Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Программа, управляемая событиями»

Выполнил работу

Студент группы РИС-23-3Б

Шуракова А.А.

Проверил

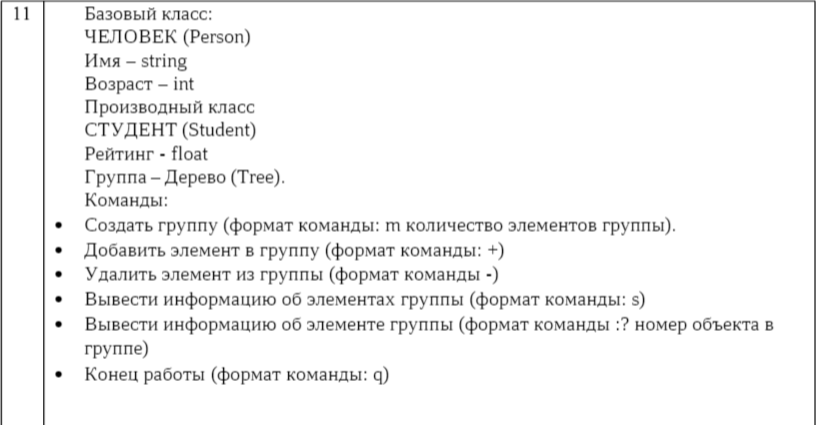
Доцент кафедры ИТАС

Полякова О.А.

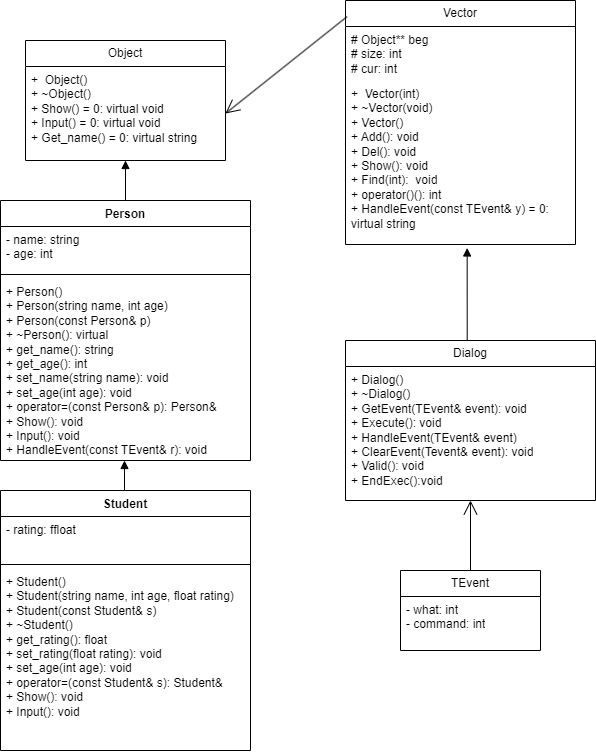
Г. Пермь-2024

**Постановка задачи:**

1. Определить иерархию пользовательских классов (см. лабораторную работу №5).
2. Во главе иерархии должен стоять абстрактный класс с чисто виртуальными методами для ввода и вывода информации об атрибутах объектов.
3. Реализовать конструкторы, деструктор, операцию присваивания, селекторы и модификаторы.
4. Определить класс-группу на основе структуры, указанной в варианте.
5. 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.
6. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.
7. Добавить методы для обработки событий группой и объектами пользовательских классов.
8. Написать тестирующую программу.
9. Нарисовать диаграмму классов и диаграмму объектов.

