**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра САПР**

# ОТЧЕТ

**по лабораторной работе №9**
**по дисциплине «Объектно-ориентированное программирование»**
**Тема: «Использование наследования с применением интерфейсов»**

Студент гр. 1302 _____ Максимов Ю. Е.

Преподаватель: _____ Новакова Н. Е.

Санкт-Петербург
2024

1. Цель работы

Изучение интерфейсов в языке C++ с помощью программного продукта компании clion.

2. Анализ задачи

Необходимо:

1. Разработать структуры данных для заданной предметной области;
2. Написать программу на основе созданных в упражнении 1 структур данных.

3. Ход выполнения работы

*3.1 Упражнение 1*

В ходе выполнения данного упражнения написана программа, которая представляет спортивные и гоночные автомобили, в ходе которой используется два базовых класса, и шесть классов, а так же фабрика.

**3.1.1 Используемые классы и методы**

В программе, написанной в данном упражнении, используются следующие методы:

- std::cout – служит для отображения на экране строк

и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку

- main() – служит для запуска программы
- Car – абстрактный класс машин
- RacingCar, SportCar – классы машин
- Detail – абстрактный класс деталей
- Body, Engine, Transmission, Wheel – класс деталей
- FactoryCar – фабрика для создания машин

Таблица 1.3.1 Описание методов класса Car

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_car | void | public | - | Отображение машины |

Таблица 1.3.2 Описание методов класса RacingCar

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_car | void | public | - | Отображение машины |

Таблица 1.3.3 Описание методов класса SportCar

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_car | void | public | - | Отображение машины |

Таблица 1.3.4 Описание методов класса Detail

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_detail | void | public | - | Отображение детали |

Таблица 1.3.4 Описание методов класса Body

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_detail | void | public | - | Отображение корпуса |

Таблица 1.3.4 Описание методов класса Engine

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_detail | void | public | - | Отображение мотора |

Таблица 1.3.4 Описание методов класса Transmission

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_detail | void | public | - | Отображение трансмиссии |

Таблица 1.3.4 Описание методов класса Wheel

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_detail | void | public | - | Отображение колеса |

Таблица 1.3.4 Описание методов фабрики FactoryCar

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| createSportCarBMW | std::unique_ptr<car::Car> | public | - | Создание спортивной машины БМВ |
| createSportCarAudi | std::unique_ptr<car::Car> | public | - | Создание спортивной машины Ауди |
| createRacingCarFerrari | std::unique_ptr<car::Car> | public | - | Создание гоночной машины Феррари |

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
|---|---|---|---|---|
| out_createRacingCarMercedes | std::unique_ptr<car::Car> | public | - | Создание гоночной машины Мерседес |

Таблица 1.3.4 Описание полей класса Car

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_made | std::string | private | Изготовитель |
| m_name | std::string | private | Имя |
| m_body | std::unique_ptr<detail::Detail> | private | Корпус |
| m_engine | std::unique_ptr<detail::Detail> | private | Двигатель |
| m_transmission | std::unique_ptr<detail::Detail> | private | Трансмиссия |
| m_wheel | std::unique_ptr<detail::Detail> | private | Колесо |

Таблица 1.3.4 Описание полей класса RacingCar

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_typeRacing | TYPE_RACING | private | Для какой гонки |

Таблица 1.3.4 Описание полей класса SportCar

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_isNitro | bool | private | Есть ли нитро |

Таблица 1.3.4 Описание полей класса Detail

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_made | std::string | private | Изготовитель |
| m_name | std::string | private | Имя |

Таблица 1.3.4 Описание полей класса Body

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_color | std::string | private | Цвет |
| m_typeMaterial | TYPE_MATERIAL | private | Тип матерьяла |

Таблица 1.3.4 Описание полей класса Engine

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_engineCapacity | int | private | Обьем двигателя |
| m_horsepower | int | private | Лошадиные силы |

Таблица 1.3.4 Описание полей класса Transmission

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_numberGears | int | private | Сколько ступеней |
| m_typeOfTransmission | TYPE_OF_TRANSMISSION | private | Тип трансмиссии |

