Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет

им. И.Раззакова

Факультет информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Направление: 710400 «Программная инженерия»

Дисциплина: «Объектно – ориентированное программирование»

**ОТЧЕТ**

По лабораторной работе №6.

Тема: «Наследование»

                                                                       Выполнила: студент группы

ПИ(б)-2-21 Марат к. Жылдыз

Проверил: Мусабаев Э. Б.

**Бишкек – 2024**

**Лабораторная работа №6.**

**Тема: Наследование.**

**Задания:**

1. Проблему экономии времени, доходов, а также увеличения надежности создаваемой объектно-ориентированной программы можно решить с помощью одного из главных принципов ООП - наследования, используя для создания класса-потомка готовый протестированный и отлаженный код класса-родителя. Для решения этой проблемы используйте преимущества **простого наследования**. Для этого создайте родительский класс работник (**Employee)**, протестируйте и отладьте его. В описании этого класса есть поля**:** имя работника – name, занимаемая должность - position, заработная плата за месяц- salary;

методы: конструктор по умолчанию, конструктор с аргументами, деструктор, расчет заработной платы за год, размещение информации о классе-родителе на консоли.

На основе отлаженного класса-родителя **Employee** создайте с помощью **механизма наследования** **public** класс-потомок менеджер (**Manager)**.

Класс - потомок Manager состоит из**:**

полей**:** ежегодный бонус **(annual\_bonus)**, машина компании (**company\_car[64]**)**,** опционы на акции (**stock\_options**) типа (**int**);

методов: конструктор по умолчанию, конструктор с аргументами, деструктор, размещение информации о классе-потомке на консоли.

*Определитесь с идентификаторами доступа к членам класса, не нарушая основного принципа ООП – инкапсуляции в наследовании.*

**Код программы:**

//#include<iostream>

#include<string>

#include<Windows.h>

using namespace std;

class Employee

{

private:

string name;

string position;

protected:

int salary;

public:

Employee() {}

Employee(string l, int j, string k) {

name = l;

salary = j;

position = k;

}

~Employee() {}

void salary1(int n) {

cout << "Заработная плата за год: " << salary \* 12 + n << endl;

}

void shown() {

cout << "Имя: " << name << endl;

cout << "Должность: " << position << endl;

cout << "Зарплата на месяц: " << salary << endl;

}

};

class Manager:public Employee {

private:

int annual\_bonus;

string company\_car;

int stock\_options;

public:

Manager(){

}

Manager(string l, string k, int j,int n,string m,int i):Employee(l,j,k) {

annual\_bonus = n;

company\_car = m;

stock\_options = i;

}

~Manager() {}

void show() {

shown();

cout << "Ежегодный бонус: " << annual\_bonus << endl;

cout << "Машина компании: " << company\_car << endl;

cout << "Опционы на акции: " << stock\_options <<"%" << endl;

salary1(annual\_bonus);

}

};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

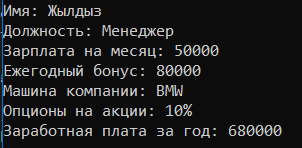
Manager a("Жылдыз", "Менеджер", 50000,80000,"BMW",10);

a.show();

return 0;

}

**Результат программы:**

****

**2.** Используйте преимущества **множественного наследования,** а именно: разработайте, протестируйте и отладьте родительские классы Дисплей **(*Display*)** и Материнская плата (M***otherBoard),*** и на их основе создайте класс-потомокКомпьютер **(*Сomputer*)**:

В описании *класса-родителя* ***Display*** есть

поля: тип монитора char type[32], количество цветов long colors, разрешение по оси х int x\_resolution, разрешение по оси у int y\_resolution.

методы: конструктор с аргументами; размещение информации о классе-родителе на консоли.

В описании *класса-родителя* ***MotherBoard*** *имеет*

поля: тип процессора int processor, скорость процессора int speed, объем оперативной памяти int RAM;

методы: конструктор с аргументами; размещение информации о классе-родителе на консоли.

