Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет», ПНИПУ

ЛАБОРАТОРНАЯ РАБОТА

ПРОГРАММА, УПРАВЛЯЕМАЯ СОБЫТИЯМИ

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**Постановка задачи**

1. Определить иерархию классов, в основе абстрактный класс с чисто виртуальными методами ввода-вывода информации об атрибутах объектов.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Определить класс-группу на основе структуры, указанной в варианте.
4. Для группы реализовать конструкторы, деструктор, методы добавления и удаления элементов группы, метод просмотра группы, перегрузить операцию для получения информации о размере группы.
5. Определить класс Диалог – наследника группы, в котором реализовать методы обработки событий.
6. Добавить методы для обработки событий группой и объектами пользовательских классов.

Для варианта 15:

Базовый класс:

ПЕЧАТНОЕ\_ИЗДАНИЕ(PRINT)

Название – string

Автор – string

Производный класс:

ЖУРНАЛ (MAGAZINE)

Количество страниц – int

Группа – вектор (изменено)

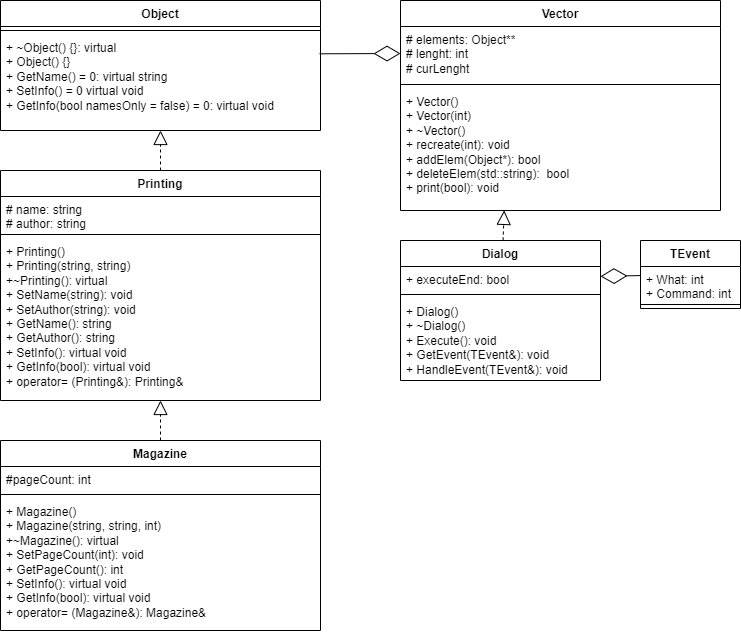
Команды:

* Создать группу (формат команды: m количество элементов группы).
* Добавить элемент в группу (формат: +)
* Удалить элемент из группы (формат: -)
* Вывести информацию об элементах группы (формат: s)
* Вывести информацию о названиях всех элементов группы (формат: z)
* Конец работы (формат: q)

**Анализ задачи**

1. Создаются файлы, в которых определяются иерархия классов единичных объектов и их методы.
2. Создаются файлы, в которых определяются класс-коллекция и его методы.
3. Полем класса коллекции является указатель на массив указателей на абстрактные классы, для поддержки работы с несколькими типами.
4. Класс диалог определяется как потомок класса-коллекции и содержит методы реагирующие на события вызванные пользователем.

**UML**



**Код**

Object.h

#pragma once

#include "Magazine.h"

#include <string>

#include <iostream>

Magazine::Magazine(string inName, string inAuthor, int inPageCount) :Printing(inName, inAuthor) {

pageCount = inPageCount;

}

void Magazine::SetPageCount(int inPageCount)

{

pageCount = inPageCount;

}

int Magazine::GetPageCount()

{

return pageCount;

}

void Magazine::SetInfo()

{

Printing::SetInfo();

cout << endl << "Enter count of printing's pages: ";

cin >> pageCount;

}

void Magazine::GetInfo(bool nameOnly = false)

{

Printing::GetInfo(nameOnly);

if (!nameOnly)

{

cout << "Count of pages: " << GetPageCount();

}

}

Magazine& Magazine::operator= (Magazine& copyBase)

{

Printing::operator=(copyBase);

pageCount = copyBase.pageCount;

return \*this;

}

Printing.h

#pragma once

#include "Object.h"

#include <string>

class Printing : public Object {

protected:

std::string name;

std::string author;

public:

Printing() {};

Printing(std::string, std::string);

virtual ~Printing() {};

void SetName(std::string);

void SetAuthor(std::string);

std::string GetName();

std::string GetAuthor();

virtual void SetInfo();

virtual void GetInfo(bool);

Printing& operator= (Printing&);

};

Printing.cpp

#pragma once

#include "Printing.h"

#include <string>

#include <iostream>

using namespace std;

Printing::Printing(string inName, string inAuthor)

{

name = inName;

author = inAuthor;

}

void Printing::SetName(string inName)

{

name = inName;

}

void Printing::SetAuthor(string inAuthor)

{

author = inAuthor;

}

string Printing::GetName()

{

return name;

}

string Printing::GetAuthor()

{

return author;

}

void Printing::SetInfo()

{

cout << endl << "Enter printing name: ";

getline(cin, name);

cout << endl << "Enter printing author: ";

getline(cin, author);

}

void Printing::GetInfo(bool nameOnly = false)

{

if (!nameOnly) {

cout << endl << "Printing information";

cout << "Name: " << GetName();

cout << "Author: " << GetAuthor();

}

else {

cout << GetName();

}

}

Printing& Printing::operator= (Printing& copyBase)

{

name = copyBase.name;

author = copyBase.author;

return \*this;

}

Magazine.h

#pragma once

#include "Printing.h"

#include <string>

using namespace std;

class Magazine : public Printing

{

protected:

int pageCount;

public:

Magazine() :Printing() { pageCount = 0; };

Magazine(string, string, int);

virtual ~Magazine() {};

void SetPageCount(int);

int GetPageCount();

virtual void SetInfo();

virtual void GetInfo(bool);

Magazine& operator= (Magazine&);

};

Magazine.cpp

#pragma once

#include "Magazine.h"

#include <string>

#include <iostream>

Magazine::Magazine(string inName, string inAuthor, int inPageCount) :Printing(inName, inAuthor) {

pageCount = inPageCount;

}

void Magazine::SetPageCount(int inPageCount)

{

pageCount = inPageCount;

}

int Magazine::GetPageCount()

{

return pageCount;

}

void Magazine::SetInfo()

{

Printing::SetInfo();

cout << endl << "Enter count of printing's pages: ";

cin >> pageCount;

}

void Magazine::GetInfo(bool nameOnly = false)

{

Printing::GetInfo(nameOnly);

if (!nameOnly)

{

cout << "Count of pages: " << GetPageCount();

}

}

Magazine& Magazine::operator= (Magazine& copyBase)

{

Printing::operator=(copyBase);

pageCount = copyBase.pageCount;

return \*this;

}

Vector.h

#include "Vector.h"

#include "Object.h"

#include <string>

#include <iostream>

Vector::Vector()

{

elements = nullptr;

length = 0;

curLength = 0;

}

Vector::Vector(int len)

{

elements = new Object\* [len];

length = len;

curLength = 0;

}

Vector::~Vector()

{

if (!(elements = nullptr)) {

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

delete elements[i];

}

delete[] elements;

}

}

void Vector::recreate(int len)

{

if (!(elements = nullptr)) {

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

delete elements[i];

}

delete[] elements;

}

length = len;

elements = new Object \* [len];

curLength = 0;

}

bool Vector::addElem(Object\* element)

{

if (curLength >= length) {

std::cout << "Коллекция переполнена" << std::endl;

return false;

}

elements[curLength] = element;

curLength++;

return true;

}

bool Vector::deleteElem(std::string name)

{

if (curLength == 0) {

std::cout << "Коллекция пуста" << std::endl;

return false;

}

bool flag = false;

for (int i = 0; i < curLength && !flag; i++) {

flag = elements[i]->GetName() == name;

if (flag) {

Object\* temp = elements[i];

for (int j = i; j < curLength - 1; j++) {

elements[j] = elements[j + 1];

}

delete temp;

curLength--;

}

}

if (!flag) std::cout << "Заданный элемент не найден" << std::endl;

return flag;

}

void Vector::print(bool namesOnly)

{

if (curLength == 0) {

std::cout << "Коллекция пуста" << std::endl;

return;

}

elements[0]->GetInfo(namesOnly);

for (int i = 1; i < curLength; i++) {

std::cout << std::endl;

elements[i]->GetInfo(namesOnly);

}

}

Vector.cpp

#include "Vector.h"

