**Частное учреждение образования «Минский колледж предпринимательства»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 27**

**по учебному предмету**

**«Технология разработки программного обеспечения»**

**Вариант 2**

*technicaldocs.ru*

*Подпись и дата*

*Взам. инв. №*

*Инв. № дубл.*

*Инв. № подл.*

*Подпись и дата*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | Выполнил учащийся 3 курса группы П-2108 |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Тростянко С.А. |
|  |  |  |
|  |  | Преподаватель |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.П.Кислюк |

2023

**Тема:** разработка программ с использованием паттернов поведения.

**Цель работы**: научиться разрабатывать программы с использованием поведенческих шаблонов проектирования Стратегия, Состояние и Шаблонный метод.

**Вариант 2**.

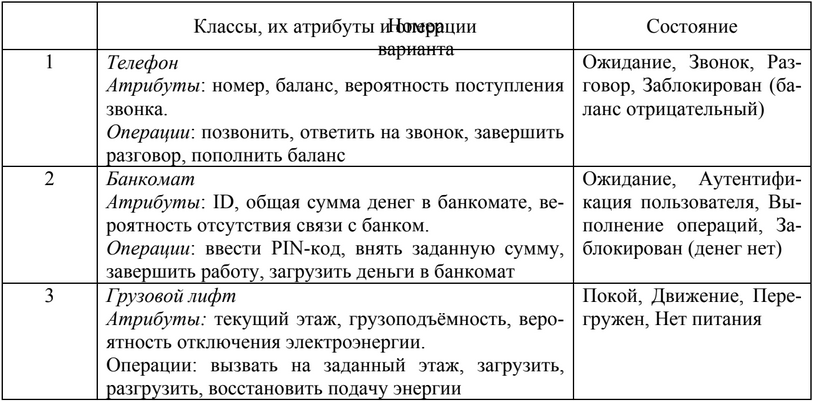


Рис. 1. Вариант задания

**Ход работы:**

Была разработана программа по заданному условию: выполнение программы отображено на Рис. 2. Диаграмма конечных автоматов представлена на Рис. 3.

