Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

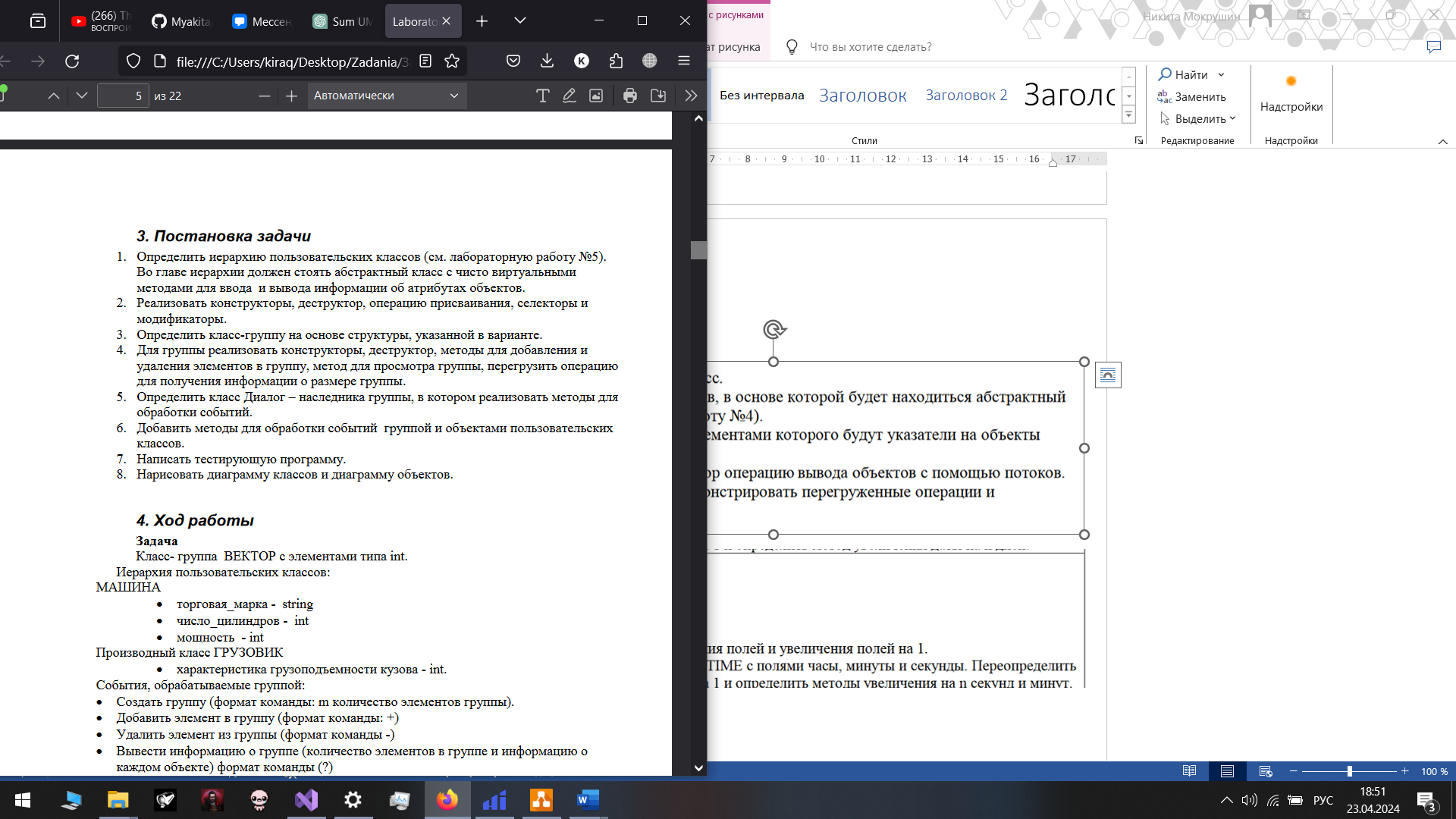
Лабораторная работа  
«Программа управляемая событиями»