Таблица 1.3.4 Описание полей класса Wheel

| Поля | Тип | Модификатор доступа | Назначение |
|---|---|---|---|
| m_diameterDisc | int | private | Диаметр Дисков |
| m_loadIndex | int | private | Максимум нагрузки |
| m_speedIndex | SPEED_INDEX | private | Максимум скорости |

### 3.1.4 Диаграмма классов

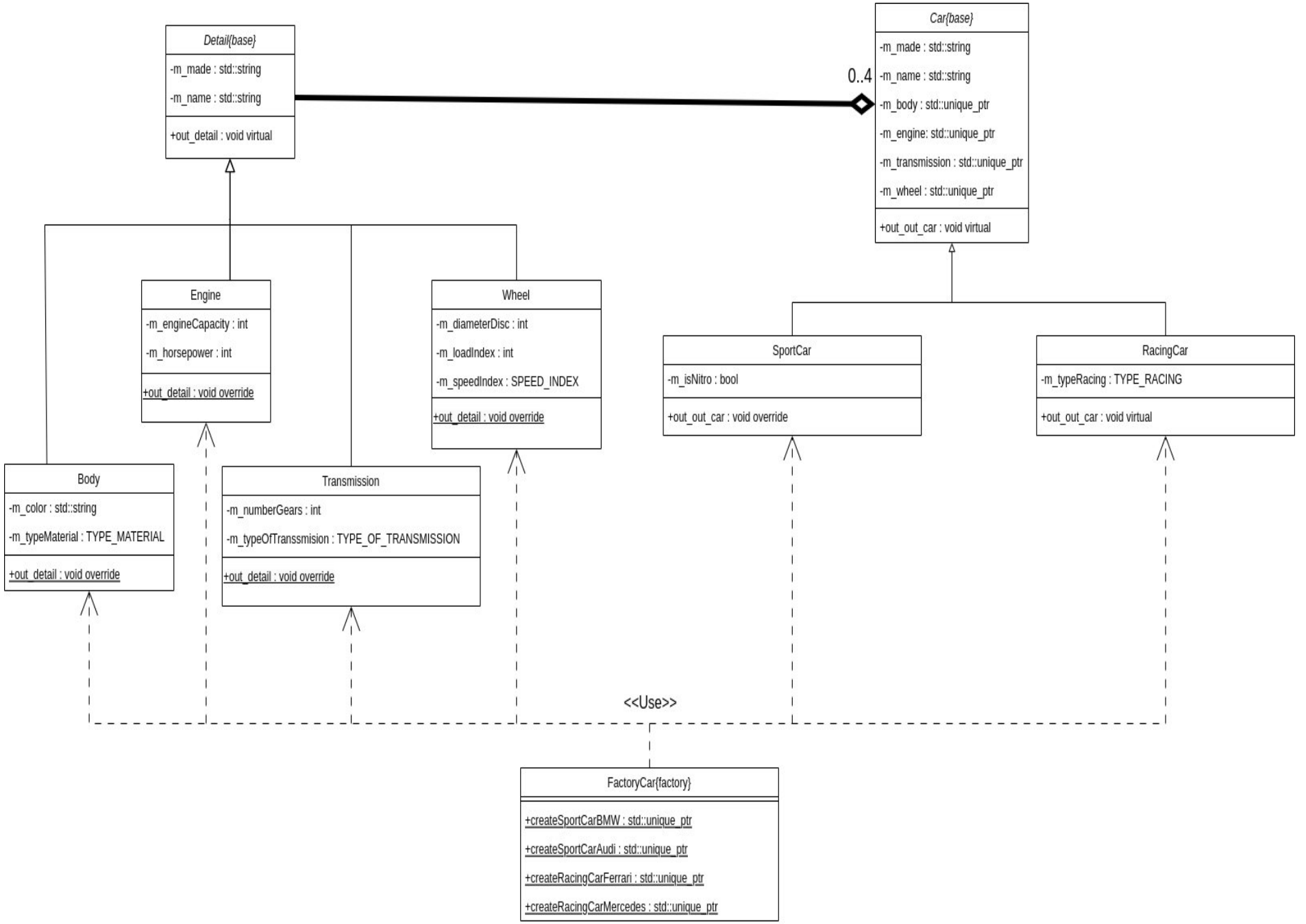
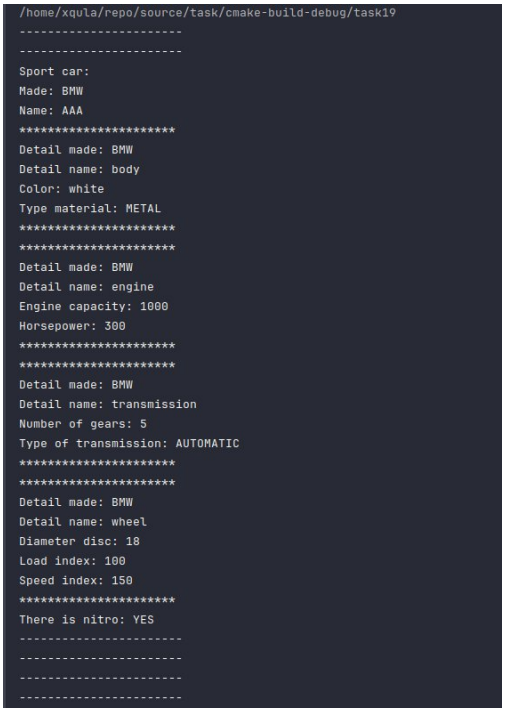