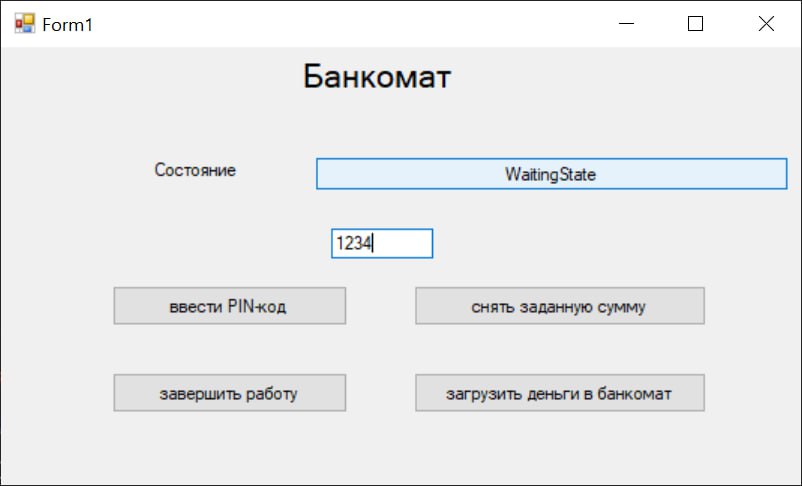


Рис. 2. Результат выполнения программы

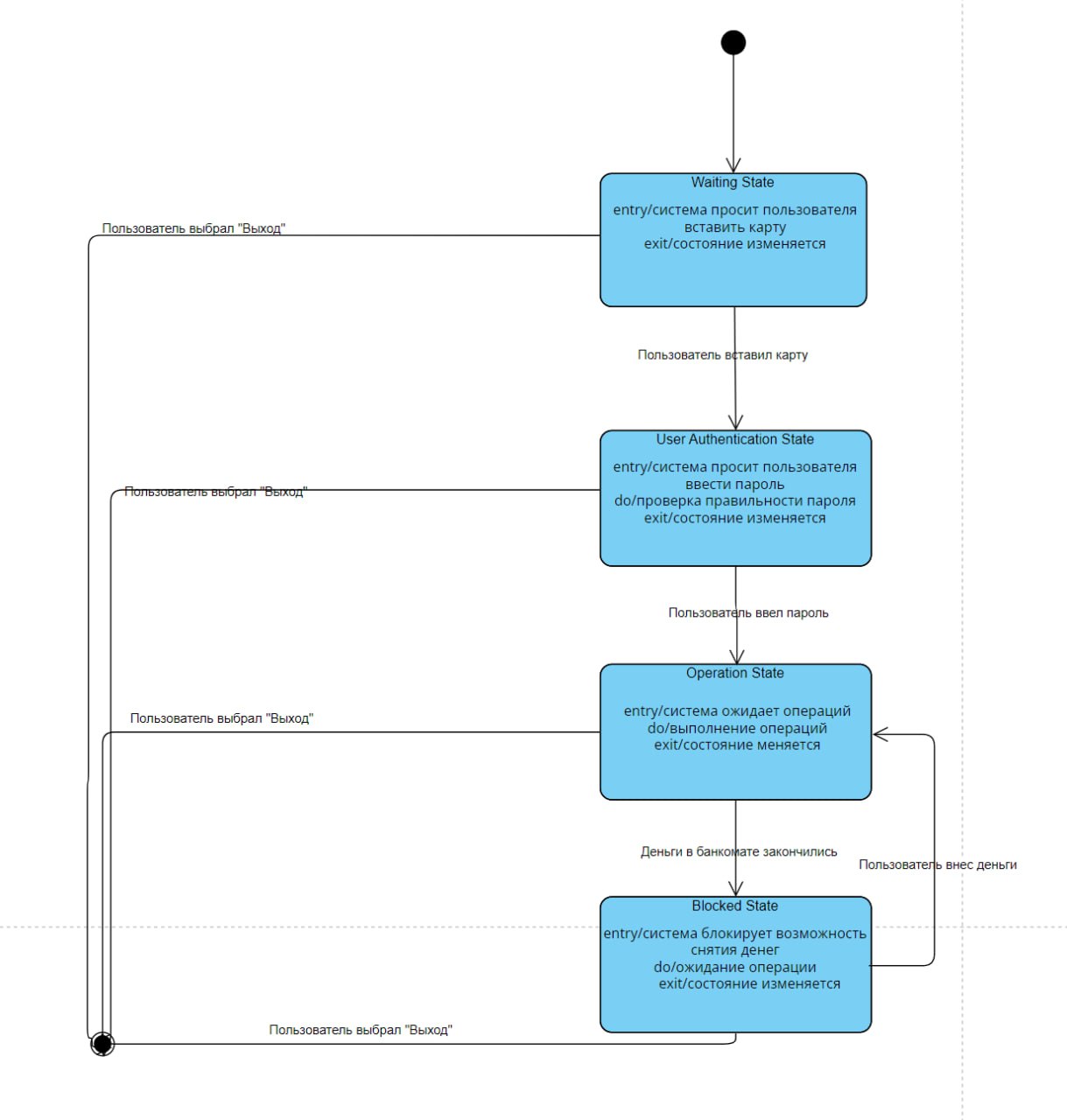
****

Рис. 3. Диаграмма конечных автоматов

**Вывод**: я научился разрабатывать программы с использованием поведенческих шаблонов проектирования Стратегия, Состояние и Шаблонный метод.

**Контрольные вопросы:**

1. Каково назначение поведенческих шаблонов проектирования?

Они определяют взаимодействие и коммуникацию между объектами в системе.

Облегчают изменение поведения системы, позволяя изменять алгоритмы или стратегии выполнения во время выполнения программы.

Повышают гибкость и переиспользуемость кода, позволяя отделять аспекты поведения от основной логики системы.

Позволяют управлять сложными взаимодействиями и упрощают понимание системы за счет организации поведения объектов вокруг общих шаблонов.

2. Какие основные элементы используются на диаграммах конечных автоматов UML?

Состояние (State): представляет определенное состояние объекта.

Переход (Transition): определяет условия и события, при которых объект переходит из одного состояния в другое.

