Пермский национальный исследовательский политехнический университет.

Лабораторная работа № 8 по ООП.

«Программа, управляемая событиями».

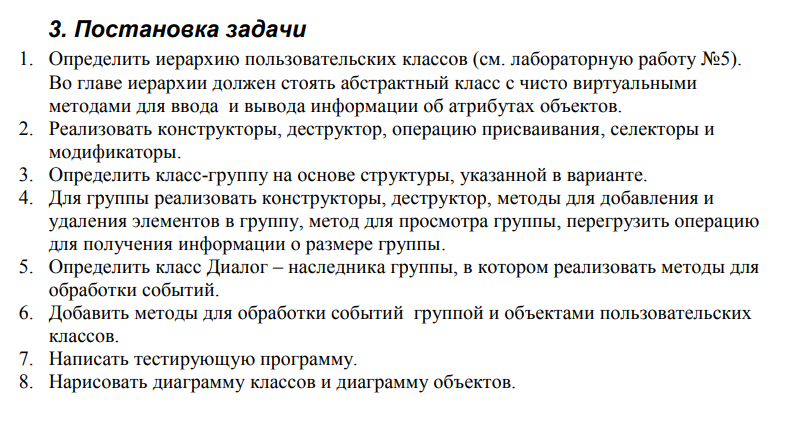
Выполнил: студент группы РИС-23-2б

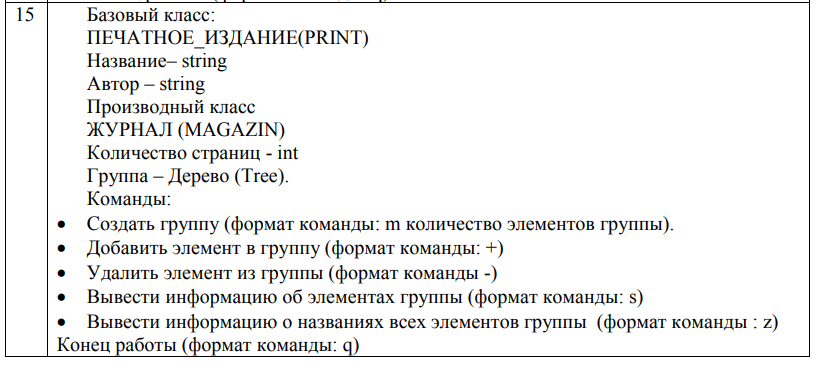
Вековшинин Иван Николаевич

Проверила: доцент кафедры ИТАС

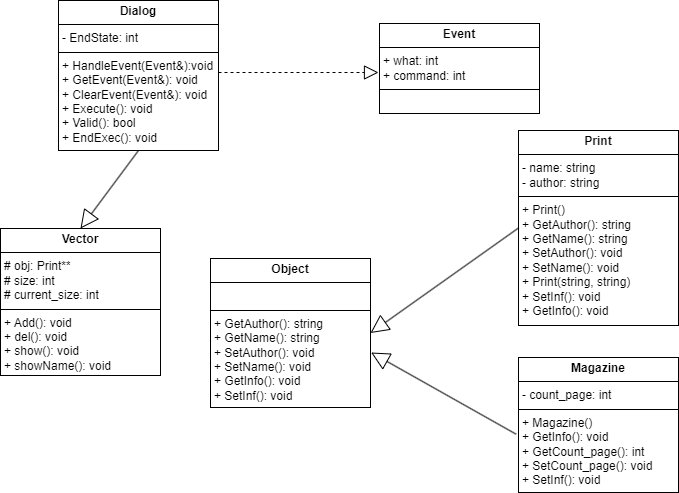
О.А. Полякова.

2024 г.