****

**Uml диаграмма:**

****

**Код программы:**

**event.h**

#pragma once

#include <iostream>

using namespace std;

const int evNothing = 0;

const int evMessage = 100;

const int cmAdd = 1;

const int cmDel = 2;

const int cmGet = 3;

const int cmShow = 4;

const int cmMake = 6;

const int cmFind = 7;

const int cmQuit = 101;

/\*struct TEvent {

int what;

int command;

int message;

int a;

};

\*/

struct TEvent {

int what;

union {

int command;

struct {

int message;

int a;

};

};

};

**object.h**

#pragma once

#include <iostream>

#include "event.h"

using namespace std;

class Object

{

public:

Object(void) {};

virtual void Show() = 0;

virtual void Input() = 0;

virtual ~Object(void) {};

virtual void HandleEvent(const TEvent& e) = 0;

};

**student.h**

#pragma once

#include <iostream>

#include <string>

#include "person.h"

using namespace std;

class Student : public Person {

protected:

float rating;

public:

Student();

Student(string name, int age, float rating);

Student(const Student& s);

~Student();

float get\_rating() { return rating; }

void set\_rating(float rating);

void Show();

void Input();

Student& operator=(const Student&);

};

**student.cpp**

#include <iostream>

#include "student.h"

using namespace std;

Student::Student() { rating = 0; }

Student::~Student() {}

void Student::Show() {

cout << "\nStudent name: " << name << "\n";

cout << "\nStudent age: " << age << "\n";

cout << "\nStudent rating: " << rating << "\n";

}

void Student::Input() {

cout << "\nEnter student name: "; cin >> name;

cout << "\nEnter student age: "; cin >> age;

cout << "\nEnter student rating: "; cin >> rating;

}

Student::Student(string N, int A, float R) {

name = N;

age = A;

rating = R;

}

Student::Student(const Student& student) {

name = student.name;

age = student.age;

rating = student.rating;

}

Student& Student::operator=(const Student& student) {

if (&student == this) { return \*this; }

name = student.name;

age = student.age;

rating = student.rating;

return \*this;

}

void Student::set\_rating(float R) {

rating = R;

}

**person.h**

#pragma once

#include <iostream>

#include <string>

#include "object.h"

#include "event.h"

using namespace std;

class Person : public Object {

protected:

string name;

int age;

public:

Person();

Person(string name, int age);

Person(const Person& p);

virtual ~Person();

string get\_name();

int get\_age();

void set\_name(string name);

void set\_age(int age);

Person& operator =(const Person& p);

void Show();

void Input();

void HandleEvent(const TEvent& r);

};

**person.cpp**

#include <iostream>

#include "person.h"

using namespace std;

Person::Person() {

name = " ";

age = 0;

}

Person::Person(string N, int A) {

name = N;

age = A;

}

Person::Person(const Person& person) {

name = person.name;

age = person.age;

}

Person::~Person() {}

string Person::get\_name() { return name; }

int Person::get\_age() { return age; }

void Person::set\_name(string N) { name = N; }

void Person::set\_age(int A) { age = A; }

Person& Person::operator=(const Person& person) {

if (&person == this) { return \*this; }

name = person.name;

age = person.age;

return \*this;

}

void Person::Show() {

cout << "\nPerson name: " << name << endl;

cout << "\nPerson age: " << age << endl;

}

void Person::Input() {

cout << "\nEnter name: "; cin >> name;

cout << "\nEnter age: "; cin >> age;

}

void Person::HandleEvent(const TEvent& r) {

if (r.what == evMessage) {

switch (r.command) {

case cmGet:cout << "Автор = " << this->get\_age() << endl;

cout << "Имя = " << this->get\_name() << endl;

break;

}

}

}

**vector.h**

#pragma once

#include "person.h"

#include "student.h"

#include "event.h"

#include<iostream>

class Vector

{

public:

Vector(int);//конструктор с параметрами

public:

~Vector(void);//деструктор

Vector();

void Add();//добавление элемента в вектор

void Del();

void Show();

void Find(int);

int operator()();//размер вектора

void HandleEvent(const TEvent& y) {

if (y.what == evMessage) {

Object\*\* p = this->beg;

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

(\*p)->HandleEvent(y);

p++;

}

}

};

protected:

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

int size;//размер

int cur;//текущая позиция

};

**vector.cpp**

#include<iostream>

#include"vector.h"

#include "student.h"

Vector::~Vector(void)

{

if (beg != 0)delete[] beg;

beg = 0;

}

Vector::Vector() {

beg = nullptr;

cur = 0;

size = 0;

}

//конструктор с параметрами

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

//добавление объекта, на который указывает указатель p в вектор

void Vector::Add()

