Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет»
Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»

ЛАБОРАТОРНАЯ РАБОТА №8
«Программа, управляемая событием»

Выполнила:
студентка группы ИВТ-23-2б
Соловьева Екатерина
Александровна
Проверила:
доцент кафедры ИТАС
О. А. Полякова

2024 г.

Постановка задачи

1. Определить иерархию пользовательских классов (см. лабораторную работу №5). 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.

11 Вариант:

Базовый класс:

ЧЕЛОВЕК (Person)

Имя – string

Возраст – int

Производный класс

СТУДЕНТ (Student)

Рейтинг - float

Группа – Дерево (Tree).

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 элементе группы (формат команды :? номер объекта в группе)
- Конец работы (формат команды: q)

Код программы на C++

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;
};
```

Odject.h

```
#pragma once
#include <iostream>
#include "Event.h"

using namespace std;

class Object {
public:
	virtual void show() = 0;
	virtual void input() = 0;
	void HandleEvent(const TEvent&);
};
```

Person.h

```
#pragma once
#include <iostream>
#include "Object.h"
#include <String>

using namespace std;

class Person : public Object {
protected:
	string name;
	int age;
public:
	Person(void);
	Person(string name, int age);
	Person(const Person& p);
	~Person(void);
	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();
};
```

Student.h

```
#pragma once
#include <iostream>
#include "Person.h"
#include <string>

using namespace std;

class Student : public Person {
protected:
        float rating;
public:
        Student(void);
        Student(string name, int age, float rating);
        Student(const Student& s);
        ~Student(void);
        float get_rating() { return rating; }
        void set_rating(float rating);
        void show();
        void input();
        Student& operator=(const Student&);
};
```

Tree.h

```
#pragma once
#include <iostream>
#include "Object.h"
#include "Person.h"
#include "Student.h"
using namespace std;

class Tree {
protected:
        Object** beg;
        int size;
        int cur;
public:
        Tree(void);
        Tree(int);
        ~Tree(void);
        void Add(void);
        void Del(void);
        void Show(void);
        void Find(int);
        int operator()();
        void HandleEvent(const TEvent& event);
};
```

Dialog.h

```
#pragma once
#include "Tree.h"
#include "Event.h"
using namespace std;

class Dialog :
        public Tree {
protected:
```

```
        int EndState;
public:
        Dialog(void);
        virtual ~Dialog(void);
        virtual void GetEvent(TEvent&);
        virtual int Execute();
        virtual void HandleEvent(TEvent&);
        virtual void ClearEvent(TEvent&);
        int Valid();
        void EndExec();
};
```

Person.cpp

```
#include "Person.h"
#include <iostream>

using namespace std;

Person::Person(void) {
        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(void) {}

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;
}
```

Dialog.cpp

```
#include <iostream>
```

```
#include "Dialog.h"
using namespace std;

Dialog::Dialog(void) : Tree() {
	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:
			size = event.a;
			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;
                }
        }
}
```

Tree.cpp

```
#include "Tree.h"
#include <iostream>

using namespace std;

Tree::Tree(void) {
        beg = nullptr;
        size = 0;
        cur = 0;
}
Tree::Tree(int n) {
        beg = new Object * [n];
        cur = 0;
        size = n;
}
Tree::~Tree(void) {
        if (beg != 0) { delete[] beg; }
        beg = 0;
}
void Tree::Add() {
        Object* p;

        cout << "\n1.Person\n2.Student\nEnter variant >> ";
        int y;
        cin >> y; cout << "\n";

        if (y == 1) {
                Person* a = new (Person);
                a->input();
                p = a;
                if (cur < size) {
                        beg[cur] = p;
                        cur++;
                }
        }
        else if (y == 2) {
                Student* b = new (Student);
                b->input();
                p = b;
                if (cur < size) {
                        beg[cur] = p;
                        cur++;
                }
        }
        else return;
}
void Tree::Show()
```

```
{
	if (cur == 0) { cout << "\nEmpty" << endl; }
	Object** p = beg;
	for (int i = 0; i < cur; i++) {
		(*p)->show();
		p++;
	}
}
int Tree::operator()() { return cur; }
void Tree::Del(void) {
	if (cur == 0) { return; }
	cur--;
}
void Tree::Find(int tmp) {
	Object** p = beg;
	for (int i = 0; i < cur; i++) {
		if (i == tmp - 1) { (*p)->show(); }
		p++;
	}
}
void Tree::HandleEvent(const TEvent& event) {
	if (event.what == evMessage) {
		Object** p = beg;
		for (int i = 0; i < cur; i++) {
			(*p)->show();
			++p;
		}
	}
}
```

Student.cpp

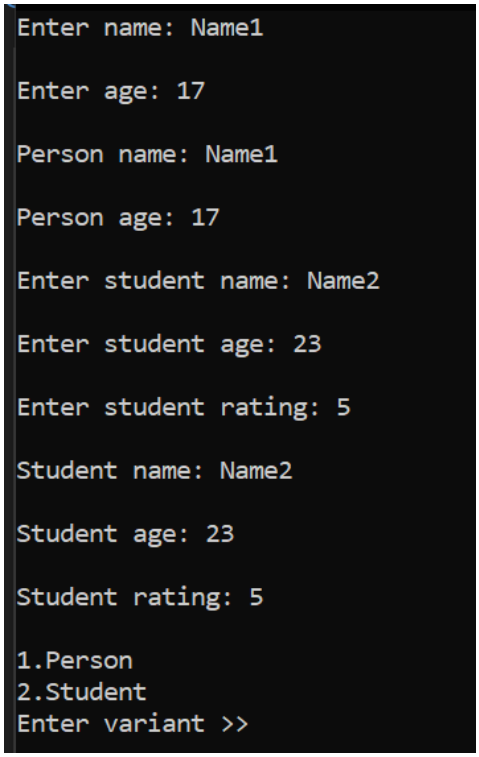
```
#include "Student.h"
#include <iostream>

using namespace std;

