Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №8

Дисциплина: Основы алгоритмизации и программирования

Тема: Программа управляемая событиями

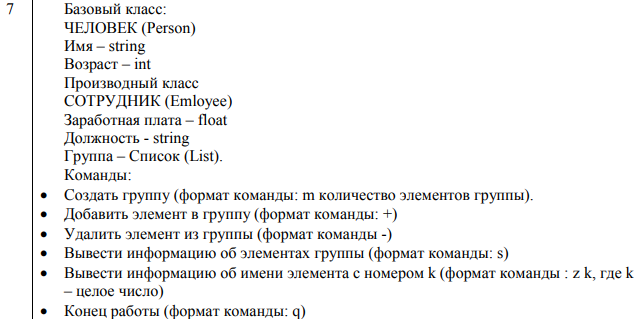
|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Бреднев М.П. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

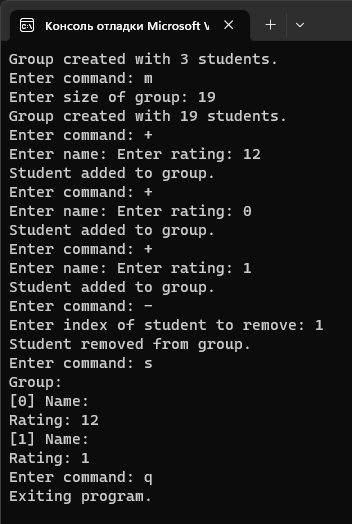
**Анализ предметной области**

**Постановка задачи**

1. Определить иерархию пользовательских классов. Во главе иерархии должен стоять абстрактный класс с чисто виртуальными функциями.
2. Реализовать конструкторы, деструкторы, 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 группа 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.



**Тестирование программы**

****

*Рис. 1 – Создание группы, добавление элементов, вывод группы в консоль, удаление элемента и вывод имен группы в консоль.*

# Контрольные вопросы

1. Что такое класс-группа? Привести примеры таких классов.

Группа это объект, в который включены другие объекты. Объекты, входящие в группу, называются элементами группы. Элементы группы, в свою очередь, могут быть группой.

В отличие от контейнера мы понимаем группу как класс, который не только хранит объекты других классов, но и обладает собственными свойствами, не вытекающими из свойств его элементов.

Группа дает второй вид иерархии - иерархию объектов (иерархию типа целое/часть), построенную на основе агрегации, первый вид иерархия классов, построенная на основе наследования.

Пример:

Класс-группа Vector. Группа содержит указатель на динамический массив указателей типа Object (Object \*\*beg), количество элементов, под которые выделена память (int size), номер последнего элемента, добавленного в группу (int cur).

class Vector {

public:

Vector(int);//конструктор с параметрами

public:

~Vector(void);//деструктор

void Add();//добавление элемента в вектор

void Del();

void Show();

int operator()();//размер вектора

protected:

Object\*\*beg;//указатель на первый элемент вектора

int size;//размер

int cur;//текущая позиция };

1. Привести пример описания класса-группы Список (List).

struct Node {

Node\* prev = nullptr, \* next = nullptr;

Object\* data;

};

class List {

public:

List();

List(int);

List(List&);

~List(void);

void puchback();

void popback();

void Show();

protected:

Node\* head = nullptr, \* tail = nullptr;

int size;

};

1. Привести пример конструктора (с параметром, без параметров, копирования) для класса-группы Список.

Без параметров:

List::List() {

size = 0;

head = nullptr;

tail = nullptr;

}

С параметрами:

List::List(int s) {

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

pushback();

}

Копирования:

List::List(const List& a) {

Node\* nodeA = a.head;

List newList(a.size);

Node\* newNode = newList.head;

while (nodeA != nullptr) {

newNode->data = nodeA->data;

nodeA = nodeA->next;

newNode = newNode->next;

}

}

1. Привести пример деструктора для класса-группы Список.

List::~List() {

Node\* curr = head;

while (curr != nullptr) {

head = curr->next;

delete curr->data;

delete curr;

curr = head;

}

}

1. Привести пример метода для просмотра элементов для класса-группы Список.

void List::Show() {

if (size == 0) cout << "Empty" << endl;

Node\* p = head;

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

p->data->Show(); //вызов метода Show() (позднее связывание)

p = p->next; //передвигаем указатель на следующий объект

}

}

1. Какой вид иерархии дает группа?

Группа дает второй вид иерархии - иерархию объектов(иерархию типа целое/часть), построенную на основе агрегации, первый вид иерархия классов, построенная на основе наследования.

1. Почему во главе иерархии классов, содержащихся в группе объектов должен находиться абстрактный класс?

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах.

Таким образом, можно создать функцию, параметром которой является указатель на абстрактный класс. На место этого параметра при выполнении программы может передаваться указатель на объект любого производного класса. Это позволяет создавать полиморфные функции, работающие с объектом любого типа в пределах одной иерархии.