{

Object\* p;

//выбор из объектов двух возможных классов

cout << "\n1.Person\n2.Student\nВведите номер выбранного варианта >> ";

int y;

cin >> y; cout << "\n";

if (y == 1)//добавление объекта класса Car

{

Person\* a = new Person();

a->Input();//ввод значений атрибутов

p = a;

if (cur < size)

{

beg[cur] = p;//добавление в вектор cur++;

cur++;

}

}

else if (y == 2) //добавление объекта класса Lorry

{

Student\* b = new Student();

b->Input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else return;

}

//просмотр вектора

void Vector::Show()

{

if (cur == 0) cout << "Пустой" << endl;

Object\*\* p = beg;//указатель на указатель типа Object

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

{

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

p++;//передвигаем указатель на следующий объект

}

}

void Vector::Find(int tmp) {

Object\*\* p = beg;

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

if (i == tmp - 1) { (\*p)->Show(); }

p++;

}

}

//операция, которая возвращает размер вектора

int Vector::operator ()()

{

return cur;

}

//удаление элемента из вектора, память не освобождается!

void Vector::Del()

{

if (cur == 0)return;//пустой

cur--;

}

**dialog.h**

#pragma once

#include "vector.h"

#include "event.h"

using namespace std;

class Dialog :

public Vector {

protected:

int EndState;

public:

Dialog();

virtual ~Dialog();

virtual void GetEvent(TEvent& event);

virtual int Execute();

virtual void HandleEvent(TEvent& event);

virtual void ClearEvent(TEvent& event);

int Valid();

void EndExec();

};

**dialog.cpp**

#include <iostream>

#include "dialog.h"

#include "student.h"

using namespace std;

Dialog::Dialog(void) : Vector() {

EndState = 0;

}

Dialog::~Dialog(void) {}

void Dialog::GetEvent(TEvent& event) {

string OpInt = "+-s?qm";

string s;

string param;

char code;

cout << '>';

cin >> s;

code = s[0];

if (OpInt.find(code) >= 0) {

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 'q': event.command = cmQuit; break;

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

}

if (s.length() > 1) {

param = s.substr(1, s.length() - 1);

int A = atoi(param.c\_str());

event.a = A;

}

}

else { event.what = evNothing; }

}

int Dialog::Execute() {

TEvent event;

do {

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (!Valid());

return EndState;

}

int Dialog::Valid() {

if (EndState == 0) { return 0; }

else { return 1; }

}

void Dialog::ClearEvent(TEvent& event) { event.what = evNothing; }

void Dialog::EndExec() { EndState = 1; }

void Dialog::HandleEvent(TEvent& event) {

if (event.what == evMessage) {

switch (event.command) {

case cmMake:

cout << "Enter size: ";

cin >> size;

beg = new Object \* [size];

cur = 0;

ClearEvent(event);

break;

case cmAdd:

Add();

ClearEvent(event);

break;

case cmDel:

Del();

ClearEvent(event);

break;

case cmShow:

Show();

ClearEvent(event);

break;

case cmQuit:

EndExec();

ClearEvent(event);

break;

case cmFind:

int tmp = event.a;

Find(tmp);

ClearEvent(event);

break;

//default:Vector::HandleEvent(event);

}

}

}

**main.cpp**

#include <iostream>

#include "dialog.h"

#include "vector.h"

#include "student.h"

using namespace std;

int main()

{

system("chcp 1251");

cout << "m: создать группу" << endl;

cout << "+: добавить элемент в группу" << endl;

cout << "-: удалить элемент из группы" << endl;

cout << "s: вывести информацию об элементах группы" << endl;

cout << "?: вывести информацию об элементе группы" << endl;

cout << "q: конец работы" << endl;

