Министерство образования и науки Российской Федерации Федеральное агентство по образованию Федеральное государственное бюджетное образовательное учреждение высшего образования «Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторная работа №6

по курсу «Технология программирования»

# «Разработка приложение на основе ОО парадигмы»

Выполнил студент группы ИВТ-21\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Птахова А.М/

Проверила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Долженкова М.Л/

Киров 2022

**1. Задание:**

В выбранной предметной области создать иерархию классов состоящую минимум из одного родительского и двух дочерних классов.  В каждом классе определить минимум два собственных член данных, две собственных, две унаследованных и две  перекрытых член функции. Разработать приложение демонстрирующее принципы полиморфизма, наследования и инкапсуляции.

**2. Разработка программы**

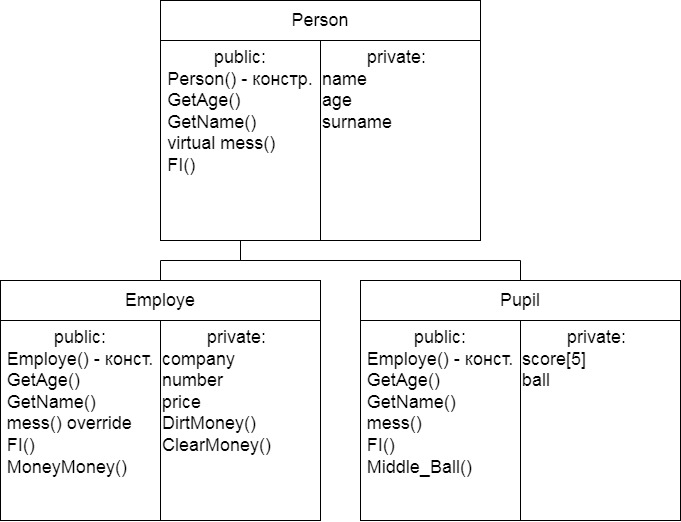


Рисунок 1 – иерархия класса

**3. Листинг программы**

#include <iostream>

#include <string>

#include <cmath>

using std::string;

using std::cout;

using std::endl;

using std::cin;

class Person // Человек - родитель

{

public:

Person(std::string n, string s) //n-name, s-surname, a-age

{

//cout << "Получаем имя, фамилию и возраст из класса Person\n";

name = n;

surname = s;

//GetAge();

//mess();

}

void GetAge() // checking

{

bool flag=false;

do {

cout << "Enter age: ";

cin >> age;

if (cin.fail())

{

flag = true;

cout << "Error!\n";

cin.clear();

cin.ignore(255, '\n');

//break;

}

else if (age<0 || age>100)

{

flag = true;

cout << "Error!\n";

cin.clear();

cin.ignore(255, '\n');

//break;

}

else { flag = false; }

} while (flag==true);

}

string GetName()

{

return name;

}

virtual void mess()

{

cout << "Name: " << name << " " << surname << "\tAge: " << age << "\n";

}

void FI()

{

cout << name[0] << surname[0] << endl;

}

private:

string name;

string surname;

int age;

};

class Employee:public Person

{

public:

Employee(string n, string s, string c) :Person(n,s) // c - company, count - number of working days

{

//cout << "This is constructor Employee\n";

company = c;

}

void GetCount()

{

bool flag = false;

do {

cout << "Enter number of working days: ";

cin >> number;

if (cin.fail())

{

flag = true;

cout << "Error!\n";

cin.clear();

cin.ignore(255, '\n');

//break;

}

else if (number < 0 || number>20)

{

flag = true;

cout << "Error!\n";

cin.clear();

cin.ignore(255, '\n');

//break;

}

else flag = false;

} while (flag==true);

}

void MoneyMoney()

{

DirtMoney();

cout << "Salary after tax " << ClearMoney() << "\n";

}

void mess() override

{

Person::mess();

std::cout << "Company: " << company << std::endl;

}

private:

string company;

int number;

double price;

void DirtMoney()

{

//cout << "This is private method of Employee";

if (number < 20 && number > 5)

{

price = 500 / 30 \* number;

cout << "Salary before tax " << price << " radishs\n";

