a multiset.
 * \brief  The multiset is used because it stores the data sorted.
 * \brief  This sorting mandatory because the entries should be date ascending.
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#ifndef DRIVE_RECORD_HPP
#define DRIVE_RECORD_HPP

#include <set>
#include "RecordEntry.hpp"
#include "Object.hpp"

// Using statement for the used container to store the record entries
using TCont = std::multiset<RecordEntry>;

class DriveRecord : public Object {
public:

   /**
    * \brief Methode for adding a record entry to a collection of drive records.
    *
    * \param entry : Record to be added to the colletion
    */
   void AddRecord(const RecordEntry & entry);

   /**
    * \brief This methode adds up all the distance of all record entries.
    *
    * \return the sum of all distances in the collection
    */
   size_t GetMilage() const;

   /**
    * \brief Formatted output of all Record Entry on an ostream.
    *
    * \param ost : Refernce to an ostream where the Entries should be printed at.
    * \return Referenced ostream
    */
   virtual std::ostream& Print(std::ostream& ost = std::cout) const override;

private:

   TCont m_driveRecords;
};


#endif // !1
```

6.5 DriveRecord.cpp

```
/*****************************************************************//**
 * \file   DriveRecord.cpp
 * \brief  Implementation of a Drive Record
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#include <numeric>
#include <algorithm>
#include "DriveRecord.hpp"

/**
* \brief Methode for adding a record entry to a collection of drive records.
*
* \param entry : Record to be added to the colletion
*/
void DriveRecord::AddRecord(const RecordEntry& entry)
{
   m_driveRecords.insert(entry);
}

/**
* \brief This methode adds up all the distance of all record entries.
*
* \return the sum of all distances in the collection
*/
size_t DriveRecord::GetMilage() const
{
   // use std accumulet + lambda to calc the total Milage
   return std::accumulate(m_driveRecords.cbegin(), m_driveRecords.cend(), static_cast<size_t>(0),
      [](const size_t val,const RecordEntry& entry) {return val + entry.GetDistance();});
}

/**
* \brief Formatted output of all Record Entry on an ostream.
*
```

```
* \param ost : Refernce to an ostream where the Entries should be printed at.
* \return Referenced ostream
*/
std::ostream& DriveRecord::Print(std::ostream& ost) const
{
   if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;

   std::for_each(m_driveRecords.cbegin(), m_driveRecords.cend(), [&](const RecordEntry& entry) {entry.Print(ost);});

   if (ost.fail()) throw Object::ERROR_FAIL_WRITE;

   return ost;
}
```

6.6 Garage.hpp

```
/*****************************************************************//**
 * \file   Vehicle.hpp
 * \brief  This Class implements a polymorph container containing
 * \brief  all derivatives of the 'Vehicle' Class.
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#include <vector>
#include <string>
#include "Object.hpp"
#include "Vehicle.hpp"

using TGarageCont = std::vector<Vehicle const *>;

class Garage : public Object {
public:

    inline static const std::string ERROR_NULLPTR= "ERROR:␣Passed␣in␣Nullptr!";

    Garage() = default;

    /**
     * \brief Adds a vehicle to a vehicle collection.
     * \brief A specific vehicle is passed in and casted to a vehicle Pointer.
     * \brief This is allowed because Car,Truck and Bike are derived from Vehicle.
     * \brief A car is a Vehicle.
     * \brief This casted Pointer is copied ito this methode and added to the collection
     * \param newVehicle : Pointer to a Vehicle.
     */
    void AddVehicle(Vehicle const * const newVehicle);

    /**
     * \brief deletes Vehicle inside garage from provided pointer.
     * \param pVehicle : Pointer to a Vehicle.
     */
    void DeleteVehicle(Vehicle const * const pVehicle);

    /**
     * \brief Functions searches for vehicle with matching plate.
     * \param pVehicle : Pointer to a Vehicle.
     * \return pointer to the vehicle inside the garage
     */
    Vehicle const * const SearchPlate(const std::string & plate) const;

    /**
     * \brief Formatted of every car and its drive record
     * \param ost : Refernce to an ostream where the Entry should be printed at.
     * \return Referenced ostream
     */
    std::ostream& Print(std::ostream& ost = std::cout) const override;