Рисунок 3.1.4. Диаграмма классов

Как мы видим по диаграмме, Детали создаются в фабрике и передаются в машины, что есть агрегация, и на выходе мы получаем готовую машину, котору мы можем создать как угодно из различных деталей.

### 3.1.5 Контрольный пример

На рис.3.1.5 представлены результаты выполнения программы.



```
----------------------
----------------------
Sport car:
Made: Audi
Name: XXX
**********************
Detail made: BMW
Detail name: body
Color: black
Type material: ALUMINUM
**********************
**********************
Detail made: BMW
Detail name: engine
Engine capacity: 1200
Horsepower: 310
**********************
**********************
Detail made: BMW
Detail name: transmission
Number of gears: 8
Type of transmission: ROBOT
**********************
**********************
Detail made: BMW
Detail name: wheel
Diameter disc: 21
Load index: 120
Speed index: 160
**********************
There is nitro: NO
----------------------
----------------------
```

```
----------------------
----------------------
Racing car:
Made: Ferrari
Name: YYY
**********************
Detail made: Ferrari
Detail name: body
Color: red
Type material: CARBON
**********************
**********************
Detail made: Ferrari
Detail name: engine
Engine capacity: 1200
Horsepower: 310
**********************
**********************
Detail made: Ferrari
Detail name: transmission
Number of gears: 12
Type of transmission: MAN
**********************
**********************
Detail made: BMW
Detail name: wheel
Diameter disc: 23
Load index: 180
Speed index: 240
**********************
type of racing car: F1
----------------------
----------------------
```

```
----------------------
----------------------
Racing car:
Made: Mercedes
Name: ZZZ
**********************
Detail made: Ferrari
Detail name: body
Color: red
Type material: CARBON
**********************
**********************
Detail made: Mercedes
Detail name: engine
Engine capacity: 1200
Horsepower: 310
**********************
**********************
Detail made: Mercedes
Detail name: transmission
Number of gears: 8
Type of transmission: MANUAL
**********************
**********************
Detail made: Mercedes
Detail name: wheel
Diameter disc: 18
Load index: 140
Speed index: 270
**********************
type of racing car: drift
----------------------
----------------------

Process finished with exit code 0
```

Рис.3.1.5 Контрольный пример для программы

Как видно из рисунка, мы видим как гоночный так и спортивный автомобиль, с разными деталями.