Событие (Event): вызывает переход между состояниями.

Действие (Action): определяет действия, выполняемые объектом при переходе между состояниями.

Начальное состояние (Initial State): определяет начальное состояние объекта.

Конечное состояние (Final State): определяет конечное состояние объекта.

3. Для чего предназначен поведенческий шаблон Состояние?

Предназначен для представления объекта, который может изменять свое поведение в зависимости от его внутреннего состояния. Этот шаблон позволяет объекту изменять свое поведение, вызывая различные методы, связанные с текущим состоянием объекта.

4. Какое решение предлагается в шаблоне Состояние?

Решение, предлагаемое в шаблоне Состояние:

Шаблон Состояние предлагает создание отдельных классов для каждого состояния объекта. Каждый класс состояния реализует одинаковый интерфейс, чтобы объект мог переключаться между различными состояниями. Переключение состояний происходит путем вызова методов, специфичных для текущего состояния объекта.

5. Что понимают под стратегией в шаблоне Стратегия?

В шаблоне Стратегия под стратегией понимается алгоритм или стратегия выполнения задачи, который может быть выбран или заменен во время выполнения программы. Стратегия определяет, как будет выполнена задача или как будет применен алгоритм.

6. Какое решение предлагается в шаблоне Стратегия?

Решение, предлагаемое в шаблоне, Стратегия: Шаблон Стратегия предлагает определение отдельных классов-стратегий, которые реализуют общий интерфейс. Клиентский объект выбирает и использует конкретную стратегию во время выполнения программы. Это позволяет клиенту изменять поведение объекта, выбирая разные стратегии.

7. Какую проблему позволяет решить шаблон Шаблонный метод?

Шаблонный метод позволяет решить проблему повторяющихся шагов в алгоритме, предоставляя общий шаблон для алгоритма и позволяя подклассам переопределить определенные шаги алгоритма по своему усмотрению. Это позволяет улучшить гибкость и переиспользуемость кода, а также избежать дублирования кода для схожих алгоритмов.

**Листинг кода:**

namespace WinFormsApp3;

public partial class Form1 : Form

{

Lift lift;

System.Threading.Timer electricityTimer;

public Form1()

{

InitializeComponent();

lift = new Lift(0, 1000, this);

electricityTimer = new System.Threading.Timer(ElectricityTimer\_Tick, null, 0, 10000);

}

private void Form1\_Load(object sender, EventArgs e)

{

Update1();

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void ElectricityTimer\_Tick(object state)

{

if (lift.HasPower)

{

lift.ChangeState(new IdleState(), "No Power");

Update1();

}

else

{

lift.ChangeState(new NoPowerState(), "Idle");

Update1();

}

}

private void button3\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

}

private void button4\_Click(object sender, EventArgs e)

{

}

public interface ILiftState

{

void MoveToFloor(Lift lift, int floor);

void Load(Lift lift, double weight);

void Unload(Lift lift, double weight);

void RestoreElectricity(Lift lift);

}

public class IdleState : ILiftState

{

public void MoveToFloor(Lift lift, int floor)

{

lift.CurrentFloor = floor;

lift.ChangeState(new MovingState(), "Moving");

}

public void Load(Lift lift, double weight)

{

if (lift.CurrentLoad + weight > lift.Capacity)

{

lift.ChangeState(new OverloadedState(), "Overloaded");

}

else

{

lift.CurrentLoad += weight;

}

}

public void Unload(Lift lift, double weight)

{

if (lift.CurrentLoad - weight < 0)

{

lift.CurrentLoad = 0;

}

else

{

lift.CurrentLoad -= weight;

}

}

public void RestoreElectricity(Lift lift)

{

lift.HasPower = true;

lift.ChangeState(new IdleState(), "Idle");

}

}

public class MovingState : ILiftState

{

public void MoveToFloor(Lift lift, int floor)

{

lift.CurrentFloor = floor;

}

public void Load(Lift lift, double weight)

{

lift.ChangeState(new OverloadedState(), "Overloaded");

}

public void Unload(Lift lift, double weight)

{

if (lift.CurrentLoad - weight < 0)

{

lift.CurrentLoad = 0;