    /**
```

```
     * \brief Calculates sum of every kilometer every vehicle has driven
     * \brief in total
     * \return size_t total kilometers
     */
    size_t GetTotalDrivenKilometers() const;


    // TODO: Copy / assignement implementation
    // TODO: overload for output operator <<

    Garage(const Garage&);
    void operator=(Garage garage);

    ~Garage();

private:
    TGarageCont m_vehicles;
};

/**
 * \brief Override for ostream operator
 * \return ostream
 */
std::ostream& operator <<(std::ostream& ost, Garage& garage);
```

6.7 Garage.cpp

```
/*****************************************************************//**
 * \file   Vehicle.c
 * \brief  Implementation of Garage.h
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#include "Garage.hpp"
#include <algorithm>
#include <numeric>

void Garage::AddVehicle(Vehicle const * const newVehicle)
{
    if (newVehicle == nullptr) throw ERROR_NULLPTR;
    // Add the new vehicle to the collection.
    m_vehicles.push_back(newVehicle);
}

/**
 * \brief deletes Vehicle inside garage from provided pointer.
 * \param pVehicle : Pointer to a Vehicle.
 */
void Garage::DeleteVehicle(Vehicle const * const pVehicle)
{
    // if pVehicle is inside m_Vehicles -> erase and free
    auto itr = std::find(m_vehicles.begin(), m_vehicles.end(), pVehicle);
    if (itr != m_vehicles.end())
    {
        m_vehicles.erase(itr);
        delete pVehicle;
    }
}

const Vehicle* const Garage::SearchPlate(const std::string & plate) const
{
    for (const auto &elem : m_vehicles)
    {
        if (elem->GetPlate() == plate)
        {
            return elem;
        }
    }

    return nullptr;
}

std::ostream& Garage::Print(std::ostream& ost) const
{
    if (!ost.good())
        throw Object::ERROR_BAD_OSTREAM;

    for (auto& elem : m_vehicles)
    {
        elem->Print(ost);
    }

    if (ost.fail())
        throw Object::ERROR_FAIL_WRITE;

    return ost;
}

size_t Garage::GetTotalDrivenKilometers() const
{
    size_t sum = std::accumulate(m_vehicles.cbegin(), m_vehicles.cend(), static_cast<size_t>(0),
        [](auto last_val, auto vehicle) {
            return last_val + vehicle->GetMilage();
        });
    return sum;
}

Garage::Garage(const Garage&)
{
    for_each(
        m_vehicles.cbegin(), m_vehicles.cend(),
        [&](auto v) {AddVehicle(v->Clone());
        });
}

void Garage::operator=(Garage garage)
{
    std::swap(m_vehicles, garage.m_vehicles);
}

Garage::~Garage()
{
    for (auto elem : m_vehicles)
    {
        delete elem;
    }

    m_vehicles.clear();
}

std::ostream& operator<<(std::ostream& ost, Garage& garage)
{
    garage.Print(ost);
    return ost;
}
```

6.8 Vehicle.hpp

```
/*******************************************************************//**
 * \file   Vehicle.hpp
 * \brief  This class imlements an abstract vehicle which is used in the
 * \brief  Garage class. It implements all the core featues of a vehicle
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
```

```
#ifndef VEHICLE_HPP
#define VEHICLE_HPP

#include "Object.hpp"
#include "DriveRecord.hpp"

// Enumeration for a fuel type
enum TFuel {
   Diesel = 0,
   Benzin = 1,
   Elektro = 2,
};

class Vehicle: public Object {
public:

   /**
    * \brief Getter for the brand member.
    *
    * \return string with the brand name
    */
   std::string GetBrand() const;

   /**
    * \brief Getter for the plate member.
    *
    * \return string with the plate name
    */
   std::string GetPlate() const;

   /**
    * \brief Getter for the fuel member.
    *
    * \return TFuel with the specified fuel type
    */
   TFuel GetFuelType() const;

   /**
    * \brief Getter for the drive record.
    *
    * \return const refernce to the drive record
    */
   const DriveRecord & GetDriveRecord() const;