Student::Student(void) : Person() { 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) : Person(N, A) {
	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;
}
```

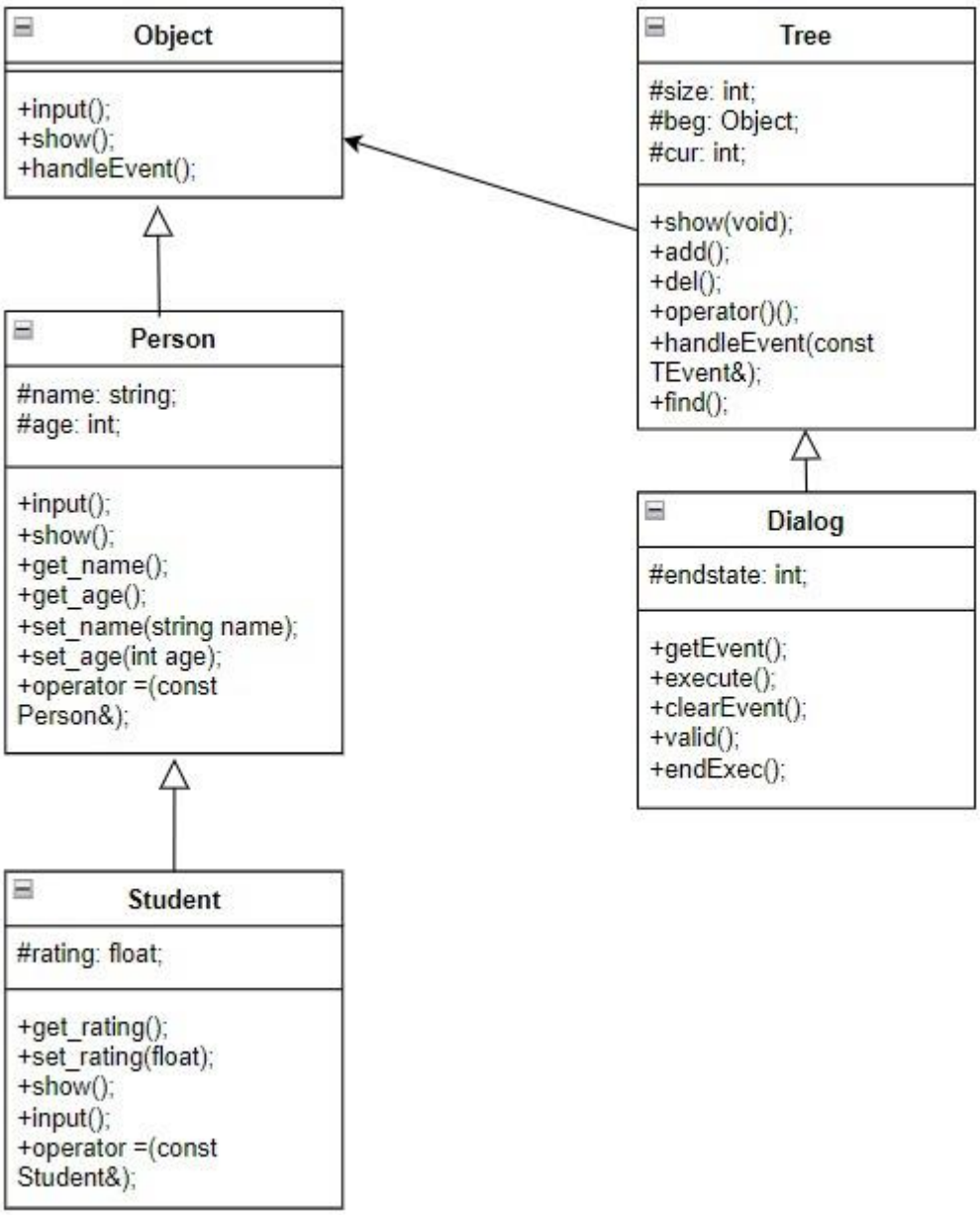
Main.cpp

```
#include <iostream>
#include "Dialog.h"
using namespace std;

int main() {
        Person* p = new Person;
        p->input();
        p->show();

        Student* s = new Student;
        s->input();
        s->show();

        Tree t(10);
        Object* o = p;
        t.Add();
        o = s;
        t.Show();
        t.Del();
        cout << "\nTree: " << t() << endl;

        Dialog D;
        D.Execute();
        return 0;
}
```

Вывод:



UML-диаграмма



Анализ результатов

Программа сработала корректно и вывела необходимые результаты.