Именно поэтому во главе иерархии классов должен находится абстрактный класс.

1. Что такое событие? Для чего используются события?

События лучше всего представить себе как пакеты информации, которыми обмениваются объекты и которые создаются объектно-ориентированной средой в ответ на те или иные действия пользователя. Нажатие на клавишу или манипуляция мышью порождают событие, которое передается по цепочке объектов, пока не найдется объект, знающий, как обрабатывать это событие. События сами по себе не производят никаких действий в программе, но в ответ на событие могут создаваться новые объекты, модифицироваться или уничтожаться существующие, что и приводит к изменению состояния программы. Иными словами все действия по обработке данных реализуются объектами, а события лишь управляют их работой.

События предназначены для того, чтобы иметь возможность предусмотреть реакцию программы на то или иное действие.

1. Какие характеристики должно иметь событие-сообщение?

Сообщение передаваемое от одних объектов другим имеет, как правило, следующие характеристики:

- код класса сообщения, отличающий сообщения объектов одного класса от объектов другого класса;

- адрес объекта, которому предназначено сообщение (м.б. не задан, тогда сообщение могут прочитать все объекты);

- информационное поле.

1. Привести пример структуры, описывающей событие.

enum Event

{

evNothing = 0, //пустое событие

evMessage = 100,//непустое событие

cmAdd = 1, // +

cmDel = 2, // -

cmGet = 3, // z

cmShow = 4, // s

cmMake = 5, // m

cmQuit = 101 // q

};

struct TEvent {

int what; //наличие события

int command; //тип комманды

};

1. Задана структура события

struct TEvent

{

int what;

union

{

MouseEventType mouse;

KeyDownEvent keyDown;

MessageEvent message;

} };

Какие значения, и в каких случаях присваиваются полю what?

Поле what задает тип события, определяющий источник данного события. Оно может принимать следующие значения:

* evNothing - это пустое событие, которое означает, что ничего делать не надо. Полю what присваивается значение evNothing, когда событие обработано каким-либо объектом.
* evMessage - событие-сообщение от объекта (непустое событие).

1. Задана структура события

struct TEvent {

int what;//тип события

union

{

int command;//код команды

struct//параметры команды {

int message;

int a; }; }; };

Какие значения, и в каких случаях присваиваются полю command?

Полю command присваиваются коды различных определённых команд. При получении того или иного сообщения, поле command принимает одно из кодов команд (см. пример из вопроса 10).

1. Задана структура события

struct TEvent {

int what;//тип события

union {

int command;//код команды

struct { //параметры команды

int message;

int a; }; };};

Для чего используются поля *a* и *message*?

Поля *a* и *message* необходимы в случае, если программа получает сообщение с параметром. Тогда поле *message* будет отвечать за то, какая команда поступила, а поле *a* будет являться параметром команды.

Пример:

Пользователь вводит следующее сообщение:

*+10*

В этом случае, программа сначала посмотрит, какую команду мы получили (+ - добавить в группу) и на основе этого запишет в поле *message* заданную команду, а затем посмотрит, есть ли после символа + целочисленное значение. Далее, программа запишет введенное значение в поле *a*, и после этого обработчик события будет знать, что необходимо добавить 5 элементов в группу.

1. Какие методы необходимы для организации обработки сообщений?

Следующие методы необходимы для организации обработки событий (названия произвольны).

* GetEvent – формирование события;
* Execute реализует главный цикл обработки событий. Он постоянно получает событие путем вызова GetEvent и обрабатывает их с помощью HandleEvent. Этот цикл завершается, когда поступит событие «конец».
* HandleEvent – обработчик событий. Обрабатывает каждое событие нужным для него образом. Если объект должен обрабатывать определенное событие (сообщение), то его метод HandleEvent должен распознавать это событие и реагировать на него должным образом. Событие может распознаваться, например, по коду команды (поле command).
* ClearEvent очищает событие, когда оно обработано, чтобы оно не обрабатывалось далее.
* Valid - проверяет, завершена ли работа.
* EndExec - завершение обработки событий (после вызова этого метода цикл обработки событий заканчивается).

1. Какой вид имеет главный цикл обработки событий-сообщений?

Главный цикл обработки событий реализуется в методе Execute главной группы-объекта “прикладная программа” по следующей схеме:

int TMyApp::Execute() {

do {

endState=0;

GetEvent(event); //получить событие

HandleEvent(event); //обработать событие

//событие осталось не обработано

if(event.what!=evNothing)

EventError(event);

} while(!Valid());

return endState;

}

1. Какую функцию выполняет метод ClearEvent()? Каким образом?

ClearEvent очищает событие, присваивая полю event.what значение evNothing.

1. Какую функцию выполняет метод HandleEvent()? Каким образом?

