Министерство образования Республики Беларусь
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

ОТЧЕТ
О ЛАБОРАТОРНОЙ РАБОТЕ № 3
Наследование

по дисциплине «Программирование на языках высокого уровня»

Выполнил ст. гр. 450503 А.П. Красько

Проверил асс. каф. ЭВМ И.Г. Скиба

Минск 2025

1 ПОСТАНОВКА ЗАДАЧИ

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

2 ДИАГРАММА КЛАССОВ

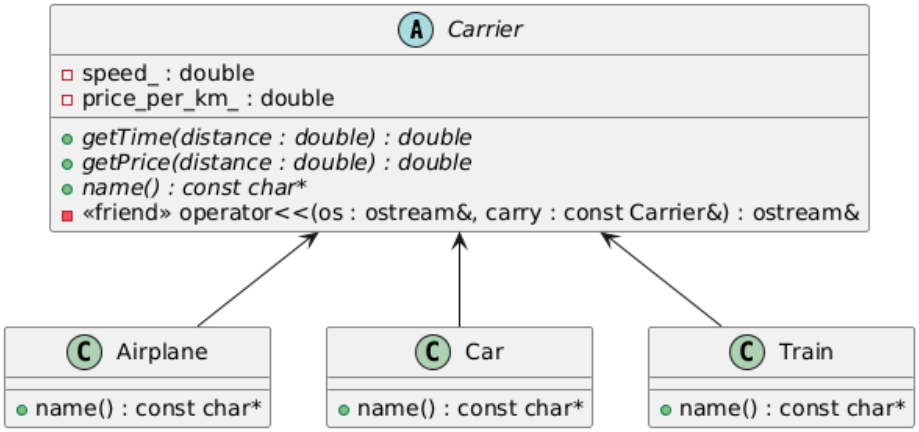

Рисунок 2.1 – Диаграмма классов

3 ЛИСТИНГ КОДА

Файл main.cc:

```
#include <consoleUtils.hh>
#include <functional>
#include <l3/include/airplane.hh>
#include <l3/include/car.hh>
#include <l3/include/screens.hh>
#include <l3/include/train.hh>
#include <memory>

using namespace std;
using namespace carriers;
using namespace console_utils;
using namespace screens;

int main(void) {
    unique_ptr<Carrier> carry_ptr;
    static array<function<bool()>, 4> actions = {
        [&carry_ptr]() { return createCarrier(carry_ptr); },
        [&carry_ptr]() { return printCarrier(carry_ptr.get()); },
        [&carry_ptr]() { return calculate(carry_ptr.get()); },
        []() { return false; },

    };
```

```
    unsigned int response;
    do {
        printMainScreen();
        readT(response, ">", [](unsigned int num) { return num > 0 && num <=
4; });
        cout << "\x{1B}[2J\x{1B}[H\n";
    } while (actions[response - 1]());

    return 0;
}
```

Файл carrier.hh:

```
#pragma once
#include <cmath>
#include <format>
#include <iostream>
#include <stdexcept>

namespace carriers {
class Carrier {
   private:
    double speed_;
    double price_per_km_;

   private:
    friend std::ostream& operator<<(std::ostream& os, const Carrier& carry) {
        os << std::format("{}: [speed: {}, price: {}]", carry.name(),
carry.speed_, carry.price_per_km_) << std::endl;
        return os;
    }

   public:
    Carrier(double speed, double price_per_km);
    virtual double getTime(double distance) const;
    virtual double getPrice(double distance) const;
    virtual const char* name() const = 0;
};

}  // namespace carriers
```

Файл train.hh:

```
#pragma once
#include <l3/include/carrier.hh>
namespace carriers {
class Train : public Carrier {
   public:
    Train(double speed, double price_per_km);
    const char* name() const override;
};
```

```
}  // namespace carriers
```

Файл airplane.hh:

```
#pragma once
#include "carrier.hh"

namespace carriers {
class Airplane : public Carrier {
   public:
    Airplane(double speed, double price_per_km);
    const char* name() const override;
};
}  // namespace carriers
```

Файл car.hh:

```
#pragma once
#include "carrier.hh"

namespace carriers {
class Car : public Carrier {
   public:
    Car(double speed, double price_per_km);
    const char* name() const override;
};
}  // namespace carriers
```

Файл screens.hh:

```
#pragma once
#include <memory>
#include "carrier.hh"
namespace screens {
void printMainScreen();

bool createCarrier(std::unique_ptr<carriers::Carrier> &carry_ptr);
bool printCarrier(const carriers::Carrier *carry_ptr);
bool calculate(const carriers::Carrier *carry_ptr);
}  // namespace screens
```

Файл carrier.cc:

```
#include <l3/include/carrier.hh>
namespace carriers {

Carrier::Carrier(double speed, double price_per_km) : speed_{speed},
price_per_km_{price_per_km} {
```

```
    if (speed_ <= 0) throw std::invalid_argument("speed should be > 0");
}

// add amount into account
double Carrier::getTime(double distance) const { return distance / speed_; }
// add amount into account
double Carrier::getPrice(double distance) const { return distance *
price_per_km_; }

}  // namespace carriers
```

