Министерство науки и высшего образования РФ

Федеральное бюджетное государственное образовательное учреждение высшего профессионального образования

Пермский национальный исследовательский политехнический университет (ПНИПУ)

Факультет: Электротехнический (ЭТФ)

Направление: 09.03.01 «Информатика и вычислительная техника (Автоматизированные системы обработки информации и управления)»

Кафедра: «Информационные технологии и автоматизированных систем» (ИТАС)

Основы алгоритмизации

Лабораторная работа №8

Программа, управляемая событиями

Студент: Балтаев Э. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Группа: АСУ-20-1бзу

Работу проверил:

доцент кафедры ИТАС

Полякова О. А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_

Пермь 2021г.

Постановка задачи

1. Определить иерархию пользовательских классов согласно варианту. Во главе абстрактный класс с чисто виртуальными методами.

2. Реализовать конструкторы, методы и перегрузить операторы

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

4. Для группы реализовать конструктора, методы.

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

Задание варианта:

Базовый класс:

ЧЕЛОВЕК (Person)

Имя – string

Возраст – int

Производный класс

СОТРУДНИК (Employee)

Заработная плата – string

Должность – int

Группа – Список (List)

Команды

* Создать группу (формат команды: m количество элементов группы)
* Добавить элемент в группу (формат команды: +)
* Удалить элемент из группы (формат команды: -)
* Вывести информацию об элементах группы (формат команды: s)
* Вывести информацию об имени элемента с номером k (формат команды: z k, где k – целое число)
* Конец работы (формат команды: q)

Описание класса

1) Описание событий

#pragma once

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 cmQuit = 10;

const int cmClear = -1;

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

2) Абстрактный класс

#pragma once

#include <iostream>

#include "TEvent.h"

using namespace std;

class Object

{

public:

Object(void) { }

virtual void show() = 0;

virtual void input() = 0;

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

};

3) Базовый класс

#pragma once

#include "Object.h"

class Person :

public Object

{

protected:

string name;

int age;

public:

Person() { name = "NoName"; age = 0; }

Person(string n, int a) { name = n; age = a; }

Person(const Person& p) { name = p.name; age = p.age; }

~Person() {}

string get\_name() { return name; }

int get\_age() { return age; }

void set\_name(string n) { name = n; }

void set\_age(int a) { age = a; }

Person& operator= (const Person& p);

void show();

void input();

void HandleEvent(const TEvent& e);

};

4) Дочерний класс

#pragma once

#include "Person.h"

class Employee :

public Person

{

protected:

int salary;

string post;

public:

Employee() : Person() {}

Employee(string n, int a, int s, string p) : Person(n, a) { salary = s; post = p; }

Employee(const Employee& em) : Person(em) { salary = em.salary; post = em.post; }

~Employee() {}

Employee& operator= (const Employee& em);

void show();

void input();

};

5) Класс группа

#pragma once

#include "Employee.h"

class List

{

protected:

Object\*\* beg;

int size;

public:

List();

List(int n);

~List();

void add();

void del();

void show();

int operator() ();

virtual void HandleEvent(const TEvent& e);

};

6) Наследник группы

#pragma once

#include "List.h"

class Dialog :

public List

{

protected:

int EndState;

public:

Dialog(void);

Dialog(int n);

~Dialog(void);

Dialog& operator= (const Dialog& d);

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

virtual int Execute(); // бесконечное выполнение

virtual void HandleEvent(TEvent& event); // обработка события

virtual void ClearEvent(TEvent& event); // очистка события после обработки

int Valid(); // проверка EndState

void EndExec(); // обарботка события "конец работы"

};

Определение компонентных функций

1) Базовый класс

#include "Person.h"

Person& Person::operator=(const Person& p)

{

if (this == &p)

return \*this;

name = p.name;

age = p.age;

return \*this;

}

void Person::show()

{

cout << "Человек: " << name << " " << age << " лет\n";

}

void Person::input()

{

cout << "Name? "; cin >> name;

cout << "Age? "; cin >> age;

}

void Person::HandleEvent(const TEvent& e)

{

if (e.what == evMessage) {

switch (e.command)

{

case cmGet:

cout << "Name - " << name << endl;

break;

}

}

}

2) Дочерний класс

#include "Employee.h"

Employee& Employee::operator=(const Employee& em)

{

if (this == &em)

return \*this;

name = em.name;

age = em.age;

salary = em.salary;

post = em.post;

return \*this;

}

void Employee::show()

{

cout << "Работник: " << name << " " << age << " лет с зарплатой = " << salary << " на должности " << post << endl;

}

void Employee::input()

{

cout << "Name? "; cin >> name;

cout << "Age? "; cin >> age;

cout << "Salary? "; cin >> salary;

cout << "Post? "; cin >> post;

