Systemdokumentation Projekt Fuhrpark

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 12. 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: geschÃ¤tzt 10 Ph / tatsÃ¤chlich x Ph
- Simon Vogelhuber: geschÃ¤tzt x Ph / tatsÃ¤chlich 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 den Kilometerstand 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 Gesamtkilomenter aller Fahrzeuge im Fuhrpark.

3 Systementwurf

3.1 Klassendiagramm

Hier wird das Klassendiagramm eingefügt. Sollte dieses nicht auf eine A4-Seite passen, so kann es in eine eingene pdf-Datei ausgelagert werden. Verweisen Sie an dieser Stelle auf diese Datei.

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 implemeniert wurde, lassen sich die einzelnen Einträge schnell und einfach erweitern.

Sie beantworten meist folgende Fragen:

- Warum wurde die Klassenhierarchie so gewÃ¤hlt?
- Wurden Design Pattern verwendet und warum?
- Wurde Abstraktion und der Polymorphismus angewendet?
- Wie kann die Klassenstruktur einfach erweitert 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)


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

```

```

Fahrzeugart:  Auto
Marke:        UAZ
Kennzeichen:  SR770BA
13.10.2025:    25 km
TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```cpp
#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 ~Object() = default;

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

protected:
   Object() = default;
};

#endif // !1
```

6.2 RecordEntry.hpp

```cpp
/*****************************************************************//**
 * \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
 *
 * \author Simon
 * \date   October 2025
 *********************************************************************/
#include "RecordEntry.hpp"
using namespace std;


size_t RecordEntry::GetDistance() const
{
   return m_distance;
}

TDate RecordEntry::GetDate() const
{
   return m_date;
}

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;
}

bool RecordEntry::operator<(const RecordEntry& rh) const
{
   return m_date < rh.m_date;
}
```

6.4 DriveRecord.hpp

```
/*******************************************************************//**
 * \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

```cpp
#include <numeric>
#include <algorithm>
#include "DriveRecord.hpp"

void DriveRecord::AddRecord(const RecordEntry& entry)
{
   m_driveRecords.insert(entry);
}

size_t DriveRecord::GetMilage() const
{
   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();});
}

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

```cpp
/*****************************************************************//**
 * \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"

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 pVehicle);

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

    /**
     * \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;

    // TODO: Copy / assignement implementation
    // is identical to the Simple Animal Project.
    Garage(const Garage&);
    void operator=(Garage garage);

    ~Garage();
private:
    std::vector<Vehicle const *> m_vehicles;
};
```

6.7 Garage.cpp

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

/**
 * \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 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* 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;
    }
}

/**
 * \brief Functions searches for vehicle with matching plate.
 * \param pVehicle : Pointer to a Vehicle.
 * \return pointer to the vehicle inside the garage
 */
```

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

    return nullptr;
}

/**
 * \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& Garage::Print(std::ostream& ost) const
{
    if (ost.fail())
        throw Object::ERROR_BAD_OSTREAM;

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

    return ost;
}

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);
}

/**
 * \brief Frees every vehicle from memory.
 * \brief Caution! pointers get invalidated.
 */
Garage::~Garage()
{
    for (auto elem : m_vehicles)
    {
        delete elem;
    }

    m_vehicles.clear();
}
```

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 Setter for the plate member of a vehicle.
    *
    * \param plate : string that represents the plate
    */
   void SetPlate(const std::string & plate);

   /**
    * \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:

   /**
```

```cpp
    * \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) : m_brand{ brand }, m_fuel{ fuelType } {}

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


#endif // !1
```

6.9 Vehicle.cpp

```cpp
#include "Vehicle.hpp"

std::string Vehicle::GetBrand() const
{
    return m_brand;
}

std::string Vehicle::GetPlate() const
{
    return m_plate;
}

TFuel Vehicle::GetFuelType() const
{
    return m_fuel;
}

const DriveRecord & Vehicle::GetDriveRecord() const
{
    return m_record;
}

void Vehicle::SetPlate(const std::string & plate)
{
    m_plate = plate;
}

void Vehicle::AddRecord(const RecordEntry& entry)
{
    m_record.AddRecord(entry);
}

size_t Vehicle::GetMilage() const
{
    return m_record.GetMilage();
}
```

6.10 Car.hpp

```cpp
#ifndef CAR_HPP
#define CAR_HPP
```

```cpp

#include "Vehicle.hpp"

class Car : public Vehicle {
public:

   Car(const std::string & brand,const TFuel & fuelType) : Vehicle(brand, fuelType) {}

   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

```cpp
#include "Car.hpp"

using namespace std;

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

   ost <<endl<< left << setw(14) << "Fahrzeugart:" << "Auto" << 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;
}

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

6.12 Truck.hpp

6.13 Truck.cpp

6.14 Bike.hpp

6.15 Bike.cpp