UML Diagram



Код:

**ClassPrint.h**

#pragma once

#include<iostream>

using namespace std;

class Magazine;

class Object {

public:

virtual string GetAuthor() = 0;

virtual string GetName() = 0;

virtual void SetAuthor(string) = 0;

virtual void SetName(string) = 0;

virtual void GetInfo() = 0;

virtual void SetInf() = 0;

};

class Print : public Object{

string name;

string author;

public:

Print() {

name = "Unknown"; author = "Unknown";

}

string GetAuthor() override;

string GetName() override;

void SetAuthor(string) override;

void SetName(string) override;

Print(string name, string author);

void SetInf() override;

void GetInfo() override {

cout << "Название: " << this->name;

cout << endl << "Автор: " << this->author;

cout << endl;

}

};

Print::Print(string name, string author) {

this->author = author;

this->name = name;

}

string Print::GetAuthor() {

return this->author;

}

string Print::GetName() {

return this->name;

}

void Print::SetName(string a) {

this->name = a;

}

void Print::SetAuthor(string a) {

this->author = a;

}

void Print::SetInf() {

string a, b;

cout << "Введите название: "; cin >> a;

this->name = a;

cout << "Введите автора: "; cin >> b;

this->author = b;

}

class Magazine : public Print {

int count\_page;

public:

Magazine() {

count\_page = 0;

}

void GetInfo() override {

cout << "Название: " << GetName();

cout << endl << "Автор: " << GetAuthor();

cout << "\nКоличество страниц: " << count\_page;

cout << endl;

}

int GetCount\_page();

void SetCount\_page(int);

void SetInf() override;

};

int Magazine::GetCount\_page() {

return this->count\_page;

}

void Magazine::SetCount\_page(int count) {

this->count\_page = count;

}

void Magazine::SetInf() {

string a, b; int k;

cout << "Введите название: "; cin >> a;

this->SetName(a);

cout << "Введите автора: "; cin >> b;

this->SetAuthor(b);

cout << "Введите количество страниц: "; cin >> k;

this->count\_page = k;

}

**Dialog.h**

#pragma once

#include"Vector.h"

#include<iostream>

using namespace std;

void menu();

const int cmCreate = 1;

const int cmAdd = 2;

const int cmDel = 3;

const int cmExit = 4;

const int cmShow = 5;

const int cmShowName = 6;

const int evNothing = 0;

const int evMessage = 100;

class Event {

public:

int what;

int command;

};

class Dialog : public Vector {

int EndState = 0;

public:

void HandleEvent(Event&);

void GetEvent(Event&);

void ClearEvent(Event&);

void Execute();

bool Valid();

void EndExec();

};

void Dialog::HandleEvent(Event& event) {

int k;

if (event.what == evMessage) {

switch (event.command) {

case cmAdd:

Add(); ClearEvent(event); break;

case cmDel:

del(); ClearEvent(event); break;

case cmShow:

show(); ClearEvent(event); break;

case cmShowName:

showName(); ClearEvent(event); break;

case cmExit:

EndExec(); ClearEvent(event); break;

case cmCreate:

cout << "Введите количество элементов: "; cin >> this->size;

obj = new Print \* [size];

ClearEvent(event);

break;

}

}

}

void Dialog::GetEvent(Event& event) {

string commands = "+-szqm";

char c;

system("pause");

system("cls");

menu();

cout << "Введите команду > "; cin >> c;

if (commands.find(c) >= 0) {

event.what = evMessage;

switch (c) {

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

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

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

case 'z': event.command = cmShowName; break;

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

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

}

}

else event.what = evNothing;

}

void Dialog::ClearEvent(Event& a) {

a.what = evNothing;

}

void Dialog::Execute() {

Event event;

do {

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (Valid());

}

bool Dialog::Valid() {

return EndState == 0;

}

void Dialog::EndExec() {

EndState = 1;

}

void menu() {

cout << "Создать группу(m)";

cout << "\nДобавить элемент в группу(+)";

cout << "\nУдалить элемент из группы(-)";

cout << "\nВывести информацию об элементах группы(s)";

cout << "\nВывести информацию о названиях всех элементов группы(z)";

cout << "\nКонец работы(q)";

cout << endl;

}

**Vector.h**

#pragma once

#include<iostream>

#include "ClassPrint.h"

using namespace std;

class Vector

{

protected:

Print\*\* obj;

int size;

int current\_size = 0;

public:

void Add() {

cout << "1. Print\n2. Magazine\n>";

int x; cin >> x;

if (x == 1) {

Print\* tmp = new Print;

tmp->SetInf();

if (current\_size < size) {

obj[current\_size] = tmp;

current\_size++;

