**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Дисциплина «**Объектно-Ориентированное Программирование**»

**Отчет**

**Лабораторная работа №6**

Выполнил: студент группы ПИ-2-21

Газиев Давид

Проверил: Мусабаев Э. Б.

**Бишкек 2024**

**Задание 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 - занимаемая должность - position[64], 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.

**Код программы**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

string position;

double salary;

public:

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

Employee(const string& name, const string& position, double salary)

: name(name), position(position), salary(salary) {}

virtual ~Employee() {}

double calculateAnnualSalary() const {

return salary \* 12;

}

virtual void display() const {

cout << "Имя: " << name << endl;

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

cout << "Заработная плата за месяц: " << salary << endl;

}

};

class Manager : public Employee {

private:

double annual\_bonus;

string company\_car;

int stock\_options;

public:

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

Manager(const string& name, const string& position, double salary, double bonus, const string& car, int options)

: Employee(name, position, salary), annual\_bonus(bonus), company\_car(car), stock\_options(options) {}

~Manager() override {}

void display() const override {

Employee::display();

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

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

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

}

};

int main() {

setlocale(LC\_ALL, "ru");

Employee emp1("Иванов", "Программист", 5000);

cout << "Информация о работнике:\n";

emp1.display();

cout << "Годовая заработная плата: " << emp1.calculateAnnualSalary() << endl;

Manager manager1("Петров", "Менеджер", 6000, 2000, "BMW", 1000);

cout << "\nИнформация о менеджере:\n";

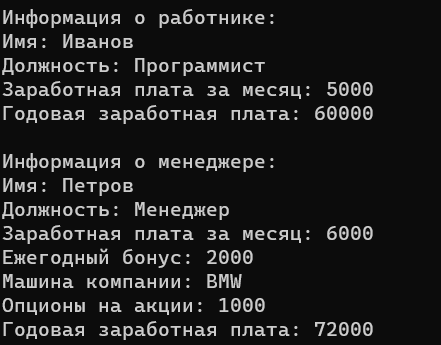
manager1.display();

cout << "Годовая заработная плата: " << manager1.calculateAnnualSalary() << endl;

return 0;

}

**Проверка выполнения**

****

**Задание 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>

using namespace std;

class Display {

protected:

char type[32];

long colors;

int x\_resolution;

int y\_resolution;

public:

Display(const char\* t, long c, int x\_res, int y\_res)

: colors(c), x\_resolution(x\_res), y\_resolution(y\_res) {

strncpy\_s(type, t, sizeof(type));

type[sizeof(type) - 1] = '\0';

}

void display() const {

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

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

cout << "Разрешение по оси X: " << x\_resolution << endl;

cout << "Разрешение по оси Y: " << y\_resolution << endl;

}

};

class MotherBoard {

protected:

int processor;

int speed;

int RAM;

public:

MotherBoard(int p, int s, int r)

: processor(p), speed(s), RAM(r) {}

void display() const {

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

cout << "Скорость процессора: " << speed << " GHz" << endl;

cout << "Объем оперативной памяти: " << RAM << " GB" << endl;

}

};

class Computer : public Display, public MotherBoard {

private:

char name[64];

int hard\_disk;

public:

Computer(const char\* n, int hd, const char\* t, long c, int x\_res, int y\_res, int p, int s, int r)

: Display(t, c, x\_res, y\_res), MotherBoard(p, s, r), hard\_disk(hd) {

strncpy\_s(name, n, sizeof(name));

name[sizeof(name) - 1] = '\0';

}

void Show() const {

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

Display::display();

MotherBoard::display();

cout << "Объем жесткого диска: " << hard\_disk << " GB" << endl;

}

};

int main() {

setlocale(LC\_ALL, "ru");

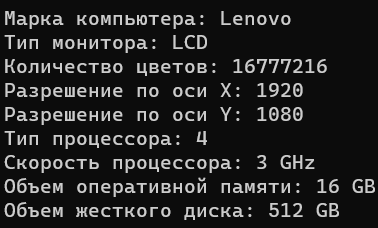
Computer comp("Lenovo", 512, "LCD", 16777216, 1920, 1080, 4, 3, 16);

comp.Show();

return 0;

}

**Проверка выполнения**

****

**Задание 3**

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>

using namespace std;

class Employee {

private:

string name;

string position;

double salary;

public:

Employee(const string& name, const string& position, double salary)

: name(name), position(position), salary(salary) {}

void display() const {

cout << "Имя: " << name << endl;

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

cout << "Заработная плата за месяц: " << salary << endl;

}

};

class ManagerPrivate : private Employee {

public:

ManagerPrivate(const string& name, const string& position, double salary)

: Employee(name, position, salary) {}

void display() const {

cout << "Информация о менеджере (защищенный доступ):\n";

Employee::display();

}

};

class AnotherClass {

public:

void displayEmployee(const ManagerPrivate& manager) const {

cout << "Информация о менеджере (дружественный класс, защищенный доступ):\n";

manager.display();

}

};

int main() {

setlocale(LC\_ALL, "ru");

ManagerPrivate managerPrivate("Иванов", "Менеджер", 5000);

managerPrivate.display();

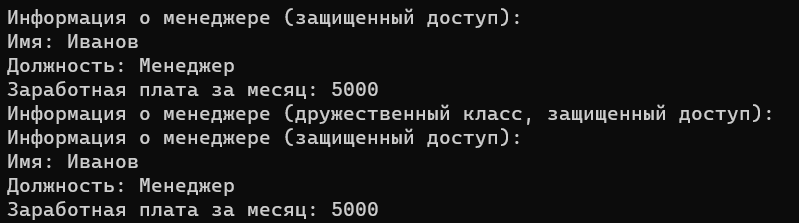
AnotherClass anotherClass;

anotherClass.displayEmployee(managerPrivate);

return 0;

}

**Проверка выполнения**

****