Технологии конструирования программного обеспечения

Отчет по лабораторной работе № 05

Группа: 221-329 **Студент:** Минчаков Аркадий Сергеевич

Задание на лабораторную работу:

Для заданного варианта задания разработать UML-диаграмму классов и диаграмму последовательности. Разработать консольное приложение (C++, C#). Допускается вводить дополнительные понятия предметной области. В программе предусмотреть тестирование функциональности созданных объектов классов.

1 Изучить пример проектирования программной системы с использованием паттерна Состояние [Турчин-Архитектура ИС.pdf [Электронный ресурс], с. 143–154].

2 Разработать диаграмму конечных автоматов (состояний) для заданного класса (таблица 1). Описать в форме таблицы варианты реакции экземпляра класса на операции, вызываемые в указанных состояниях.

3 Разработать библиотеку классов, включающую необходимые классы для реализации паттерна Состояние (класс Конечный автомат, интерфейс Состояние, классы Конкретные состояния).

4 Для заданных вариантов разработать UML-диаграммы классов, состояний и диаграммы последовательности

5 <Опция на дополнительные баллы>: Разработать приложение Windows Forms для управления состояниями экземпляров класса Конечный автомат. При использовании Windows-форм вместо исходного кода в отчет вставить ссылку на репозиторий GitHub с проектом.

Номер в списке группы - 15, вариант 3

3	*Грузовой лифт* *Атрибуты:* текущий этаж, грузоподъёмность, вероятность отключения электроэнергии. Операции: вызвать на заданный этаж, загрузить, разгрузить, восстановить подачу энергии	Покой, Движение, Перегружен, Нет питания, Авария

Figure 1: Таблица 1

UML-диаграмма классов:

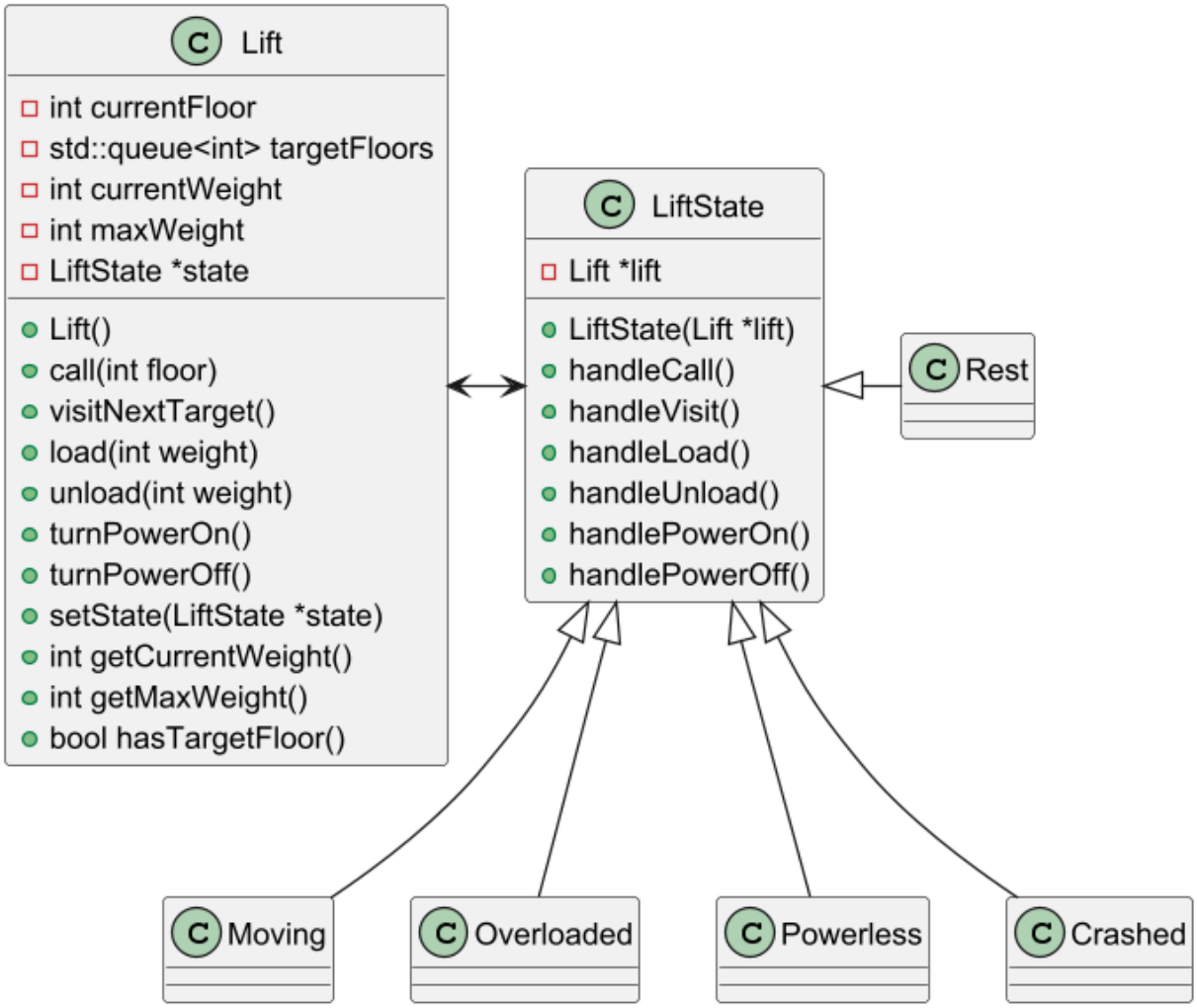

Figure 2: UML-диаграмма классов

Диаграмма последовательности:

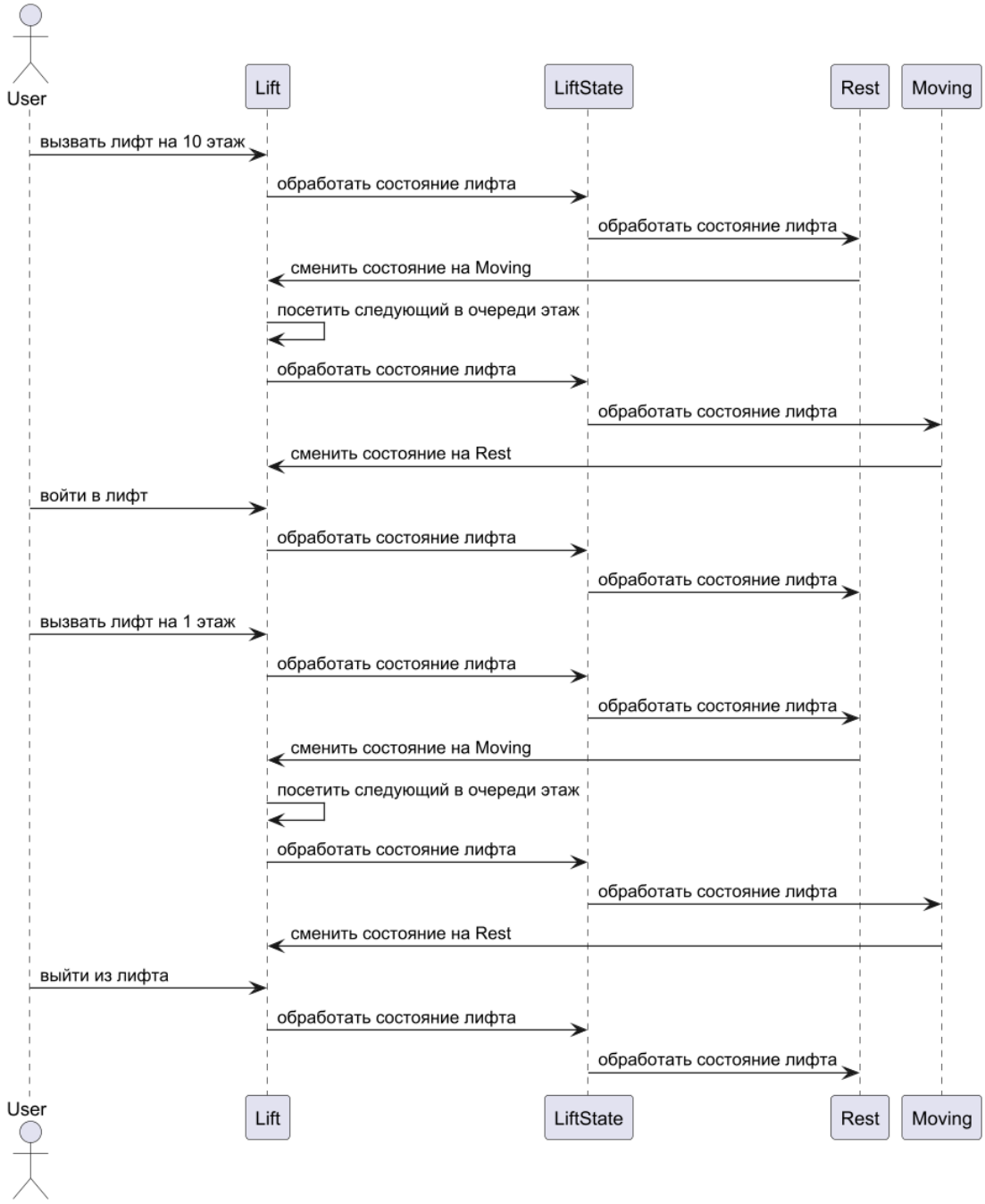

Figure 3: Диаграмма последовательности

Диаграмма состояний:

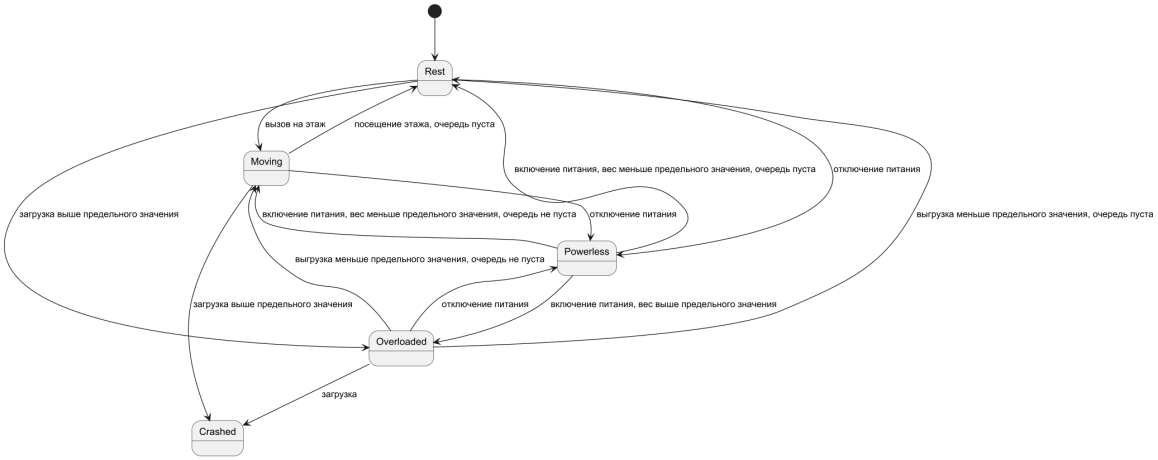

Figure 4: Диаграмма состояний

Исходный код программы:

LiftState.h

```
#ifndef SOFTWARE_DESIGN_TECHNOLOGIES_LIFTSTATE_H
#define SOFTWARE_DESIGN_TECHNOLOGIES_LIFTSTATE_H


#include <iostream>
#include <queue>

class Lift;

class LiftState {
protected:
    Lift *lift;

public:
    explicit LiftState(Lift *lift) : lift(lift) {}

    virtual void handleCall() {};

    virtual void handleVisit() {};

    virtual void handleLoad() {};

    virtual void handleUnload() {};

    virtual void handlePowerOn() {};

    virtual void handlePowerOff() {};
};

class Rest : public LiftState {
public:
    explicit Rest(Lift *lift);

    void handleCall() override;

    void handleLoad() override;

    void handlePowerOff() override;
};

class Moving : public LiftState {
public:
    explicit Moving(Lift *lift);

    void handleVisit() override;

    void handleLoad() override;

    void handlePowerOff() override;
};

class Overloaded : public LiftState {
public:
    explicit Overloaded(Lift *lift);

    void handleLoad() override;

    void handleUnload() override;

    void handlePowerOff() override;
};
```

```
62
63  class Powerless : public LiftState {
64  public:
65      explicit Powerless(Lift *lift);
66
67      void handlePowerOn() override;
68  };
69
70  class Crashed : public LiftState {
71  public:
72      explicit Crashed(Lift *lift);
73  };
74
75  class Lift {
76  private:
77      int currentFloor;
78      std::queue<int> targetFloors;
79      int currentWeight;
80      int maxWeight;
81      LiftState *state;
82
83  public:
84      Lift();
85
86      void call(int floor);
87
88      void visitNextTarget();
89
90      void load(int weight);
91
92      void unload(int weight);
93
94      void turnPowerOn();
95
96      void turnPowerOff();
97
98      void setState(LiftState *state);
99
100     int getCurrentWeight();
101
102     int getMaxWeight();
103
104     bool hasTargetFloor();
105 };
106
107
108 #endif //SOFTWARE_DESIGN_TECHNOLOGIES_LIFTSTATE_H
```

