Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе № 15.8**

Дисциплина: «Информатика»

Тема Объектно-ориентированное программирование. Программа, управляемая событиями

Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь 2023**

**Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Разработка программы, управляемой событиями.

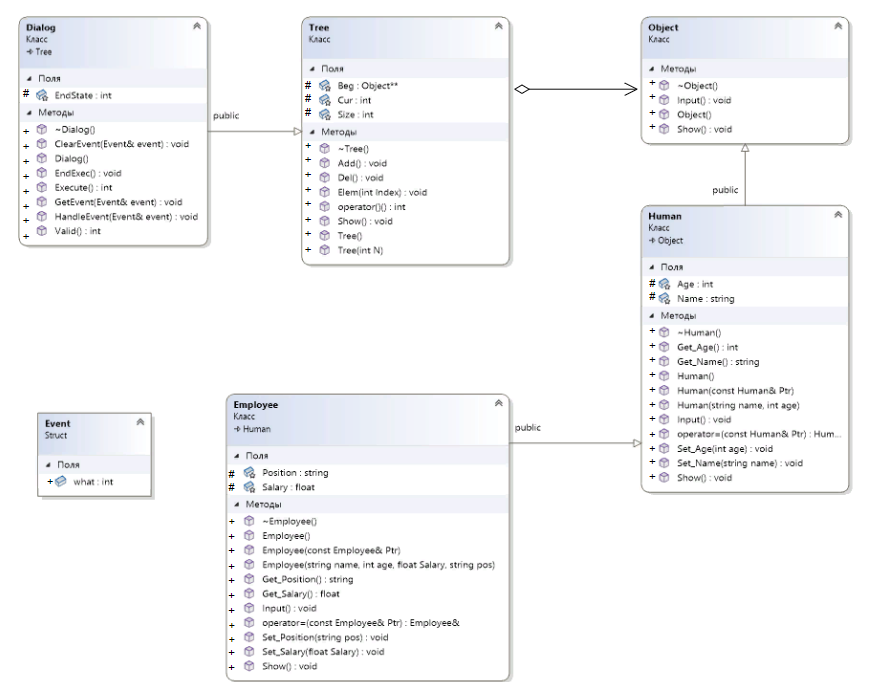
**Постановка задачи**

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.

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение абстрактного класса Object и определение необходимых методов данного класса.
   2. Реализовать определение производного от класса Object класс Human и определение необходимых методов и операторов данного класса.
   3. Реализовать определение производного от класса Human класс Employee и определение необходимых методов и операторов данного класса.
   4. Реализовать определение класса Tree и определение необходимых методов и операторов данного класса
   5. Реализовать определение производного от класса Tree класс Dialog и определение необходимых методов и операторов данного класса.
   6. Реализовать применение этих функций в главной функции.
2. Для решения задачи данные были представлены в следующем виде:
   1. Для работы с данными используются атрибуты класса.
3. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Ввод данных реализован с помощью функции getline и оператора cin, используемых при реализации в главной функции и функций перегрузки оператора.
   2. Вывод данных реализован с помощью оператора cout, используемого при реализации в главной функции и функций перегрузки операторов ввода-вывода.
4. Поставленные задачи решены следующими действиями:
   1. В классе Object находится реализация чисто виртуальных функций.
   2. В классе Human находится реализация объекта человек и методы для ввода-вывода.
   3. В классе Employee находится реализация объекта студент и методы для ввода-вывода.
   4. В классе Tree находится реализация структуры дерева и методов для работы с ним.
   5. В классе Dialog находится реализация обработчика событий.