}

else {

Print\*\* temp = new Print \* [size + 1];

for (int i = 0; i < size; ++i)

temp[i] = obj[i];

temp[size] = tmp;

delete[] obj;

obj = temp;

current\_size++;

}

}

if (x == 2) {

Magazine\* tmp = new Magazine;

tmp->SetInf();

if (current\_size < size) {

obj[current\_size] = tmp;

current\_size++;

}

else {

Print\*\* temp = new Print \* [size + 1];

for (int i = 0; i < size; ++i)

temp[i] = obj[i];

temp[size] = tmp;

delete[] obj;

obj = temp;

}

}

}

void del() {

Print\*\* temp = new Print \* [size - 1];

for (int i = 0; i < size-1; ++i)

temp[i] = obj[i];

delete[] obj;

obj = temp;

current\_size--;

size--;

}

void show() {

for (int i = 0; i < current\_size; ++i)

obj[i]->GetInfo();

}

void showName() {

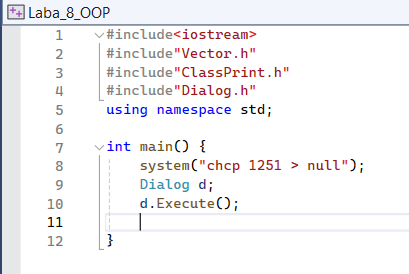
for (int i = 0; i < current\_size; ++i)

obj[i]->GetName();