   /**
    * \brief Methode for adding a record entry to the drive record collection.
    *
    * \param entry : Entry which should be added to the drive recod
    */
   void AddRecord(const RecordEntry& entry);

   /**
    * \brief Getter for the total milage of a vehicle.
    *
    * \return Total milage of a vehicle
    */
   size_t GetMilage() const;

   /**
    * @brief Creates a clone of the vehicle.
    *
    * \return a excat replicate of a vehicle
    */
   virtual Vehicle const* Clone() const = 0;

protected:

   /**
    * \brief protected CTOR of a vehicle.
    * \brief protected because it is a abstract class
    *
    * \param brand : string that represents the brand of the vehicle
    * \param fuelType : Fuel type of the vehicle
    */
   Vehicle(const std::string& brand, const TFuel& fuelType, const std::string& plate) : m_brand{ brand }, m_fuel{ fuelType }, m_plate{p

private:
   std::string m_brand;
   std::string m_plate;
   TFuel m_fuel;
   DriveRecord m_record;
};


#endif // !1
```

6.9 Vehicle.cpp

```
/*****************************************************************//**
 * \file   Vehicle.cpp
 * \brief  Implementation of the abstract vehicle class
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#include "Vehicle.hpp"

/**
* \brief Getter for the brand member.
*
* \return string with the brand name
*/
std::string Vehicle::GetBrand() const
{
    return m_brand;
}

/**
* \brief Getter for the plate member.
*
* \return string with the plate name
*/
std::string Vehicle::GetPlate() const
{
    return m_plate;
}

/**
* \brief Getter for the fuel member.
*
* \return TFuel with the specified fuel type
*/
TFuel Vehicle::GetFuelType() const
{
    return m_fuel;
}

/**
* \brief Getter for the drive record.
*
* \return const refernce to the drive record
*/
const DriveRecord & Vehicle::GetDriveRecord() const
{
    return m_record;
}

/**
* \brief Methode for adding a record entry to the drive record collection.
*
* \param entry : Entry which should be added to the drive recod
*/
void Vehicle::AddRecord(const RecordEntry& entry)
```

```
{
    m_record.AddRecord(entry);
}

/**
* \brief Getter for the total milage of a vehicle.
*
* \return Total milage of a vehicle
*/
size_t Vehicle::GetMilage() const
{
    return m_record.GetMilage();
}
```

6.10 Car.hpp

```
#ifndef CAR_HPP
#define CAR_HPP

#include "Vehicle.hpp"

class Car : public Vehicle {
public:

   /**
    * \brief CTOR of a CAR -> calles the Base Class vehicle CTOR.
    *
    * \param brand string that identifies the brand.
    * \param fuelType Fueltype of the Car
    * \param plate string that identifies the plate.
    */
   Car(const std::string & brand,const TFuel & fuelType, const std::string & plate) : Vehicle(brand, fuelType,plate) {}

   /**
    * \brief Function that print all the vehicle specific info with the drive record.
    *
    * \param ost where the data should be printed at
    * \return referenced ostream
    */
   virtual std::ostream& Print(std::ostream& ost = std::cout) const override;

   /**
    * @brief Creates a clone of the vehicle.
    *
    * \return a excat replicate of a vehicle
    */
   virtual Vehicle const* Clone() const;

private:
};


#endif // !1
```

6.11 Car.cpp

```
/*****************************************************************//**
 * \file   Car.cpp
 * \brief  Implemetation of a Car
 *
 * \author Simon
 * \date   October 2025
 ********************************************************************/
#include "Car.hpp"

using namespace std;

/**
* \brief Function that print all the vehicle specific info with the drive record.
*
* \param ost where the data should be printed at
* \return referenced ostream
*/
std::ostream& Car::Print(std::ostream& ost) const
{
   if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;

   ost <<endl<< left << setw(14) << "Fahrzeugart:" << "PKW" << endl;
   ost << left << setw(14) << "Marke:" << GetBrand() << endl;
   ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;
   GetDriveRecord().Print(ost);

      if (ost.fail()) throw Object::ERROR_FAIL_WRITE;

   return ost;
}

/**
* @brief Creates a clone of the vehicle.
*
* \return a excat replicate of a vehicle
*/
Vehicle const* Car::Clone() const
{
   return new Car(*this);
}
```

6.12 Truck.hpp