**UML-Диаграмма**



**Реализация задачи на языке С++**

**Заголовочный файл Dialog.h**

#pragma once

#include "Tree.h"

#include "Event.h"

class Dialog : public Tree

{

protected:

int EndState;

public:

Dialog();

virtual ~Dialog();

virtual void GetEvent(Event& event); // Получить событие

virtual int Execute(); // Главынй цикл обработки событий

virtual void HandleEvent(Event& event); // Обработчик

virtual void ClearEvent(Event& event); // Очистить событие

int Valid(); // Проверка атрибута EndState

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

};

**Файл с описанием методов класса Dialog.cpp**

#include "Dialog.h"

#include "Event.h"

#include <iostream>

#include <string>

using namespace std;

Dialog::Dialog() : Tree()

{

EndState = 0;

}

Dialog::~Dialog()

{

}

void Dialog::GetEvent(Event& event)

{

string OperationCodes = "+-&/qam";

string Str;

string Parameter;

char Code;

cout << "Введите код операции\n\n";

cout << "m целое\_число - создать группу из некоторого количества элементов\n";

cout << "+ - добавить элемент в группу\n";

cout << "- - удалить элемент из группы\n";

cout << "s - вывод информации об элементах группы\n";

cout << "z целое\_число - вывод информации об имени элемента с заданным номером\n";

cout << "q - конец работы\n\n";

cout << "> ";

getline(cin, Str);

Code = Str[0];

if (OperationCodes.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 'z':

{

event.command = cmElem;

break;

}

case 'q':

{

event.command = cmQuit;

break;

}

default:

{

cout << "\nОперация не выполнена!\n\n";

}

}

}

if (Str.length() > 1)

{

Parameter = Str.substr(1, Str.length() - 1);

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

if (A < 0)

{

cout << "\nЧисло элементов в группе не может быть отрицательным\n\n";

event.what = evNothing;

}

else

{

event.parameter = A;

}

}

else

{

if (Str.length() == 0)

{

event.what = evNothing;

}

}

}

int Dialog::Execute()

{

Event event;

do

{

EndState = 0;

GetEvent(event); // Получение события

HandleEvent(event); // Обработка события

} while (!Valid());

return EndState;

}

int Dialog::Valid()

{

if (EndState == 0)

{

return 0;

}

else

{

return 1;

}

}

void Dialog::ClearEvent(Event& event)

{

event.what = evNothing;

}

void Dialog::EndExec()

{

cout << "\nВыход из программы...\n\n";

EndState = 1;

}

void Dialog::HandleEvent(Event& event)

{

if (event.what == evMessage)

{

switch (event.command)

{

case cmMake: // Создание группы

{

if (event.parameter > 0)

{

Size = event.parameter;

Beg = new Object \* [Size];

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

{

Beg[i] = nullptr;

}

ClearEvent(event);

cout << endl;

}

else

{

cout << "\nВы не ввели количество элементов! Введите код операции в соответствии с шаблоном m целое\_число\n\n";

}

break;

}

case cmAdd: // Добавление элемента в группу

{

Add();

ClearEvent(event);

break;

}

case cmDel: // Удаление элемента из группы

{

Del();

ClearEvent(event);

break;

}

case cmShow: // Вывод элементов группы

{

Show();

ClearEvent(event);

break;

}

case cmElem: // Вывод элемента группы по номеру

{

int Index = event.parameter;

Elem(Index);

ClearEvent(event);

break;

}

case cmQuit: // Выход из программы

{

EndExec();

ClearEvent(event);

break;

}

}

}

}

**Заголовочный файл Tree.h**

#pragma once

#include "Object.h"

class Tree

{

protected:

Object\*\* Beg;

int Size;

int Cur;

public:

Tree();

Tree(int N);

~Tree();

void Add();

void Del();

void Show();

void Elem(int Index);

int operator() ();

};

**Файл с описанием методов класса Tree.cpp**

#include "Tree.h"

#include "Human.h"

#include "Employee.h"

#include <iostream>

using namespace std;

Tree::Tree()

{

Beg = 0;

Cur = 0;

Size = 0;

}

Tree::Tree(int N)

{

Beg = new Object \* [N];

Object\* ptr = nullptr;

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

{

Beg[i] = ptr;

}

Cur = 0;

Size = N;

}

Tree::~Tree()

{

if (Beg != 0)

{

delete[] Beg;

}

Beg = 0;

}

void Tree::Add()

{

if (Size > 0)

