**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 |
| 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

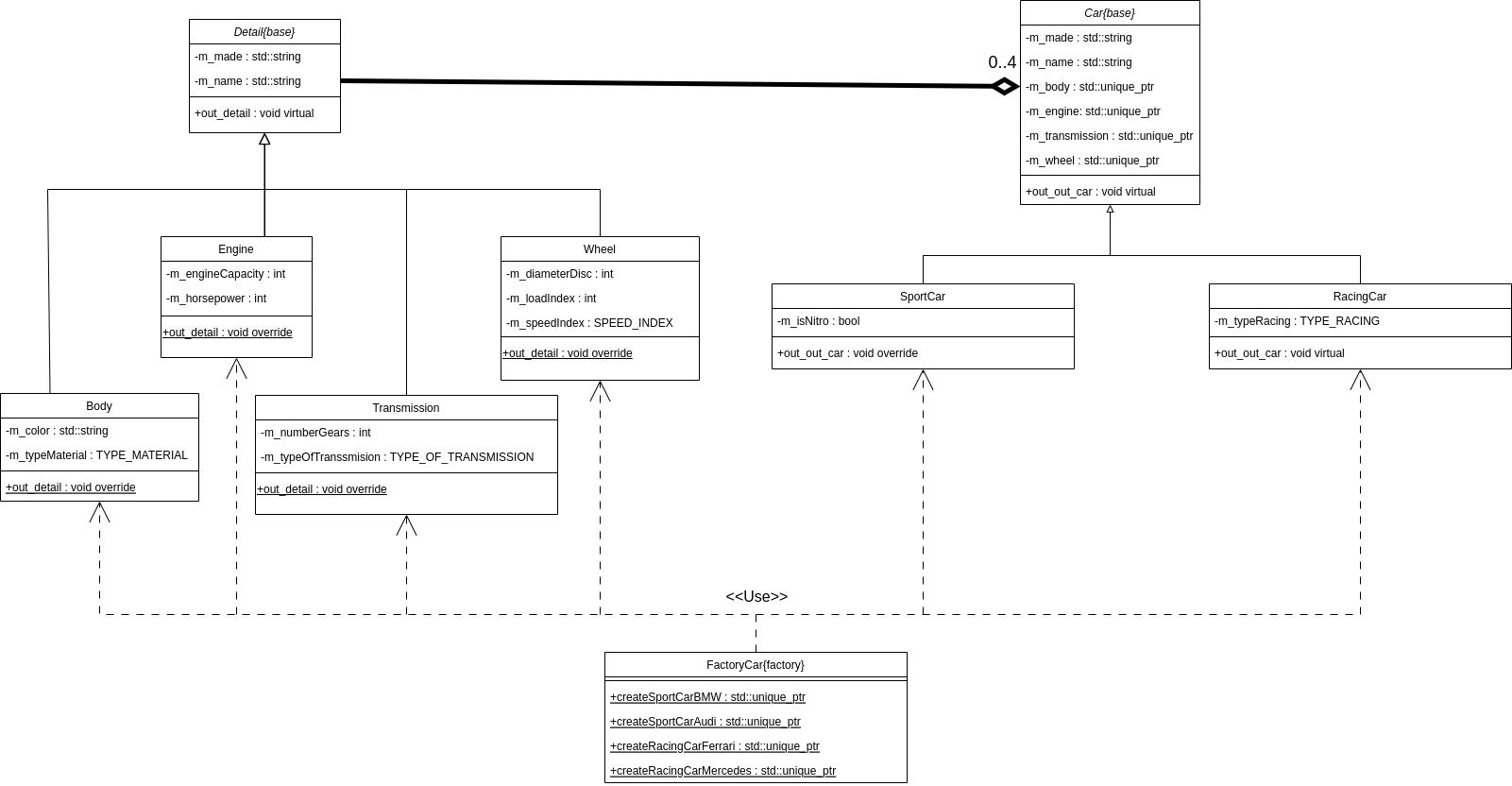


Рисунок 3.1.4. Диаграмма классов

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.