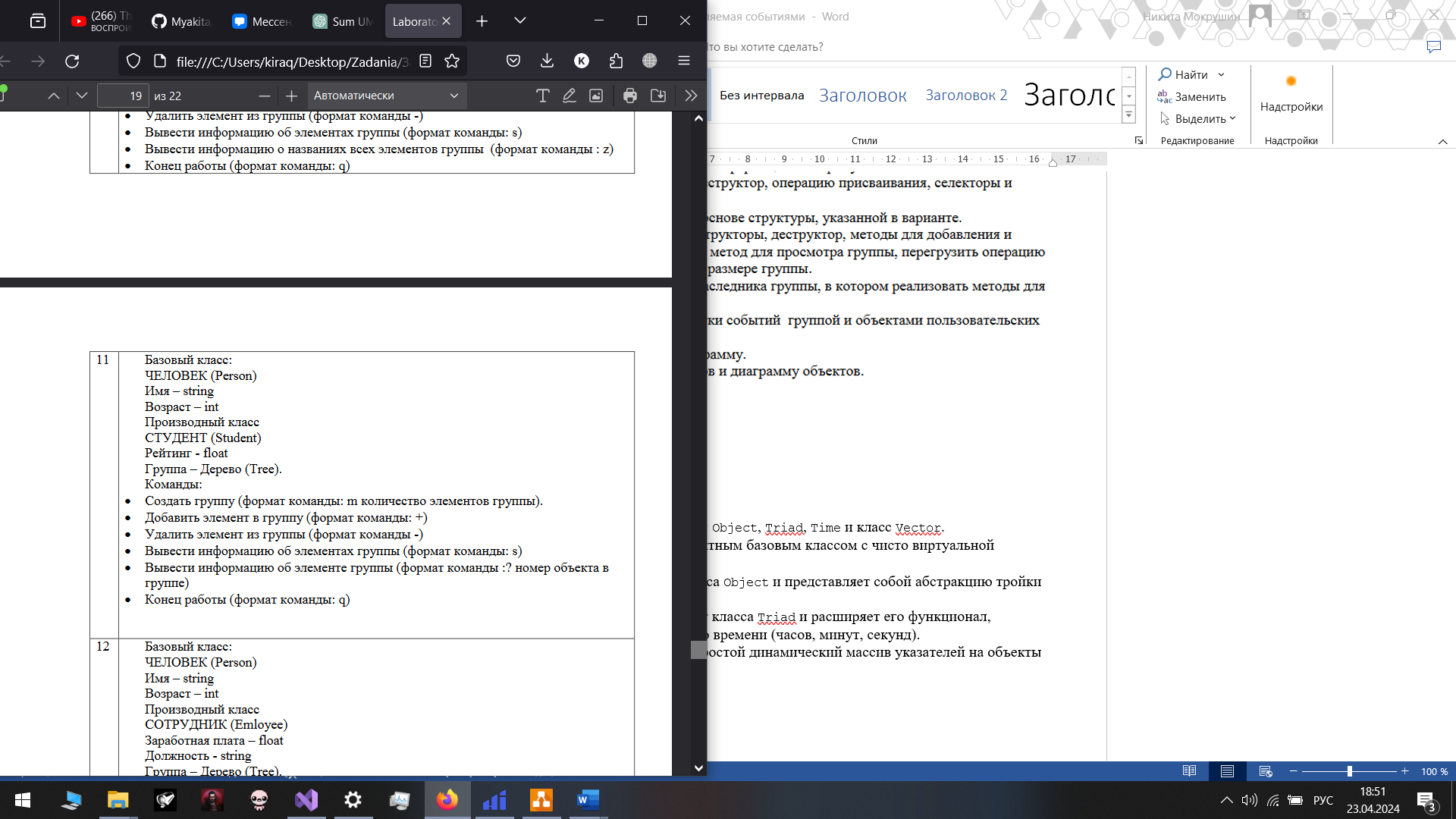
Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

Постановка задачи:





Анализ задачи:

1. **Описание классов:**
   * Есть абстрактный класс Object, который представляет объект и имеет виртуальные функции show(), input() и getname().
   * Person - класс, наследующий от Object, представляет человека с полями имени и возраста.
   * Student - наследник Person, расширяет его, добавляя рейтинг студента.
   * Vector - класс, содержащий массив указателей на объекты класса Object.
2. **Описание интерфейса:**
   * Пользователь может взаимодействовать с программой через консоль, используя определенные команды.
   * Ввод и вывод осуществляется с помощью стандартных потоков ввода-вывода (cin и cout).
3. **Использование динамической памяти:**
   * Для хранения объектов класса Object используется динамический массив указателей.
4. **Управление диалогом:**
   * Диалоговый класс Dialog отвечает за взаимодействие с пользователем и обработку команд.
   * Он обрабатывает события и вызывает соответствующие методы класса Vector.
5. **Управление жизненным циклом:**
   * Управление памятью осуществляется в классе Vector, который отвечает за удаление созданных объектов при удалении самого себя.