}

3) Класс группа

#include "List.h"

List::List()

{

size = 10;

beg = new Object \* [size];

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

beg[i] = 0;

}

List::List(int n)

{

size = n;

beg = new Object \* [size];

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

beg[i] = 0;

}

List::~List()

{

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

if (beg[i] == 0)

break;

delete beg[i];

}

delete[] beg;

beg = 0;

}

void List::add()

{

// ищем пустое место

int ind = -1;

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

if (beg[i] == 0) {

ind = i;

break;

}

if (ind == -1) {

cout << "Ошибка: заполнено" << endl;

return;

}

// выбираем тип

cout << "Добавить человека (1) или работника (остальные числа)? ";

int x; cin >> x;

Object\* obj;

if (x == 1)

obj = new Person();

else

obj = new Employee();

obj->input();

// добавляем

beg[ind] = obj;

cout << "Успешно\n";

}

void List::del()

{

// ищем конец

int ind;

for (ind = 0; ind < size; ind++)

if (beg[ind] == 0)

break;

if (ind == 0) {

cout << "Ошибка: пусто" << endl;

return;

}

delete beg[ind - 1];

beg[ind - 1] = 0;

cout << "Успешно\n";

}

void List::show()

{

if ((\*this)() == 0)

cout << "Пусто";

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

if (beg[i] == 0)

break;

beg[i]->show();

}

cout << endl;

}

int List::operator()()

{

int count = 0;

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

if (beg[i] == 0)

break;

count++;

}

return count;

}

void List::HandleEvent(const TEvent& e)

{

if (e.what == evMessage)

{

bool ok = false;

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

// если конец

if (beg[i] == 0)

break;

// если номер совпадает

if (e.a == (i + 1)) {

beg[i]->HandleEvent(e);

ok = true;

}

}

if (ok == false)

cout << "Не верный номер\n";

}

}

4) Наследник группы

#include "Dialog.h"

Dialog::Dialog(void)

{

EndState = 0;

}

Dialog::Dialog(int n) : List(n)

{

}

Dialog::~Dialog(void)

{

}

Dialog& Dialog::operator=(const Dialog& d)

{

if (this == &d)

return \*this;

if ((\*this)() != 0)

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

if (beg[i] == 0)

break;

delete beg[i];

}

delete[] beg;

size = d.size;

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

beg[i] = d.beg[i];

return \*this;

}

void Dialog::GetEvent(TEvent& event)

{

string s, param, OpInt = "m+-szqc"; // строка с символами операций

char code;

cout << '>'; cin >> s; code = s[0];

if (OpInt.find(code) >= 0) // является ли символ кодом операции

{

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 'z': event.command = cmGet; break;

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

case 'c': event.command = cmClear; break;

}

if (s.length() > 1) // выделяем параметры команды, если они есть

{

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

int A = atoi(param.c\_str()); // преобразуем параметр в число

event.a = A; // записываем в сообщение

}

}

else

event.what = evNothing; // пустое событие

}

int Dialog::Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (!Valid());

return EndState;

}

void Dialog::HandleEvent(TEvent& event)

{

if (event.what == evMessage)

{

switch (event.command)

{

case cmMake:

\*this = Dialog(event.a);

cout << "Успешно\n";

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 cmClear:

system("cls");

ClearEvent(event);

break;

default:

List::HandleEvent(event);

}

}

}

void Dialog::ClearEvent(TEvent& event)

{

event.what = evNothing;

}

int Dialog::Valid()

{

if (EndState == 0)

return 0;

else

return 1;

}

void Dialog::EndExec()

{

EndState = 1;

}

Определение главного файла

#include <iostream>

#include "Dialog.h"

using namespace std;

int main()

{

system("chcp 1251");

Dialog d;

d.Execute();

return 0;