}

else

{

cout << "Error!";

price = 0;

}

}

double ClearMoney()

{

return (price - price \* 0.13);

}

};

class pupil :public Person

{

int score[5];

double ball;

public:

pupil(string n, string s) :Person(n, s)

{

cout << "Enter grade: \n";

for (int i = 0; i < 5; i++)

{

bool flag = false;

do {

cin >> score[i];

if (cin.fail())

{

flag = true;

cout << "Error!\n";

cin.clear();

cin.ignore(255, '\n');

//break;

}

else if (score[i]<=0 || score[i]>5)

{

flag = true;

cout << "Error!\n";

cin.clear();

cin.ignore(255, '\n');

//break;

}

else flag = false;

}while (flag==true);

}

}

double Middle\_ball()

{

int sum = 0;

int k = 0;

for (int i = 0; i < 5; i++)

{

sum = sum + score[i];

k++;

}

ball = sum / k;

return round(ball);

}

};

int main()

{

string name, surname, company;

int age, count;

int n;

cout << "Select character: \n";

cout << "1. Person\n2. Employe\n3. Pupil\n4. Exit\n";

cin >> n;

while (n != 4)

{

switch (n) {

case 1:

{

int f;

cout << "Enter name: ";

cin >> name;

cout << "Enter Surname: ";

cin >> surname;

Person\* tom = new Person(name, surname);

tom->GetAge();

cout << "Choose an action: \n";

cout << "1. get name\n2. display a message\n3. get initials\n";

cin >> f;

switch (f)

{

case 1:

cout << tom->GetName() << endl;

break;

case 2:

tom->mess();

break;

case 3:

tom->FI();

break;

default:

break;

}

}

break;

case 2:

{

int f;

cout << "Enter name: ";

cin >> name;

cout << "Enter Surname: ";

cin >> surname;

cout << "Enter company name: ";

cin >> company;

Person\* bob = new Employee(name,surname, company);

((Employee\*)bob)->GetAge();

((Employee\*)bob)->GetCount();

cout << "Choose an action: \n";

cout << "1. get name\n2. display a message\n3. get initials\n4. Calculate salary\n";

cin >> f;

switch (f)

{

case 1:

cout << ((Employee\*)bob)->GetName() << endl;

break;

case 2:

((Employee\*)bob)->mess();

break;

case 3:

((Employee\*)bob)->FI();

break;

case 4:

((Employee\*)bob)->MoneyMoney();

break;

default:

break;

}

break;

}

case 3:

{

int f;

cout << "Enter name: ";

cin >> name;

cout << "Enter Surname: ";

cin >> surname;

Person\* james = new pupil(name, surname);

((pupil\*)james)->GetAge();

cout << "Choose an action: \n";

cout << "1. get name\n2. display a message\n3. get initials\n4. Calculate GPA\n";

cin >> f;

switch (f)

{

case 1:

cout << ((pupil\*)james)->GetName() << endl;

break;

case 2:

((pupil\*)james)->mess();

break;

case 3:

((pupil\*)james)->FI();

break;

case 4:

cout << ((pupil\*)james)->Middle\_ball() << endl;

break;

default:

break;

}

break;

}

default:

break;

}

cout << "Select character: \n";

cout << "1. Person\n2. Employe\n3. Pupil\n4. Exit\n";

cin >> n;

}

return 0;

}

**4. Вывод**

В ходе выполнения лабораторной работы были изучены принципы ООП: инкапсуляция, полиморфизм и наследование.

Для реализации инкапсуляции использовался спецификатор доступа private, скрывающий данные и (или) код, манипулирующий с этими данными. Главная цель инкапсуляции – защита от внешнего вмешательства и неправильного использования.

Полиморфизм продемонстрирован при помощи виртуальной функции mess, которая переопределена в производном классе Employee. При переопределении стоит учитывать, что функция в производном классе должна иметь столько же параметров и возвращать объект такого же типа, как и в базовом классе.

Оставшийся принцип позволяет наследовать функциональность базового класса в производном классе. Это позволяет не переписывать одну и ту же функцию для разных классов. Стоит отметить, что наследуются только данные и функции, которые имеют спецификаторы private, protected.