Код:

#include <iostream>

#include <string>

using namespace std;

class Object {

public:

Object() {}

virtual void show() = 0;

virtual void input() = 0;

virtual string getname() = 0;

virtual ~Object() {}

};

class Person : public Object {

public:

Person() : name("noname"), age(0) {}

Person(string n, int a) : name(n), age(a) {}

Person(const Person& p) : name(p.name), age(p.age) {}

~Person() {}

int getage() { return age; }

string getname() override { return name; }

void setage(int a) { age = a; }

void setname(string n) { name = n; }

void show() override {

cout << "Возраст: " << age << endl;

cout << "Имя: " << name << endl;

}

void input() override {

cout << "Введите возраст: ";

cin >> age;

cout << endl;

cout << "Введите имя: ";

cin >> name;

cout << endl;

}

private:

string name;

int age;

};

class Student : public Person {

public:

Student() : Person(), reiting(0) {}

Student(string name,int age, float r) : Person(name, age), reiting(r) {}

Student(const Student& s) : Person(s), reiting(s.reiting) {}

~Student() {}

void setreit(float r) { reiting = r; }

float getreit() { return reiting; }

void show() override {

Person::show();

cout << "Рейтинг данного студента: " << reiting << endl;

}

void input() override {

Person::input();

cout << "Введите рейтинг студента: ";

cin >> reiting;

cout << endl;

}

private:

float reiting;

};

class Vector {

public:

Vector(int s) : size(s), cur(0) {

beg = new Object \* [size];

}

~Vector() {

for (int i = 0; i < cur; ++i) {

delete beg[i];

}

delete[] beg;

}

Object\*& operator[](int index) {

if (index >= 0 && index < cur) {

return beg[index];

}

else {

static Object\* null\_ptr = nullptr;

return null\_ptr;

}

}

void add() {

if (cur < size) {

cout << "1. Student" << endl;

int x;

cin >> x;

if (x == 1) {

Student\* s = new Student;

s->input();

beg[cur] = s;

cur++;

}

else {

cout << "Неизвестная операция" << endl;

}

}

else {

cout << "Массив переполнен" << endl;

}

}

void del() {

if (cur == 0)

{

cout << "Пусто"<< endl;

}

else

{

delete beg[cur-1];

cur--;

}

}

void show() {

if (cur == 0) {

cout << "Пусто" << endl;

}

else {

for (int i = 0; i < cur; ++i) {

beg[i]->show();

}

}

}

int operator()() { return cur; }

private:

Object\*\* beg;

int size;

int cur;

};

const int evnothing = 0;

const int evmessage = 100;

const int cmadd = 1;

const int cmdel = 2;

const int cmshow = 3;

const int cmshowelem = 4;

const int cmmake = 6;

const int cmquit = 101;

struct Tevent {

int what;

int command;

};

class Dialog {

public:

Dialog() : EndState(0) {}

void getevent(Tevent& event) {

string OpInt = "m+-s?q";

string s;

char code;

cout << '>';

cin >> s;

code = s[0];

if (OpInt.find(code) != string::npos) {

event.what = evmessage;

switch (code) {

case 'm': event.command = cmmake; break;

case '+': event.command = cmadd; break;

case '-': event.command = cmdel; break;

case 's': event.command = cmshow; break;

case '?': event.command = cmshowelem; break;

case 'q': event.command = cmquit; break;

}

}

else {

event.what = evnothing;

}

}

void execute() {

Tevent event;

do {

EndState = 0;

getevent(event);

handleevent(event);

} while (valid());

}

void handleevent(Tevent& event) {

int num;

if (event.what == evmessage) {

switch (event.command) {

case cmmake:

cout << "Введите размер массива: ";

cin >> size;

vec = new Vector(size);

clearevent(event);

break;

case cmadd:

vec->add();

clearevent(event);

break;

case cmdel:

vec->del();

cout << "Очищен последний студент" << endl;

clearevent(event);

break;

case cmshow:

vec->show();

clearevent(event);

break;

case cmshowelem:

cout << "Введите номер элемента: ";

cin >> num;

num--;

if (num >= 0 && num < (\*vec)()) {

((\*vec)[num])->show();