}

else

{

lift.CurrentLoad -= weight;

}

}

public void RestoreElectricity(Lift lift)

{

lift.HasPower = true;

}

}

public class OverloadedState : ILiftState

{

public void MoveToFloor(Lift lift, int floor)

{

// Lift cannot move when overloaded

}

public void Load(Lift lift, double weight)

{

// No more load can be added

}

public void Unload(Lift lift, double weight)

{

lift.CurrentLoad -= weight;

if (lift.CurrentLoad <= lift.Capacity)

{

lift.ChangeState(new IdleState(), "Idle");

}

}

public void RestoreElectricity(Lift lift)

{

lift.HasPower = true;

}

}

public class NoPowerState : ILiftState

{

public void MoveToFloor(Lift lift, int floor)

{

// Lift cannot move without power

}

public void Load(Lift lift, double weight)

{

// Lift cannot load without power

}

public void Unload(Lift lift, double weight)

{

// Lift cannot unload without power

}

public void RestoreElectricity(Lift lift)

{

lift.HasPower = true;

lift.ChangeState(new IdleState(), "Idle");

}

}

public class Lift

{

public int CurrentFloor { get; set; }

public double Capacity { get; set; }

public double CurrentLoad { get; set; }

public bool HasPower { get; set; }

private ILiftState \_state;

public Form1 \_form;

public Lift(int currentFloor, double capacity, Form1 form)

{

CurrentFloor = currentFloor;

Capacity = capacity;

\_state = new IdleState();

HasPower = true;

\_form = form;

\_form.label1.Text = "Idle";

}

public void ChangeState(ILiftState newState, string stateName)

{

\_state = newState;

\_form.label1.Text = stateName;

}

public void MoveToFloor(int floor)

{

if (!HasPower)

{

\_form.label1.Text = "No Power";

return;

}

\_state.MoveToFloor(this, floor);

}

public void Load(double weight)

{

if (!HasPower)

{

\_form.label1.Text = "No Power";

return;

}

\_state.Load(this, weight);

}

public void Unload(double weight)

{

if (!HasPower)

{

\_form.label1.Text = "No Power";

return;

}

\_state.Unload(this, weight);

}

public void RestoreElectricity()

{

\_state.RestoreElectricity(this);

}

}

private void timer1\_Tick(object sender, EventArgs e)

{

Update1();

if (lift.HasPower)

{

lift.ChangeState(new NoPowerState(), "No Power");

}

else

{

lift.ChangeState(new IdleState(), "Idle");

}

}

public void Update1()

{

if (label1.Text == "No Power")

{

button1.Enabled = false;

button2.Enabled = false;

button3.Enabled = false;

button4.Enabled = true;

}

if (label1.Text == "Idle")

{

button1.Enabled = true;

button2.Enabled = true;

button3.Enabled = true;

button4.Enabled = false;

}

if (label1.Text == "Overloaded")

{

button1.Enabled = false;

button2.Enabled = true;

button3.Enabled = false;

button4.Enabled = false;

}

if (label1.Text == "Moving")

{

button1.Enabled = false;

button2.Enabled = false;

button3.Enabled = false;

button4.Enabled = false;

}

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

electricityTimer.Dispose();

}

private void button3\_Click\_1(object sender, EventArgs e)

{

MessageBox.Show("Едем к этажу: " + textBox3.Text);

int floor = int.Parse(textBox3.Text);

lift.MoveToFloor(floor);

label1.Text = "Moving";

Update1();

}

private void button2\_Click\_1(object sender, EventArgs e)

{

double weight = int.Parse(textBox2.Text);

lift.Unload(weight);

MessageBox.Show($"Remaining Load: {lift.Capacity - lift.CurrentLoad}");

Update1();

}

private void button1\_Click\_1(object sender, EventArgs e)

{

double weight = int.Parse(textBox1.Text);

lift.Load(weight);

MessageBox.Show($"Remaining Load: {lift.Capacity - lift.CurrentLoad}");

Update1();

}

private void button4\_Click\_1(object sender, EventArgs e)

{

lift.RestoreElectricity();

lift.ChangeState(new IdleState(), "Idle");

label1.Text = "Idle";

Update1();

}

}