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 12

Выполнил:

студент группы РИС-20-2б

Морозова Е.М.

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.

**Задание варианта:** Базовый класс - Человек (поля : имя(string) и возраст(int)). Производный класс – сотрудник (поля: заработная плата(float) , должность(string) и группа. Группа – дерево(tree). Команды: создать группу, добавить элемент в группу, удалить элемент из группы, вывести информацию об элементах группы, вывести информацию о, имени элемента с указанным номером, завершить работу.

**Анализ задачи**

1. Описание класса:

class Person :

public Object

{

public:

Person(void);

public:

virtual ~Person(void);

void Show();

void Input();

Person(string, int);

Person(const Person&);

string Get\_name() { return name; }

int Get\_age() { return age; }

void set\_name(string);

void set\_age(int);

Person& operator=(const Person&);

virtual void HandleEvent(const TEvent& e);

protected:

string name;

int age;

};

class Dialog:

public Tree

{

public:

Dialog(int);

public:

virtual ~Dialog(void);

virtual void GetEvent(TEvent& event);

virtual int Execute();

virtual void HandleEvent(TEvent& event);

virtual void ClearEvent(TEvent& event);

int Valid();

void EndExec();

protected:

int EndState;

};

class Object

{

public:

Object(void);

virtual void Show() = 0;

virtual void Input() = 0;

virtual ~Object(void);

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

};

class Employee :

public Person

{

public:

Employee(void);

public:

~Employee(void);

void Show();

void Input();

Employee(string, int, float, string);

Employee(const Employee&);

float Get\_plata() { return plata; }

string Get\_dolzh() { return dolzh; }

void Set\_plata(float);

void Set\_dolzh(string);

Employee&operator=(const Employee&);

protected:

float plata;

string dolzh;

};

const int evNothing = 10;

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 = 101;

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

2. Описание компонентных функций:

Tree::~Tree(void)

{

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

beg = 0;

}

Tree::Tree(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

void Tree::Add() {

Object\* p;

cout << "\n1.Person" << endl;

cout << "2.Employee" << endl;

int y;

cin >> y;

if (y == 1)

{

Person\* a = new Person;

a->Input();

p = a;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else

if (y == 2)

{

Employee\* b = new Employee;

b->Input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else return;

}

void Tree::Show()

{

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

Object \* \*p = beg;

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

{

(\*p)->Show();

p++;

}

}

int Tree::operator()()

{

return cur;

}

void Tree::Del()

{

if (cur == 0)

return;

cur--;

}

void Tree::HandleEvent(const TEvent& e)

{

if (e.what == evMessage)

{

Object\*\* p = beg;

if (e.a >= 0 && e.a <= cur)

{

for (int i = 0; i < e.a; i++)

p++;

}

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

}

}

Employee::Employee(void) :Person()

{

plata = 0;

dolzh = "";

}

Employee::~Employee(void)

{

}

Employee::Employee(string N, int A, float P, string D) :Person(N, A)

{

plata = P;

dolzh = D;

}

Employee::Employee(const Employee& E)

{

name = E.name;

age = E.age;

plata = E.plata;

dolzh = E.dolzh;

}

void Employee::Set\_plata(float P)

{

plata = P;

}

void Employee::Set\_dolzh(string D)

{

dolzh = D;

}

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

{

if (&e == this) return\*this;

name = e.name;

age = e.age;

}

void Employee::Show()

{

cout << "\nname=" << name;

cout << "\nage=" << age;

cout << "\nplata=" << plata;

cout << "\ndolzh=" << dolzh;

}

void Employee::Input()

{

cout << "\nname=";

cin >> name;

cout << "\nage=";

cin >> age;

cout << "\nplata=";

cin >> plata;

cout << "\ndolzh=";

cin >> dolzh;

}

Person::Person(void)

{

name = "";

age = 0;

}

Person::~Person(void)

{

}

Person::Person(string N, int A)

{

name = N;

age = A;

}

Person::Person(const Person& person)

{

name = person.name;

age = person.age;

}

void Person::set\_name(string N)

{

name = N;

}

void Person::set\_age(int A)

{

age = A;

}

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

{

if (&p==this) return \*this;

name = p.name;

age = p.age;

return \*this;

}

void Person::Show()

{

cout << "\nname=" << name;

cout << "\nage=" << age;

}

void Person::Input() {

cout << "\nname=";

cin >> name;

cout << "\nage=";

cin >> age;

}

void Person::HandleEvent(const TEvent &e)

{

if (e.what == evMessage)

{

switch (e.command)

{

case cmGet:cout << "name=" << Get\_name() << endl;

break;

}

}

}

Object::Object(void)

{

}

Object::~Object(void)

{

}

3. Функция main():

void main()

{

Person\* a = new Person;

a->Input();

a->Show();

Employee\* b = new Employee;

b->Input();

b->Show();

Tree t(10);

t.Add();

t.Add();

t.Show();

t.Del();

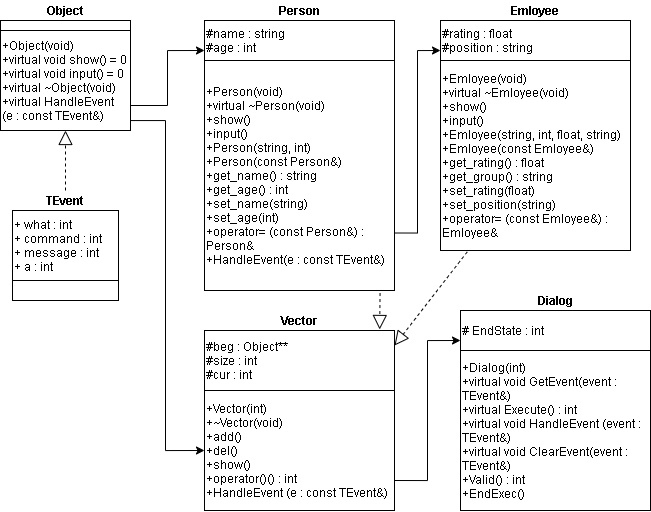
cout << "\nTree size=" << t();

Dialog D(10);

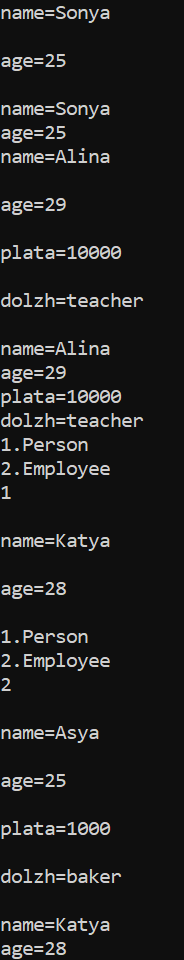
D.Execute();

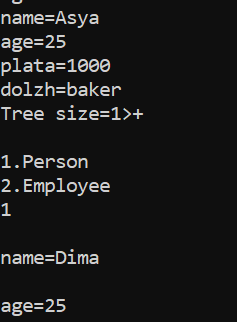
}

**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».