}

else {

cout << "Неверный номер элемента" << endl;

}

clearevent(event);

break;

case cmquit:

endexec();

break;

default:

cout << "Неизвестная команда" << endl;

break;

}

}

}

void clearevent(Tevent& event) { event.what = evnothing; }

bool valid() { return EndState == 0; }

void endexec() { EndState = 1; }

~Dialog() { delete vec; }

private:

int EndState;

int size;

Vector\* vec;

};

int main() {

setlocale(0, "");

system("chcp 1251>nul");

cout << "m: Создать группу\n+: Добавить элемент\n";

cout << "-: Удалить элемент\ns: Информация о членах группы\n";

cout << "?: Информация об элементе группы(? - номер)\nq: Выход\n";

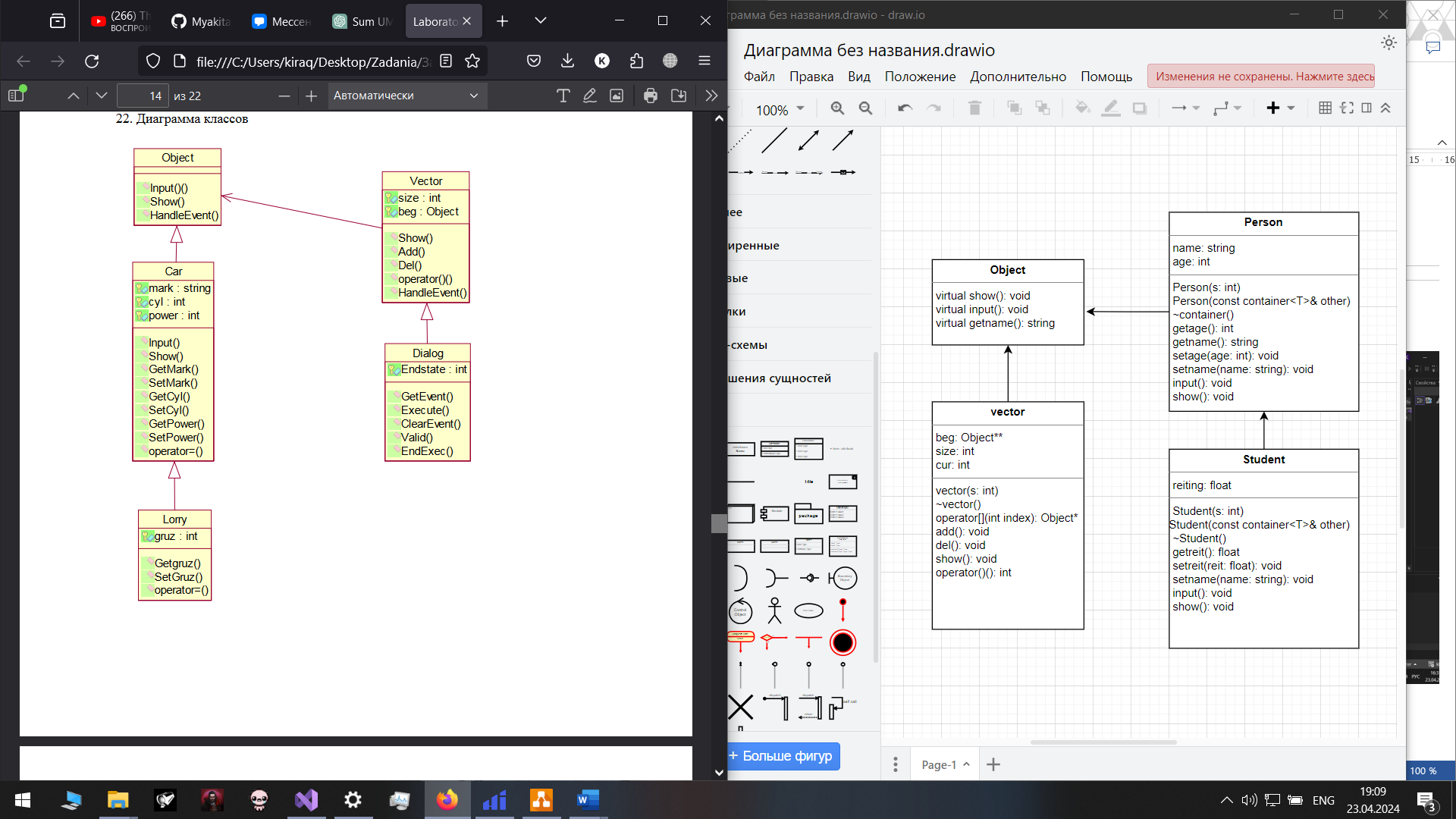
Dialog D;