Получив событие (структуру типа TEvent), обработчик событий для класса TDerivedClass обрабатывает его по следующей схеме:

void TDerivedClass::HandleEvent(TEvent& event) {

//Вызов обработчика событий базового класса

TBaseClass::HandleEvent( event );

if (event.what == evCommand) { // Если обработчик базового

// событие не обработал

switch( event.message.command ) {

case cmCommand1:

// Обработка команды cmCommand1;

СlearEvent( event );

break;

case cmCommand2:

// Обработка команды cmCommand2;

СlearEvent( event );

break; …

case cmCommandN:

// Обработка команды cmCommandN

СlearEvent( event );

break;

default: // событие не обработано

break;

} };

}

1. Какую функцию выполняет метод GetEvent()?

Метод GetEvent() выполняет функцию формирования события на основе полученного сообщения.

1. Для чего используется поле EndState? Какой класс (объект) содержит это поле?

Поле EndState используется в классе Dialog и отвечает за состояние работы диалога (идёт/завершён).

class Dialog :

public Vector

{

public:

Dialog(void);

~Dialog(void);//деструктор

void GetEvent(TEvent& event);//получить событие

void Execute();//главный цикл обработки событий

void HandleEvent (TEvent& event); //обработчик

void ClearEvent (TEvent & event);//очистить событие

bool Valid();//проверка атрибута EndState

void EndExec();//обработка события «конец работы»

protected:

int EndState;

};

1. Для чего используется функция Valid()?

Функция Valid проверяет состояние поля EndState. Она возвращает значение True, если программа не завершена, False - если завершена. Используется в главном обработчике событий Execute.

bool Dialog::Valid() {

return EndState == 0;

}

void Dialog::Execute() {

TEvent event;

do {

EndState = 0;

GetEvent(event); //получить событие

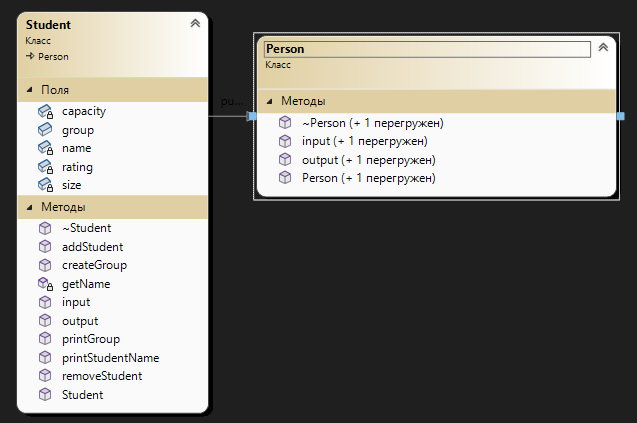
HandleEvent(event); //обработать событие

} while (Valid());

}

**Приложения**

UML-диаграмма:



Приложение Б – код программы

Main.cpp:

int main() {

Student student;

student.createGroup(3);

char command;

do {

cout << "Enter command: ";

cin >> command;

switch (command) {

case 'm': {

int size;

cout << "Enter size of group: ";

cin >> size;

student.createGroup(size);

break;

}

case '+':

student.addStudent();

break;

case '-':

student.removeStudent();

break;

case 's':

student.printGroup();

break;

case 'z': {

int index;

cout << "Enter index: ";

cin >> index;

student.printStudentName(index);

break;

}

case 'q':

cout << "Exiting program." << endl;

break;

default:

cout << "Invalid command." << endl;

break;

}

} while (command != 'q');

delete[] student.group;

return 0;

}

Person.h:

class Person {

public:

Person() {}

virtual ~Person() {}

virtual void input() = 0;

virtual void output() const = 0;

};

Student.h:

class Student : public Person {

public:

Student() {}

~Student() {}

void input() override {

cout << "Enter name: ";

getline(cin, name);

cout << "Enter rating: ";

cin >> rating;

}

void output() const override {

cout << "Name: " << name << endl;

cout << "Rating: " << rating << endl;

}

void createGroup(int size) {

capacity = size;

group = new Student[size];

cout << "Group created with " << size << " students." << endl;

}

void addStudent() {

if (size == capacity) {

cout << "Group is full." << endl;

return;

}

Student student;

student.input();

group[size] = student;

size++;

cout << "Student added to group." << endl;

}

void removeStudent() {

int index;

cout << "Enter index of student to remove: ";

cin >> index;

if (index < 0 || index >= size) {

cout << "Invalid index." << endl;

return;

}

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

group[i] = group[i + 1];

}

size--;

cout << "Student removed from group." << endl;

}

void printGroup() const {

cout << "Group:" << endl;

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

cout << "[" << i << "] ";

group[i].output();

}

}

void printStudentName(int index) const {

if (index < 0 || index >= size) {

cout << "Invalid index." << endl;

return;

}

cout << "Name of student at index " << index << ": " << group[index].getName() << endl;

}

Student\* group = nullptr;

private:

string name;

float rating;

int size = 0;

int capacity = 0;

string getName() const { return name; }

};