Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №18.8**

Дисциплина: «Основы теории алгоритмов и структуры данных»  
Тема: Программа, управляемая событиями

Вариант 14

Выполнил:

студент группы РИС-20-2б

Вичугов Алексей Дмитриевич

Проверила:

доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь, 2021

**Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов.

2. Разработка программы, управляемой событиями.

**Постановка задачи**

1. Определить иерархию пользовательских классов. Во главе иерархии должен стоять абстрактный класс с чисто виртуальными методами для ввода и вывода информации об атрибутах объектов.

2. Реализовать конструкторы, деструктор, операцию присваивания, селекторы и модификаторы.

3. Определить класс-группу на основе структуры, указанной в варианте.

4. Для группы реализовать конструкторы, деструктор, методы для добавления и удаления элементов в группу, метод для просмотра группы, перегрузить операцию для получения информации о размере группы.

5. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.

6. Добавить методы для обработки событий группой и объектами пользовательских классов.

7. Написать тестирующую программу.

8. Нарисовать диаграмму классов и диаграмму объектов.

9. Задание варианта: Базовый класс - печатное издание (поля : название и автор). Производный класс – книга (поля: количество страниц и издательство). Группа – дерево. Команды: создать группу, добавить элемент в группу, удалить элемент из группы, вывести информацию об элементах группы, вывести информацию о названии элемента группы с указанным номером, завершить работу.

**Анализ задачи**

1. Описание класса:

static const int leave=0;

static const int make=1;

static const int add=2;

static const int del=3;

static const int show=4;

static const int name=5;

struct Event{

int type;

string message;

int nubmer;

};

class Basic{

virtual void Init()=0;

virtual void Show()=0;

};

class Book:public Basic{

public:

int pages;

string publisher;

Book(){pages=0;publisher="No";}

Book(int p, string pub){pages=p;publisher=pub;}

Book(Book &b) {pages=b.pages;publisher=b.publisher;}

~Book(){}

void Init(){

cout << "Pages: ";

cin >> pages;

cout << "Publisher: ";

cin >> publisher;

}

void Show(){

cout << "Publisher: " << publisher << endl << "Pages: " << pages << endl;

}

void operator =(Book &b){

pages=b.pages;

publisher=b.publisher;

}

};

class Printer:public Basic{

public:

string publication;

string avtor;

Book \*first=NULL;

Book \*second=NULL;

Book \*third=NULL;

Printer(){publication="No";avtor="No";}

Printer(string n, string a, Book \*f, Book \*s, Book \*t){publication=n;avtor=a;first=f;second=s;third=t;}

Printer(Printer &p) {publication=p.publication; avtor=p.avtor; first=p.first; second=p.second; third=p.third;}

~Printer(){}

void Init(){

cout << "Publication: ";

cin >> publication;

cout << "Author: ";

cin >> avtor;

Book \*kniga = new Book;

cout << "First book: " << endl;

kniga->Init();

first=kniga;

kniga = new Book;

cout << "Second book: " << endl;

kniga->Init();

second=kniga;

kniga = new Book;

cout << "Third book: " << endl;

kniga->Init();

third=kniga;

}

void Show(){

cout << "Publication: " << publication << endl << "Author: " << avtor << endl;

cout << "Books: " << endl;

first->Show();

second->Show();

third->Show();

}

void operator =(Printer &p){

publication=p.publication;

avtor=p.avtor;

first=p.first;

second=p.second;

third=p.third;

}

};

class Group{

public:

int size;

Printer \*first;

Printer \*second;

Printer \*third;

Group(){size=0;first=NULL; second=NULL; third=NULL;}

Group(Printer \*p){first=p;size++;}

Group(Group &g){first=g.first;second=g.second;third=g.third;}

~Group(){}

void Add(){

if (size<3){

Printer \*print = new Printer; print->Init();

switch (size){

case 0: first=print; size++; break;

case 1: second=print; size++; break;

case 2: third=print; size++; break;

default: cout << "Size error! " << endl; break;

}

}

else cout << "Group if full! " << endl;

