1 ПОСТАНОВКА ЗАДАЧИ

Создать абстрактный базовый класс «грузоперевозчик» и производные классы «Самолёт», «Поезд», «Автомобиль». Определить время и стоимость перевозки для указанных городов и расстояний.

2 ДИАГРАММА КЛАССОВ

Диаграмма классов представлена на рисунке 2.1.

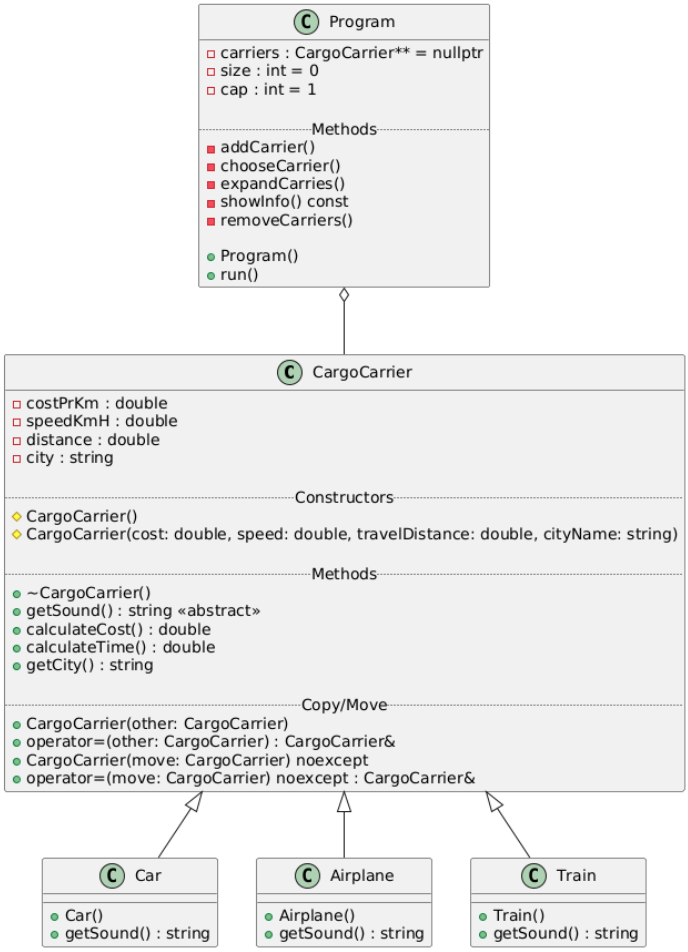

Рисунок 2.1 – Диаграмма классов

3 ЛИСТИНГ КОДА

Файл main.cpp

```
#include "program.h"

int main() {
    Program program;

    program.run();
    return 0;
}
```

Файл cargo_carrier.h

```
#pragma once

#include <iostream>

class CargoCarrier {
    double costPrKm;
    double speedKmH;
    double distance;
    std::string city;

  protected:
    CargoCarrier();

    explicit CargoCarrier(double cost, double speed, double travelDistance,
std::string cityName);

  public:
    virtual ~CargoCarrier();
    virtual std::string getSound() const = 0;

    CargoCarrier(const CargoCarrier &other);
    CargoCarrier &operator=(const CargoCarrier &other);
    CargoCarrier(CargoCarrier &&move) noexcept;
    CargoCarrier &operator=(CargoCarrier &&move) noexcept;
    double calculateCost() const;
    double calculateTime() const;
    std::string getCity() const;
};
```

Файл cargo_carrier.cpp

```
#include "cargo_carrier.h"

CargoCarrier::CargoCarrier() : costPrKm(0.0), speedKmH(0.0), distance(0.0) {
}

CargoCarrier::CargoCarrier(double cost, double speed, double travelDistance,
std::string cityName)
    :     costPrKm(cost),     speedKmH(speed),     distance(travelDistance),
city(std::move(cityName)) {
}

CargoCarrier::~CargoCarrier() {
    costPrKm = 0;
    speedKmH = 0;
```

```
    distance = 0;
}

CargoCarrier::CargoCarrier(const CargoCarrier &other) = default;

CargoCarrier::CargoCarrier(CargoCarrier &&move) noexcept = default;

CargoCarrier &CargoCarrier::operator=(const CargoCarrier &other) = default;

CargoCarrier &CargoCarrier::operator=(CargoCarrier &&move) noexcept = default;

double CargoCarrier::calculateCost() const {
    return distance * costPrKm;
}

double CargoCarrier::calculateTime() const {
    return distance / speedKmH;
}

std::string CargoCarrier::getCity() const {
    return city;
}
```