#include "Object.h"

#include <string>

#include <iostream>

Vector::Vector()

{

elements = nullptr;

length = 0;

curLength = 0;

}

Vector::Vector(int len)

{

elements = new Object\* [len];

length = len;

curLength = 0;

}

Vector::~Vector()

{

if (!(elements = nullptr)) {

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

delete elements[i];

}

delete[] elements;

}

}

void Vector::recreate(int len)

{

if (!(elements = nullptr)) {

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

delete elements[i];

}

delete[] elements;

}

length = len;

elements = new Object \* [len];

curLength = 0;

}

bool Vector::addElem(Object\* element)

{

if (curLength >= length) {

std::cout << "Коллекция переполнена" << std::endl;

return false;

}

elements[curLength] = element;

curLength++;

return true;

}

bool Vector::deleteElem(std::string name)

{

if (curLength == 0) {

std::cout << "Коллекция пуста" << std::endl;

return false;

}

bool flag = false;

for (int i = 0; i < curLength && !flag; i++) {

flag = elements[i]->GetName() == name;

if (flag) {

Object\* temp = elements[i];

for (int j = i; j < curLength - 1; j++) {

elements[j] = elements[j + 1];

}

delete temp;

curLength--;

}

}

if (!flag) std::cout << "Заданный элемент не найден" << std::endl;

return flag;

}

void Vector::print(bool namesOnly)

{

if (curLength == 0) {

std::cout << "Коллекция пуста" << std::endl;

return;

}

elements[0]->GetInfo(namesOnly);

for (int i = 1; i < curLength; i++) {

std::cout << std::endl;

elements[i]->GetInfo(namesOnly);

}

}

EventHandle.h

#pragma once

#include "Vector.h"

class TEvent

{

public:

int What;

int Command;

};

class Dialog : public Vector

{

protected:

bool executeEnd;

public:

Dialog();

~Dialog();

void Execute();

void GetEvent(TEvent&);

void HandleEvent(TEvent&);

};

EventHandle.cpp

#pragma once

#include "EventHandle.h"

#include "Magazine.h"

#include <iostream>

const int eventNothing = 0;

const int eventMessage = 100;

const int cmdCreate = 1;

const int cmdAdd = 2;

const int cmdDelete = 3;

const int cmdPrint = 4;

const int cmdPrintNames = 5;

const int cmdQuit = 6;

Dialog::Dialog()

{

executeEnd = false;

}

Dialog::~Dialog()

{

}

void Dialog::Execute()

{

TEvent custEvent;

while (!executeEnd)

{

std::cout << "\nm - создать новую группу элементов\n+ - добавить элемент\n- - удалить элемент\ns - вывод всех элементов\nz - вывод только имен\nq - завершение работы\n";

GetEvent(custEvent);

HandleEvent(custEvent);

}

}

void Dialog::GetEvent(TEvent& custEvent)

{

char CommandSym;

std::cout << "> ";

std::cin >> CommandSym;

custEvent.What = eventMessage;

switch (CommandSym)

{

case 'm':

custEvent.Command = cmdCreate;

break;

case '+':

custEvent.Command = cmdAdd;

break;

case '-':

custEvent.Command = cmdDelete;

break;

case 's':

custEvent.Command = cmdPrint;

break;

case 'z':

custEvent.Command = cmdPrintNames;

break;

case 'q':

custEvent.Command = cmdQuit;

break;

default:

custEvent.What = eventNothing;

break;

}

}

void Dialog::HandleEvent(TEvent& custEvent)

{

if (custEvent.What == eventNothing) {

std::cout << "Некорректный ввод, повторите попытку" << std::endl;

return;

}

std::string Name;

switch (custEvent.Command)

{

case cmdCreate:

int Size;

do

{

std::cout << "Введите размер коллекции: " << std::endl;

std::cin >> Size;

} while (Size <= 0);

recreate(Size);

break;

case cmdAdd:

if (elements == nullptr)

{

std::cout << "Необходимо предварительно создать коллекцию" << std::endl;

return;

}

int number;

do

{

std::cout << "Добавить: 1 - печатное издание, 2 - журнал" << std::endl;

std::cin >> number;

} while (number != 1 && number != 2);

std::cin.ignore();

if (number == 1)

{

Printing\* printing = new Printing;

printing->SetInfo();

addElem(printing);