{

Object\* ptr;

cout << "\nВыберите класс нового объекта:\n\n";

cout << "1. Класс Human\n";

cout << "2. Класс Employee\n";

cout << "\nВведите номер операции: ";

int Code;

cin >> Code;

if (Code == 1)

{

Human\* elem1 = new (Human);

elem1->Input();

ptr = elem1;

if (Cur < Size)

{

Beg[Cur] = ptr;

Cur++;

}

else

{

Size++;

Beg[Cur] = ptr;

Cur++;

}

cin.ignore();

}

else

{

if (Code == 2)

{

Employee\* elem2 = new Employee;

elem2->Input();

ptr = elem2;

if (Cur < Size)

{

Beg[Cur] = ptr;

Cur++;

}

else

{

Size++;

Beg[Cur] = ptr;

Cur++;

}

cin.ignore();

}

else

{

cout << "\nОшибка: введен неверный номер. Элемент не может быть добавлен...\n\n";

}

}

}

else

{

cout << "\nВы не создали группу! Выполните команду m целое\_число, прежде чем работать с другими командами (исключение: q)\n\n";

}

}

void Tree::Show()

{

if (Size > 0)

{

if (Beg[0] == nullptr)

{

cout << "\nМножество не заполнено...\n\n";

}

else

{

cout << "\nВывожу элементы дерева...\n\n";

Object\*\* ptr = Beg;

int i = 0;

while (i < Size)

{

cout << "Объект # " << i + 1 << endl;

if (\*ptr != nullptr)

{

(\*ptr)->Show();

}

else

{

cout << "\nОбъект пуст!\n\n";

}

ptr++;

i++;

}

}

}

else

{

cout << "\nВы не создали группу! Выполните команду m целое\_число, прежде чем работать с другими командами (исключение: q)\n\n";

}

}

void Tree::Elem(int Index)

{

Index -= 1; // т.к номер элемента

if (Size > 0)

{

if (Index >= Size)

{

cout << "\nИндекс за пределами размерности дерева!\n\n";

}

else

{

Object\* ptr = Beg[Index];

if (ptr != nullptr)

{

cout << "\nВывожу данные элемента с индексом " << Index << endl << endl;

ptr->Show();

}

else

{

cout << "\nЭлемента под данным индексом не существует. Заполните данный элемент!\n\n";

}

}

}

else

{

cout << "Вы не создали группу! Выполните команду m целое\_число, прежде чем работать с другими командами (исключение: q)\n";

}

}

int Tree::operator () ()

{

return Size;

}

void Tree::Del() // "Удаление", память не освобождается!

{

if (Size > 0)

{

int temp = Size - 1;

while (Beg[temp] == nullptr)

{

Size--;

temp--;

}

Size--;

Cur--;

}

else

{

cout << "Удаление невозможно: размер группы равен 0 или группа не создана\n";

Size = 0;

}

}

**Заголовочный файл Human.h**

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Human : public Object

{

protected:

string Name;

int Age;

public:

Human();

Human(string name, int age);

Human(const Human& Ptr);

virtual ~Human();

void Show();

void Input();

string Get\_Name() { return Name; }

int Get\_Age() { return Age; }

void Set\_Name(string name) { Name = name; }

void Set\_Age(int age) { Age = age; }

Human& operator =(const Human& Ptr);

};

**Файл с описанием методов класса Human.cpp**

#include "Human.h"

#include <iostream>

using namespace std;

Human::Human()

{

Name = "";

Age = 0;

}

Human::Human(string name, int age)

{

Name = name;

Age = age;

}

Human::Human(const Human& Ptr)

{

Name = Ptr.Name;

Age = Ptr.Age;

}

Human::~Human()

{

}

Human& Human::operator =(const Human& Ptr)

{

if (&Ptr != this)

{

Name = Ptr.Name;

Age = Ptr.Age;

}

return \*this;

}

void Human::Show()

{

cout << "\nВывод атрибутов...\n";

cout << "Имя: " << Name << endl;

cout << "Возраст: " << Age << endl << endl;

}

void Human::Input()