```
#ifndef TRUCK_HPP
#define TRUCK_HPP

#include "Vehicle.hpp"

class Truck : public Vehicle {
public:

   /**
    * \brief CTOR of a Truck -> calles the Base Class vehicle CTOR.
    *
    * \param brand string that identifies the brand.
    * \param fuelType Fueltype of the Truck
    * \param plate string that identifies the plate.
    */
   Truck(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}

   /**
    * \brief Function that print all the vehicle specific info with the drive record.
    *
    * \param ost where the data should be printed at
    * \return referenced ostream
    */
   virtual std::ostream& Print(std::ostream& ost = std::cout) const override;

   /**
    * @brief Creates a clone of the vehicle.
    *
    * \return a excat replicate of a vehicle
    */
   virtual Vehicle const* Clone() const;

private:
};


#endif
```

6.13 Truck.cpp

```
/*****************************************************************//**
 * \file   Truck.cpp
 * \brief  Implementation of a Truck
 *
 * \author Simon
 * \date   October 2025
 *********************************************************************/
#include "Truck.hpp"

using namespace std;

/**
* \brief Function that print all the vehicle specific info with the drive record.
*
* \param ost where the data should be printed at
* \return referenced ostream
*/
std::ostream& Truck::Print(std::ostream& ost) const
{
   if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;

   ost << endl << left << setw(14) << "Fahrzeugart:" << "LKW" << endl;
   ost << left << setw(14) << "Marke:" << GetBrand() << endl;
   ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;
   GetDriveRecord().Print(ost);

   if (ost.fail()) throw Object::ERROR_FAIL_WRITE;

   return ost;
}

/**
* @brief Creates a clone of the vehicle.
*
* \return a excat replicate of a vehicle
*/
Vehicle const* Truck::Clone() const
{
   return new Truck(*this);
}
```

6.14 Bike.hpp

```
#ifndef BIKE_HPP
#define BIKE_HPP

#include "Vehicle.hpp"

class Bike : public Vehicle {
public:

   /**
    * \brief CTOR of a Bike -> calles the Base Class vehicle CTOR.
    *
    * \param brand string that identifies the brand.
    * \param fuelType Fueltype of the Bike
    * \param plate string that identifies the plate.
    */
   Bike(const std::string& brand, const TFuel& fuelType, const std::string& plate) : Vehicle(brand, fuelType, plate) {}

   /**
    * \brief Function that print all the vehicle specific info with the drive record.
    *
    * \param ost where the data should be printed at
    * \return referenced ostream
    */
   virtual std::ostream& Print(std::ostream& ost = std::cout) const override;

   /**
    * @brief Creates a clone of the vehicle.
    *
    * \return a excat replicate of a vehicle
    */
   virtual Vehicle const* Clone() const;

private:
};


#endif
```

6.15 Bike.cpp

```
/*****************************************************************//**
 * \file   Bike.cpp
 * \brief  Implementation of the Bike Class
 *
 * \author Simon
 * \date   October 2025
 *********************************************************************/
#include "Bike.hpp"

using namespace std;

/**
* \brief Function that print all the vehicle specific info with the drive record.
*
* \param ost where the data should be printed at
* \return referenced ostream
*/
std::ostream& Bike::Print(std::ostream& ost) const
{
    if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;

    ost << endl << left << setw(14) << "Fahrzeugart:" << "Motorrad" << endl;
    ost << left << setw(14) << "Marke:" << GetBrand() << endl;
    ost << left << setw(14) << "Kennzeichen:" << GetPlate() << endl;
    GetDriveRecord().Print(ost);

    if (ost.fail()) throw Object::ERROR_FAIL_WRITE;

    return ost;
}

/**
*@brief Creates a clone of the vehicle.
*
* \return a excat replicate of a vehicle
*/
```

```
Vehicle const* Bike::Clone() const
{
   return new Bike(*this);
}
```