LiftState.cpp

```
1   #include "LiftState.h"
2
3   void Lift::call(int floor) {
4       std::wcout << L"Лифт вызвали на этаж " << floor << std::endl;
5       if (floor == currentFloor) {
6           return;
7       }
8       targetFloors.push(floor);
9       state->handleCall();
10  }
11
12  void Lift::visitNextTarget() {
13      if (!targetFloors.empty()) {
```

```
14          std::wcout << L"Лифт посетил следующий этаж из очереди: " <<
                targetFloors.front() << std::endl;
15          targetFloors.pop();
16      }
17      std::wcout << L"Нет этажа, который лифт мог бы посетить" << std::endl;
18      state->handleVisit();
19  }
20
21  void Lift::load(int weight) {
22      currentWeight += weight;
23      std::wcout << L"В лифт загрузили " << weight << L" кг, текущий вес: " <<
            currentWeight << L" кг" << std::endl;
24      state->handleLoad();
25  }
26
27  void Lift::unload(int weight) {
28      currentWeight -= weight;
29      std::wcout << L"Из лифта выгрузили " << weight << L" кг, текущий вес: " <<
            currentWeight << L" кг" << std::endl;
30      state->handleUnload();
31  }
32
33  void Lift::turnPowerOn() {
34      std::wcout << L"Питание лифта выключено" << std::endl;
35      state->handlePowerOn();
36  }
37
38  void Lift::turnPowerOff() {
39      std::wcout << L"Питание лифта включено" << std::endl;
40      state->handlePowerOff();
41  }
42
43  void Lift::setState(LiftState *state) {
44      this->state = state;
45  }
46
47  int Lift::getCurrentWeight() {
48      return currentWeight;
49  }
50
51  int Lift::getMaxWeight() {
52      return maxWeight;
53  }
54
55  bool Lift::hasTargetFloor() {
56      return !targetFloors.empty();
57  }
58
59  Lift::Lift() :
60      currentFloor(1),
61      targetFloors(),
62      currentWeight(0),
63      maxWeight(1000),
64      state(new Rest(this)) {}
65
66  void Rest::handleCall() {
67      lift->setState(new Moving(lift));
68      delete this;
69  }
70
71  void Rest::handleLoad() {
72      if (lift->getCurrentWeight() > lift->getMaxWeight()) {
73          lift->setState(new Overloaded(lift));
74          delete this;
75      }
76  }
```

```
77
78  void Rest::handlePowerOff() {
79      lift->setState(new Powerless(lift));
80      delete this;
81  }
82
83  Rest::Rest(Lift *lift) : LiftState(lift) {
84      std::wcout << L"Лифт в состоянии 'Покой'" << std::endl;
85  }
86
87  void Moving::handleLoad() {
88      if (lift->getCurrentWeight() > lift->getMaxWeight()) {
89          lift->setState(new Crashed(lift));
90          delete this;
91      }
92  }
93
94  void Moving::handlePowerOff() {
95      lift->setState(new Powerless(lift));
96      delete this;
97  }
98
99  Moving::Moving(Lift *lift) : LiftState(lift) {
100     std::wcout << L"Лифт в состоянии 'Движение'" << std::endl;
101 }
102
103 void Moving::handleVisit() {
104     if (!lift->hasTargetFloor()) {
105         lift->setState(new Rest(lift));
106         delete this;
107     }
108 }
109
110 void Overloaded::handleLoad() {
111     lift->setState(new Crashed(lift));
112     delete this;
113 }
114
115 void Overloaded::handleUnload() {
116     if (lift->getCurrentWeight() <= lift->getMaxWeight()) {
117         if (lift->hasTargetFloor()) {
118             lift->setState(new Moving(lift));
119         } else {
120             lift->setState(new Rest(lift));
121         }
122         delete this;
123     }
124 }
125
126 void Overloaded::handlePowerOff() {
127     lift->setState(new Powerless(lift));
128 }
129
130 Overloaded::Overloaded(Lift *lift) : LiftState(lift) {
131     std::wcout << L"Лифт в состоянии 'Перегружен'" << std::endl;
132 }
133
134 void Powerless::handlePowerOn() {
135     if (lift->getCurrentWeight() > lift->getMaxWeight()) {
136         lift->setState(new Overloaded(lift));
137     } else {
138         if (lift->hasTargetFloor()) {
139             lift->setState(new Moving(lift));
140         } else {
141             lift->setState(new Rest(lift));
142         }
```

```
    }
    delete this;
}

Powerless::Powerless(Lift *lift) : LiftState(lift) {
    std::wcout << L"Лифт в состоянии 'Нет питания'" << std::endl;
}

Crashed::Crashed(Lift *lift) : LiftState(lift) {
    std::wcout << L"Лифт в состоянии 'Авария'" << std::endl;
}
```

main.cpp

```
#include "LiftState.h"

int main() {
    setlocale(LC_ALL, "Russian");


    std::wcout << L"Сценарий использования 'Поездка с 10 этажа на 1'" << std::endl <<
        std::endl;

    Lift lift;
    lift.call(10);
    lift.visitNextTarget();
    lift.load(60);
    lift.call(1);
    lift.visitNextTarget();
    lift.unload(60);

    std::wcout << std::endl << std::endl << L"Сценарий использования 'Аварийная
        перегрузка'" << std::endl << std::endl;

    lift = Lift();
    lift.load(1000);
    lift.load(500);
    lift.call(5);
    lift.call(7);
    lift.unload(500);
    lift.visitNextTarget();
    lift.visitNextTarget();
    lift.unload(1000);
    lift.load(1001);
    lift.load(1);

    return 0;
}
```