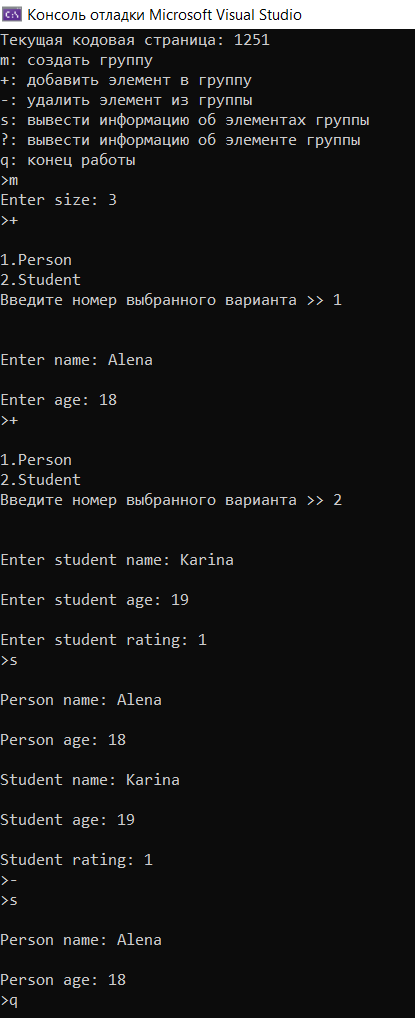
Dialog D;

D.Execute();

return 0;

}

**Результат работы программы:**



**Ответы на контрольные вопросы:**

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

Класс-группа - это класс, который содержит в себе коллекцию объектов других классов и предоставляет методы для работы с этой коллекцией. Например, можно создать класс-группу "Список студентов", который будет содержать объекты класса "Студент". Класс-группа будет предоставлять методы для добавления/удаления студентов, получения списка студентов и т.д.

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

template <typename T>

class List {

public:

// Конструкторы и деструктор

List();

List(const List<T>& other);

~List();

// Операторы

List<T>& operator=(const List<T>& other);

T& operator[](int index);

const T& operator[](int index) const;

// Методы

void insert(int index, const T& value);

void remove(int index);

int size() const;

bool isEmpty() const;

private:

// Внутренний класс узла списка

struct Node {

T data;

Node\* next;

Node(const T& d) : data(d), next(nullptr) {}

};

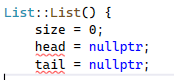
Node\* head;

int length;

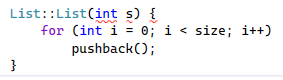
};

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

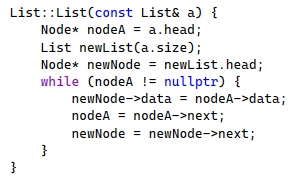
без параметров:



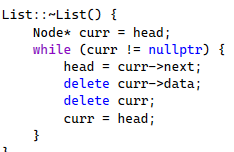
с параметром:



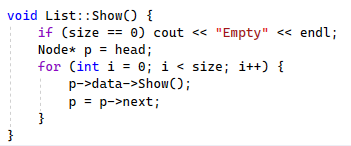
копирования:



1. Привести пример деструктора для класса-группы Список.



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



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?

В иерархии классов, содержащихся в группе объектов, на вершине должен находиться абстрактный класс, так как он определяет общий интерфейс, который должен быть реализован всеми классами-наследниками. Это позволяет упростить дальнейшую работу с объектами классов-наследников и обеспечить единообразие в их использовании. Кроме того, такая организация позволяет обеспечить полиморфизм и использовать указатели на базовый абстрактный класс для работы с объектами различных классов-наследников через общий интерфейс.

1. Что такое событие? Для чего используются события?

Событие (event) представляет собой сигнал, сообщающий программе о каком-то действии, произошедшем в системе или приложении. События используются для организации обработки пользовательских действий, взаимодействия компонентов приложения, реализации асинхронной обработки и многих других задач.

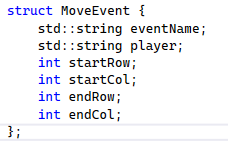
1. Какие характеристики должно иметь событие-сообщение?

Объект TEvent состоит из двух частей. Первая (what) задает тип события, определяющий источник данного события. Вторая задает информацию, передаваемую с событием. Для разных типов событий содержание информации различно. Поле what 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.

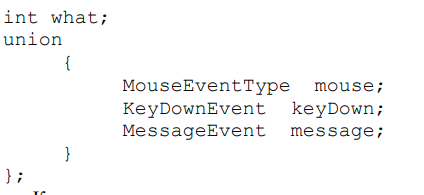
• evMessage - событие-сообщение от объекта.

