

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

Кафедра электронных вычислительных машин

ОТЧЕТ  
О ЛАБОРАТОРНОЙ РАБОТЕ № 6  
Генерация и обработка исключительных ситуаций  
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

На основе созданных в л.р. №3 классов, продемонстрировать генерацию исключений. Промоделировать ситуацию, в которой будет осуществляться проверка ввода строк. Например, если в строку необходимо записать цифры, пробелы, строчные и прописные латинские символы, то при вводе другой информации сгенерировать исключительную ситуацию. В сообщении об ошибке должна входить информация о том, что введена не указанная информация.

## 2 ДИАГРАММА КЛАССОВ

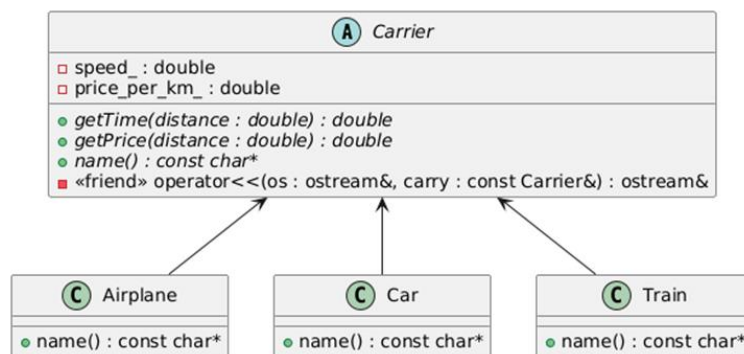


Рисунок 2.1 – Диаграмма классов

## 3 ЛИСТИНГ КОДА

Файл `main.cc`:

```
#include <functional>
#include <l3/include/carrier.hh>
#include <l6/include/screens.hh>
#include <l6/include/throwingConsoleUtils.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 {
    bool failed;
    printMainScreen();
    do {
        failed = false;
        try {
            readT(response, ">", [](unsigned int number) { return number
> 0 && number <= 4; });
        } catch (const std::runtime_error& e) {
            std::cerr << e.what() << '\n';
            failed = true;
        }
    }while (failed);
    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 <l3/include/airplane.hh>
#include <l3/include/car.hh>
#include <l3/include/train.hh>
#include <l6/include/throwingConsoleUtils.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;
    println("What type of carrier to create?");
    println("    1. Airplane");
    println("    2. Car");
    println("    3. Train");
    try {
        readT(response, ">", [](unsigned int num) { return num > 0 && num <=
3; });
    }
}
```

```

    } catch (const console_utils::invalid_input_error &) {
        std::cerr << "Invalid input" << '\n';

        return true;
    }

    switch (response) {
        case 1:
            carry_ptr = make_unique<Airplane>();
            break;
        case 2:
            carry_ptr = make_unique<Car>();
            break;
        case 3:
            carry_ptr = make_unique<Train>();
            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;
    try {
        readT(distance, "Pleaes enter distance: ");
    } catch (const console_utils::invalid_input_error &) {
        std::cerr << "Invalid input" << '\n';
        return true;
    }

    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 throwingConsoleUtils.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
}
```

```
#endif
}
} // namespace console_utils
```

## Файл throwingConsoleUtils.hh

```
#pragma once
#include <format>
#include <iostream>
#include <limits>
#include <exception>
namespace console_utils {

std::pair<int, int> getConsoleDimensions();
struct invalid_input_error : public std::exception {};
template <typename T, typename CT>
void readT(T& data, const std::string& message, CT bound) {
    std::cout << message;
    std::cin >> data;
    if (std::cin.fail() || !bound(data)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

        throw invalid_input_error();
    }
}
template <typename T>
void readT(T& data, const std::string& message) {
    std::cout << message;
    std::cin >> data;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    if (std::cin.fail()) {
        std::cin.clear();
        throw invalid_input_error();
    }
}
}
```

## 4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

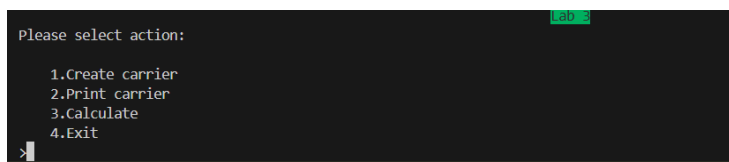


Рисунок 4.1 – Главное меню



```
What type of carrier to create?
1. Airplane
2. Car
3. Train
>1
Please enter speed (speed > 0): 88
Please enter cost per km: 99
```

Рисунок 4.2 – Меню создания перевозчика

```
Airplane: [speed: 88, price: 99]

Please select action:

1.Create carrier
2.Print carrier
3.Calculate
4.Exit
>
```

Рисунок 4.3 – Вывод перевозчика

```
Pleaes enter distance: 9
time: 0.10 cost: 891.00

Please select action:

1.Create carrier
2.Print carrier
3.Calculate
4.Exit
>
```

Рисунок 4.4 – Расчёт времени и цены

## 5 ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была успешно продемонстрирована генерация и обработка исключительных ситуаций в программе. На основе ранее созданных классов транспортных средств была реализована система валидации ввода, которая проверяет корректность введённых данных и генерирует исключения при нарушении заданных условий.