## 4. Листинг программы

car.cpp

```cpp
//
// Created by xqula on 21.06.24.
//
#pragma once
#include "memory"
#include "car/car.h"
namespace factory {
    class FactoryCar {
    public:
        FactoryCar() = default;

        virtual ~FactoryCar() = default;

        static auto createSportCarBMW() -> std::unique_ptr<car::Car>;

        static auto createSportCarAudi() -> std::unique_ptr<car::Car>;

        static auto createRacingCarFerrari() -> std::unique_ptr<car::Car>;

        static auto createRacingCarMercedes() -> std::unique_ptr<car::Car>;
    };

}
```

**car.h**

```cpp
//
// Created by xqula on 21.06.24.
//
#pragma once

#include "string"
#include "detail/detail.h"
#include "memory"
namespace car {
    class Car {
    public:
        explicit Car(
                std::string made,
                std::string name,
                std::unique_ptr<detail::Detail> body,
                std::unique_ptr<detail::Detail> engine,
                std::unique_ptr<detail::Detail> transmission,
                std::unique_ptr<detail::Detail> wheel);
        virtual ~Car() = default;
        virtual auto out_car() -> void;
    private:
        std::string m_made;
```

```
        std::string m_name;
        std::unique_ptr<detail::Detail> m_body;
        std::unique_ptr<detail::Detail> m_engine;
        std::unique_ptr<detail::Detail> m_transmission;
        std::unique_ptr<detail::Detail> m_wheel;
    };

}
```

## racingcar.h

```
//
// Created by xqula on 21.06.24.
//
#pragma once
#include "car.h"
namespace car{
    enum TYPE_RACING{
        F1 = 0,
        DRIFT = 1,
        RALLY = 2
    };
    class RacingCar: public Car{
    public:
        explicit RacingCar(
                std::string made,
                std::string name,
                std::unique_ptr<detail::Detail> body,
                std::unique_ptr<detail::Detail> engine,
                std::unique_ptr<detail::Detail> transmission,
                std::unique_ptr<detail::Detail> wheel,
                TYPE_RACING typeRacing
        );
        ~RacingCar() override = default;
        auto out_car() -> void override;
    private:
        TYPE_RACING m_typeRacing;
    };

}
```

## racingcar.cpp

```
//
// Created by xqula on 21.06.24.
//
#include "racingcar.h"
#include "iostream"
car::RacingCar::RacingCar(std::string made,
                          std::string name,
                          std::unique_ptr<detail::Detail> body,
                          std::unique_ptr<detail::Detail> engine,
                          std::unique_ptr<detail::Detail> transmission,
                          std::unique_ptr<detail::Detail> wheel,
                          car::TYPE_RACING typeRacing):
    Car(std::move(made), std::move(name), std::move(body), std::move(engine), std::move(transmission),
std::move(wheel)),
    m_typeRacing(typeRacing)
{
}
auto car::RacingCar::out_car() -> void {
```

```
    std::cout << "----------------------" << std::endl;
    std::cout << "----------------------" << std::endl;
    std::cout << "Racing car:" << std::endl;
    Car::out_car();
    std::cout << "type of racing car: ";
    switch (static_cast<int>(m_typeRacing)) {
        case 0:
            std::cout << "F1" << std::endl;
            break;
        case 1:
            std::cout << "drift" << std::endl;
            break;
        case 2:
            std::cout << "rally" << std::endl;
            break;
        default:
            std::cout << "unknown" << std::endl;
            break;
    }
    std::cout << "----------------------" << std::endl;
    std::cout << "----------------------" << std::endl;
}
```

## sportcar.h

```
//
// Created by xqula on 21.06.24.
//
#pragma once

#include "car.h"
namespace car {
    class SportCar: public Car {
    public:
        explicit SportCar(
                std::string made,
                std::string name,
                std::unique_ptr<detail::Detail> body,
                std::unique_ptr<detail::Detail> engine,
                std::unique_ptr<detail::Detail> transmission,
                std::unique_ptr<detail::Detail> wheel,
                bool isNitro
        );
        ~SportCar() override = default;
        auto out_car() -> void override;
    private:
        bool m_isNitro;
    };

}
```

## sportcar.cpp

```
//
// Created by xqula on 21.06.24.
//
#include "sportcar.h"
#include "iostream"
car::SportCar::SportCar(std::string made,
                        std::string name,
                        std::unique_ptr<detail::Detail> body,
                        std::unique_ptr<detail::Detail> engine,
```

```
                    std::unique_ptr<detail::Detail> transmission,
                    std::unique_ptr<detail::Detail> wheel,
                    bool isNitro):
    Car(std::move(made), std::move(name), std::move(body), std::move(engine), std::move(transmission),
std::move(wheel)),
    m_isNitro(isNitro)
{
}
auto car::SportCar::out_car() -> void {
    std::cout << "----------------------" << std::endl;
    std::cout << "----------------------" << std::endl;
    std::cout << "Sport car:" << std::endl;
    Car::out_car();
    std::cout << "There is nitro: ";
    if(m_isNitro){
        std::cout << "YES" << std::endl;
    } else {
        std::cout << "NO" << std::endl;
    }
    std::cout << "----------------------" << std::endl;
    std::cout << "----------------------" << std::endl;
}
```