Файл screens.cc:

```
#include <consoleUtils.hh>
#include <l3/include/airplane.hh>
#include <l3/include/car.hh>
#include <l3/include/train.hh>
#include <memory>
#include <print>
using namespace std;
using namespace carriers;
using namespace console_utils;

namespace screens {
void printMainScreen() {
    auto [cols, rows] = getConsoleDimensions();
    println("{:^{}}", "\x{1B}[48;5;35mLab 3\x{1B}[0m", cols);
    println("Please select action:\n");
    println("    1.Create carrier");
    println("    2.Print carrier");
    println("    3.Calculate");
    println("    4.Exit");
}

bool createCarrier(unique_ptr<Carrier> &carry_ptr) {
    unsigned int response;
    double speed;
    double cost;
    println("What type of carrier to create?");
    println("   1. Airplane");
    println("   2. Car");
    println("   3. Train");
    readT(response, ">", [](unsigned int num) { return num > 0 && num <= 3;
});
    readT(speed, "Please enter speed (speed > 0): ", [](double num) { return
num > 0; });
    readT(cost, "Please enter cost per km: ");

    switch (response) {
        case 1:
            carry_ptr = make_unique<Airplane>(speed, cost);
            break;
        case 2:
```

```
            carry_ptr = make_unique<Car>(speed, cost);
            break;
        case 3:
            carry_ptr = make_unique<Train>(speed, cost);
            break;
        default:
            break;
    }
    return true;
}
bool printCarrier(const Carrier *carry_ptr) {
    if (!carry_ptr) {
        cout << "None, please create one first" << endl;
        return true;
    }
    cout << *carry_ptr;
    return true;
}
bool calculate(const Carrier *carry_ptr) {
    if (!carry_ptr) {
        cout << "No carrier, please create one first" << endl;
        return true;
    }
    size_t distance;
    readT(distance, "Pleaes enter distance: ");
    cout << format("time: {:.2f} cost: {:.2f}", carry_ptr->getTime(distance),
carry_ptr->getPrice(distance)) << endl;

    return true;
}
}  // namespace screens
```

Файл train.cc:

```
#include <l3/include/train.hh>
namespace carriers
{
    Train::Train(double speed, double price_per_km) : Carrier{speed,
price_per_km} {};
    const char* Train::name() const { return "Train"; }
} // namespace carriers
```

Файл airplane.cc:

```
#include <l3/include/airplane.hh>

namespace carriers
{
    Airplane::Airplane(double speed, double price_per_km) : Carrier{speed,
price_per_km} {};
```

```
    const char* Airplane::name() const { return "Airplane"; }
} // namespace carriers
```

Файл car.cc:

```
#include <l3/include/car.hh>

namespace carriers
{
    Car::Car(double speed, double price_per_km) : Carrier{speed,
price_per_km} {};
    const char* Car::name() const { return "Car"; }
} // namespace carriers
```

Файл consoleUtils.cc

```
#include <iostream>

#ifdef __linux__
#include <sys/ioctl.h>
#include <unistd.h>
#endif

#ifdef _WIN32
#include <windows.h>
#endif

namespace console_utils {

std::pair<int, int> getConsoleDimensions() {
#ifdef _WIN32
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
    return std::make_pair(csbi.srWindow.Right - csbi.srWindow.Left + 1,
csbi.srWindow.Bottom - csbi.srWindow.Top + 1);
#endif
#ifdef __linux__

    struct winsize w;
    ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);
    return std::make_pair(w.ws_col, w.ws_row);

#endif
}
}  // namespace console_utils
```

Файл consoleUtils.hh

```
#pragma once
#include <iostream>
#include <limits>

namespace console_utils {


std::pair<int, int> getConsoleDimensions();
```

```
template <typename T, typename CT>
void readT(T& data, const std::string& massage, CT bound) {
    std::cout << massage;
    while (((std::cin >> data).fail()) || !bound(data)) {
        std::cout << "Invalid input. Reread input requierments\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), ' ');
        std::cout << massage;
    }
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
template <typename T>
void readT(T& data, const std::string& massage) {
    std::cout << massage;
    while ((std::cin >> data).fail()) {
        std::cout << "Invalid input. Reread input requierments\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), ' ');
        std::cout << massage;
    }
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
template <typename T, typename CT>
void readT(T& data, const std::string& message, CT bound, const std::string&
errmess) {
    std::cout << message;
    while (((std::cin >> data).fail()) || !bound(data)) {
        std::cout << errmess;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << message;
    }
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
}  // namespace console_utils
```

4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Рисунок 4.1 – Главное меню

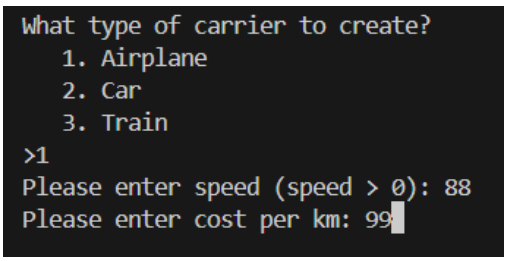

Рисунок 4.2 – Меню создания перевозчика

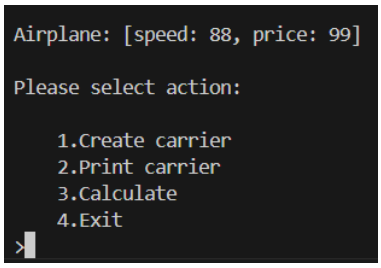

Рисунок 4.3 – Вывод перевозчика

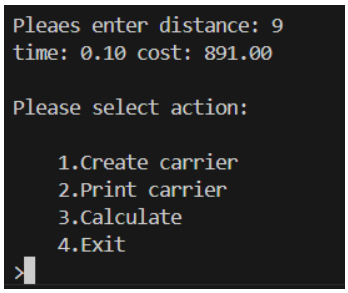

Рисунок 4.4 – Расчёт времени и цены

5 ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были успешно применены принципы объектно-ориентированного программирования, в частности, наследование. Был создан базовый класс Carrier, от которого унаследованы специализированные классы Airplane, Train и Car. Это позволило избежать дублирования кода и организовать логическую иерархию сущностей.