D.execute();

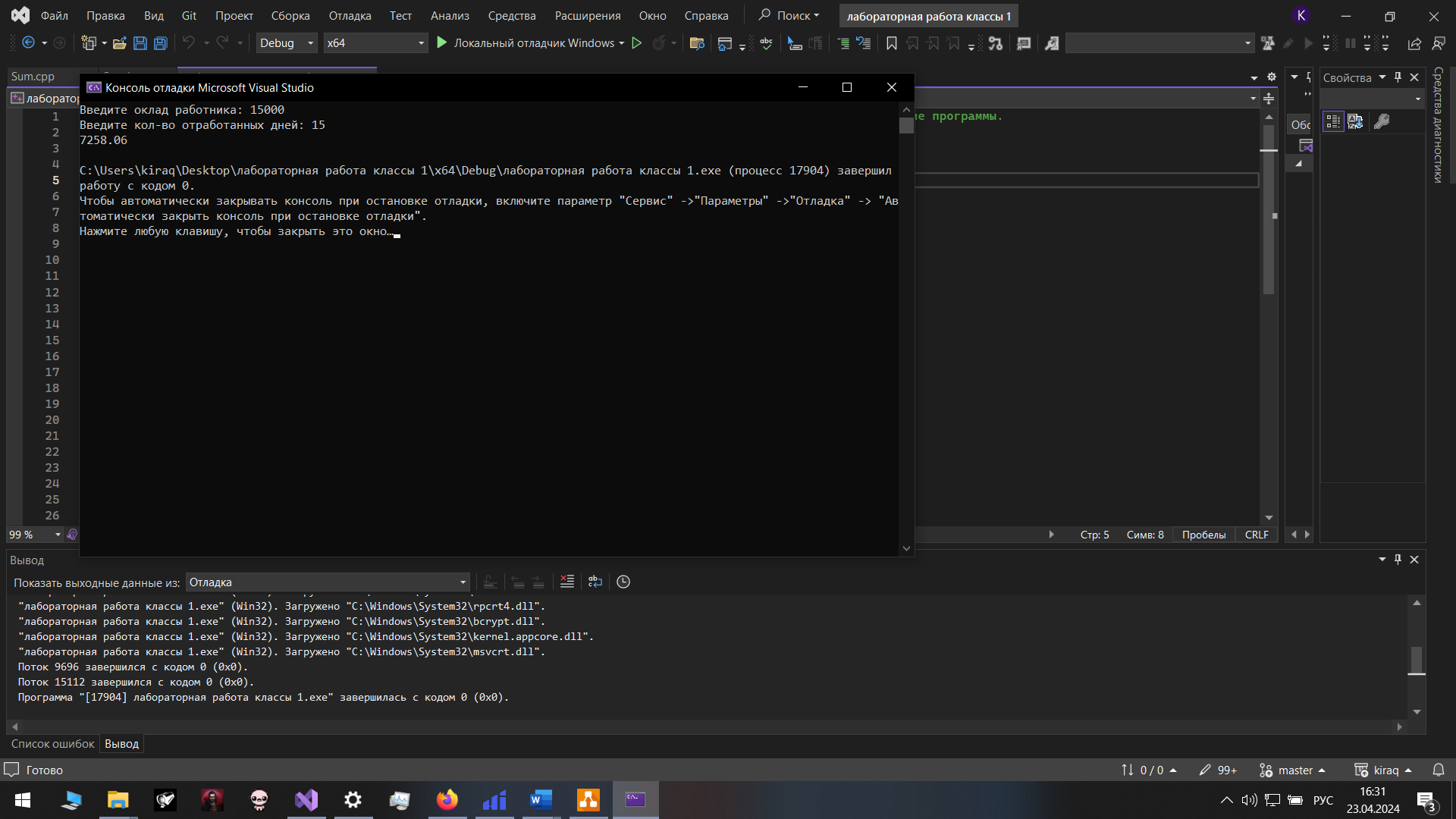
return 0;