{

cout << "\nВвод данных...\n\n";

cout << "Введите имя: ";

cin >> Name;

cout << "Введите возраст: ";

cin >> Age;

while (Age <= 0)

{

cout << "Ошибка: возраст не может быть отрицательным. Введите новый возраст: ";

cin >> Age;

}

cout << endl;

}

**Заголовочный файл Employee.h**

#pragma once

#include "Human.h"

#include <iostream>

#include <string>

using namespace std;

class Employee : public Human

{

protected:

float Salary;

string Position;

public:

Employee();

Employee(string name, int age, float Salary, string pos);

Employee(const Employee& Ptr);

~Employee();

float Get\_Salary() { return Salary; }

void Set\_Salary(float Salary) { this->Salary = Salary; }

string Get\_Position() { return Position; }

void Set\_Position(string pos) { this->Position = pos; }

void Show();

void Input();

Employee& operator =(const Employee& Ptr);

};

**Файл с описанием методов класса Employee.cpp**

#include "Employee.h"

#include "Employee.h"

#include <iostream>

using namespace std;

Employee::Employee() : Human()

{

Salary = 0;

Position = "-";

}

Employee::Employee(string name, int age, float Salary, string pos) : Human(name, age)

{

this->Salary = Salary;

this->Position = pos;

}

Employee::Employee(const Employee& Ptr)

{

Name = Ptr.Name;

Age = Ptr.Age;

Salary = Ptr.Salary;

Position = Ptr.Position;

}

Employee::~Employee() {}

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

{

if (&Ptr != this)

{

Name = Ptr.Name;

Age = Ptr.Age;

Salary = Ptr.Salary;

Position = Ptr.Position;

}

return \*this;

}

void Employee::Show()

{

cout << "\nВывод атрибутов...\n";

cout << "Имя: " << Name << endl;

cout << "Возраст: " << Age << endl;

cout << "ЗП: " << Salary << endl << endl;

cout << "Должность: " << Position << endl << endl;

}

void Employee::Input()

{

cout << "\nВвод данных...\n\n";

cout << "Введите имя: ";

cin >> Name;

cout << "Введите возраст: ";

cin >> Age;

while (Age <= 0)

{

cout << "Ошибка! Возраст не может быть отрицательным. Введите новый возраст: ";

cin >> Age;

}

cout << "Введите ЗП: ";

cin >> Salary;

while (Salary <= 0)

{

cout << "Очень жаль, что ЗП отрицательная. Попытайтель исправить это ^\_^ : ";

cin >> Salary;

}

cout << "Введите должность: ";

cin >> Position;

cout << endl;

}

**Заголовочный файл Object.h**

#pragma once

class Object

{

public:

Object();

virtual ~Object();

virtual void Show() = 0;

virtual void Input() = 0;

};

**Файл с главной программой main.cpp**

#include <iostream>

#include "Object.h"

#include "Human.h"

#include "Employee.h"

#include "Tree.h"

#include "Dialog.h"

#include "Event.h"

using namespace std;

int main()

{

system("chcp 1251>nul");

Dialog D;

D.Execute();

}

**Ответы на вопросы**

1. Класс, содержащий множество объектов другого класса
2. Class List {list <Student> data; int size;}
3. List() {size=0;}
4. ~List() {delete data;}
5. Void Show() {for (int i=0;i<size;i++) cout<<data[i];}
6. Ассоциация
7. Чтобы применять один указатель к объекту любого класса
8. Структура, передаваемая между классами и пользователем
9. Индикатор пустоты сообщения, само сообщение
10. Struct e{int what;union{int message;struct{int command,a;}}}
11. 0 для пустого события, 1 для не пустого
12. Целое число в зависимости от типа передаваемого события
13. Для передач параметров события
14. Обработчик, цикл обработки, конструктор и деструктор
15. Цикл while до тех пор, пока не придет события с кодом выхода
16. Удаляет событие
17. Обрабатывает событие, запускает функцию, соответствующую типу события
18. Формирует новое событие
19. Для информирования обработчика о завершении цикла
20. Для проверки значения endstate