Файл car.h

```
#pragma once

#include "cargo_carrier.h"

class Car : public CargoCarrier {
  public:
    const char *getSound() const override;

    Car();
};
```

Файл car.cpp

```
#include "car.h"
#include "consts.h"
#include "utils.h"

std::string Car::getSound() const {
    return kCarSound;
}

Car::Car()
    :  CargoCarrier(getNumber("\nPlease enter car cost (per km): ",
kCarMinCostPrKm, kCarMaxCostPrKm),
                    getNumber("\nPlease enter car speed (km/h): ",
kCarMinSpeedKmH, kCarMaxSpeedKmH),
                    getNumber("\nPlease enter car travel distance(km): ",
kCarMinDistanceKm, kCarMaxDistanceKm),
                    getString("\nPlease enter the name of the city to travel to
by car: ")) {
}
```

Файл train.h

```
#pragma once
```

```
#include "cargo_carrier.h"

class Train : public CargoCarrier {
  public:
    const char *getSound() const override;

    Train();
};
```

Файл train.cpp

```
#include "train.h"
#include "consts.h"
#include "utils.h"

std::string Train::getSound() const {
    return kTrainSound;
}

Train::Train()
    :  CargoCarrier(getNumber("\nPlease  enter  train  cost  (per  km):  ",
kTrainMinCostPrKm, kTrainMaxCostPrKm),
                    getNumber("\nPlease  enter  train  speed  (km/h):  ",
kTrainMinSpeedKmH, kTrainMaxSpeedKmH),
                    getNumber("\nPlease enter train travel distance(km): ",
kTrainMinDistanceKm, kTrainMaxDistanceKm),
                    getString("\nPlease enter the name of the city to travel to
by train: ")) {
}
```

Файл airplane.h

```
#pragma once

#include "cargo_carrier.h"

class Airplane : public CargoCarrier {
  public:
    std::string getSound() const override;

    Airplane();
};
```

Файл airplane.cpp

```
#include "airplane.h"
#include "consts.h"
#include "utils.h"

std::string Airplane::getSound() const {
    return kAirplaneSound;
}

Airplane::Airplane()
    : CargoCarrier(
          getNumber("\nPlease  enter  airplane  cost  (per  km):  ",
kAirplaneMinCostPrKm, kAirplaneMaxCostPrKm),
          getNumber("\nPlease  enter  airplane  speed  (km/h):  ",
kAirplaneMinSpeedKmH, kAirplaneMaxSpeedKmH),
```

```
          getNumber("\nPlease   enter   airplane   travel   distance(km):   ",
kAirplaneMinDistanceKm, kAirplaneMaxDistanceKm),
          getString("\nPlease enter the name of the city to travel to by plane:
")) {
}
```

Файл menus.h

```
#pragma once

void showTaskMenu();
void showCarriersMenu();
```

Файл menus.cpp

```
#include "menus.h"
#include <iostream>

void showTaskMenu() {
    std::cout << "\n\t\t\t\tTASK" << std::endl;
    std::cout << "Create an abstract base class 'CargoCarrier' and derived
classes 'Airplane', 'Train', 'Car'."
              << std::endl;
    std::cout << "Define methods to calculate travel time and cost for the given
cities and distances." << std::endl;

    std::cout << "\n\t\t\t\tMENU" << std::endl;
    std::cout << "1. Add a cargo carrier." << std::endl;
    std::cout << "2. Show information (cost and time for a given distance and
city)." << std::endl;
    std::cout << "3. Exit the program." << std::endl;
}

void showCarriersMenu() {
    std::cout << "\n\t\t\t\tSELECT CARRIER TYPE" << std::endl;
    std::cout << "1. Car" << std::endl;
    std::cout << "2. Train" << std::endl;
    std::cout << "3. Airplane" << std::endl;
}
```