## body.cpp

```
//
// Created by xqula on 21.06.24.
//
#include "body.h"
#include "iostream"
detail::Body::Body(std::string made, std::string name, std::string color, detail::TYPE_MATERIAL
typeMaterial)
    : Detail(std::move(made), std::move(name)),
    m_color(std::move(color)),
    m_typeMaterial(typeMaterial)
{
}
auto detail::Body::out_detail() -> void {
    std::cout << "**********************" << std::endl;
    Detail::out_detail();
    std::cout << "Color: " << m_color << std::endl;
    std::cout << "Type material: ";
    switch (static_cast<int>(m_typeMaterial)) {
        case 0:
            std::cout << "METAL" << std::endl;
            break;
        case 1:
            std::cout << "ALUMINUM" << std::endl;
            break;
        case 2:
            std::cout << "CARBON" << std::endl;
            break;
    }
    std::cout << "**********************" << std::endl;
}
```

## body.h

```
//
// Created by xqula on 21.06.24.
//
```

```cpp
#include "detail.h"
#include "string"
namespace detail {
    enum class TYPE_MATERIAL {
        METAL = 0,
        ALUMINUM = 1,
        CARBON = 2
    };
    class Body: public Detail {
    public:
        explicit Body(std::string made, std::string name, std::string color, TYPE_MATERIAL typeMaterial);
        auto out_detail() -> void override;
    private:
        std::string m_color;
        TYPE_MATERIAL m_typeMaterial;
    };
}
```

## detail.cpp

```cpp
//
// Created by xqula on 21.06.24.
//
#include "detail.h"
#include <utility>
#include "iostream"
detail::Detail::Detail(std::string made, std::string name) : m_made(std::move(made)),
m_name(std::move(name)) {}
auto detail::Detail::out_detail() -> void {
    std::cout << "Detail made: " << m_made << "\nDetail name: " << m_name << std::endl;
}
```

## detail.h

```cpp
//
// Created by xqula on 21.06.24.
//
#pragma once

#include "string"
namespace detail {
    class Detail {
    public:
        explicit Detail(std::string made, std::string name);
        virtual ~Detail() = default;
        virtual auto out_detail() -> void;

    private:
        std::string m_made;
        std::string m_name;
    };
}
```

## engine.cpp

```cpp
//
// Created by xqula on 21.06.24.
//
```

```cpp
#include "engine.h"
#include <iostream>
detail::Engine::Engine(std::string made, std::string name, const int engineCapacity, const int
m_horsepower):
        Detail(std::move(made), std::move(name)),
        m_engineCapacity(engineCapacity),
        m_horsepower(m_horsepower)
{
}
auto detail::Engine::out_detail() -> void {
    std::cout << "**********************" << std::endl;
    Detail::out_detail();
    std::cout << "Engine capacity: " << m_engineCapacity << std::endl;
    std::cout << "Horsepower: " << m_horsepower << std::endl;
    std::cout << "**********************" << std::endl;
}
```

## engine.h

```cpp
//
// Created by xqula on 21.06.24.
//
#pragma once

#include "detail.h"
namespace detail {
    class Engine : public Detail {
    public:
        explicit Engine(std::string made, std::string name, int engineCapacity, int m_horsepower);
        auto out_detail() -> void override;
    private:
        int m_engineCapacity;
        int m_horsepower;
    };

}
```

## transmission.h

```cpp
//
// Created by xqula on 21.06.24.
//
#pragma once

#include "detail.h"
namespace detail {
    enum class TYPE_OF_TRANSMISSION {
        MANUAL = 0,
        AUTOMATIC = 1,
        VARIATION = 2,
        ROBOT = 3
    };

    class Transmission : public Detail {
    public:
        explicit Transmission(
                std::string made,
                std::string name,
                int numberGears,
```

```
            TYPE_OF_TRANSMISSION typeOfTransmission);
        auto out_detail() -> void override;
    private:
        int m_numberGears;
        TYPE_OF_TRANSMISSION m_typeOfTransmission;
    };
}
```