}

else

{

Magazine\* magaz = new Magazine;

magaz->SetInfo();

addElem(magaz);

}

break;

case cmdDelete:

if (elements == nullptr)

{

std::cout << "Необходимо предварительно создать коллекцию" << std::endl;

return;

}

std::cin.ignore();

std::cout << "Введите имя работы на удаление" << std::endl;

getline(cin, Name);

deleteElem(Name);

case cmdPrint:

print(false);

std::cout << std::endl;

break;

case cmdPrintNames:

print(true);

std::cout << std::endl;

break;

case cmdQuit:

executeEnd = true;

break;

default:

break;

}

custEvent.What = eventNothing;

}

Main.cpp

#include "EventHandle.h"

int main()

{

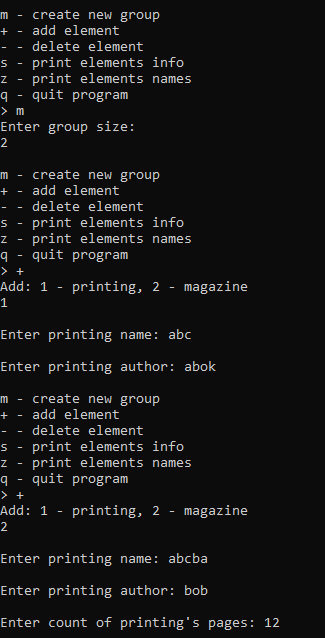
system("chcp 1251");

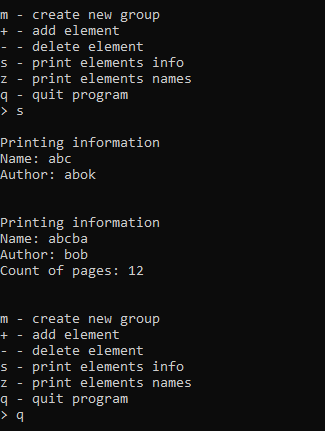
Dialog dialog;

dialog.Execute();

}

**Решение**





**Выводы**

В ходе выполнения работы были изучены механизмы работы программы, управляемой событиями.

**Ответы на контрольные вопросы:**

1. Объект, в который включены другие объекты. Окно программы, содержащее экземпляры отдельных элементов.
2. Пример класса-группы

class Elem {

int data;

Elem\* next;

Elem\* prev;

}

class List {

Elem\* begin;

Elem\* end;

bool priorityFront;

public:

void pushElem (Elem\* elem, bool priorityFront = false);

void popElem(bool priorityFront = false);

void setPriority(bool flag);

}

1. Пример конструкторов

List(){

begin = nullptr;

end = nullptr;

priorityFront = false;

}

List(Elem\* elem, bool priorityFrontCust = false){

begin = elem;

end = elem;

priorityFront = priorityFrontCust;

}

List(List& list){

begin = list.begin;

end = list.end;

priorityFront = list.priorityFront;

}

1. Пример деструктора

~List() {

Elem\* curr = begin;

Elem\* del;

while(curr != nullptr){

del = curr;

curr = curr->next;

delete curr;

}

}

1. Пример метода просмотра элементов

void Show() {

Elem\* curr = begin;

while(curr != nullptr){

cout << curr->data << endl;

curr = curr->next;

}

}

1. Иерархия объектов.
2. Для возможности доступа к объектам с использованием одних и тех же методов.
3. Информация об изменении чего-либо в программе. Используется в программах, которые реагируют на определенные действия со стороны пользователя.
4. Что это за событие (чем вызвано), информация о том, какие действия предпринять.
5. Пример структуры

Class TEvent{

int what;

int command;

}

1. Целочисленные. При событии вызванных мышью, нажатием клавиши, получением сообщения.
2. Целочисленный тип события, и или код команды, или сообщение.
3. О
4. Формирование события, поддержка главного цикла обработки событий, обработчик событий, очистка событий.
5. Получение события -> обработка события -> проверка необходимости продолжения события.
6. Сброс данных о событии. Переводит значения полей данных отвечающих за события в состояние по умолчанию.
7. Обработка событий. В соответствии с данными о событии выполняются определенные действия.
8. Получение данных о событии.
9. Используется для определения окончания выполнения программы. Содержится в классе, поддерживающем работу главного цикла.
10. Для определения корректности данных о событии.

**Github**

<https://github.com/Hitikov/Lab_OOP_8>