Файл consts.h

```
#pragma once

inline constexpr const char *kCarSound = "BIP BIP!";
inline constexpr const char *kTrainSound = "CHOOH CHOOH!";
inline constexpr const char *kAirplaneSound = "WHOOOSH!";

inline constexpr const double kCarMinCostPrKm = 0.1;
inline constexpr const double kCarMaxCostPrKm = 5.0;
inline constexpr const double kCarMinSpeedKmH = 10.0;
inline constexpr const double kCarMaxSpeedKmH = 250.0;
inline constexpr const double kCarMinDistanceKm = 1.0;
inline constexpr const double kCarMaxDistanceKm = 1000.0;

inline constexpr const double kTrainMinCostPrKm = 0.05;
inline constexpr const double kTrainMaxCostPrKm = 1.0;
inline constexpr const double kTrainMinSpeedKmH = 30.0;
inline constexpr const double kTrainMaxSpeedKmH = 350.0;
inline constexpr const double kTrainMinDistanceKm = 10.0;
```

```
inline constexpr const double kTrainMaxDistanceKm = 2000.0;

inline constexpr const double kAirplaneMinCostPrKm = 0.5;
inline constexpr const double kAirplaneMaxCostPrKm = 10.0;
inline constexpr const double kAirplaneMinSpeedKmH = 300.0;
inline constexpr const double kAirplaneMaxSpeedKmH = 1200.0;
inline constexpr const double kAirplaneMinDistanceKm = 100.0;
inline constexpr const double kAirplaneMaxDistanceKm = 10000.0;

inline constexpr const char *kWhiteColor = "\o{33}[0m";
inline constexpr const char *kRedColor = "\o{33}[31m";
inline constexpr const char *kGreenColor = "\o{33}[32m";
```

Файл program.h

```
#pragma once

#include "passenger_carrier.h"

class Program {
    PassengerCarrier **carriers = nullptr;
    int size = 0;
    int cap = 1;

    void addCarrier();
    void chooseCarrier();
    void expandCarries();
    void showInfo() const;
    void removeCarriers();

  public:
    Program();

    void run();
};
```

Файл program.cpp

```
#include "program.h"
#include "airplane.h"
#include "car.h"
#include "consts.h"
#include "menus.h"
#include "train.h"
#include "utils.h"

Program::Program() = default;

void Program::addCarrier() {
    expandCarries();

    chooseCarrier();

    std::cout << kGreenColor << "\nYou have successfully added carrier!" <<
kWhiteColor << std::endl;
}

void Program::expandCarries() {
    if (size < 0) {
        size = 0;
    }
```

```
    if (size + 1 == cap) {
        cap *= 2;
        auto **new_carriers = new CargoCarrier *[cap];

        for (int i = 0; i < size; i++) {
            new_carriers[i] = carriers[i];
        }

        delete[] carriers;

        carriers = new_carriers;
    }

    size++;
}

void Program::chooseCarrier() {
    int opt = 0;

    showCarriersMenu();

    while (true) {
        opt = getNumber("\nPlease enter carriers menu option ", 1, 3);

        switch (opt) {
        case 1:
            carriers[size - 1] = new Car;
            return;
        case 2:
            carriers[size - 1] = new Train;
            return;
        case 3:
            carriers[size - 1] = new Airplane;
            return;
        default:
            std::cout << kRedColor << "\nError, you picked is an incorrect
carrier menu option. Please try again."
                      << kWhiteColor << std::endl;
        }
    }
}

void Program::showInfo() const {
    if (size == 0) {
        std::cout << kRedColor << "\nError, you have not added any carriers
yet. Please use option 1 to add a carrier."
                  << kWhiteColor << std::endl;
        return;
    }

    std::cout << "\n\t\t\t\tINFO" << std::endl;

    for (int i = 0; i < size; i++) {
        printInfo(*carriers[i]);
        std::cout << std::endl;
    }

    std::cout << kGreenColor << "You have successfully showed all info about
carriers!" << kWhiteColor << std::endl;
}

void Program::removeCarriers() {
    for (int i = 0; i < size; i++) {
```

```
        delete carriers[i];
    }

    delete[] carriers;
    carriers = nullptr;
    size = 0;
    cap = 0;
}

void Program::run() {
    int opt = 0;

    system("clear");
    showTaskMenu();

    while (true) {
        opt = getNumber("\nPlease enter menu option ", 1, 3);

        switch (opt) {
        case 1:
            addCarrier();
            break;
        case 2:
            showInfo();
            break;
        case 3:
            removeCarriers();
            std::cout << kGreenColor << "\nYou have successfully exited the
program." << kWhiteColor << std::endl;
            return;
        default:
            std::cout << kRedColor << "\nError, you picked is an incorrect menu
option. Please try again."
                      << kWhiteColor << std::endl;
        }
    }
}
```