1. Привести пример структуры, описывающей событие.



1. Задана структура события





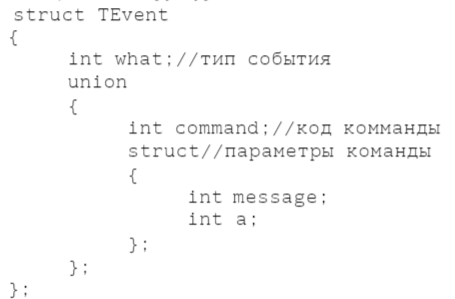
Какие значения, и в каких случаях присваиваются полю what?

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.

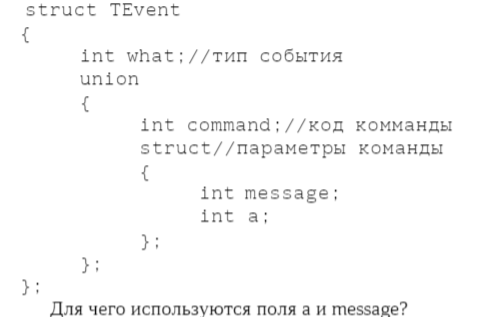
evMessage - событие-сообщение от объекта (непустое событие).

1. Задана структура события



Полю command присваиваются коды различных определённых команд. При получении того или иного сообщения, поле command принимает одно из кодов команд.

1. Задана структура события

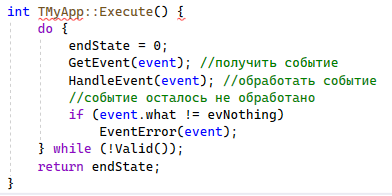


Полю command присваиваются коды различных определённых команд. При получении того или иного сообщения, поле command принимает одно из кодов команд

14. Какие методы необходимы для организации обработки сообщений?

* 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).

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



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



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

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

Метод HandleEvent() выполняет обработку события-сообщения, которое было передано в качестве аргумента метода. Обычно он содержит логику для обработки конкретного типа событий.

void Dialog::HandleEvent(TEvent& event) {

if (event.what == evMessage) {

switch (event.command) {

case cmMake:

cout << "Enter size: ";

cin >> size;

beg = new Object \* [size];

cur = 0;

ClearEvent(event);

break;

case cmAdd:

Add();

ClearEvent(event);

break;

case cmDel:

Del();

ClearEvent(event);

break;

case cmShow:

Show();

ClearEvent(event);

break;

case cmQuit:

EndExec();

ClearEvent(event);

break;

case cmFind:

int tmp = event.a;

Find(tmp);

ClearEvent(event);

break;

}

}

}

18. Какую функцию выполняет метод GetEvent ()?

Метод GetEvent() используется для получения информации о следующем событии в очереди событий. Если в очереди нет ни одного события, метод блокируется до появления нового события. Если в очереди есть события, метод возвращает информацию о первом событии в очереди и удаляет его из очереди. Обычно этот метод вызывается в главном цикле обработки событий.

void Dialog::GetEvent(TEvent& event) {

string OpInt = "+-s?qm";

string s;

string param;

char code;

cout << '>';

cin >> s;

code = s[0];

if (OpInt.find(code) >= 0) {

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 'q': event.command = cmQuit; break;

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

}

if (s.length() > 1) {

param = s.substr(1, s.length() - 1);

int A = atoi(param.c\_str());

event.a = A;

}

}

else { event.what = evNothing; }

}

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

Поле EndState используется для указания конечного состояния системы после выполнения события. Обычно это поле заполняется в обработчике события, который анализирует информацию, полученную из события, и принимает решение о том, как изменить состояние системы.

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

20. Для чего используется функция Valid()?

Функция Valid() используется для проверки, является ли текущее состояние объекта допустимым. Она возвращает булево значение true, если текущее состояние объекта допустимо, и false в противном случае.

**Ссылка на Git:**

https://github.com/Ananasic07/labs\_2sem/tree/main/Класс%208