- *Класс-потомок* ***Сomputer*** создать на основе родительских классовс помощью механизма наследования **public** и отобразить его на экране**,** добавив:

поля: марка компьютера (name [64]), объем жесткого диска (hard\_disk**)**.

методы: конструктор с аргументами, метод Show().

**Код программы:**

#include<iostream>

#include<string>

#include<Windows.h>

using namespace std;

class Display{

private:

string type;

long colors;

int x\_resolution;

int y\_resolution;

public:

Display(string i, long j, int k, int l) {

type = i;

colors = j;

x\_resolution = k;

y\_resolution = l;

}

~Display(){}

void shown(){

cout << "Тип монитора: " << type << endl;

cout << "Количество цветов: " << colors << endl;

cout << "разрешение по оси х: " << x\_resolution << endl;

cout << "разрешение по оси y: " << y\_resolution << endl;

}

};

class MotherBoard{

private:

string processor;

double speed;

int RAM;

public:

MotherBoard(string a, double b, int c) {

processor = a;

speed = b;

RAM = c;

}

~MotherBoard(){}

void showm() {

cout << "Тип процессора: " << processor << endl;

cout << "скорость процессора: " << speed << " ГГц" << endl;

cout << "объем оперативной памяти: " << RAM <<" ГБ"<< endl;

}

};

class Computer :public Display, public MotherBoard {

private:

string name;

int hard\_disk;

public:

Computer(string x, int z, string a, double b, int c, string i, long j, int k, int l) :Display(i, j, l, k), MotherBoard(a, b, c) {

name = x;

hard\_disk = z;

}

~Computer(){}

void Show() {

cout << "марка компьютера: " << name << endl;

cout << "объем жесткого диска: " << hard\_disk << " ГБ" << endl;

showm();

shown();

}

};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

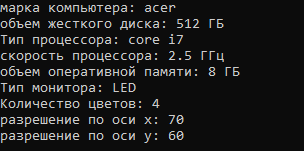
Computer a("acer", 512, "core i7", 2.5, 8, "LED", 4, 60, 70);

a.Show();

return 0;

}

**Результат программы:**



1. В задании № 1 изменить механизм наследования так, чтобы закрыть доступ к членам базового класса из производного класса. А затем восстановить этот доступ всеми известным Вам способами.

**Код программы:**

#include<iostream>

#include<string>

#include<Windows.h>

using namespace std;

class Employee

{

private:

string name;

string position ;

protected:

int salary;

public:

Employee():name(""),position(""),salary(0){}

Employee(string l, int j, string k) {

name = l;

salary = j;

position = k;

}

~Employee() {}

void salary1(int n) {

cout << "Заработная плата за год: " << salary \* 12 + n << endl;

}

void shown() {

cout << "Имя: " << name << endl;

cout << "Должность: " << position << endl;

cout << "Зарплата на месяц: " << salary << endl;

}

};

class Manager :private Employee {

private:

int annual\_bonus;

string company\_car;

int stock\_options;

public:

Employee::salary1;

Employee::shown;

Manager(): annual\_bonus(0),company\_car(""),stock\_options(0) {

}

Manager(string l, string k, int j, int n, string m, int i) :Employee(l, j, k) {

annual\_bonus = n;

company\_car = m;

stock\_options = i;

}

~Manager() {}

void show() {

shown();

cout << "Ежегодный бонус: " << annual\_bonus << endl;

cout << "Машина компании: " << company\_car << endl;

cout << "Опционы на акции: " << stock\_options << "%" << endl;

salary1(annual\_bonus);

}

};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

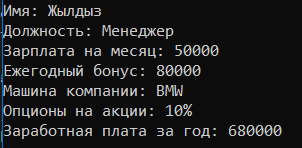
Manager a("Жылдыз", "Менеджер", 50000, 80000, "BMW", 10);

a.show();

return 0;

}

**Результат программы:**

****