}

void Show(){

int s=1;

cout << endl << "Group show: " << endl;

if (first!=NULL){

cout << s++ << ": " << endl;

first->Show();

}

if (second!=NULL){

cout << s++ << ": " << endl;

second->Show();

}

if (third!=NULL){

cout << s << ": " << endl;

third->Show();

}

}

void Remove(int s){

if (s>size) cout << "Not found! " << endl;

else switch(s){

case 1: first=second;second=third;third=NULL; size--; break;

case 2: second=third; third=NULL; size--; break;

case 3: third=NULL; size--; break;

default: cout << "Not found! " << endl;

}

}

int operator ()(){

return size;

}

};

class Dialog:public Group{

public:

int size;

Group \*g;

Dialog(){size=0;g=NULL;}

void Worker(){

cout << endl << "Enter the symbol of command: " << endl;

cout << "1. Make a group (m); \n2. Add element in group (+); \n3. Delete element of group (-); \n4. Show group (s); \n5. Show name of element (z); \n6. Exit (q). " << endl;

char c;

Event e;

cin >> c;

switch(c){

case 'm': e.type=1; e.message="Empty groub is created. ";Executor(e);break;

case '+': e.type=2; e.message="Add new element: ";Executor(e);break;

case '-': e.type=3; e.message="Enter the number of deleted element: ";Executor(e);break;

case 's': e.type=4; e.message="Group: ";Executor(e);break;

case 'z': e.type=5; e.message="Enter the nubmer of element: ";Executor(e);break;

case 'q': e.type=0; e.message="Exit. ";Executor(e);break;

default: cout << "Command not found. "; Worker(); break;

}

}

void Executor(Event &e){

bool ex=0;

switch(e.type){

case make: g = new Group; cout << e.message << endl; break;

case add: g->Add(); break;

case del: cout << e.message; int k; cin >> k; g->Remove(k);g->Show(); break;

case show: cout << e.message << endl; g->Show(); break;

case name:

cout << e.message; int p; cin >> p;

cout << g->size << endl;

if (p>g->size) cout << "Not found. " << endl;

else switch(p){

case 1: cout << g->first->publication << endl; break;

case 2: cout << g->second->publication << endl; break;

case 3: cout << g->third->publication << endl; break;

}

break;

case leave: ex=1;

}

if (!ex) Worker();

}

};

2. Функция main():

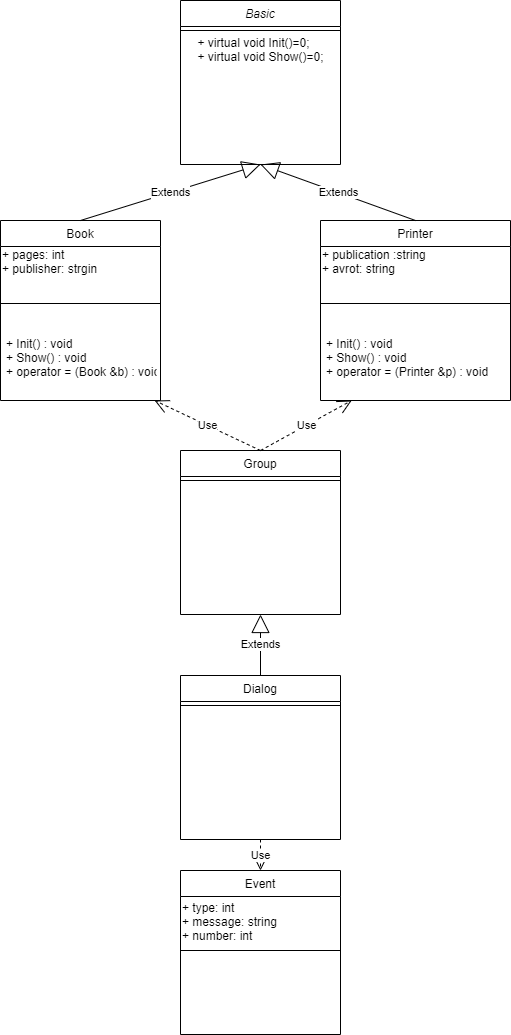
main(){

Dialog d;