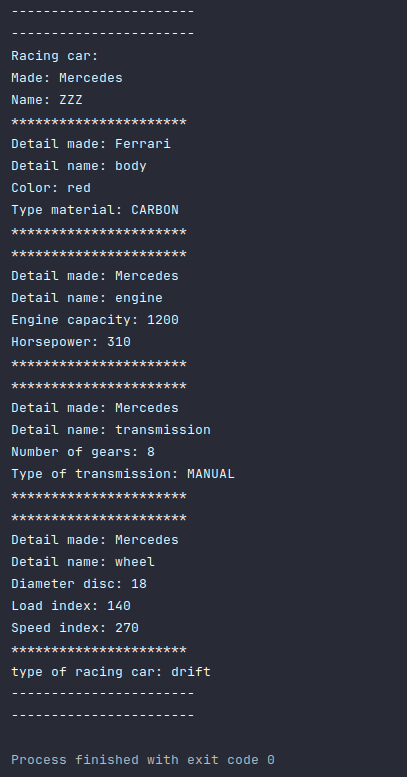
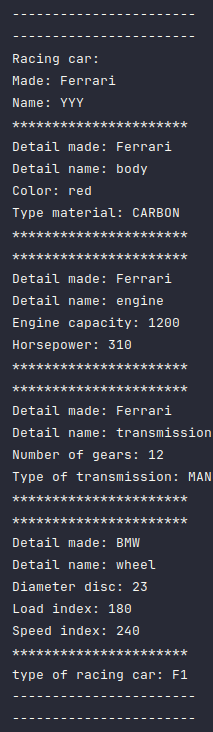
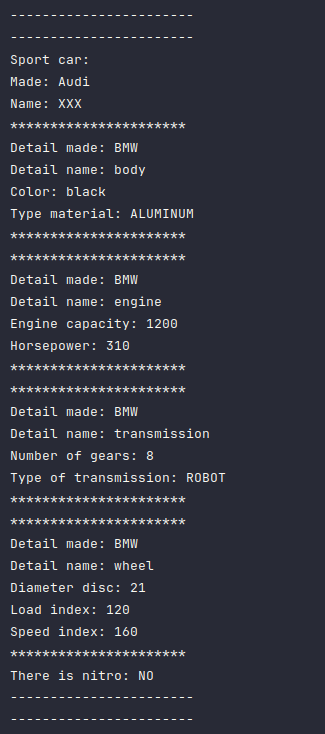
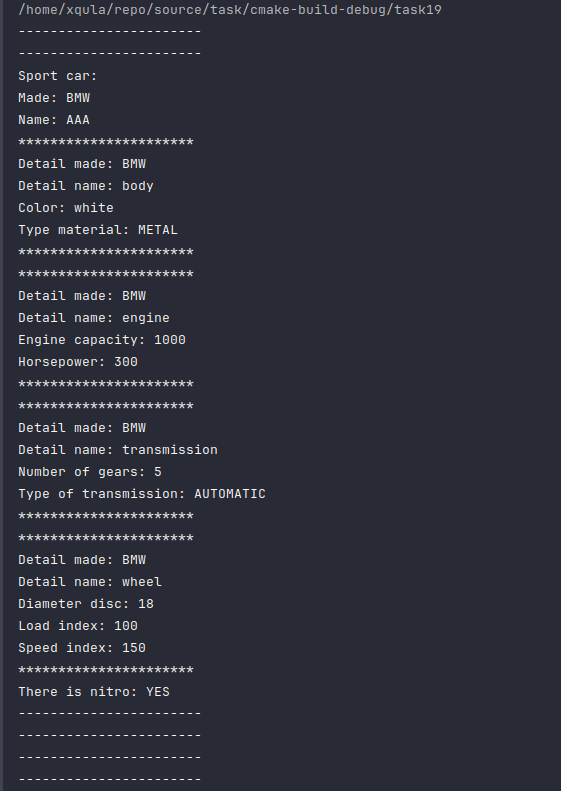


Рис.3.1.5 Контрольный пример для программы

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

4. Листинг программы

car.cpp

//  
// 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**

//  
// 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.  
//  
#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**

//  
// 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**

//  
// 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**

//  
// Created by xqula on 21.06.24.  
//  
#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**

//  
// 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**

//  
// 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.  
//  
#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.