Файл utils.h

```
#pragma once

#include "cargo_carrier.h"
#include "consts.h"
#include <iostream>

template <typename T>

T getNumber(const std::string &msg, T min, T max) {
    T num;
    int sym = 0;

    std::cout << msg;
    std::cout << "(" << min << " <= input <= " << max << "): ";

    while (true) {
        if (std::cin.peek() != '\n' && std::cin.peek() != ' ' && (std::cin >>
num).good()) {
            sym = std::cin.peek();
            if (((char)sym == '\n' || (char)sym == EOF) && num >= min && num
<= max) {
                std::cin.get();
                return num;
```

```
            }
        }

        std::cin.clear();
        while (std::cin.get() != '\n' && !std::cin.eof())
            ;
        std::cout << kRedColor << "\nError, invalid input. Please try again: "
<< kWhiteColor;
    }
}

std::string getString(const std::string &msg);

void printInfo(const CargoCarrier &carrier);
```

Файл utils.cpp

```
#include "utils.h"

std::string getString(const std::string &msg) {
    std::string str;

    std::cout << msg;
    while (true) {
        std::getline(std::cin, str);

        if (!str.empty()) { // строка не пустая
            return str;
        }

        std::cout << kRedColor << "\nError, the string cannot be empty. Please
try again: " << kWhiteColor;
    }
}

void printInfo(const CargoCarrier &carrier) {
    std::cout << carrier.getSound() << " The cost(in BYN) per distance travelled
to " << carrier.getCity() + " is "
              << carrier.calculateCost() << std::endl;
    std::cout << carrier.getSound() << " The time(in Hours) per distance
travelled to " << carrier.getCity() + " is "
              << carrier.calculateTime() << std::endl;
}
```

4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

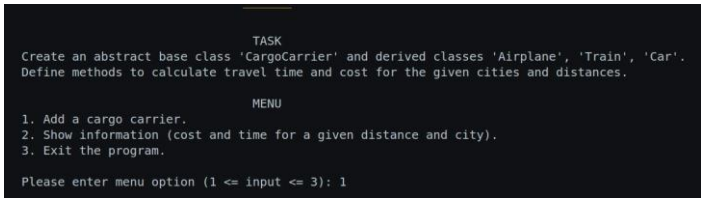

Рисунок 4.1 – Описание задания и главное меню программы

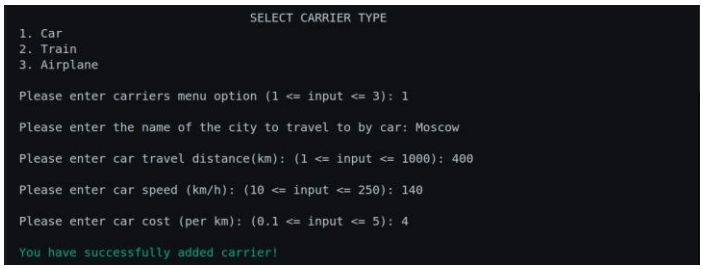

Рисунок 4.2 – Меню выбора грузоперевозчика и добавление машины

Рисунок 4.3 – Вывод всей информации о пассажироперевозчиках

Рисунок 4.4 – Завершение программы

5 ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были закреплены навыки объектно-ориентированного программирования в C++ с использованием принципов наследования и полиморфизма. Был разработан абстрактный базовый класс «Грузоперевозчик», а также производные классы «Самолёт», «Поезд» и «Автомобиль», реализующие общие и специфические характеристики различных видов транспорта. В классах были определены методы для расчёта времени и стоимости передвижения в зависимости от заданных параметров. Проведённое тестирование подтвердило корректность работы иерархии классов, правильность переопределения виртуальных функций и корректное взаимодействие базового и производных классов.