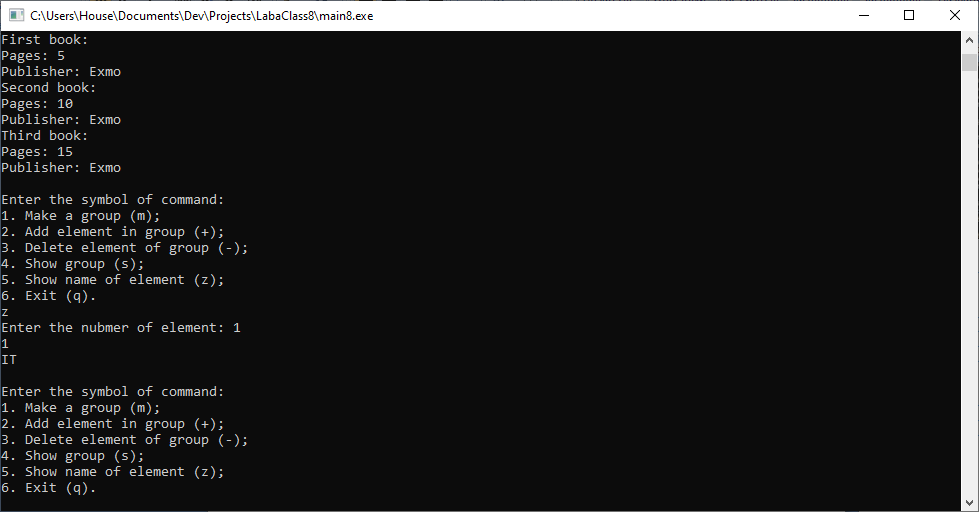
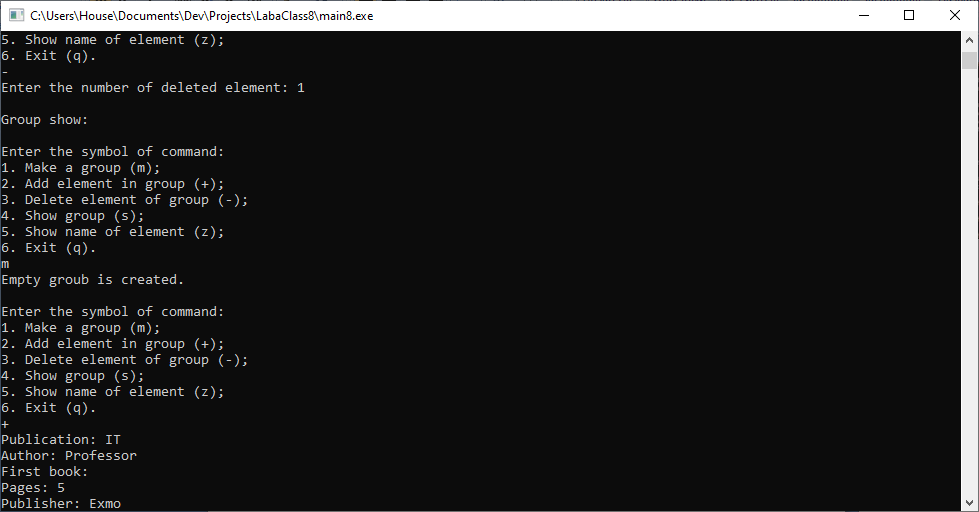
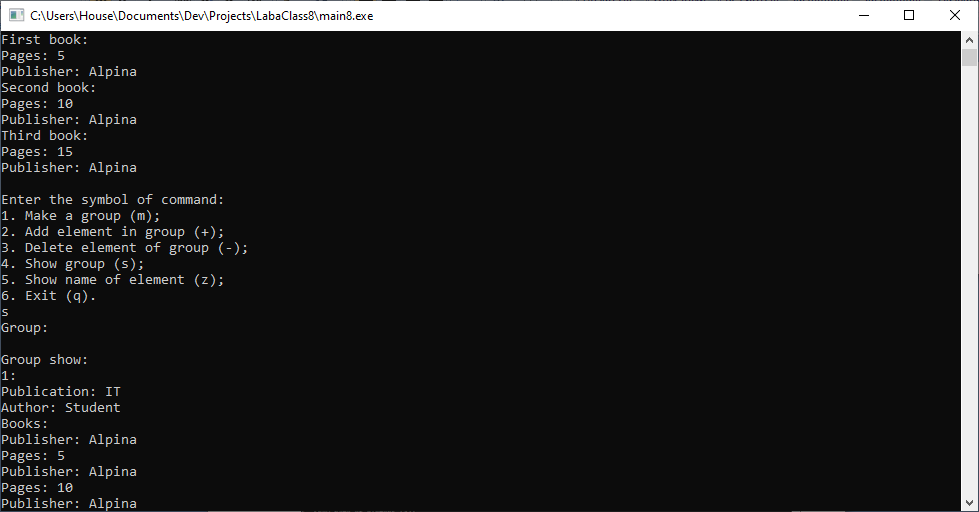
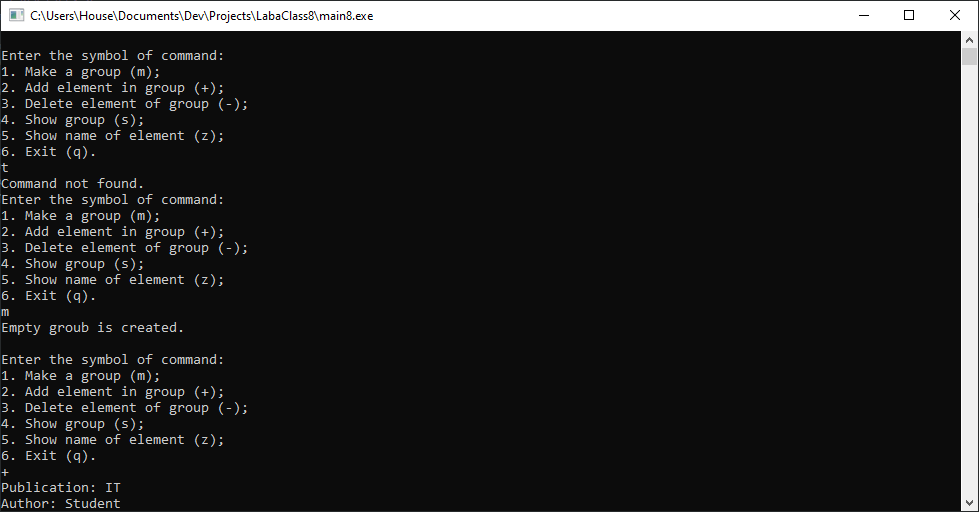
d.Worker();