## transmission.cpp

```
//
// Created by xqula on 21.06.24.
//
#include "transmission.h"
#include "iostream"
detail::Transmission::Transmission(std::string made,
                                   std::string name,
                                   const int numberGears,
                                   const detail::TYPE_OF_TRANSMISSION typeOfTransmission)
    : Detail(std::move(made), std::move(name)),
    m_numberGears(numberGears),
    m_typeOfTransmission(typeOfTransmission)
{
}
auto detail::Transmission::out_detail() -> void {
    std::cout << "**********************" << std::endl;
    Detail::out_detail();
    std::cout << "Number of gears: " << m_numberGears << std::endl;
    std::cout << "Type of transmission: ";
    switch (static_cast<int>(m_typeOfTransmission)) {
        case 0:
            std::cout << "MANUAL" << std::endl;
            break;
        case 1:
            std::cout << "AUTOMATIC" << std::endl;
            break;
        case 2:
            std::cout << "VARIATION" << std::endl;
            break;
        case 3:
            std::cout << "ROBOT" << std::endl;
            break;
    }
    std::cout << "**********************" << std::endl;
}
```

## Wheel.cpp

```
//
// Created by xqula on 21.06.24.
//
#include "wheel.h"
#include <utility>
#include "iostream"
detail::Wheel::Wheel(
        std::string made,
        std::string name,
        const int diameterDisc,
        const int loadIndex,
        const detail::SPEED_INDEX speedIndex) :
    m_diameterDisc(diameterDisc),
```

```
    m_loadIndex(loadIndex),
    m_speedIndex(speedIndex),
    Detail(std::move(made),std::move(name))
{
}
/*
 * Outputs the details of a car to the console.
 *
 * @param car The car to output the details of.
 */
auto detail::Wheel::out_detail() -> void {
    std::cout << "**********************" << std::endl;
    Detail::out_detail();
    std::cout << "Diameter disc: " << m_diameterDisc << std::endl;
    std::cout << "Load index: " << m_loadIndex << std::endl;
    std::cout << "Speed index: " << static_cast<int>(m_speedIndex) << std::endl;
    std::cout << "**********************" << std::endl;
}
```

## Wheel.h

```
//
// Created by xqula on 21.06.24.
//
#pragma once

#include "detail.h"
namespace detail {
    enum class SPEED_INDEX {
        P = 150,
        Q = 160,
        R = 170,
        S = 180,
        T = 190,
        U = 200,
        H = 210,
        V = 240,
        W = 270,
        Y = 300
    };

    class Wheel: public Detail {
    public:
        explicit Wheel( std::string made,
                        std::string name,
                        int diameterDisc,
                        int loadIndex,
                        detail::SPEED_INDEX speedIndex);
        auto out_detail() -> void override;
    private:
        int m_diameterDisc{};
        int m_loadIndex{};
        SPEED_INDEX m_speedIndex;
    };

}
```