}

Диаграмма классов

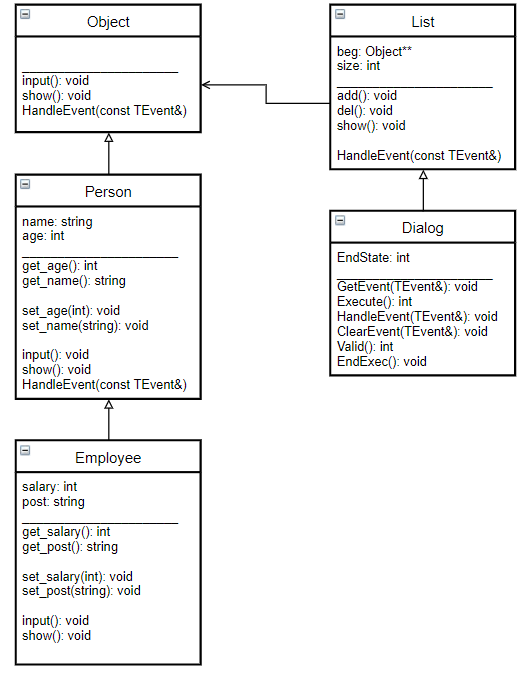
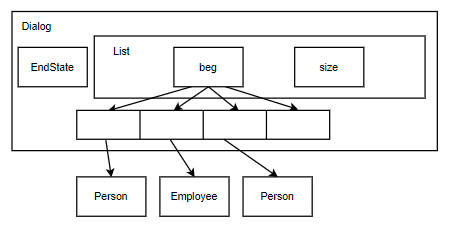
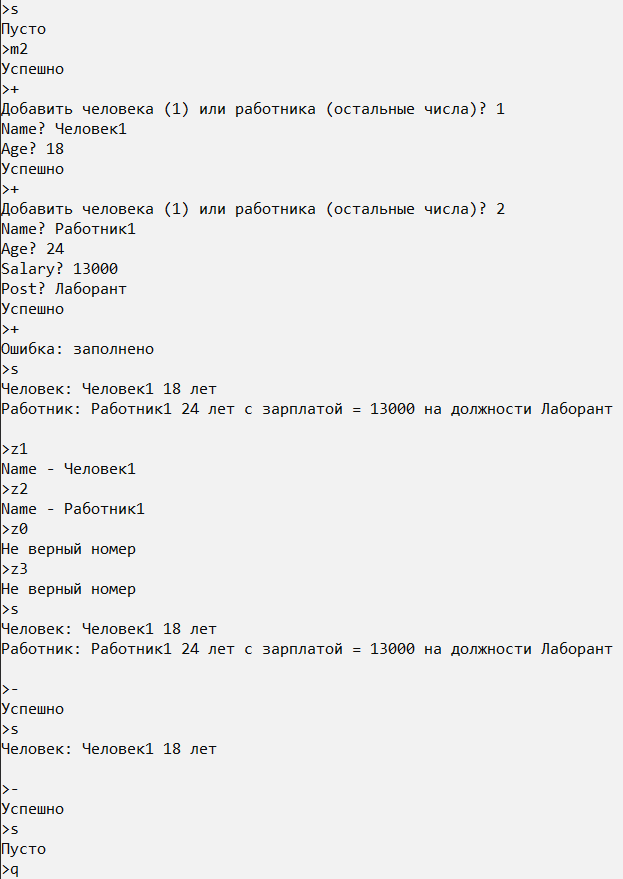


Диаграмма объектов



Результаты работы программы:



Ответы на контрольные вопросы

1) Группа – объект, в который включены другие объекты, чаще всего с похожими свойствами. Примеры: факультет-кафедра-студ.группа; огород

2)

class List

{

private:

Object\*\* beg;

int size;

int curr;

…

}

3)

List()

{

size=10;

beg=new Object\*[size];

curr=0;

}

List(int n)

{

size=n;

beg=new Object\*[size];

curr=0;

}

List(const List& l)

{

size=l.size;

beg=new Object\*[size];

for(curr=0; curr<size; curr++)

beg[curr] = l.beg[curr];

}

4)

~List()

{

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

delete beg[i];

delete[] beg;

}

5)

void show()

{

if(curr==0)

cout<<”Empty\n”;

Object\*\* p = beg;

for(int i=0; i<curr; i++, p++)

cout<<(\*p)<<endl;

}

6) Иерархия объектов (типа целое/часть), построенную на основе агрегации и Иерархию классов, построенную на основе наследования

7) Чтобы можно было определить контейнер объектов абстрактного типа в группе и использовать общие методы для всей иерархии

8) Событие – пакет информации, которыми обмениваются объекты и которые позволяют управлять программой будто «автоматизированной»

9) Код класса; адрес объекта кому предназначено; информационное поле

10)

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

11) Значения: нажата клавиша, нажата кнопка мыши, ничего.

12) Если пришло событие – сообщение, то присваивается код команды для выполнения

13) Для получения строковой информации о выполняемой команде

14) GetEvent – формирование; Execute – цикл обработки событий; HandleEvent – обработка события; ClearEvent – очищение события;

15)

Выполнить

{

Получние;

Обработка;

}Пока(Команда != Конец);

16) Очищает события после обработки; what = evNothing

17) Обработка события; switch(код команды) { … }

18) Формирует событие; Вводится команда и событию присваиваются данные данной команды

19) Класс Dialog – обрабатывающий события. Для понимания, заканчивать цикл обработки или нет

20) Для проверки результата обработки после каждой итерации, когда нужно выйти.