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

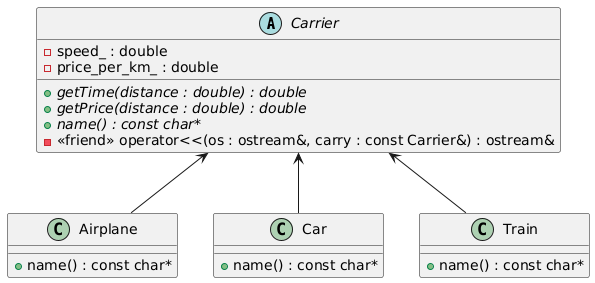


Рисунок 2.1 – Диаграмма классов

# 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**

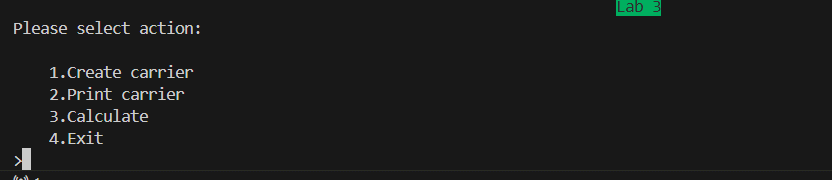


Рисунок 4.1 – Главное меню

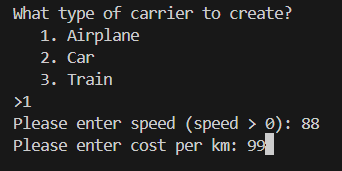


Рисунок 4.2 – Меню создания перевозчика

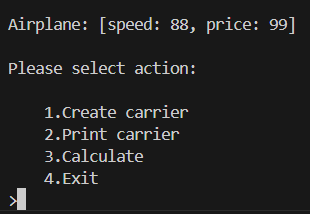


Рисунок 4.3 – Вывод перевозчика

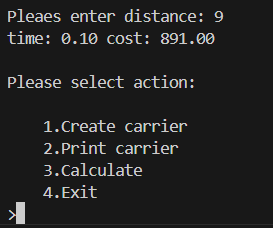


Рисунок 4.4 – Расчёт времени и цены

**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.