}

**Диаграмма классов**



**Результаты работы программы**

****

**Ответы на вопросы**

1. Это класс, в который включены объекты (или другие классы).
2. Class people{}; class List {people \*first;};
3. Class people{}; class List{people \*first; List(){first=NULL;} List(people \*p) {first=p;} List (List &l) {first = l.first;}};
4. Class people{}; class List {people \*first; ~List(){delete first;}};
5. Class people{public: int b;}; class List {people \*first; void Show(){cout << \*(first->b);}};
6. Иерархия объектов.
7. Потому что элементы группы должны иметь общие свойства.
8. Пакет информации, которыми обмениваются события. Используется для реакции программы на действия пользователя.
9. Тип события и информацию, передаваемую с событием.
10. Struct event {int what; string message;};
11. Целочисленные, в случае возникновения события.
12. Целочисленные, в случае возникновения события.
13. Для передачи сообщения о событии и параметра команды.
14. Формирование события, определение события, обработка события нужным образом и очистка события после его обработки.
15. Вид цикла, определяющего, какое произошло событие и вызывающего соответствующую функцию, обрабатывающую событие.
16. Очищает событие после его обработки сбросом значения, отвечающего за код события.
17. Определяет тип события и обрабатывает его соответствующим образом, вызывая определённые функции.
18. Формирует событие.
19. Для создания события, вызывающего завершение работы. Объект, хранящий коды событий.
20. Для проверки состояния «конец работы».