}

UML Диаграммы:



Вывод:



программа работает корректно и выдаёт требуемый результат.  
Ответы на вопросы:

1. Класс-группа - это абстрактный класс, который объединяет другие классы в логическую или функциональную группу. Он предоставляет общий интерфейс для работы с этой группой классов. Примеры классов-групп могут включать классы для работы с коллекциями (например, список, стек, очередь), классы для работы с графическими элементами (например, элементы управления), классы для работы с событиями и др.
2. Пример описания класса-группы "Список" может выглядеть следующим образом:

cpp

class List {

public:

virtual void add() = 0;

virtual void remove() = 0;

virtual void show() = 0;

virtual ~List() {}

};

1. Примеры конструкторов для класса-группы "Список":
   * Конструктор без параметров:

cpp

List() {}

* Конструктор с параметром:

cpp

List(int size) {}

* Конструктор копирования:

cpp

1. List(const List& other) {}
2. Пример деструктора для класса-группы "Список":

cpp

virtual ~List() {}

1. Пример метода для просмотра элементов для класса-группы "Список":

cpp

virtual void show() = 0;

1. Группа классов образует иерархию, которая позволяет структурировать и организовать код программы. Она обеспечивает удобное разделение функциональности на логически связанные компоненты.
2. Во главе иерархии классов, содержащихся в группе объектов, должен находиться абстрактный класс, так как он определяет общий интерфейс для всех классов этой группы. Это позволяет использовать полиморфизм для работы с объектами разных конкретных классов через общий интерфейс.
3. Событие представляет собой сообщение о возникновении какого-либо события или действия в программе. События используются для управления потоком выполнения программы, обработки пользовательских действий и реакции на внешние воздействия.
4. Характеристики события-сообщения:
   * Тип события (например, нажатие клавиши, перемещение мыши, команда от пользователя).
   * Параметры события (дополнительные данные, например, координаты мыши, код клавиши, дополнительные параметры команды).
   * Код события (определяет конкретное действие или команду).
5. Пример структуры, описывающей событие:

cpp

struct Event {

int type; // Тип события

union {

MouseEvent mouse; // Параметры мыши

KeyDownEvent keyDown; // Параметры нажатия клавиши

MessageEvent message; // Параметры сообщения

};

};

1. В структуре события TEvent полю what присваиваются значения типа события. Например:

* what = 0 может означать общее сообщение или отсутствие события.
* what = 1 может означать сообщение о нажатии клавиши.
* what = 2 может означать сообщение о перемещении мыши.
* what = 3 может означать сообщение пользовательской команды.

1. В структуре события TEvent полю command присваиваются значения кодов команд. Например:

* command = 1 может означать добавление элемента.
* command = 2 может означать удаление элемента.
* command = 3 может означать выполнение определенной команды.

1. Поля a и message в структуре события TEvent используются для передачи дополнительных параметров команды. Например, a может содержать дополнительный аргумент или индекс, а message может содержать текстовое сообщение или дополнительные данные.
2. Для организации обработки сообщений необходимы методы для обработки каждого типа событий, например, методы для обработки нажатия клавиши, перемещения мыши, пользовательских команд и т. д.
3. Главный цикл обработки событий-сообщений часто является циклом, в котором происходит ожидание событий и их обработка. Он отвечает за получение событий из очереди событий, их анализ и вызов соответствующих обработчиков событий.
4. Метод ClearEvent() выполняет очистку поля события, устанавливая его тип в значение, обозначающее отсутствие события (например, evnothing).
5. Метод HandleEvent() выполняет обработку полученного события, выполняя соответствующие действия в зависимости от типа события и его параметров.
6. Метод GetEvent() используется для получения нового события из источника событий (например, пользовательского ввода, системных сообщений).
7. Поле EndState используется для определения состояния выполнения программы (например, завершение работы программы). Оно содержится в классе Dialog.
8. Функция Valid() используется для проверки текущего состояния программы или события. Например, она может возвращать true, если программа находится в активном состоянии, и false, если программа завершилась или находится в состоянии ожидания.