## factorycar.cpp

```
//
// Created by xqula on 21.06.24.
//
```

```cpp
#include "factorycar.h"
#include "detail/body.h"
#include "detail/engine.h"
#include "detail/transmission.h"
#include "detail/wheel.h"
#include "car/sportcar.h"
#include "car/racingcar.h"
using namespace factory;

/*
 * Creates a new BMW sport car.
 *
 * @return A unique pointer to a new BMW sport car.
 */
auto FactoryCar::createSportCarBMW() -> std::unique_ptr<car::Car> {
    return std::make_unique<car::SportCar>( "BMW",
                                "AAA",
                                std::make_unique<detail::Body>( "BMW",
                                                "body",
                                                "white",
                                                detail::TYPE_MATERIAL::METAL),
                                std::make_unique<detail::Engine>("BMW",
                                                 "engine",
                                                 1000,
                                                 300 ),
                                std::make_unique<detail::Transmission>("BMW",
                                                    "transmission",\
                                                    5,
                                                    detail::TYPE_OF_TRANSMISSION::AUTOMATIC),
                                std::make_unique<detail::Wheel>("BMW",
                                                "wheel",
                                                18,
                                                100,
                                                detail::SPEED_INDEX::P),
                                true
                                );
}
/*
 * Creates a new Audi sport car.
 *
 * @return A unique pointer to a new Audi sport car.
 */
auto FactoryCar::createSportCarAudi() -> std::unique_ptr<car::Car> {
    return std::make_unique<car::SportCar>( "Audi",
                                 "XXX",
                                 std::make_unique<detail::Body>( "BMW",
                                                 "body",
                                                 "black",
                                                 detail::TYPE_MATERIAL::ALUMINUM),
                                 std::make_unique<detail::Engine>("BMW",
                                                  "engine",
                                                  1200,
                                                  310 ),
                                 std::make_unique<detail::Transmission>("BMW",
                                                     "transmission",\
                                                     8,
                                                     detail::TYPE_OF_TRANSMISSION::ROBOT),
                                 std::make_unique<detail::Wheel>("BMW",
                                                 "wheel",
                                                 21,
                                                 120,
                                                 detail::SPEED_INDEX::Q),
                                 false
```

```
                                    );
}
/*
 * Creates a new Ferrari racing car.
 *
 * @return A unique pointer to a new Ferrari racing car.
 */
auto FactoryCar::createRacingCarFerrari() -> std::unique_ptr<car::Car> {
    return std::make_unique<car::RacingCar>( "Ferrari", "YYY", std::make_unique<detail::Body>( "Ferrari",
                                                              "body",
                                                              "red",
                                                              detail::TYPE_MATERIAL::CARBON),
                                    std::make_unique<detail::Engine>("Ferrari",
                                                                     "engine",
                                                                     1200,
                                                                     310 ),
                                    std::make_unique<detail::Transmission>("Ferrari",
                                                                           "transmission",\
                                                                           12,
                                                                           detail::TYPE_OF_TRANSMISSION::MANUAL),
                                    std::make_unique<detail::Wheel>("BMW",
                                                                    "wheel",
                                                                    23,
                                                                    180,
                                                                    detail::SPEED_INDEX::V),
                                    car::TYPE_RACING::F1
                                    );
}
/*
 * Creates a new Mercedes racing car.
 *
 * @return A unique pointer to a new Mercedes racing car.
 */
auto FactoryCar::createRacingCarMercedes() -> std::unique_ptr<car::Car> {
    return std::make_unique<car::RacingCar>( "Mercedes", "ZZZ",
std::make_unique<detail::Body>( "Ferrari",
                                                              "body",
                                                              "red",
                                                              detail::TYPE_MATERIAL::CARBON),
                                    std::make_unique<detail::Engine>("Mercedes",
                                                                     "engine",
                                                                     1200,
                                                                     310 ),
                                    std::make_unique<detail::Transmission>("Mercedes",
                                                                           "transmission",\
                                                                           8,
                                                                           detail::TYPE_OF_TRANSMISSION::MANUAL),
                                    std::make_unique<detail::Wheel>("Mercedes",
                                                                    "wheel",
                                                                    18,
                                                                    140,
                                                                    detail::SPEED_INDEX::W),
                                    car::TYPE_RACING::DRIFT
                                    );
}
```

## factorycar.h

```
//
// Created by xqula on 21.06.24.
//
#pragma once
```

```
#include "memory"
#include "car/car.h"
namespace factory {
    class FactoryCar {
    public:
        FactoryCar() = default;

        virtual ~FactoryCar() = default;

        static auto createSportCarBMW() -> std::unique_ptr<car::Car>;

        static auto createSportCarAudi() -> std::unique_ptr<car::Car>;

        static auto createRacingCarFerrari() -> std::unique_ptr<car::Car>;

        static auto createRacingCarMercedes() -> std::unique_ptr<car::Car>;
    };

}
```

5. Полученные результаты

В ходе выполнения лабораторной работы была реализована иерархия классов, представляющая спортивные и гоночные автомобили.

6. Выводы

В ходе выполнения данной лабораторной работы были изучены абстрактные классы, наследование и интерфейсы, была реализована иерархия классов для представления спортивных и гоночных автомобили.