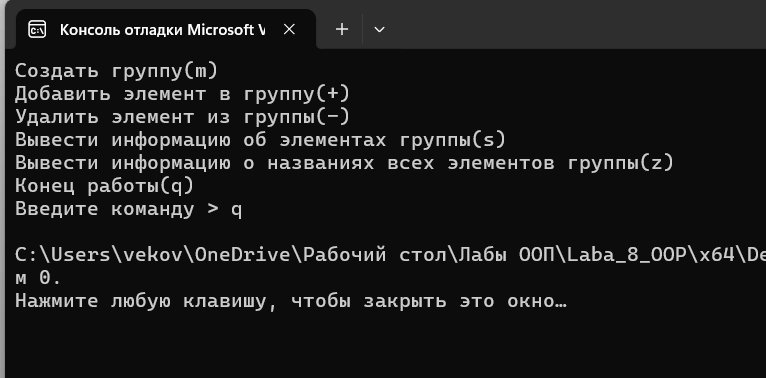
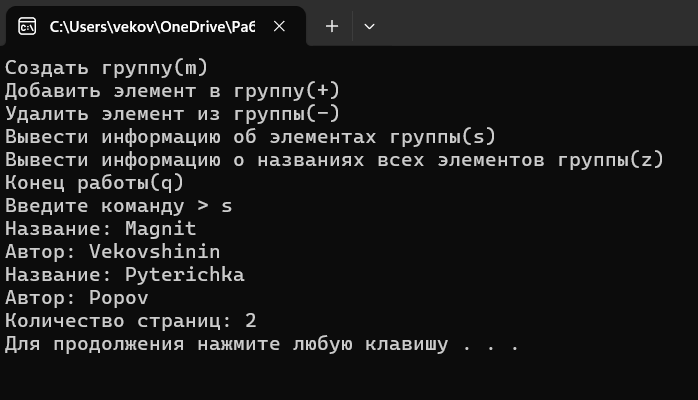
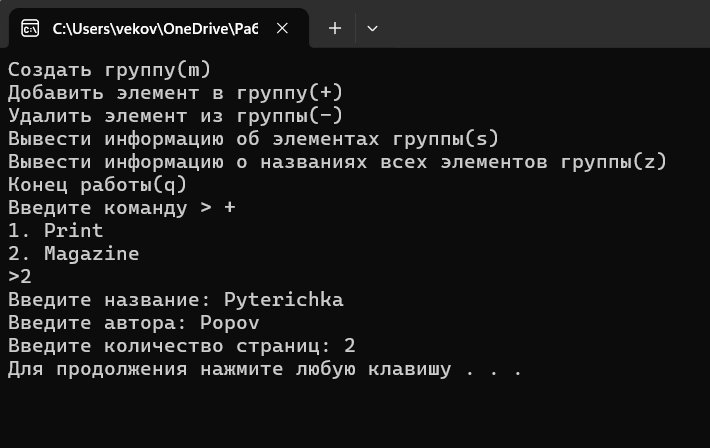
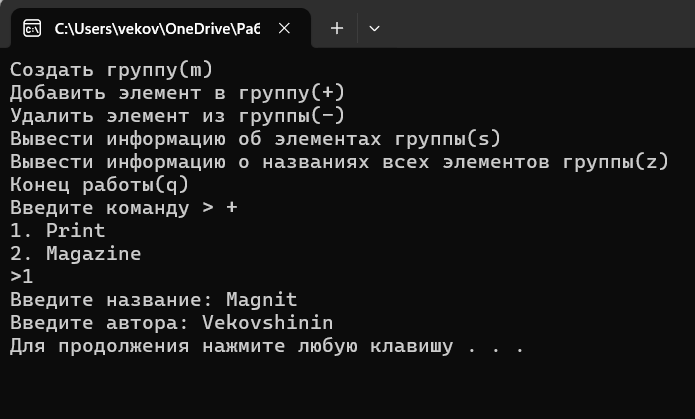
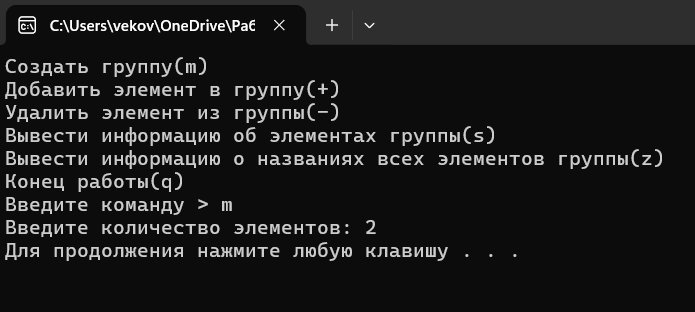
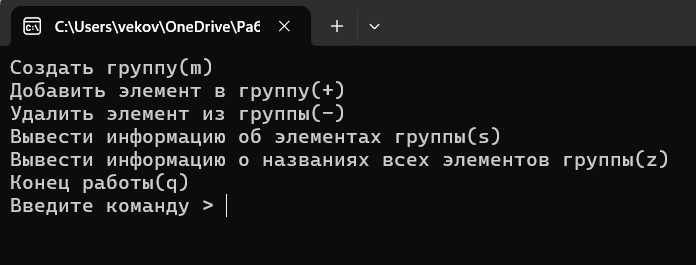
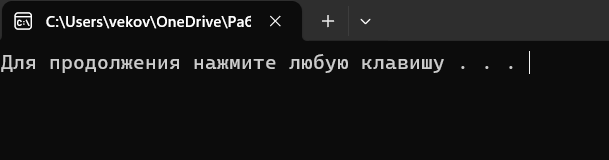
}

};

**Main.cpp**



Результат работы:



Ответы на вопросы:

1)Группа – это объект, в который включены другие объекты. Примеры групп: окно в интерактивной программе, организационная структура(пример: кофейня, кухня, обслуживание).

2)Класс-группа list:

|  |
| --- |
| struct Node  {  //каждый узел списка хранит данные, которые являются указателем на абстрактный класс  Object\* data;  Node\* next\_ptr;  Node() {}  Node(Object\* Data)  {  data = Data;  next\_ptr = nullptr;  }  };  struct List  {  Node\* head;  Node\* tail;  int size;    //конструктор без параметров  List() {  head = nullptr;  tail = nullptr;  }  //конструктор создает список на size элементов  List(int size) {  for (int i = 0; i < size; ++i) {  Node\* node = new Node;  if (i == 0) {  head = node; tail = node;  }  else {  for (auto i = head; i != tail; i = i->next\_ptr) {  i->next\_ptr = node;  }  tail = node;  }  }  }  List(const List& a) {  //выделяем место под список того же размера, как список, который копируем  for (int i = 0; i < a.size; ++i) {  Node\* node = new Node;  if (i == 0) {  head = node; tail = node;  }  else {  for (auto i = head; i != tail; i = head->next\_ptr) {  i->next\_ptr = node;  }  tail = node;  }  }  //копируем данные  Node\* temp = a.head;  for (auto i = head; i != tail; i = i->next\_ptr) {  i->data = temp->data;  temp = temp->next\_ptr;  }  }  void del() //удаление элемента  {  if (head == nullptr) return;  if (head == tail)  {  delete tail;  head = tail = nullptr;  size--;  return;  }  Node\* node = head;  head = node->next\_ptr;  size--;  delete node;  }  void add(Object\* data) //добавление элемента  {  Node\* node = new Node(data);  if (head == nullptr) head = node;  if (tail != nullptr) tail->next\_ptr = node;  tail = node;  size++;  }    ~List()  {  while (head != nullptr) del();  }  }; |

3)Конструкторы (по умолчанию, с параметрами, копирования)

|  |
| --- |
| //конструктор без параметров  List() {  head = nullptr;  tail = nullptr;  }  //конструктор создает список на size элементов  List(int size) {  for (int i = 0; i < size; ++i) {  Node\* node = new Node;  if (i == 0) {  head = node; tail = node;  }  else {  for (auto i = head; i != tail; i = i->next\_ptr) {  i->next\_ptr = node;  }  tail = node;  }  }  }  List(const List& a) {  //выделяем место под список того же размера, как список, который копируем  for (int i = 0; i < a.size; ++i) {  Node\* node = new Node;  if (i == 0) {  head = node; tail = node;  }  else {  for (auto i = head; i != tail; i = head->next\_ptr) {  i->next\_ptr = node;  }  tail = node;  }  }  //копируем данные  Node\* temp = a.head;  for (auto i = head; i != tail; i = i->next\_ptr) {  i->data = temp->data;  temp = temp->next\_ptr;  }  } |

4)Деструктор для класса-группы список

|  |
| --- |
| ~List()  {  while (head != nullptr) del();  } |

5)Метод для просмотра данных списка:

|  |
| --- |
| void getInfo() {  for (auto i = head; i != tail; i = i->next\_ptr) {  i->data->GetInfo();  cout << endl;  }  } |

6)Вид иерархии: агрегация

7) Назначение абстрактного класса заключается в предоставлении общего определения для базового класса, которое могут совместно использовать несколько производных классов.

8)Событие – это ситуация, которая при определенных условиях заставляет базовый алгоритм перейти на другой путь выполнения программы.

9)Код класса сообщения; адрес объекта, которому предназначено сообщение; информационное поле.

10)

|  |
| --- |
| class Event { //класс события  public:  int what; //есть ли событие  int command; //какое событие  }; |

11)evNothing – пустое событие, или evMessage – сообщение о том, что какое-то событие выбрано пользователем.

12) Полю command присваивается целочисленное значение в зависимости от события, которые выбрал пользователь.

13) message – сообщение о событии, a – параметр события.

14)GetEvent – формирование события, Execute – главный цикл обработки сообщений, HandleEvent – обработчик событий, ClearEvent – очищает событие.

15)Главный цикл обработки событий постоянно получает событие путем вызова GetEvent и обрабатывает их с помощью HandleEvent. Цикл завершается, когда поступит событие об окончании работы.

16)Очищает событие, когда оно обработано, чтобы оно не срабатывало далее. Присваивает полю what = evNothing.

17)Метод должен распознать событие и реагировать, событие может распозноваться по полю command. Когда поле command совпало с существующей командой, описанной ранее, вызывается нужый метод выбранный пользователем.

18) GetEvent – формирует событие. В зависимости от выбранного пользователем события присваивает полю command значение, которое отвечает за выполнение выбранного события.

19) EndState отвечает за условие выхода из цикла, т.е. завершение работы программы. Это поле содержит класс Dialog.

20)функция Valid – возвращает true или false, в зависимости от значение EndState, и отвечает за условие выхода из цикла обработки событий.